IEC 62541-5

Edition 2.0 2015-03

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 5: Information Model

Architecture unifiée OPC – Partie 5: Modèle d'informations

IEC 62541-5:2015-03(en-fr)

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

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

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

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 It also gives information on projects, replaced and withdrawn publications.

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

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

IEC Glossary - std.iec.ch/glossary
More than 60 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

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

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

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

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

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

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

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

Glossaire IEC - std.iec.ch/glossary
Plus de 60 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

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

IEC 62541-5

Edition 2.0 2015-03

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 5: Information Model

Architecture unifiée OPC – Partie 5: Modèle d'informations

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

ICS 25.040.40; 35.100

ISBN 978-2-8322-2384-0

CONTENTS

FOREWORD 12
1 Scope 14
2 Normative references 14
3 Terms, definitions and conventions 14
3.1 Terms and definitions 14
3.2 Abbreviations and symbols 14
3.3 Conventions for Node descriptions 15
4 NodeIds and BrowseNames 16
4.1 NodeIds 16
4.2 BrowseNames 16
5 Common Attributes 17
5.1 General 17
5.2 Objects 17
5.3 Variables 17
5.4 VariableTypes 17
6 Standard ObjectTypes 18
6.1 General 18
6.2 BaseObjectType 18
6.3 ObjectTypes for the Server Object 18
6.3.1 ServerType 18
6.3.2 ServerCapabilitiesType 20
6.3.3 ServerDiagnosticsType 22
6.3.4 SessionsDiagnosticsSummaryType 23
6.3.5 SessionDiagnosticsObjectType 24
6.3.6 VendorServerInfoType 25
6.3.7 ServerRedundancyType 25
6.3.8 TransparentRedundancyType 25
6.3.9 NonTransparentRedundancyType 26
6.3.10 NonTransparentNetworkRedundancyType 26
6.3.11 OperationLimitsType 27
6.3.12 AddressSpaceFileType 29
6.3.13 NamespaceMetadataType 29
6.3.14 NamespacesType 31
6.4 ObjectTypes used as EventTypes 31
6.4.1 General 31
6.4.2 BaseEventType 31
6.4.3 AuditEventType 33
6.4.4 AuditSecurityEventType 34
6.4.5 AuditChannelEventType 35
6.4.6 AuditOpenSecureChannelEventType 35
6.4.7 AuditSessionEventType 36
6.4.8 AuditCreateSessionEventType 36
6.4.9 AuditUrlMismatchEventType 37
6.4.10 AuditActivateSessionEventType 38
6.4.11 AuditCancelEventType 38
6.4.12 AuditCertificateEventType 39

6.4.13 AuditCertificateDataMismatchEventType 39
6.4.14 AuditCertificateExpiredEventType 39
6.4.15 AuditCertificateInvalidEventType 40
6.4.16 AuditCertificateUntrustedEventType 40
6.4.17 AuditCertificateRevokedEventType 40
6.4.18 AuditCertificateMismatchEventType 41
6.4.19 AuditNodeManagementEventType 41
6.4.20 AuditAddNodesEventType 42
6.4.21 AuditDeleteNodesEventType 42
6.4.22 AuditAddReferencesEventType 42
6.4.23 AuditDeleteReferencesEventType 43
6.4.24 AuditUpdateEventType 43
6.4.25 AuditWriteUpdateEventType 44
6.4.26 AuditHistoryUpdateEventType 44
6.4.27 AuditUpdateMethodEventType 45
6.4.28 SystemEventType 45
6.4.29 DeviceFailureEventType 45
6.4.30 SystemStatusChangeEventType 46
6.4.31 BaseModelChangeEventType 46
6.4.32 GeneralModelChangeEventType 46
6.4.33 SemanticChangeEventType 47
6.4.34 EventQueueOverflowEventType 47
6.4.35 ProgressEventType 48
6.5 ModellingRuleType 48
6.6 FolderType 48
6.7 DataTypeEncodingType 49
6.8 DataTypeSystemType 49
6.9 AggregateFunctionType 49
7 Standard VariableTypes 50
7.1 General 50
7.2 BaseVariableType 50
7.3 PropertyType 50
7.4 BaseDataVariableType 50
7.5 ServerVendorCapabilityType 51
7.6 DataTypeDictionaryType 51
7.7 DataTypeDescriptionType 52
7.8 ServerStatusType 52
7.9 BuildInfoType 52
7.10 ServerDiagnosticsSummaryType 53
7.11 SamplingIntervalDiagnosticsArrayType 53
7.12 SamplingIntervalDiagnosticsType 54
7.13 SubscriptionDiagnosticsArrayType 54
7.14 SubscriptionDiagnosticsType 54
7.15 SessionDiagnosticsArrayType 55
7.16 SessionDiagnosticsVariableType 56
7.17 SessionSecurityDiagnosticsArrayType 57
7.18 SessionSecurityDiagnosticsType 58
7.19 OptionSetType 58
8 Standard Objects and their Variables 59

8.1 General 59
8.2 Objects used to organise the AddressSpace structure 59
8.2.1 Overview 59
8.2.2 Root 60
8.2.3 Views 60
8.2.4 Objects 61
8.2.5 Types 61
8.2.6 ObjectTypes 62
8.2.7 VariableTypes 63
8.2.8 ReferenceTypes 64
8.2.9 DataTypes 64
8.2.10 OPC Binary 66
8.2.11 XML Schema 66
8.2.12 EventTypes 66
8.3 Server Object and its containing Objects 67
8.3.1 General 67
8.3.2 Server Object 68
8.4 ModellingRule Objects 69
8.4.1 ExposesItsArray 69
8.4.2 Mandatory 69
8.4.3 Optional 69
8.4.4 OptionalPlaceholder 70
8.4.5 MandatoryPlaceholder 70
9 Standard Methods 70
9.1 GetMonitoredItems 70
10 Standard Views 71
11 Standard ReferenceTypes 71
11.1 References 71
11.2 HierarchicalReferences 71
11.3 NonHierarchicalReferences 71
11.4 HasChild 72
11.5 Aggregates 72
11.6 Organizes 72
11.7 HasComponent 73
11.8 HasOrderedComponent 73
11.9 HasProperty 73
11.10 HasSubtype 73
11.11 HasModellingRule 74
11.12 HasTypeDefinition 74
11.13 HasEncoding 74
11.14 HasDescription 75
11.15 HasEventSource 75
11.16 HasNotifier 75
11.17 GeneratesEvent 75
11.18 AlwaysGeneratesEvent 76
12 Standard DataTypes 76
12.1 Overview 76
12.2 DataTypes defined in IEC 62541-3 76

12.3 DataTypes defined in IEC 62541-4 81
12.4 BuildInfo 82
12.5 RedundancySupport 82
12.6 ServerState 83
12.7 RedundantServerDataType 83
12.8 SamplingIntervalDiagnosticsDataType 84
12.9 ServerDiagnosticsSummaryDataType 84
12.10 ServerStatusDataType 85
12.11 SessionDiagnosticsDataType 86
12.12 SessionSecurityDiagnosticsDataType 87
12.13 ServiceCounterDataType 88
12.14 StatusResult 88
12.15 SubscriptionDiagnosticsDataType 89
12.16 ModelChangeStructureDataType 90
12.17 SemanticChangeStructureDataType 90
12.18 BitFieldMaskDataType 91
12.19 NetworkGroupDataType 91
12.20 EndpointUrlListDataType 92
Annex A (informative) Design decisions when modelling the server information 93
A.1 Overview 93
A.2 ServerType and Server Object 93
A.3 Typed complex Objects beneath the Server Object 93
A.4 Properties versus DataVariables 93
A.5 Complex Variables using complex DataTypes 94
A.6 Complex Variables having an array 94
A.7 Redundant information 94
A.8 Usage of the BaseDataVariableType 95
A.9 Subtyping 95
A.10 Extensibility mechanism 95
Annex B (normative) StateMachines 96
B.1 General 96
B.2 Examples of finite state machines 96
B.2.1 Simple state machine 96
B.2.2 State machine containing substates 97
B.3 Definition of state machine 98
B.4 Representation of state machines in the AddressSpace 98
B.4.1 Overview 98
B.4.2 StateMachineType 99
B.4.3 StateVariableType 100
B.4.4 TransitionVariableType 101
B.4.5 FiniteStateMachineType 101
B.4.6 FiniteStateVariableType 102
B.4.7 FiniteTransitionVariableType 103
B.4.8 StateType 103
B.4.9 InitialStateType 104
B.4.10 TransitionType 105
B.4.11 FromState 105
B.4.12 ToState 106
B.4.13 HasCause 106

B.4.14 HasEffect 106
B.4.15 HasSubStateMachine 107
B.4.16 TransitionEventType 107
B.4.17 AuditUpdateStateEventType 108
B.4.18 Special Restrictions on subtyping StateMachines 108
B.4.19 Specific StatusCodes for StateMachines 109
B.5 Examples of StateMachines in the AddressSpace 110
B.5.1 StateMachineType using inheritance 110
B.5.2 StateMachineType with a sub-machine using inheritance 111
B.5.3 StateMachineType using containment 112
B.5.4 Example of a StateMachine having Transition to SubStateMachine 113
Annex C (normative) File Transfer 115
C.1 Overview 115
C.2 FileType 115
C.3 Open 116
C.4 Close 117
C.5 Read 117
C.6 Write 118
C.7 GetPosition 118
C.8 SetPosition 119

Figure 1 – Standard AddressSpace Structure 59
Figure 2 – Views Organization 60
Figure 3 – Objects Organization 61
Figure 4 – ObjectTypes Organization 62
Figure 5 – VariableTypes Organization 63
Figure 6 – ReferenceType Definitions 64
Figure 7 – DataTypes Organization 65
Figure 8 – EventTypes Organization 67
Figure 9 – Excerpt of Diagnostic Information of the Server 68
Figure B.1 – Example of a simple state machine 97
Figure B.2 – Example of a state machine having a sub-machine 97
Figure B.3 – The StateMachine Information Model 99
Figure B.4 – Example of an initial State in a sub-machine 104
Figure B.5 – Example of a StateMachineType using inheritance 110
Figure B.6 – Example of a StateMachineType with a SubStateMachine using inheritance 111
Figure B.7 – Example of a StateMachineType using containment 112
Figure B.8 – Example of a state machine with transitions from sub-states 113
Figure B.9 – Example of a StateMachineType having Transition to SubStateMachine 114

Table 1 – Examples of DataTypes 15
Table 2 – Type Definition Table 16
Table 3 – Common Node Attributes 17
Table 4 – Common Object Attributes 17
Table 5 – Common Variable Attributes 17

Table 6 – Common VariableType Attributes 18
Table 7 – BaseObjectType Definition 18
Table 8 – ServerType Definition 19
Table 9 – ServerCapabilitiesType Definition 21
Table 10 – ServerDiagnosticsType Definition 23
Table 11 – SessionsDiagnosticsSummaryType Definition 24
Table 12 – SessionDiagnosticsObjectType Definition 24
Table 13 – VendorServerInfoType Definition 25
Table 14 – ServerRedundancyType Definition 25
Table 15 – TransparentRedundancyType Definition 25
Table 16 – NonTransparentRedundancyType Definition 26
Table 17 – NonTransparentNetworkRedundancyType Definition 27
Table 18 – OperationLimitsType Definition 28
Table 19 – AddressSpaceFileType Definition 29
Table 20 – NamespaceMetadataType Definition 30
Table 21 – NamespacesType Definition 31
Table 22 – BaseEventType Definition 31
Table 23 – AuditEventType Definition 34
Table 24 – AuditSecurityEventType Definition 34
Table 25 – AuditChannelEventType Definition 35
Table 26 – AuditOpenSecureChannelEventType Definition 35
Table 27 – AuditSessionEventType Definition 36
Table 28 – AuditCreateSessionEventType Definition 37
Table 29 – AuditUrlMismatchEventType Definition 37
Table 30 – AuditActivateSessionEventType Definition 38
Table 31 – AuditCancelEventType Definition 38
Table 32 – AuditCertificateEventType Definition 39
Table 33 – AuditCertificateDataMismatchEventType Definition 39
Table 34 – AuditCertificateExpiredEventType Definition 40
Table 35 – AuditCertificateInvalidEventType Definition 40
Table 36 – AuditCertificateUntrustedEventType Definition 40
Table 37 – AuditCertificateRevokedEventType Definition 41
Table 38 – AuditCertificateMismatchEventType Definition 41
Table 39 – AuditNodeManagementEventType Definition 41
Table 40 – AuditAddNodesEventType Definition 42
Table 41 – AuditDeleteNodesEventType Definition 42
Table 42 – AuditAddReferencesEventType Definition 43
Table 43 – AuditDeleteReferencesEventType Definition 43
Table 44 – AuditUpdateEventType Definition 43
Table 45 – AuditWriteUpdateEventType Definition 44
Table 46 – AuditHistoryUpdateEventType Definition 44
Table 47 – AuditUpdateMethodEventType Definition 45
Table 48 – SystemEventType Definition 45

Table 49 – DeviceFailureEventType Definition 46
Table 50 – SystemStatusChangeEventType Definition 46
Table 51 – BaseModelChangeEventType Definition 46
Table 52 – GeneralModelChangeEventType Definition 47
Table 53 – SemanticChangeEventType Definition 47
Table 54 – EventQueueOverflowEventType Definition 47
Table 55 – ProgressEventType Definition 48
Table 56 – ModellingRuleType Definition 48
Table 57 – FolderType Definition 49
Table 58 – DataTypeEncodingType Definition 49
Table 59 – DataTypeSystemType Definition 49
Table 60 – AggregateFunctionType Definition 49
Table 61 – BaseVariableType Definition 50
Table 62 – PropertyType Definition 50
Table 63 – BaseDataVariableType Definition 51
Table 64 – ServerVendorCapabilityType Definition 51
Table 65 – DataTypeDictionaryType Definition 51
Table 66 – DataTypeDescriptionType Definition 52
Table 67 – ServerStatusType Definition 52
Table 68 – BuildInfoType Definition 53
Table 69 – ServerDiagnosticsSummaryType Definition 53
Table 70 – SamplingIntervalDiagnosticsArrayType Definition 54
Table 71 – SamplingIntervalDiagnosticsType Definition 54
Table 72 – SubscriptionDiagnosticsArrayType Definition 54
Table 73 – SubscriptionDiagnosticsType Definition 55
Table 74 – SessionDiagnosticsArrayType Definition 55
Table 75 – SessionDiagnosticsVariableType Definition 56
Table 76 – SessionSecurityDiagnosticsArrayType Definition 58
Table 77 – SessionSecurityDiagnosticsType Definition 58
Table 78 – OptionSetType Definition 59
Table 79 – Root Definition 60
Table 80 – Views Definition 61
Table 81 – Objects Definition 61
Table 82 – Types Definition 62
Table 83 – ObjectTypes Definition 63
Table 84 – VariableTypes Definition 63
Table 85 – ReferenceTypes Definition 64
Table 86 – DataTypes Definition 66
Table 87 – OPC Binary Definition 66
Table 88 – XML Schema Definition 66
Table 89 – EventTypes Definition 67
Table 90 – Server Definition 69
Table 91 – ExposesItsArray Definition 69

Table 92 – Mandatory Definition 69
Table 93 – Optional Definition 70
Table 94 – OptionalPlaceholder Definition 70
Table 95 – MandatoryPlaceholder Definition 70
Table 96 – GetMonitoredItems Method AddressSpace Definition 71
Table 97 – References ReferenceType 71
Table 98 – HierarchicalReferences ReferenceType 71
Table 99 – NonHierarchicalReferences ReferenceType 72
Table 100 – HasChild ReferenceType 72
Table 101 – Aggregates ReferenceType 72
Table 102 – Organizes ReferenceType 73
Table 103 – HasComponent ReferenceType 73
Table 104 – HasOrderedComponent ReferenceType 73
Table 105 – HasProperty ReferenceType 73
Table 106 – HasSubtype ReferenceType 74
Table 107 – HasModellingRule ReferenceType 74
Table 108 – HasTypeDefinition ReferenceType 74
Table 109 – HasEncoding ReferenceType 74
Table 110 – HasDescription ReferenceType 75
Table 111 – HasEventSource ReferenceType 75
Table 112 – HasNotifier ReferenceType 75
Table 113 – GeneratesEvent ReferenceType 76
Table 114 – AlwaysGeneratesEvent ReferenceType 76
Table 115 – IEC 62541-3 DataType Definitions 77
Table 116 – BaseDataType Definition 78
Table 117 – Structure Definition 78
Table 118 – Enumeration Definition 79
Table 119 – ByteString Definition 79
Table 120 – Number Definition 79
Table 121 – Double Definition 79
Table 122 – Integer Definition 80
Table 123 – DateTime Definition 80
Table 124 – String Definition 80
Table 125 – UInteger Definition 80
Table 126 – Image Definition 80
Table 127 – UInt64 Definition 81
Table 128 – IEC 62541-4 DataType Definitions 81
Table 129 – UserIdentityToken Definition 82
Table 130 – BuildInfo Structure 82
Table 131 – BuildInfo Definition 82
Table 132 – RedundancySupport Values 82
Table 133 – RedundancySupport Definition 83
Table 134 – ServerState Values 83

Table 135 – ServerState Definition 83
Table 136 – RedundantServerDataType Structure 83
Table 137 – RedundantServerDataType Definition 84
Table 138 – SamplingIntervalDiagnosticsDataType Structure 84
Table 139 – SamplingIntervalDiagnosticsDataType Definition 84
Table 140 – ServerDiagnosticsSummaryDataType Structure 85
Table 141 – ServerDiagnosticsSummaryDataType Definition 85
Table 142 – ServerStatusDataType Structure 85
Table 143 – ServerStatusDataType Definition 86
Table 144 – SessionDiagnosticsDataType Structure 86
Table 145 – SessionDiagnosticsDataType Definition 87
Table 146 – SessionSecurityDiagnosticsDataType Structure 88
Table 147 – SessionSecurityDiagnosticsDataType Definition 88
Table 148 – ServiceCounterDataType Structure 88
Table 149 – ServiceCounterDataType Definition 88
Table 150 – StatusResult Structure 89
Table 151 – StatusResult Definition 89
Table 152 – SubscriptionDiagnosticsDataType Structure 89
Table 153 – SubscriptionDiagnosticsDataType Definition 90
Table 154 – ModelChangeStructureDataType Structure 90
Table 155 – ModelChangeStructureDataType Definition 90
Table 156 – SemanticChangeStructureDataType Structure 91
Table 157 – SemanticChangeStructureDataType Definition 91
Table 158 – BitFieldMaskDataType Definition 91
Table 159 – NetworkGroupDataType Structure 91
Table 160 – NetworkGroupDataType Definition 91
Table 161 – EndpointUrlListDataType Structure 92
Table 162 – EndpointUrlListDataType Definition 92
Table B.1 – StateMachineType Definition 100
Table B.2 – StateVariableType Definition 100
Table B.3 – TransitionVariableType Definition 101
Table B.4 – FiniteStateMachineType Definition 102
Table B.5 – FiniteStateVariableType Definition 103
Table B.6 – FiniteTransitionVariableType Definition 103
Table B.7 – StateType Definition 104
Table B.8 – InitialStateType Definition 105
Table B.9 – TransitionType Definition 105
Table B.10 – FromState ReferenceType 105
Table B.11 – ToState ReferenceType 106
Table B.12 – HasCause ReferenceType 106
Table B.13 – HasEffect ReferenceType 107
Table B.14 – HasSubStateMachine ReferenceType 107
Table B.15 – TransitionEventType 108

Table B.16 – AuditUpdateStateEventType 108
Table B.17 – Specific StatusCodes for StateMachines 109
Table C.1 – FileType 115
Table C.2 – Open Method AddressSpace Definition 117
Table C.3 – Close Method AddressSpace Definition 117
Table C.4 – Read Method AddressSpace Definition 118
Table C.5 – Write Method AddressSpace Definition 118
Table C.6 – GetPosition Method AddressSpace Definition 119
Table C.7 – SetPosition Method AddressSpace Definition 119

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

OPC UNIFIED ARCHITECTURE –

Part 5: Information Model

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 62541-5 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

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

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

a) Defined ProgressEventType in 6.4.35 identifying the progress of an operation such as a service call (issue number 0057);
b) Defined DataType called BitFieldMaskDataType in 12.18 representing a bit field where individual fields can be written without redefining other fields (issue number 0188);
c) Delete Property SamplingRateCount in ServerDiagnosticSummaryDataType (12.9) as it was not needed (issue number 0635);

d) Added the Property “EffectiveTransitionTime” to TransitionVariableType in B.4.4 (issue number 0728);

e) Introduced VariableType OptionSetType in 7.19 representing a bit mask and text defining the semantic of the individual bits (issue number 0983);

f) Added a new EventType called SystemStatusChangeEventType in 6.4.30 that can be used to indicate connection to the underlying system is lost (issue number 1416);

g) Added properties to ServerCapabilitiesType (6.3.2) describing the max array length and string length for variables as well as added an object for operation limits (max size of arrays when calling services (e.g. read)). Added type OperationLimitsType (6.3.11) containing that information (issue number 1451);

h) Added SecureChannelId to AuditActivateSessionEventType (6.4.10) and adapted text in various places (issue number 1492);

i) Added normative Annex C defining FileType and Methods used to transfer files (issue number 1502);

j) Added a Method GetMonitoredItems on ServerType (6.3.1) to receive information on monitored items (issue 1543);

k) Removed the concept of *ModelParent* from document as it is not that useful. The *NodeId* of the *ReferenceType* will be kept not breaking existing applications (issue numbers 1555 and 1556).

l) Added meta data for namespaces in ServerType (6.3.1) and created types for managing that (issue number 1702).

m) Added representations for ModellingRules OptionalPlaceholder in 8.4.4 and MandatoryPlaceholder in 8.4.5 (issue number 1831);

n) Added new types NonTransparentNetworkRedundancyType (6.3.10), NetworkGroupDataType (12.19) and EndpointUrlListDataType (12.20) to manage HotAndMiorrored redundancy. Added more description on redundancy and updated RedundancySupport enumeration in 12.5 (issue number 2031);

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

CDV	Report on voting
65E/376/CDV	65E/404/RVC

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

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

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

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

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

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

OPC UNIFIED ARCHITECTURE –

Part 5: Information Model

1 Scope

This part of IEC 62541 defines the Information Model of the OPC Unified Architecture. The Information Model describes standardised *Nodes* of a *Server*'s *AddressSpace*. These *Nodes* are standardised types as well as standardised instances used for diagnostics or as entry points to server-specific *Nodes*. Thus, the Information Model defines the *AddressSpace* of an empty OPC UA *Server*. However, it is not expected that all *Servers* will provide all of these *Nodes*.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

IEC 62541-3, *OPC unified architecture – Part 3: Address Space Model*

IEC 62541-4, *OPC unified architecture – Part 4: Services*

IEC 62541-6, *OPC unified architecture – Part 6: Mappings*

IEC 62541-7, *OPC unified architecture – Part 7: Profiles*

IEC 62541-9, *OPC unified architecture – Part 9: Alarms and conditions*

IEC 62541-10, *OPC unified architecture – Part 10: Programs*

IEC 62541-11, *OPC unified architecture – Part 11: Historical Access*

3 Terms, definitions and conventions

3.1 Terms and definitions

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

3.1.1
ClientUserId
string that identifies the user of the client requesting an action

Note 1 to entry: The *ClientUserId* is obtained directly or indirectly from the *UserIdentityToken* passed by the *Client* in the *ActivateSession Service* call. See 6.4.3 for details.

3.2 Abbreviations and symbols

UA Unified Architecture

XML Extensible Markup Language

3.3 Conventions for Node descriptions

Node definitions are specified using tables (see Table 2).

Attributes are defined by providing the *Attribute* name and a value, or a description of the value.

References are defined by providing the *ReferenceType* name, the *BrowseName* of the *TargetNode* and its *NodeClass*.

- If the *TargetNode* is a component of the *Node* being defined in the table the *Attributes* of the composed *Node* are defined in the same row of the table.
- The *DataType* is only specified for *Variables*; "[<number>]" indicates a single-dimensional array, for multi-dimensional arrays the expression is repeated for each dimension (e.g. [2][3] for a two-dimensional array). For all arrays the *ArrayDimensions* is set as identified by <number> values. If no <number> is set, the corresponding dimension is set to 0, indicating an unknown size. If no number is provided at all the *ArrayDimensions* can be omitted. If no brackets are provided, it identifies a scalar *DataType* and the *ValueRank* is set to the corresponding value (see IEC 62541-3). In addition, *ArrayDimensions* is set to null or is omitted. If it can be Any or ScalarOrOneDimension, the value is put into "{<value>}", so either "{Any}" or "{ScalarOrOneDimension}" and the *ValueRank* is set to the corresponding value (see IEC 62541-3) and the *ArrayDimensions* is set to null or is omitted. Examples are given in Table 1.

Table 1 – Examples of DataTypes

Notation	Data-Type	Value-Rank	Array-Dimensions	Description
Int32	Int32	−1	omitted or null	A scalar Int32.
Int32[]	Int32	1	omitted or {0}	Single-dimensional array of Int32 with an unknown size.
Int32[][]	Int32	2	omitted or {0,0}	Two-dimensional array of Int32 with unknown sizes for both dimensions.
Int32[3][]	Int32	2	{3,0}	Two-dimensional array of Int32 with a size of 3 for the first dimension and an unknown size for the second dimension.
Int32[5][3]	Int32	2	{5,3}	Two-dimensional array of Int32 with a size of 5 for the first dimension and a size of 3 for the second dimension.
Int32{Any}	Int32	−2	omitted or null	An Int32 where it is unknown if it is scalar or array with any number of dimensions.
Int32{ScalarOrOneDimension}	Int32	−3	omitted or null	An Int32 where it is either a single-dimensional array or a scalar.

- The TypeDefinition is specified for *Objects* and *Variables*.
- The TypeDefinition column specifies a symbolic name for a *NodeId*, i.e. the specified *Node* points with a *HasTypeDefinition Reference* to the corresponding *Node*.
- The *ModellingRule* of the referenced component is provided by specifying the symbolic name of the rule in the *ModellingRule* column. In the *AddressSpace*, the *Node* shall use a *HasModellingRule Reference* to point to the corresponding *ModellingRule Object*.

If the *NodeId* of a *DataType* is provided, the symbolic name of the *Node* representing the *DataType* shall be used.

Nodes of all other *NodeClasses* cannot be defined in the same table; therefore only the used *ReferenceType*, their *NodeClass* and their *BrowseName* are specified. A reference to another part of this document points to their definition.

Table 2 illustrates the table. If no components are provided, the DataType, TypeDefinition and ModellingRule columns may be omitted and only a Comment column is introduced to point to the *Node* definition.

Table 2 – Type Definition Table

Attribute	**Value**				
Attribute name	Attribute value. If it is an optional Attribute that is not set "--" will be used.				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
ReferenceType name	*NodeClass* of the Target*Node*.	*BrowseName* of the target *Node*. If the *Reference* is to be instantiated by the server, then the value of the target Node's BrowseName is "--".	*Attributes* of the referenced *Node*, only applicable for *Variables* and *Objects*.		Referenced *ModellingRule* of the referenced *Object*.
NOTE Notes referencing footnotes of the table content.					

Components of *Nodes* can be complex, that is containing components by themselves. The *TypeDefinition*, *NodeClass*, *DataType* and *ModellingRule* can be derived from the type definitions, and the symbolic name can be created as defined in 4.1. Therefore those containing components are not explicitly specified; they are implicitly specified by the type definitions.

4 NodeIds and BrowseNames

4.1 NodeIds

The *NodeIds* of all *Nodes* described in this standard are only symbolic names. IEC 62541-6 defines the actual *NodeIds*.

The symbolic name of each *Node* defined in this standard is its *BrowseName*, or, when it is part of another *Node*, the *BrowseName* of the other *Node*, a ".", and the *BrowseName* of itself. In this case "part of" means that the whole has a *HasProperty* or *HasComponent Reference* to its part. Since all *Nodes* not being part of another *Node* have a unique name in this standard, the symbolic name is unique. For example, the *ServerType* defined in 6.3.1 has the symbolic name "ServerType". One of its *InstanceDeclarations* would be identified as "ServerType.ServerCapabilities". Since this *Object* is complex, another *InstanceDeclaration* of the *ServerType* is "ServerType.ServerCapabilities.MinSupportedSampleRate". The *Server Object* defined in 8.3.2 is based on the ServerType and has the symbolic name "Server". Therefore, the instance based on the *InstanceDeclaration* described above has the symbolic name "Server.ServerCapabilities.MinSupportedSampleRate".

The NamespaceIndex for all *NodeIds* defined in this standard is 0. The namespace for this NamespaceIndex is specified in IEC 62541-3.

Note that this standard not only defines concrete *Nodes*, but also requires that some Nodes have to be generated, for example one for each Session running on the *Server*. The *NodeIds* of those *Nodes* are server-specific, including the Namespace. However the NamespaceIndex of those *Nodes* cannot be the NamespaceIndex 0, because they are not defined by the OPC Foundation but generated by the *Server*.

4.2 BrowseNames

The text part of the *BrowseNames* for all *Nodes* defined in this standard is specified in the tables defining the *Nodes*. The NamespaceIndex for all *BrowseNames* defined in this standard is 0.

5 Common Attributes

5.1 General

For all *Nodes* specified in this standard, the *Attributes* named in Table 3 shall be set as specified in Table 3.

Table 3 – Common Node Attributes

Attribute	Value
DisplayName	The *DisplayName* is a *LocalizedText*. Each server shall provide the *DisplayName* identical to the *BrowseName* of the *Node* for the LocaleId "en". Whether the server provides translated names for other LocaleIds is vendor specific.
Description	Optionally a vendor specific description is provided.
NodeClass	Shall reflect the *NodeClass* of the *Node.*
NodeId	The *NodeId* is described by *BrowseNames* as defined in 4.1 and defined in IEC 62541-6.
WriteMask	Optionally the *WriteMask Attribute* can be provided. If the *WriteMask Attribute* is provided, it shall set all *Attributes* to not writeable that are not said to be vendor-specific. For example, the *Description Attribute* may be set to writeable since a *Server* may provide a server-specific description for the *Node*. The *NodeId* shall not be writeable, because it is defined for each *Node* in this standard.
UserWriteMask	Optionally the *UserWriteMask Attribute* can be provided. The same rules as for the *WriteMask Attribute* apply.

5.2 Objects

For all *Objects* specified in this standard, the *Attributes* named in Table 4 shall be set as specified in Table 4.

Table 4 – Common Object Attributes

Attribute	Value
EventNotifier	Whether the *Node* can be used to subscribe to *Events* or not is vendor specific.

5.3 Variables

For all *Variables* specified in this standard, the *Attributes* named in Table 5 shall be set as specified in Table 5.

Table 5 – Common Variable Attributes

Attribute	Value
MinimumSamplingInterval	Optionally, a vendor-specific minimum sampling interval is provided.
AccessLevel	The access level for *Variables* used for type definitions is vendor-specific, for all other *Variables* defined in this standard, the access level shall allow a current read; other settings are vendor specific.
UserAccessLevel	The value for the *UserAccessLevel Attribute* is vendor-specific. It is assumed that all *Variables* can be accessed by at least one user.
Value	For *Variables* used as *InstanceDeclarations,* the value is vendor-specific; otherwise it shall represent the value described in the text.
ArrayDimensions	If the *ValueRank* does not identify an array of a specific dimension (i.e. *ValueRank* <= 0) the *ArrayDimensions* can either be set to null or the *Attribute* is missing. This behaviour is vendor-specific. If the *ValueRank* specifies an array of a specific dimension (i.e. *ValueRank* > 0) then the *ArrayDimensions Attribute* shall be specified in the table defining the *Variable*.

5.4 VariableTypes

For all *VariableTypes* specified in this standard, the *Attributes* named in Table 6 shall be set as specified in Table 6.

Table 6 – Common VariableType Attributes

Attributes	Value
Value	Optionally a vendor-specific default value can be provided.
ArrayDimensions	If the *ValueRank* does not identify an array of a specific dimension (i.e. *ValueRank* <= 0) the *ArrayDimensions* can either be set to null or the *Attribute* is missing. This behaviour is vendor-specific. If the *ValueRank* specifies an array of a specific dimension (i.e. *ValueRank* > 0) then the *ArrayDimensions Attribute* shall be specified in the table defining the *VariableType*.

6 Standard ObjectTypes

6.1 General

Typically, the components of an *ObjectType* are fixed and can be extended by subtyping. However, since each *Object* of an *ObjectType* can be extended with additional components, this standard allows extending the standard *ObjectTypes* defined in this document with additional components. Thereby, it is possible to express the additional information in the type definition that would already be contained in each *Object*. Some *ObjectTypes* already provide entry points for server-specific extensions. However, it is not allowed to restrict the components of the standard *ObjectTypes* defined in this standard. An example of extending the *ObjectTypes* is putting the standard *Property NodeVersion* defined in IEC 62541-3 into the *BaseObjectType*, stating that each *Object* of the *Server* will provide a *NodeVersion*.

6.2 BaseObjectType

The *BaseObjectType* is used as type definition whenever there is an *Object* having no more concrete type definitions available. *Servers* should avoid using this *ObjectType* and use a more specific type, if possible. This *ObjectType* is the base *ObjectType* and all other *ObjectTypes* shall either directly or indirectly inherit from it. However, it might not be possible for *Servers* to provide all *HasSubtype References* from this *ObjectType* to its subtypes, and therefore it is not required to provide this information.

There are no *References* except for *HasSubtype References* specified for this *ObjectType*. It is formally defined in Table 7.

Table 7 – BaseObjectType Definition

Attribute	Value				
BrowseName	BaseObjectType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasSubtype	ObjectType	ServerType	Defined in 6.3.1		
HasSubtype	ObjectType	ServerCapabilitiesType	Defined in 6.3.2		
HasSubtype	ObjectType	ServerDiagnosticsType	Defined in 6.3.3		
HasSubtype	ObjectType	SessionsDiagnosticsSummaryType	Defined in 6.3.4		
HasSubtype	ObjectType	SessionDiagnosticsObjectType	Defined in 6.3.5		
HasSubtype	ObjectType	VendorServerInfoType	Defined in 6.3.6		
HasSubtype	ObjectType	ServerRedundancyType	Defined in 6.3.7		
HasSubtype	ObjectType	BaseEventType	Defined in 6.4.2		
HasSubtype	ObjectType	ModellingRuleType	Defined in 6.5		
HasSubtype	ObjectType	FolderType	Defined in 6.6		
HasSubtype	ObjectType	DataTypeEncodingType	Defined in 6.7		
HasSubtype	ObjectType	DataTypeSystemType	Defined in 6.8		

6.3 ObjectTypes for the Server Object

6.3.1 ServerType

This *ObjectType* defines the capabilities supported by the OPC UA *Server*. It is formally defined in Table 8.

Table 8 – ServerType Definition

Attribute	Value				
BrowseName	ServerType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2					
HasProperty	Variable	ServerArray	String[]	PropertyType	Mandatory
HasProperty	Variable	NamespaceArray	String[]	PropertyType	Mandatory
HasComponent	Variable	ServerStatus[1]	ServerStatusDataType	ServerStatusType	Mandatory
HasProperty	Variable	ServiceLevel	Byte	PropertyType	Mandatory
HasProperty	Variable	Auditing	Boolean	PropertyType	Mandatory
HasComponent	Object	ServerCapabilities[1]	-	ServerCapabilitiesType	Mandatory
HasComponent	Object	ServerDiagnostics[1]	-	ServerDiagnosticsType	Mandatory
HasComponent	Object	VendorServerInfo	-	VendorServerInfoType	Mandatory
HasComponent	Object	ServerRedundancy[1]	-	ServerRedundancyType	Mandatory
HasComponent	Object	Namespaces	-	NamespacesType	Optional
HasComponent	Method	GetMonitoredItems	Defined in 9		Optional
NOTE Containing *Objects* and *Variables* of these *Objects* and *Variables* are defined by their *BrowseName* defined in the corresponding *TypeDefinitionNode*. The *NodeId* is defined by the composed symbolic name described in 4.1.					

ServerArray defines an array of *Server* URIs. This *Variable* is also referred to as the *server table*. Each URI in this array represents a globally-unique logical name for a *Server* within the scope of the network in which it is installed. Each OPC UA *Server* instance has a single URI that is used in the *server table* of other OPC UA *Servers*. Index 0 is reserved for the URI of the local *Server*. Values above 0 are used to identify remote *Servers* and are specific to a *Server*. IEC 62541-4 describes discovery mechanism that can be used to resolve URIs into URLs. The *Server* URI is case sensitive.

The URI of the *ServerArray* with Index 0 shall be identical to the URI of the *NamespaceArray* with Index 1, since both represent the local *Server*.

The indexes into the *server table* are referred to as *server indexes* or *server names*. They are used in OPC UA *Services* to identify *TargetNodes* of *References* that reside in remote *Servers*. Clients may read the entire table or they may read individual entries in the table. The *Server* shall not modify or delete entries of this table while any client has an open session to the *Server*, because clients may cache the *server table*. A *Server* may add entries to the *server table* even if clients are connected to the *Server*.

NamespaceArray defines an array of namespace URIs. This *Variable* is also referred as *namespace table*. The indexes into the *namespace table* are referred to as *NamespaceIndexes*. *NamespaceIndexes* are used in *NodeIds* in OPC UA *Services*, rather than the longer namespace URI. Index 0 is reserved for the OPC UA namespace, and index 1 is reserved for the local *Server*. Clients may read the entire *namespace table* or they may read individual entries in the *namespace table*. The *Server* shall not modify or delete entries of the *namespace table* while any client has an open session to the *Server*, because clients may cache the *namespace table*. A *Server* may add entries to the *namespace table* even if clients are connected to the *Server*. It is recommended that *Servers* not change the indexes of the *namespace table* but only add entries, because the client may cache *NodeIds* using the indexes. Nevertheless, it might not always be possible for *Servers* to avoid changing indexes in the *namespace table*. Clients that cache *NamespaceIndexes* of *NodeIds* should always check when starting a session to verify that the cached *NamespaceIndexes* have not changed.

ServerStatus contains elements that describe the status of the *Server*. See 12.10 for a description of its elements.

ServiceLevel describes the ability of the *Server* to provide its data to the client. The value range is from 0 to 255, where 0 indicates the worst and 255 indicates the best. The concrete

values are vendor-specific. The intent is to provide the clients an indication of availability among redundant *Servers*.

Auditing is a Boolean specifying if the *Server* is currently generating audit events. It is set to TRUE if the *Server* generates audit events, otherwise to false. The *Profiles* defined in IEC 62541-7 specify what kind of audit events are generated by the *Server*.

ServerCapabilities defines the capabilities supported by the OPC UA *Server*. See 6.3.2 for its description.

ServerDiagnostics defines diagnostic information about the OPC UA *Server*. See 6.3.3 for its description.

VendorServerInfo represents the browse entry point for vendor-defined *Server* information. This *Object* is required to be present even if there are no vendor-defined *Objects* beneath it. See 6.3.6 for its description.

ServerRedundancy describes the redundancy capabilities provided by the *Server*. This *Object* is required even if the *Server* does not provide any redundancy support. If the *Server* supports redundancy, then a subtype of *ServerRedundancyType* is used to describe its capabilities. Otherwise, it provides an *Object* of type *ServerRedundancyType* with the *Property* RedundancySupport set to none. See 6.3.7 for the description of *ServerRedundancyType*.

Namespaces provides a list of *NamespaceMetadataType Objects* with additional information about the namespaces used in the *Server*. See 6.3.14 for the description of *NamespaceMetadataType.*

The *GetMonitoredItems Method* is used to identify the *MonitoredItems* of a subscription. It is defined in 9; the intended usage is defined in IEC 62541-4.

6.3.2 ServerCapabilitiesType

This *ObjectType* defines the capabilities supported by the OPC UA *Server*. It is formally defined in Table 9.

Table 9 – ServerCapabilitiesType Definition

Attribute	Value			
BrowseName	ServerCapabilitiesType			
IsAbstract	False			
References	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2				
HasProperty	Variable	ServerProfileArray	String[] PropertyType	Mandatory
HasProperty	Variable	LocaleIdArray	LocaleId[] PropertyType	Mandatory
HasProperty	Variable	MinSupportedSampleRate	Duration PropertyType	Mandatory
HasProperty	Variable	MaxBrowseContinuationPoints	UInt16 PropertyType	Mandatory
HasProperty	Variable	MaxQueryContinuationPoints	UInt16 PropertyType	Mandatory
HasProperty	Variable	MaxHistoryContinuationPoints	UInt16 PropertyType	Mandatory
HasProperty	Variable	SoftwareCertificates	SignedSoftwareCertificate[] PropertyType	Mandatory
HasProperty	Variable	MaxArrayLength	UInt32 PropertyType	Optional
HasProperty	Variable	MaxStringLength	UInt32 PropertyType	Optional
HasComponent	Object	OperationLimits	-- OperationLimitsType	Optional
HasComponent	Object	ModellingRules	-- FolderType	Mandatory
HasComponent	Object	AggregateFunctions	-- FolderType	Mandatory
HasComponent	Variable	Vendor specific *Variables* of a subtype of the ServerVendorCapabilityType defined in 7.5		--

ServerProfileArray lists the *Profiles* that the *Server* supports. See IEC 62541-7 for the definitions of *Server Profiles*. This list should be limited to the *Profiles* the *Server* supports in its current configuration.

LocaleIdArray is an array of LocaleIds that are known to be supported by the *Server*. The *Server* might not be aware of all LocaleIds that it supports because it may provide access to underlying servers, systems or devices that do not report the LocaleIds that they support.

MinSupportedSampleRate defines the minimum supported sample rate, including 0, which is supported by the *Server*.

MaxBrowseContinuationPoints is an integer specifying the maximum number of parallel continuation points of the Browse *Service* that the *Server* can support per session. The value specifies the maximum the *Server* can support under normal circumstances, so there is no guarantee the *Server* can always support the maximum. The client should not open more Browse calls with open continuation points than exposed in this *Variable*. The value 0 indicates that the *Server* does not restrict the number of parallel continuation points the client should use.

MaxQueryContinuationPoints is an integer specifying the maximum number of parallel continuation points of the QueryFirst *Services* that the *Server* can support per session. The value specifies the maximum the *Server* can support under normal circumstances, so there is no guarantee the *Server* can always support the maximum. The client should not open more QueryFirst calls with open continuation points than exposed in this *Variable*. The value 0 indicates that the *Server* does not restrict the number of parallel continuation points the client should use.

MaxHistoryContinuationPoints is an integer specifying the maximum number of parallel continuation points of the HistoryRead *Services* that the *Server* can support per session. The value specifies the maximum the *Server* can support under normal circumstances, so there is no guarantee the *Server* can always support the maximum. The client should not open more HistoryRead calls with open continuation points than exposed in this *Variable*. The value 0 indicates that the *Server* does not restrict the number of parallel continuation points the client should use.

SoftwareCertificates is an array of *SignedSoftwareCertificates* containing all *SoftwareCertificates* supported by the *Server*. A *SoftwareCertificate* identifies capabilities of the *Server*. It contains the list of *Profiles* supported by the *Server*. *Profiles* are described in IEC 62541-7.

The *MaxArrayLength Property* indicates the maximum length of a one or multidimensional array supported by Variables of the *Server*. In a multidimensional array it indicates the overall length. For example, a three-dimensional array of 2x3x10 has the array length of 60. The *Server* might further restrict the length for individual Variables without notice to the client. *Servers* may use the *Property MaxArrayLength* defined in IEC 62541-3 on individual *DataVariables* to specify the size on individual values. The individual *Property* may have a larger or smaller value than *MaxArrayLength*.

The *MaxStringLength Property* indicates the maximum length of Strings supported by *Variables* of the *Server*. The *Server* might further restrict the String length for individual *Variables* without notice to the client. *Servers* may use the *Property MaxStringLength* defined in IEC 62541-3 on individual *DataVariables* to specify the length on individual values. The individual *Property* may have larger or smaller values than *MaxStringLength*.

OperationLimits is an entry point to access information on operation limits of the *Server*, for example the maximum length of an array in a read *Service* call.

ModellingRules is an entry point to browse to all *ModellingRules* supported by the *Server*. All *ModellingRules* supported by the *Server* should be able to be browsed starting from this *Object*.

AggregateFunctions is an entry point to browse to all *AggregateFunctions* supported by the *Server*. All *AggregateFunctions* supported by the *Server* should be able to be browsed starting from this *Object*. AggregateFunctions are Objects of *AggregateFunctionType*.

The remaining components of the *ServerCapabilitiesType* define the server-specific capabilities of the *Server*. Each is defined using a *HasComponent Reference* whose target is an instance of a vendor-defined subtype of the abstract *ServerVendorCapabilityType* (see 7.5). Each subtype of this type defines a specific *Server* capability. The *NodeIds* for these *Variables* and their *VariableTypes* are server-defined.

6.3.3 ServerDiagnosticsType

This *ObjectType* defines diagnostic information about the OPC UA *Server*. This *ObjectType* is formally defined in Table 10.

Table 10 – ServerDiagnosticsType Definition

Attribute	Value			
BrowseName	ServerDiagnosticsType			
IsAbstract	False			
References	**Node Class**	**BrowseName**	**DataType / TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2				
HasComponent	Variable	ServerDiagnosticsSummary	ServerDiagnosticsSummaryDataType ServerDiagnosticsSummaryType	Mandatory
HasComponent	Variable	SamplingIntervalDiagnosticsArray	SamplingIntervalDiagnosticsDataType[] SamplingIntervalDiagnosticsArrayType	Optional
HasComponent	Variable	SubscriptionDiagnosticsArray	SubscriptionDiagnosticsDataType[] SubscriptionDiagnosticsArrayType	Mandatory
HasComponent	Object	SessionsDiagnosticsSummary	-- SessionsDiagnosticsSummaryType	Mandatory
HasProperty	Variable	EnabledFlag	Boolean PropertyType	Mandatory

ServerDiagnosticsSummary contains diagnostic summary information for the *Server*, as defined in 12.9.

SamplingIntervalDiagnosticsArray is an array of diagnostic information per sampling rate as defined in 12.8. There is one entry for each sampling rate currently used by the *Server*. Its *TypeDefinitionNode* is the *VariableType SamplingIntervalDiagnosticsArrayType*, providing a *Variable* for each entry in the array, as defined in 7.11.

The sampling interval diagnostics are only collected by *Servers* which use a fixed set of sampling intervals. In these cases, length of the array and the set of contained *Variables* will be determined by the *Server* configuration and the *NodeId* assigned to a given sampling interval diagnostics variable shall not change as long as the *Server* configuration does not change. A *Server* may not expose the SamplingIntervalDiagnosticsArray if it does not use fixed sampling rates.

SubscriptionDiagnosticsArray is an array of Subscription diagnostic information per subscription, as defined in 12.15. There is one entry for each Notification channel actually established in the *Server*. Its *TypeDefinitionNode* is the *VariableType* SubscriptionDiagnosticsArrayType, providing a *Variable* for each entry in the array as defined in 7.13. Those *Variables* are also used as *Variables* referenced by other *Variables*.

SessionsDiagnosticsSummary contains diagnostic information per session, as defined in 6.3.4.

EnabledFlag identifies whether or not diagnostic information is collected by the *Server*. It can also be used by a client to enable or disable the collection of diagnostic information of the *Server*. The following settings of the Boolean value apply: TRUE indicates that the *Server* collects diagnostic information, and setting the value to TRUE leads to resetting and enabling the collection. FALSE indicates that no statistic information is collected, and setting the value to FALSE disables the collection without resetting the statistic values.

Static diagnostic *Nodes* that always appear in the *AddressSpace* will return Bad_NotReadable when the *Value Attribute* of such a *Node* is read or subscribed to and diagnostics are turned off. Dynamic diagnostic *Nodes* (such as the *Session Nodes*) will not appear in the *AddressSpace* when diagnostics are turned off.

6.3.4 SessionsDiagnosticsSummaryType

This *ObjectType* defines diagnostic information about the sessions of the OPC UA *Server*. This *ObjectType* is formally defined in Table 11.

Table 11 – SessionsDiagnosticsSummaryType Definition

Attribute		Value		
BrowseName		SessionsDiagnosticsSummaryType		
IsAbstract		False		
References	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2				
HasComponent	Variable	SessionDiagnosticsArray	SessionDiagnosticsDataType[] SessionDiagnosticsArrayType	Mandatory
HasComponent	Variable	SessionSecurityDiagnosticsArray	SessionSecurityDiagnosticsDataType[] SessionSecurityDiagnosticsArrayType	Mandatory
HasComponent	Object	<ClientName>	-- SessionDiagnosticsObjectType	Optional Placeholder
NOTE This row represents no *Node* in the *AddressSpace*. It is a placeholder pointing out that instances of the *ObjectType* will have those *Objects*.				

SessionDiagnosticsArray provides an array with an entry for each session in the *Server* having general diagnostic information about a session.

SessionSecurityDiagnosticsArray provides an array with an entry for each active session in the *Server* having security-related diagnostic information about a session. Since this information is security-related, it should not be made accessible to all users, but only to authorised users.

For each session of the *Server*, this *Object* also provides an *Object* representing the session, indicated by <*ClientName*>. The BrowseName could be derived from the *sessionName* defined in the *CreateSession Service* (IEC 62541-4) or some other server-specific mechanisms. It is of the *ObjectType* SessionDiagnosticsObjectType, as defined in 6.3.5.

6.3.5 SessionDiagnosticsObjectType

This *ObjectType* defines diagnostic information about a session of the OPC UA *Server*. This *ObjectType* is formally defined in Table 12.

Table 12 – SessionDiagnosticsObjectType Definition

Attribute	Value			
BrowseName	SessionDiagnosticsObjectType			
IsAbstract	False			
References	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2				
HasComponent	Variable	SessionDiagnostics	SessionDiagnosticsDataType SessionDiagnosticsVariableType	Mandatory
HasComponent	Variable	SessionSecurityDiagnostics	SessionSecurityDiagnosticsDataType SessionSecurityDiagnosticsType	Mandatory
HasComponent	Variable	SubscriptionDiagnosticsArray	SubscriptionDiagnosticsDataType[] SubscriptionDiagnosticsArrayType	Mandatory

SessionDiagnostics contains general diagnostic information about the session; the *SessionSecurityDiagnostics Variable* contains security-related diagnostic information. Because the information of the second *Variable* is security-related, it should not be made accessible to all users, but only to authorised users.

SubscriptionDiagnosticsArray is an array of Subscription diagnostic information per opened subscription, as defined in 12.15. Its *TypeDefinitionNode* is the *VariableType* SubscriptionDiagnosticsArrayType providing a *Variable* for each entry in the array, as defined in 7.13.

6.3.6 VendorServerInfoType

This *ObjectType* defines a placeholder *Object* for vendor-specific information about the OPC UA *Server*. This *ObjectType* defines an empty *ObjectType* that has no components. It shall be subtyped by vendors to define their vendor-specific information. This *ObjectType* is formally defined in Table 13.

Table 13 – VendorServerInfoType Definition

Attribute	Value				
BrowseName	VendorServerInfoType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2					

6.3.7 ServerRedundancyType

This *ObjectType* defines the redundancy capabilities supported by the OPC UA *Server*. It is formally defined in Table 14.

Table 14 – ServerRedundancyType Definition

Attribute	Value				
BrowseName	ServerRedundancyType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**Type Definition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2					
HasProperty	Variable	RedundancySupport	RedundancySupport	PropertyType	Mandatory
HasSubtype	ObjectType	TransparentRedundancyType	Defined in 6.3.8		
HasSubtype	ObjectType	NonTransparentRedundancyType	Defined in 6.3.9		

RedundancySupport indicates what redundancy is supported by the *Server*. Its values are defined in 12.5. It shall be set to NONE_0 for all instances of the *ServerRedundancyType* using the *ObjectType* directly (no subtype).

6.3.8 TransparentRedundancyType

This *ObjectType* is a subtype of *ServerRedundancyType* and is used to identify the capabilities of the OPC UA *Server* for server-controlled redundancy with a transparent switchover for the client. It is formally defined in Table 15.

Table 15 – TransparentRedundancyType Definition

Attribute	Value				
BrowseName	TransparentRedundancyType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ServerRedundancyType defined in 6.3.7, i.e. inheriting the InstanceDeclarations of that Node.					
HasProperty	Variable	CurrentServerId	String	PropertyType	Mandatory
HasProperty	Variable	RedundantServerArray	RedundantServerDataType[]	PropertyType	Mandatory

RedundancySupport is inherited from the *ServerRedundancyType*. It shall be set to TRANSPARENT_4 for all instances of the *TransparentRedundancyType*.

Although, in a transparent switchover scenario, all redundant *Servers* serve under the same URI to the client, it may be required to track the exact data source on the client. Therefore, *CurrentServerId* contains an identifier of the currently-used *Server* in the redundant set. This

Server is valid only inside a session; if a client opens several sessions, different *Servers* of the redundant set of *Servers* may serve it in different sessions. The value of the *CurrentServerId* may change due to failover or load balancing, so a client that needs to track its data source shall subscribe to this *Variable*.

As diagnostic information, the *RedundantServerArray* contains an array of available *Servers* in the redundant set; including their service levels (see 12.7). This array may change during a session.

6.3.9 NonTransparentRedundancyType

This *ObjectType* is a subtype of *ServerRedundancyType* and is used to identify the capabilities of the OPC UA *Server* for non-transparent redundancy. It is formally defined in Table 16.

Table 16 – NonTransparentRedundancyType Definition

Attribute	**Value**				
BrowseName	NonTransparentRedundancyType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the ServerRedundancyType defined in 6.3.7, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ServerUriArray	String[]	PropertyType	Mandatory
HasSubtype	ObjectType	NonTransparentNetworkRedundancyType	Defined in 6.3.10		

ServerUriArray is an array with the URI of all redundant *Servers* of the OPC UA *Server*. See IEC 62541-4 for the definition of redundancy in this standard. In a non-transparent redundancy environment, the client is responsible to subscribe to the redundant *Servers*. Therefore the Client might open a session to one or more redundant *Servers* of this array. The *SeverUriArray* shall contain the local *Server*.

RedundancySupport is inherited from the *ServerRedundancyType*. It shall be set to COLD_1, WARM_2, HOT_3 or HOT_AND_MIRRORED_5 for all instances of the *NonTransparentRedundancyType*. It defines the redundancy support provided by the *Server*. The *Client* is allowed to access the redundant *Server* only as described there, however, ”hot” switchover implies the support of “warm” switchover and “warm” switchover implies the support of “cold” switchover. Support for HotAndMirrored redundancy implies the support of “hot” switchover, however, for Servers supporting HotandMirrored redundancy it is strongly recommended that *Clients* use the HotAndMirrored mechanisms.

If the *Server* supports only a “cold” switchover, the *ServiceLevel Variable* of the *Server Object* should be considered to identify the primary *Server*. In this scenario, only the primary *Server* may be able to access the underlying system, because the underlying system may support access only from a single *Server*. In this case, all other *Servers* will be identified with a *ServiceLevel* of zero.

6.3.10 NonTransparentNetworkRedundancyType

This *ObjectType* is a subtype of *NonTransparentRedundancyType* and is used to identify the capabilities of the OPC UA *Server* for non-transparent network redundancy. It is formally defined in Table 17.

Table 17 – NonTransparentNetworkRedundancyType Definition

Attribute	Value				
BrowseName	NonTransparentNetworkRedundancyType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the NonTransparentRedundancyType defined in 6.3.9, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ServerNetworkGroups	NetworkGroupDataType[]	PropertyType	Mandatory

Clients switching between network paths to the same *Server* behave the same as HotAndMirrored redundancy. *Server* and network redundancy can be combined. In the combined approach it is important for the *Client* to know which ServerUris belong to the same *Server* representing different network paths and which ServerUris represent different Servers. Therefore, a *Server* implementing non-transparent network redundancy shall use the *NonTransparentNetworkRedundancyType* to identify its redundancy support.

RedundancySupport is inherited from the *ServerRedundancyType*. It shall be set to COLD_1, WARM_2, HOT_3 or HOT_AND_MIRRORED_5 for all instances of the *NonTransparentNetworkRedundancyType*. If no server redundancy is supported (the *ServerUriArray* only contains one entry), the *RedundancySupport* shall be set to HOT_AND_MIRRORED_5.

The *ServerNetworkGroups* contains an array of *NetworkGroupDataType*. The URIs of the *Servers* in that array (in the *serverUri* of the structure) shall be exactly the same as the ones provided in the *ServerUriArray*. However, the order might be different. Thus the array represents a list of HotAndMirrored redundant *Servers*. If a server only supports network redundancy, it has only one entry in the *ServerNetworkGroups*. The *networkPaths* in the structure represents the redundant network paths for each of the *Servers*. The *networkPaths* describes the different paths (one entry for each path) ordered by priority. Each network path contains an *endpointUrlList* having an array of Strings each containing a URL of an *Endpoint.* This allows using different protocol options for the same network path.

The *Endpoints* provided shall match with the *Endpints* provided by the *GetEndpoints Service* of the corresponding *Server*.

6.3.11 OperationLimitsType

This *ObjectType* is a subtype of *FolderType* and is used to identify the operation limits of the OPC UA *Server*. It is formally defined in Table 18.

Table 18 – OperationLimitsType Definition

Attribute	Value				
BrowseName	OperationLimitsType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the FolderType defined in 6.6, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	MaxNodesPerRead	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerHistoryReadData	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerHistoryReadEvents	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerWrite	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerHistoryUpdateData	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerHistoryUpdateEvents	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerMethodCall	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerBrowse	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerRegisterNodes	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerTranslateBrowsePaths ToNodeIds	UInt32	PropertyType	Optional
HasProperty	Variable	MaxNodesPerNodeManagement	UInt32	PropertyType	Optional
HasProperty	Variable	MaxMonitoredItemsPerCall	UInt32	PropertyType	Optional

The *MaxNodesPerRead Property* indicates the maximum size of the nodesToRead array when a *Client* calls the Read *Service*.

The *MaxNodesPerHistoryReadData Property* indicates the maximum size of the nodesToRead array when a *Client* calls the HistoryRead *Service* using the histroyReadDetails RAW, PROCESSED, MODIFIED or ATTIME.

The *MaxNodesPerHistoryReadEvents Property* indicates the maximum size of the nodesToRead array when a *Client* calls the HistoryRead *Service* using the historyReadDetails EVENTS.

The *MaxNodesPerWrite Property* indicates the maximum size of the nodesToWrite array when a *Client* calls the Write *Service*.

The *MaxNodesPerHistoryUpdateData Property* indicates the maximum size of the historyUpdateDetails array supported by the *Server* when a *Client* calls the HistoryUpdate *Service* using historyReadDetails RAW, PROCESSED, MODIFIED or ATTIME.

The *MaxNodesPerHistoryUpdateEvents Property* indicates the maximum size of the historyUpdateDetails array when a *Client* calls the HistoryUpdate *Service* using historyReadDetails EVENTS.

The *MaxNodesPerMethodCall Property* indicates the maximum size of the methodsToCall array when a *Client* calls the Call *Service*.

The *MaxNodesPerBrowse Property* indicates the maximum size of the nodesToBrowse array when calling the Browse *Service* or the continuationPoints array when a *Client* calls the BrowseNext *Service*.

The *MaxNodesPerRegisterNodes Property* indicates the maximum size of the nodesToRegister array when a *Client* calls the RegisterNodes *Service* and the maximum size of the nodesToUnregister when calling the UnregisterNodes *Service*.

The *MaxNodesPerTranslateBrowsePathsToNodeIds Property* indicates the maximum size of the browsePaths array when a *Client* calls the TranslateBrowsePathsToNodeIds *Service*.

The *MaxNodesPerNodeManagement Property* indicates the maximum size of the nodesToAdd array when a *Client* calls the AddNodes *Service*, the maximum size of the referencesToAdd

array when a *Client* calls the AddReferences *Service*, the maximum size of the nodesToDelete array when a *Client* calls the DeleteNodes *Service*, and the maximum size of the referencesToDelete array when a *Client* calls the DeleteReferences *Service*.

The *MaxMonitoredItemsPerCall Property* indicates the maximum size of the itemsToCreate array when a *Client* calls the CreateMonitoredItems *Service*, the maximum size of the itemsToModify array when a *Client* calls the ModifyMonitoredItems *Service*, the maximum size of the monitoredItemIds array when a *Client* calls the SetMonitoringMode *Service*, and the maximum size of the linksToAdd and the linksToRemove arrays when a *Client* calls the SetTriggering *Service*.

6.3.12 AddressSpaceFileType

This *ObjectType* defines the file for a namespace provided by the OPC UA *Server*. It is formally defined in Table 19. It represents an XML address space file using the XML schema defined in IEC 62541-6.

Table 19 – AddressSpaceFileType Definition

Attribute	**Value**				
BrowseName	AddressSpaceFileType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the FileType defined in C.2					
HasComponent	Method	ExportNamespace	The method has no parameters.		Optional

The *ExportNamespace Method* provides a way to export the namespace from the *Server AddressSpace* to the XML file represented by the *AddressSpaceFileType*. *Value Attributes* are only exported if they represent static configuration information. The client is expected to call the *ExportNamespace Method* first to update the XML file and then access the file with the *Methods* defined in the *FileType*.

Servers might provide some vendor-specific mechanisms importing parts of an address space as subtype of this *ObjectType*, for example by defining appropriate *Methods*.

6.3.13 NamespaceMetadataType

This *ObjectType* defines the metadata for a namespace provided by the *Server*. It is formally defined in Table 20.

Instances of this *Object* allow *Servers* to provide more information like version information in addition to the namespace URI. Important information for aggregating *Servers* is provided by the *StaticNodeIdTypes, StaticNumericNodeIdRange and StaticStringNodeIdPattern Properties*.

Table 20 – NamespaceMetadataType Definition

Attribute	Value				
BrowseName	NamespaceMetadataType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2					
HasProperty	Variable	NamespaceUri	String	PropertyType	Mandatory
HasProperty	Variable	NamespaceVersion	String	PropertyType	Mandatory
HasProperty	Variable	NamespacePublicationDate	DateTime	PropertyType	Mandatory
HasProperty	Variable	IsNamespaceSubset	Boolean	PropertyType	Mandatory
HasProperty	Variable	StaticNodeIdTypes	IdType[]	PropertyType	Mandatory
HasProperty	Variable	StaticNumericNodeIdRange	NumericRange[]	PropertyType	Mandatory
HasProperty	Variable	StaticStringNodeIdPattern	String	PropertyType	Mandatory
HasComponent	Object	NamespaceFile	-	AddressSpaceFileType	Optional

The *BrowseName* of instances of this type shall be derived from the represented namespace. This can, for example, be done by using the index of the namespace in the *NamespaceArray* as *namespaceIndex* of the *QualifiedName* and the namespace URI as *name* of the *QualifiedName*.

The *NamespaceUri Property* contains the namespace represented by an instance of the MetaDataType.

The *NamespaceVersion Property* provides version information for the namespace. It is intended for display purposes and shall not be used to programmatically identify the latest version.

The *NamespacePublicationDate* Property provides the publication date of the namespace version. This *Property* value can be used by *Clients* to determine the latest version if different versions are provided by different *Servers*.

The *IsNamespaceSubset Property* defines whether all *Nodes* of the namespace are accessible in the *Server* or only a subset. It is set to FALSE if the full namespace is provided and TRUE if not.

Static *Nodes* are identical for all *Attributes* in all *Servers*, including the *Value Attribute*. For *TypeDefinitionNodes*, also the *InstanceDeclarations* shall be identical. That means that for static *Nodes* the semantic is always the same. Namespaces with static *Nodes* are for example namespaces defined by standard bodies like the OPC Foundation. This is important information for aggregating *Servers.* If the namespace is dynamic and used in several *Servers* the aggregating *Server* needs to distinguish the namespace for each aggregated *Server*. The static *Nodes* of a namespace only need to be handled once, even if it they are used by several aggregated *Servers*.

The *StaticNodeIdTypes Property* provides a list of *IdTypes* used for static *Nodes*. All *Nodes* in the *AddressSpace* of the namespace using one of the *IdTypes* in the array shall be static *Nodes*.

The *StaticNumericNodeIdRange Property* provides a list of *NumericRanges* used for numeric *NodeIds* of static *Nodes*. If the *StaticNodeIdTypes Property* contains an entry for numeric NodeIds then this Property is ignored.

The *StaticStringNodeIdPattern Property* provides a regular expression as defined for the *Like Operator* defined in IEC 62541-4 to filter for string *NodeIds* of static *Nodes*. If the *StaticNodeIdTypes Property* contains an entry for string *NodeIds* then this *Property* is ignored.

The *Object NamespaceFile* contains all *Nodes* and *References* of the namespace in an XML file where the XML schema is defined in IEC 62541-6. The XML file is provided through an *AddressSpaceFileType Object*.

6.3.14 NamespacesType

This *ObjectType* defines a list of *NamespaceMetadataType Objects* provided by the *Server*. It is formally defined in Table 21.

Table 21 – NamespacesType Definition

Attribute	**Value**				
BrowseName	NamespacesType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2					
HasComponent	Object	<NamespaceIdentifier>	-	NamespaceMetadataType	OptionalPlaceholder

The *ObjectType* contains a list of *NamespaceMetadataType Objects* representing the namespaces in the *Server*. The *BrowseName* of an *Object* shall be derived from the namespace represented by the *Object*. This can, for example, be done by using the index of the namespace in the *NamespaceArray* as *namespaceIndex* of the *QualifiedName* and the namespace URI as *name* of the *QualifiedName*.

6.4 ObjectTypes used as EventTypes

6.4.1 General

This International Standard defines standard *EventTypes*. They are represented in the *AddressSpace* as *ObjectTypes*. The *EventTypes* are already defined in IEC 62541-3. The following subclauses specify their representation in the *AddressSpace*.

6.4.2 BaseEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 22.

Table 22 – BaseEventType Definition

Attribute	**Value**				
BrowseName	BaseEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseObjectType* defined in 6.2					
HasSubtype	ObjectType	AuditEventType	Defined in 6.4.3		
HasSubtype	ObjectType	SystemEventType	Defined in 6.4.28		
HasSubtype	ObjectType	BaseModelChangeEventType	Defined in 6.4.31		
HasSubtype	ObjectType	SemanticChangeEventType	Defined in 6.4.33		
HasSubtype	ObjectType	EventQueueOverflowEventType	Defined in 6.4.34		
HasSubtype	ObjectType	ProgressEventType	Defined in 6.4.35		
HasProperty	Variable	EventId	ByteString	PropertyType	Mandatory
HasProperty	Variable	EventType	NodeId	PropertyType	Mandatory
HasProperty	Variable	SourceNode	NodeId	PropertyType	Mandatory
HasProperty	Variable	SourceName	String	PropertyType	Mandatory
HasProperty	Variable	Time	UtcTime	PropertyType	Mandatory
HasProperty	Variable	ReceiveTime	UtcTime	PropertyType	Mandatory
HasProperty	Variable	LocalTime	TimeZoneDataType	PropertyType	Optional
HasProperty	Variable	Message	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	Severity	UInt16	PropertyType	Mandatory

EventId is generated by the *Server* to uniquely identify a particular *Event Notification*. The *Server* is responsible to ensure that each *Event* has its unique *EventId*. It may do this, for example, by putting GUIDs into the ByteString. Clients can use the *EventId* to assist in minimizing or eliminating gaps and overlaps that may occur during a redundancy failover. The *EventId* shall always be returned as value and the *Server* is not allowed to return a *StatusCode* for the *EventId* indicating an error.

EventType describes the specific type of *Event*. The *EventType* shall always be returned as value and the *Server* is not allowed to return a *StatusCode* for the *EventType* indicating an error.

SourceNode identifies the *Node* that the *Event* originated from. If the *Event* is not specific to a *Node* the *NodeId* is set to null. Some subtypes of this *BaseEventType* may define additional rules for *SourceNode*.

SourceName provides a description of the source of the *Event*. This could be the string-part of the *DisplayName* of the *Event* source using the default locale of the server, if the *Event* is specific to a *Node,* or some server-specific notation.

Time provides the time the *Event* occurred. This value is set as close to the event generator as possible. It often comes from the underlying system or device. Once set, intermediate OPC UA *Servers* shall not alter the value.

ReceiveTime provides the time the OPC UA *Server* received the *Event* from the underlying device of another *Server*. *ReceiveTime* is analogous to *ServerTimestamp* defined in IEC 62541-4, i.e. in the case where the OPC UA *Server* gets an *Event* from another OPC UA *Server*, each *Server* applies its own *ReceiveTime*. That implies that a *Client* may get the same *Event,* having the same *EventId*, from different *Servers* having different values of the *ReceiveTime*. The *ReceiveTime* shall always be returned as value and the *Server* is not allowed to return a *StatusCode* for the *ReceiveTime* indicating an error.

LocalTime is a structure containing the Offset and the DaylightSavingInOffset flag. The Offset specifies the time difference (in minutes) between the *Time Property* and the time at the location in which the event was issued. If DaylightSavingInOffset is TRUE, then Standard/Daylight savings time (DST) at the originating location is in effect and Offset includes the DST correction. If FALSE then the Offset does not include DST correction and DST may or may not have been in effect.

Message provides a human-readable and localizable text description of the *Event*. The *Server* may return any appropriate text to describe the *Event*. A null string is not a valid value; if the *Server* does not have a description, it shall return the string part of the *BrowseName* of the *Node* associated with the *Event*.

Severity is an indication of the urgency of the *Event*. This is also commonly called “priority”. Values will range from 1 to 1 000, with 1 being the lowest severity and 1 000 being the highest. Typically, a severity of 1 would indicate an *Event* which is informational in nature, while a value of 1 000 would indicate an *Event* of catastrophic nature, which could potentially result in severe financial loss or loss of life.

It is expected that very few *Server* implementations will support 1 000 distinct severity levels. Therefore, *Server* developers are responsible for distributing their severity levels across the 1 to 1 000 range in such a manner that clients can assume a linear distribution. For example, a client wishing to present five severity levels to a user should be able to do the following mapping:

Client Severity	OPC Severity
HIGH	801 – 1 000
MEDIUM HIGH	601 – 800
MEDIUM	401 – 600
MEDIUM LOW	201 – 400
LOW	1 – 200

In many cases a strict linear mapping of underlying source severities to the OPC Severity range is not appropriate. The *Server* developer will instead intelligently map the underlying source severities to the 1 to 1 000 OPC Severity range in some other fashion. In particular, it is recommended that *Server* developers map *Events* of high urgency into the OPC severity range of 667 to 1 000, *Events* of medium urgency into the OPC severity range of 334 to 666 and *Events* of low urgency into OPC severities of 1 to 333.

For example, if a source supports 16 severity levels that are clustered such that severities 0 to 2 are considered to be LOW, 3 to 7 are MEDIUM and 8 to 15 are HIGH, then an appropriate mapping might be as follows:

OPC Range	Source Severity	OPC Severity
HIGH (667 – 1 000)	15	1 000
	14	955
	13	910
	12	865
	11	820
	10	775
	9	730
	8	685
MEDIUM (334 – 666)	7	650
	6	575
	5	500
	4	425
	3	350
LOW (1 – 333)	2	300
	1	150
	0	1

Some *Servers* might not support any *Events* which are catastrophic in nature, so they may choose to map all of their severities into a subset of the 1 to 1 000 range (for example, 1 to 666). Other *Servers* might not support any *Events* which are merely informational, so they may choose to map all of their severities into a different subset of the 1 to 1 000 range (for example, 334 to 1 000).

The purpose of this approach is to allow clients to use severity values from multiple *Servers* from different vendors in a consistent manner. Additional discussions of severity can be found in IEC 62541-9.

6.4.3 AuditEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 23.

Table 23 – AuditEventType Definition

Attribute		Value			
BrowseName		AuditEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditSecurityEventType	Defined in 6.4.4		
HasSubtype	ObjectType	AuditNodeManagementEventType	Defined in 6.4.19		
HasSubtype	ObjectType	AuditUpdateEventType	Defined in 6.4.24		
HasSubtype	ObjectType	AuditUpdateMethodEventType	Defined in 6.4.27		
HasProperty	Variable	ActionTimeStamp	UtcTime	PropertyType	Mandatory
HasProperty	Variable	Status	Boolean	PropertyType	Mandatory
HasProperty	Variable	ServerId	String	PropertyType	Mandatory
HasProperty	Variable	ClientAuditEntryId	String	PropertyType	Mandatory
HasProperty	Variable	ClientUserId	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2.

ActionTimeStamp identifies the time the user initiated the action that resulted in the *AuditEvent* being generated. It differs from the *Time Property* because this is the time the server generated the *AuditEvent* documenting the action.

Status identifies whether the requested action could be performed (set *Status* to TRUE) or not (set *Status* to FALSE).

ServerId uniquely identifies the *Server* generating the *Event*. It identifies the *Server* uniquely even in a server-controlled transparent redundancy scenario where several *Servers* may use the same URI.

ClientAuditEntryId contains the human-readable *AuditEntryId* defined in IEC 62541-3.

The *ClientUserId* identifies the user of the client requesting an action. The *ClientUserId* can be obtained from the *UserIdentityToken* passed in the *ActivateSession* call. If the *UserIdentityToken* is a *UserNameIdentityToken* then the *ClientUserId* is the *UserName.* If the *UserIdentityToken* is an *X509IdentityToken* then the *ClientUserId* is the X509 Subject Name of the *Certificate*. If the *UserIdentityToken* is an *IssuedIdentityToken* then the *ClientUserId* should be a string that represents the owner of the token. The best choice for the string depends on the type of *IssuedIdentityToken.* If an *AnonymousIdentityToken* was used, the value is null.

6.4.4 AuditSecurityEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 24.

Table 24 – AuditSecurityEventType Definition

Attribute		Value			
BrowseName		AuditSecurityEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the AuditEventType defined in 6.4.3, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditChannelEventType	Defined in 6.4.5		
HasSubtype	ObjectType	AuditSessionEventType	Defined in 6.4.7		
HasSubtype	ObjectType	AuditCertificateEventType	Defined in 6.4.12		

This *EventType* inherits all *Properties* of the *AuditEventType*. Their semantic is defined in 6.4.3. There are no additional *Properties* defined for this *EventType*.

6.4.5 AuditChannelEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 25.

Table 25 – AuditChannelEventType Definition

Attribute	Value				
BrowseName	AuditChannelEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the AuditSecurityEventType defined in 6.4.4, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditOpenSecureChannelEventType	Defined in 6.4.6		
HasProperty	Variable	SecureChannelId	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSecurityEventType*. Their semantic is defined in 6.4.4. There are no additional *Properties* defined for this *EventType*. The *SourceNode* for *Events* of this type should be assigned to the *Server Object*. The *SourceName* for *Events* of this type should be “SecureChannel/” and the *Service* that generates the *Event* (e.g. SecureChannel/*OpenSecureChannel* or SecureChannel/*CloseSecureChannel*). If the *ClientUserId* is not available for a *CloseSecureChannel* call, then this parameter shall be set to ”System/CloseSecureChannel”.

The *SecureChannelId* shall uniquely identify the SecureChannel. The application shall use the same identifier in all *AuditEvents* related to the Session Service Set (AuditCreateSessionEventType, AuditActivateSessionEventType and their subtypes) and the SecureChannel Service Set (AuditChannelEventType and its subtypes).

6.4.6 AuditOpenSecureChannelEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 26.

Table 26 – AuditOpenSecureChannelEventType Definition

Attribute	Value				
BrowseName	AuditOpenSecureChannelEventType				
IsAbstract	True				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *AuditChannelEventType* defined in 6.4.5, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ClientCertificate	ByteString	PropertyType	Mandatory
HasProperty	Variable	ClientCertificateThumbprint	String	PropertyType	Mandatory
HasProperty	Variable	RequestType	SecurityTokenRequestType	PropertyType	Mandatory
HasProperty	Variable	SecurityPolicyUri	String	PropertyType	Mandatory
HasProperty	Variable	SecurityMode	MessageSecurityMode	PropertyType	Mandatory
HasProperty	Variable	RequestedLifetime	Duration	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditChannelEventType*. Their semantic is defined in 6.4.5. The *SourceName* for *Events* of this type should be “SecureChannel/OpenSecureChannel”. The *ClientUserId* is not available for this call, thus this parameter shall be set to ”System/OpenSecureChannel”.

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

ClientCertificate is the clientCertificate parameter of the OpenSecureChannel *Service* call.

ClientCertificateThumbprint is a thumbprint of the *ClientCertificate.* See IEC 62541-6 for details on thumbprints.

RequestType is the requestType parameter of the OpenSecureChannel *Service* call.

SecurityPolicyUri is the securityPolicyUri parameter of the OpenSecureChannel *Service* call.

SecurityMode is the securityMode parameter of the OpenSecureChannel *Service* call.

RequestedLifetime is the requestedLifetime parameter of the OpenSecureChannel *Service* call.

6.4.7 AuditSessionEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 27.

Table 27 – AuditSessionEventType Definition

Attribute	**Value**				
BrowseName	AuditSessionEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditEventType* defined in 6.4.4, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditCreateSessionEventType	Defined in 6.4.8		
HasSubtype	ObjectType	AuditActivateSessionEventType	Defined in 6.4.10		
HasSubtype	ObjectType	AuditCancelEventType	Defined in 6.4.11		
HasProperty	Variable	SessionId	NodeId	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditEventType*. Their semantic is defined in 6.4.4.

If the *Event* is generated by a *TransferSubscriptions Service* call, the *SourceNode* should be assigned to the *SessionDiagnostics Object* that represents the session. The *SourceName* for *Events* of this type should be “Session/TransferSubscriptions”.

Otherwise, the *SourceNode* for *Events* of this type should be assigned to the *Server Object*. The *SourceName* for *Events* of this type should be “Session/” and the *Service* that generates the *Event* (e.g. *CreateSession*, *ActivateSession* or *CloseSession*).

The *SessionId* should contain the *SessionId* of the session that the *Service* call was issued on In the *CreateSession Service* this shall be set to the newly created *SessionId*. If no session context exists (e.g. for a failed *CreateSession Service* call) the *SessionId* is set to null.

6.4.8 AuditCreateSessionEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 28.

Table 28 – AuditCreateSessionEventType Definition

Attribute	Value				
BrowseName	AuditCreateSessionEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditSessionEventType* defined in 6.4.7, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditUrlMismatchEventType	Defined in 6.4.9		
HasProperty	Variable	SecureChannelId	String	PropertyType	Mandatory
HasProperty	Variable	ClientCertificate	ByteString	PropertyType	Mandatory
HasProperty	Variable	ClientCertificateThumbprint	String	PropertyType	Mandatory
HasProperty	Variable	RevisedSessionTimeout	Duration	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSessionEventType*. Their semantic is defined in 6.4.7. The *SourceName* for *Events* of this type should be “Session/CreateSession”. The *ClientUserId* is not available for this call thus this parameter shall be set to the ”System/CreateSession”.

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

SecureChannelId shall uniquely identify the SecureChannel. The application shall use the same identifier in all *AuditEvents* related to the Session Service Set (AuditCreateSessionEventType, AuditActivateSessionEventType and their subtypes) and the SecureChannel Service Set (AuditChannelEventType and its subtypes).

ClientCertificate is the clientCertificate parameter of the CreateSession *Service* call.

ClientCertificateThumbprint is a thumbprint of the *ClientCertificate.* See IEC 62541-6 for details on thumbprints.

RevisedSessionTimeout is the returned revisedSessionTimeout parameter of the CreateSession *Service* call.

6.4.9 AuditUrlMismatchEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 29.

Table 29 – AuditUrlMismatchEventType Definition

Attribute	Value				
BrowseName	AuditUrlMismatchEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the AuditCreateSessionEventType defined in 6.4.8 which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	EndpointUrl	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSessionEventType*. Their semantic is defined in 6.4.8.

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

EndpointUrl is the endpointUrl parameter of the CreateSession *Service* call.

6.4.10 AuditActivateSessionEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 30.

Table 30 – AuditActivateSessionEventType Definition

Attribute	Value				
BrowseName	AuditActivateSessionEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *AuditSessionEventType* defined in 6.4.7, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ClientSoftwareCertificates	SignedSoftwareCertificate[]	PropertyType	Mandatory
HasProperty	Variable	UserIdentityToken	UserIdentityToken	PropertyType	Mandatory
HasProperty	Variable	SecureChannelId	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSessionEventType*. Their semantic is defined in 6.4.7. The *SourceName* for *Events* of this type should be "Session/ActivateSession".

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

ClientSoftwareCertificates is the clientSoftwareCertificates parameter of the ActivateSession *Service* call.

UserIdentityToken reflects the userIdentityToken parameter of the ActivateSession *Service* call. For Username/Password tokens the password should NOT be included.

SecureChannelId shall uniquely identify the SecureChannel. The application shall use the same identifier in all *AuditEvents* related to the Session Service Set (AuditCreateSessionEventType, AuditActivateSessionEventType and their subtypes) and the SecureChannel Service Set (AuditChannelEventType and its subtypes).

6.4.11 AuditCancelEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 31.

Table 31 – AuditCancelEventType Definition

Attribute		Value			
BrowseName		AuditCancelEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditSessionEventType* defined in 6.4.7, i.e. inheriting the InstanceDeclarations of that Node.					
HasProperty	Variable	RequestHandle	UInt32	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSessionEventType*. Their semantic is defined in 6.4.7. The *SourceName* for *Events* of this type should be "Session/Cancel".

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

RequestHandle is the requestHandle parameter of the Cancel *Service* call.

6.4.12 AuditCertificateEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 32.

Table 32 – AuditCertificateEventType Definition

Attribute		Value			
BrowseName		AuditCertificateEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *AuditSecurityEventType* defined in 6.4.7, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditCertificateDataMismatchEventType	Defined in 6.4.13		
HasSubtype	ObjectType	AuditCertificateExpiredEventType	Defined in 6.4.14		
HasSubtype	ObjectType	AuditCertificateInvalidEventType	Defined in 6.4.15		
HasSubtype	ObjectType	AuditCertificateUntrustedEventType	Defined in 6.4.16		
HasSubtype	ObjectType	AuditCertificateRevokedEventType	Defined in 6.4.17		
HasSubtype	ObjectType	AuditCertificateMismatchEventType	Defined in 6.4.18		
HasProperty	Variable	Certificate	ByteString	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditSecurityEventType*. Their semantic is defined in 6.4.4. The *SourceName* for *Events* of this type should be “Security/Certificate”.

Certificate is the certificate that encountered a validation issue. Additional subtypes of this *EventType* will be defined representing the individual validation errors. This certificate can be matched to the *Service* that passed it (Session or SecureChannel Service Set) since the *AuditEvents* for these *Services* also included the Certificate.

6.4.13 AuditCertificateDataMismatchEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 33.

Table 33 – AuditCertificateDataMismatchEventType Definition

Attribute		Value			
BrowseName		AuditCertificateDataMismatchEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the AuditCertificateEventType defined in 6.4.12, i.e. inheriting the InstanceDeclarations of that Node.					
HasProperty	Variable	InvalidHostname	String	PropertyType	Mandatory
HasProperty	Variable	InvalidUri	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type should be “Security/Certificate”.

InvalidHostname is the string that represents the host name passed in as part of the URL that is found to be invalid. If the host name was not invalid it can be null.

InvalidUri is the URI that was passed in and found to not match what is contained in the certificate. If the URI was not invalid it can be null.

Either the *InvalidHostname* or *InvalidUri* shall be provided.

6.4.14 AuditCertificateExpiredEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 34.

Table 34 – AuditCertificateExpiredEventType Definition

Attribute		Value			
BrowseName		AuditCertificateExpiredEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditCertificateEventType* defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type should be “Security/Certificate”. The *Message Variable* shall include a description of why the certificate was expired (i.e. time before start or time after end). There are no additional *Properties* defined for this *EventType*.

6.4.15 AuditCertificateInvalidEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 35.

Table 35 – AuditCertificateInvalidEventType Definition

Attribute		Value			
BrowseName		AuditCertificateInvalidEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditCertificateEventType* defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type should be “Security/Certificate”. The *Message* shall include a description of why the certificate is invalid. There are no additional *Properties* defined for this *EventType*.

6.4.16 AuditCertificateUntrustedEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 36.

Table 36 – AuditCertificateUntrustedEventType Definition

Attribute		Value			
BrowseName		AuditCertificateUntrustedEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditCertificateEventType* defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type should be “Security/Certificate”. The *Message Variable* shall include a description of why the certificate is not trusted. If a trust chain is involved then the certificate that failed in the trust chain should be described. There are no additional *Properties* defined for this *EventType*.

6.4.17 AuditCertificateRevokedEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 37.

Table 37 – AuditCertificateRevokedEventType Definition

Attribute		Value			
BrowseName		AuditCertificateRevokedEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditCertificateEventType* defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type should be "Security/Certificate". The *Message Variable* shall include a description of why the certificate is revoked (was the revocation list unavailable or was the certificate on the list). There are no additional *Properties* defined for this *EventType*.

6.4.18 AuditCertificateMismatchEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 38.

Table 38 – AuditCertificateMismatchEventType Definition

Attribute		Value			
BrowseName		AuditCertificateMismatchEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditCertificateEventType* defined in 6.4.12, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *AuditCertificateEventType*. Their semantic is defined in 6.4.12. The *SourceName* for *Events* of this type should be "Security/Certificate". The *Message Variable* shall include a description of misuse of the certificate. There are no additional *Properties* defined for this *EventType*.

6.4.19 AuditNodeManagementEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 39.

Table 39 – AuditNodeManagementEventType Definition

Attribute		Value			
BrowseName		AuditNodeManagementEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditEventType* defined in 6.4.3, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditAddNodesEventType			
HasSubtype	ObjectType	AuditDeleteNodesEventType			
HasSubtype	ObjectType	AuditAddReferencesEventType			
HasSubtype	ObjectType	AuditDeleteReferencesEventType			

This *EventType* inherits all *Properties* of the *AuditEventType*. Their semantic is defined in 6.4.3. There are no additional *Properties* defined for this *EventType*. The *SourceNode* for *Events* of this type should be assigned to the *Server Object*. The *SourceName* for *Events* of this type should be "NodeManagement/" and the *Service* that generates the *Event* (e.g. *AddNodes*, *AddReferences*, *DeleteNodes*, *DeleteReferences*).

6.4.20 AuditAddNodesEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 40.

Table 40 – AuditAddNodesEventType Definition

Attribute	Value				
BrowseName	AuditAddNodesEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditNodeManagementEventType* defined in 6.4.19, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	NodesToAdd	AddNodesItem[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditNodeManagementEventType*. Their semantic is defined in 6.4.19. The *SourceName* for *Events* of this type should be "NodeManagement/AddNodes".

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

NodesToAdd is the NodesToAdd parameter of the AddNodes *Service* call.

6.4.21 AuditDeleteNodesEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 41.

Table 41 – AuditDeleteNodesEventType Definition

Attribute	Value				
BrowseName	AuditDeleteNodesEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditNodeManagementEventType* defined in 6.4.19, i.e. inheriting the InstanceDeclarations of that Node.					
HasProperty	Variable	NodesToDelete	DeleteNodesItem[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditNodeManagementEventType*. Their semantic is defined in 6.4.19. The *SourceName* for *Events* of this type should be "NodeManagement/DeleteNodes".

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

NodesToDelete is the nodesToDelete parameter of the DeleteNodes *Service* call.

6.4.22 AuditAddReferencesEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 42.

Table 42 – AuditAddReferencesEventType Definition

Attribute	Value				
BrowseName	AuditAddReferencesEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditNodeManagementEventType* defined in 6.4.19, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ReferencesToAdd	AddReferencesItem[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditNodeManagementEventType*. Their semantic is defined in 6.4.19. The *SourceName* for *Events* of this type should be "NodeManagement/AddReferences".

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

ReferencesToAdd is the referencesToAdd parameter of the AddReferences *Service* call.

6.4.23 AuditDeleteReferencesEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 43.

Table 43 – AuditDeleteReferencesEventType Definition

Attribute	Value				
BrowseName	AuditDeleteReferencesEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditNodeManagementEventType* defined in 6.4.19, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ReferencesToDelete	DeleteReferencesItem[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditNodeManagementEventType*. Their semantic is defined in 6.4.19. The *SourceName* for *Events* of this type should be "NodeManagement/DeleteReferences".

The additional *Properties* defined for this *EventType* reflect parameters of the *Service* call that triggers the *Event*.

ReferencesToDelete is the referencesToDelete parameter of the DeleteReferences *Service* call.

6.4.24 AuditUpdateEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 44.

Table 44 – AuditUpdateEventType Definition

Attribute	Value				
BrowseName	AuditUpdateEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditEventType* defined in 6.4.3, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	AuditWriteUpdateEventType	Defined in 6.4.25		
HasSubtype	ObjectType	AuditHistoryUpdateEventType	Defined in 6.4.26		

This *EventType* inherits all *Properties* of the *AuditEventType*. Their semantic is defined in 6.4.3. The *SourceNode* for *Events* of this type should be assigned to the *NodeId* that was changed. The *SourceName* for *Events* of this type should be “Attribute/” and the *Service* that generated the event (e.g. *Write*, *HistoryUpdate*). Note that one *Service* call may generate several *Events* of this type, one per changed value.

6.4.25 AuditWriteUpdateEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 45.

Table 45 – AuditWriteUpdateEventType Definition

Attribute	Value				
BrowseName	AuditWriteUpdateEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateEventType* defined in 6.4.24, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	AttributeId	UInt32	PropertyType	Mandatory
HasProperty	Variable	IndexRange	NumericRange	PropertyType	Mandatory
HasProperty	Variable	NewValue	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	OldValue	BaseDataType	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditUpdateEventType*. The *SourceName* for *Events* of this type should be “Attribute/Write”. Their semantic is defined in 6.4.24.

AttributeId identifies the *Attribute* that was written on the *SourceNode*.

IndexRange identifies the index range of the written *Attribute* if the *Attribute* is an array. If the *Attribute* is not an array or the whole array was written, the *IndexRange* is set to null.

NewValue identifies the value that was written to the *SourceNode*. If the *IndexRange* is provided, only the values in the provided range are shown.

OldValue identifies the value that the *SourceNode* contained before the write. If the *IndexRange* is provided, only the value of that range is shown. It is acceptable for a *Server* that does not have this information to report a null value.

Both the *NewValue* and the *OldValue* will contain a value in the *DataType* and encoding used for writing the value.

6.4.26 AuditHistoryUpdateEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 46.

Table 46 – AuditHistoryUpdateEventType Definition

Attribute	Value				
BrowseName	AuditHistoryUpdateEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateEventType* defined in 6.4.24, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	ParameterDataTypeId	NodeId	PropertyType	New

This *EventType* inherits all *Properties* of the *AuditUpdateEventType*. Their semantic is defined in 6.4.24.

The *ParameterDataTypeId* identifies the *DataTypeId* for the extensible parameter used by the HistoryUpdate. This parameter indicates the type of HistoryUpdate being performed.

Subtypes of this *EventType* are defined in IEC 62541-11 representing the different possibilities to manipulate historical data.

6.4.27 AuditUpdateMethodEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 47.

Table 47 – AuditUpdateMethodEventType Definition

Attribute	Value				
BrowseName	AuditUpdateMethodEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditEventType* defined in 6.4.3, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	MethodId	NodeId	PropertyType	Mandatory
HasProperty	Variable	InputArguments	BaseDataType[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditEventType*. Their semantic is defined in 6.4.3. The *SourceNode* for *Events* of this type should be assigned to the *NodeId* of the object that the method resides on. The *SourceName* for *Events* of this type should be “Attribute/Call”. Note that one *Service* call may generate several *Events* of this type, one per method called. This EventType should be further subtyped to better reflect the functionality of the method and to reflect changes to the address space or updated values triggered by the method.

MethodId identifies the method that was called.

InputArguments identifies the input Arguments for the method. This parameter can be null if no input arguments where provided.

6.4.28 SystemEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 48.

Table 48 – SystemEventType Definition

Attribute	Value				
BrowseName	SystemEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasSubtype	ObjectType	DeviceFailureEventType	Defined in 6.4.29		
HasSubtype	ObjectType	SystemStatusChangeEventType	Defined in 6.4.30		
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2. There are no additional *Properties* defined for this *EventType*.

6.4.29 DeviceFailureEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 49.

Table 49 – DeviceFailureEventType Definition

Attribute		Value			
BrowseName		DeviceFailureEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *SystemEventType* defined in 6.4.28, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *SystemEventType*. Their semantic is defined in 6.4.28. There are no additional *Properties* defined for this *EventType*.

6.4.30 SystemStatusChangeEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 50.

Table 50 – SystemStatusChangeEventType Definition

Attribute		Value			
BrowseName		SystemStatusChangeEventType			
IsAbstract		True			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *SystemEventType* defined in 6.4.28, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	SystemState	ServerState	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *SystemEventType*. Their semantic is defined in 6.4.28. The *SourceNode* and the *SourceName* shall identify the system. The system can be the *Server* itself or some underlying system.

The *SystemState* specifies the current state of the system. Changes to the *ServerState* of the system shall trigger a *SystemStatusChangeEvent*, when the event is supported by the system.

6.4.31 BaseModelChangeEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 51.

Table 51 – BaseModelChangeEventType Definition

Attribute	Value				
BrowseName	BaseModelChangeEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					
HasSubtype	ObjectType	GeneralModelChangeEventType	Defined in 6.4.32		

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2. There are no additional *Properties* defined for this *EventType*. The *SourceNode* for Events of this type should be the *Node* of the *View* that gives the context of the changes. If the whole *AddressSpace* is the context, the *SourceNode* is set to the *NodeId* of the *Server Object*. The *SourceName* for *Events* of this type should be the *String* part of the *BrowseName* of the *View*; for the whole *AddressSpace* it should be “Server”.

6.4.32 GeneralModelChangeEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 52.

Table 52 – GeneralModelChangeEventType Definition

Attribute	Value				
BrowseName	GeneralModelChangeEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseModelChangeEventType* defined in 6.4.31, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	Changes	ModelChangeStructureDataType[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *BaseModelChangeEventType*. Their semantic is defined in 6.4.31.

The additional *Property* defined for this *EventType* reflects the changes that issued the *ModelChangeEvent*. It shall contain at least one entry in its array. Its structure is defined in 12.16.

6.4.33 SemanticChangeEventType

This *EventType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is formally defined in Table 53.

Table 53 – SemanticChangeEventType Definition

Attribute	Value				
BrowseName	SemanticChangeEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	Changes	SemanticChangeStructureDataType[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2. There are no additional *Properties* defined for this *EventType*. The *SourceNode* for Events of this type should be the *Node* of the *View* that gives the context of the changes. If the whole *AddressSpace* is the context, the *SourceNode* is set to the *NodeId* of the *Server Object*. The *SourceName* for *Events* of this type should be the *String* part of the *BrowseName* of the *View*, for the whole *AddressSpace* it should be “Server”.

The additional *Property* defined for this *EventType* reflects the changes that issued the *SemanticChangeEvent*. Its structure is defined in 12.17.

6.4.34 EventQueueOverflowEventType

EventQueueOverflow Events are generated when an internal queue of a MonitoredItem subscribing for *Events* in the *Server* overflows. IEC 62541-4 defines when the internal EventQueueOverflow *Events* shall be generated.

The *EventType* for *EventQueueOverflow Events* is formally defined in Table 54.

Table 54 – EventQueueOverflowEventType Definition

Attribute	Value				
BrowseName	EventQueueOverflowEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2. The *SourceNode* for *Events* of this type shall be assigned to the *NodeId* of the *Server Object*. The *SourceName* for *Events* of this type shall be “Internal/EventQueueOverflow”.

6.4.35 ProgressEventType

ProgressEvents are generated to identify the progress of an operation. An operation can be a *Service* call or something application specific like a program execution.

The *EventType* for *Progress Events* is formally defined in Table 55.

Table 55 – ProgressEventType Definition

Attribute	Value				
BrowseName	ProgressEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseEventType* defined in 6.4.2, which means it inherits the InstanceDeclarations of that Node.					
HasProperty	Variable	Context	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	Progress	UInt16	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in 6.4.2. The *SourceNode* for *Events* of this type shall be assigned to the *NodeId* of the *Session Object* where the operation was initiated. The *SourceName* for *Events* of this type shall be “Service/<Service Name as defined in IEC 62541-4>” when the progress of a *Service* call is exposed.

The additional *Property Context* contains context information about what operation progress is reported. In the case of *Service* calls it shall be a UInt32 containing the *requestHandle* of the *RequestHeader* of the *Service* call.

The additional *Property Progress* contains the percentage completed of the progress. The value shall be between 0 and 100, where 100 identifies that the operation has been finished.

It is recommended that *Servers* only expose *ProgressEvents* for *Service* calls to the *Session* that invoked the *Service*.

6.5 ModellingRuleType

ModellingRules are defined in IEC 62541-3. This *ObjectType* is used as the type for the *ModellingRules*. It is formally defined in Table 56.

Table 56 – ModellingRuleType Definition

Attribute		Value			
BrowseName		ModellingRuleType			
IsAbstract		False			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2					
HasProperty	Variable	NamingRule	NamingRuleType	PropertyType	Mandatory

The *Property NamingRule* identifies the *NamingRule* of a *ModellingRule* as defined in IEC 62541-3.

6.6 FolderType

Instances of this *ObjectType* are used to organise the *AddressSpace* into a hierarchy of *Nodes*. They represent the *root Node* of a subtree, and have no other semantics associated

with them. However, the *DisplayName* of an instance of the *FolderType*, such as "ObjectTypes", should imply the semantics associated with the use of it. There are no References specified for this *ObjectType*. It is formally defined in Table 57.

Table 57 – FolderType Definition

Attribute	Value				
BrowseName	FolderType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2.					

6.7 DataTypeEncodingType

DataTypeEncodings are defined in IEC 62541-3. This *ObjectType* is used as type for the *DataTypeEncodings*. There are no *References* specified for this *ObjectType*. It is formally defined in Table 58.

Table 58 – DataTypeEncodingType Definition

Attribute	Value				
BrowseName	DataTypeEncodingType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2.					

6.8 DataTypeSystemType

DataTypeSystems are defined in IEC 62541-3. This *ObjectType* is used as type for the *DataTypeSystems*. There are no *References* specified for this *ObjectType*. It is formally defined in Table 59.

Table 59 – DataTypeSystemType Definition

Attribute	Value				
BrowseName	DataTypeSystemType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2.					

6.9 AggregateFunctionType

This *ObjectType* defines an *AggregateFunction* supported by a UA *Server*. It is formally defined in Table 60.

Table 60 – AggregateFunctionType Definition

Attribute	Value				
BrowseName	AggregateFunctionType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2.					

For the *AggregateFunctionType*, the *Description Attribute* is mandatory. The *Description Attribute* provides a localized description of the *AggregateFunction.* Specific *AggregateFunctions* may be defined in further parts of this series of standards.

7 Standard VariableTypes

7.1 General

Typically, the components of a complex *VariableType* are fixed and can be extended by subtyping. However, because each *Variable* of a *VariableType* can be extended with additional components this standard allows the extension of the standard *VariableTypes* defined in this document with additional components. This allows the expression of additional information in the type definition that would be contained in each *Variable* anyway. However, it is not allowed to restrict the components of the standard *VariableTypes* defined in this International Standard. An example of extending *VariableTypes* would be putting the standard *Property NodeVersion*, defined in IEC 62541-3, into the *BaseDataVariableType*, stating that each *DataVariable* of the *Server* will provide a *NodeVersion*.

7.2 BaseVariableType

The *BaseVariableType* is the abstract base type for all other *VariableTypes*. However, only the *PropertyType* and the *BaseDataVariableType* directly inherit from this type.

There are no *References*, except for *HasSubtype References*, specified for this *VariableType*. It is formally defined in Table 61.

Table 61 – BaseVariableType Definition

Attribute	Value				
BrowseName	BaseVariableType				
IsAbstract	True				
ValueRank	−2 (−2 = Any)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasSubtype	VariableType	PropertyType	Defined in 7.3		
HasSubtype	VariableType	BaseDataVariableType	Defined in 7.4		

7.3 PropertyType

The *PropertyType* is a subtype of the *BaseVariableType*. It is used as the type definition for all *Properties*. *Properties* are defined by their *BrowseName* and therefore they do not need a specialised type definition. It is not allowed to subtype this *VariableType*.

There are no *References* specified for this *VariableType*. It is formally defined in Table 62.

Table 62 – PropertyType Definition

Attribute	Value				
BrowseName	PropertyType				
IsAbstract	False				
ValueRank	−2 (−2 = Any)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseVariableType defined in 7.2.					

7.4 BaseDataVariableType

The *BaseDataVariableType* is a subtype of the *BaseVariableType*. It is used as the type definition whenever there is a *DataVariable* having no more concrete type definition available. This *VariableType* is the base *VariableType* for *VariableTypes* of *DataVariables*, and all other *VariableTypes* of *DataVariables* shall either directly or indirectly inherit from it. However, it might not be possible for *Servers* to provide all *HasSubtype References* from this *VariableType* to its subtypes, and therefore it is not required to provide this information.

There are no *References* except for *HasSubtype References* specified for this *VariableType*. It is formally defined in Table 63.

Table 63 – BaseDataVariableType Definition

Attribute		Value	
BrowseName		BaseDataVariableType	
IsAbstract		False	
ValueRank		−2 (−2 = Any)	
DataType		BaseDataType	
References	**NodeClass**	**BrowseName**	**Comment**
Subtype of the BaseVariableType defined in 7.2.			
HasSubtype	VariableType	ServerVendorCapabilityType	Defined in 7.5
HasSubtype	VariableType	DataTypeDictionaryType	Defined in 7.6
HasSubtype	VariableType	DataTypeDescriptionType	Defined in 7.7
HasSubtype	VariableType	ServerStatusType	Defined in 7.8
HasSubtype	VariableType	BuildInfoType	Defined in 7.9
HasSubtype	VariableType	ServerDiagnosticsSummaryType	Defined in 7.10
HasSubtype	VariableType	SamplingIntervalDiagnosticsArrayType	Defined in 7.11
HasSubtype	VariableType	SamplingIntervalDiagnosticsType	Defined in 7.12
HasSubtype	VariableType	SubscriptionDiagnosticsArrayType	Defined in 7.13
HasSubtype	VariableType	SubscriptionDiagnosticsType	Defined in 7.14
HasSubtype	VariableType	SessionDiagnosticsArrayType	Defined in 7.15
HasSubtype	VariableType	SessionDiagnosticsVariableType	Defined in 7.16
HasSubtype	VariableType	SessionSecurityDiagnosticsArrayType	Defined in 7.17
HasSubtype	VariableType	SessionSecurityDiagnosticsType	Defined in 7.18
HasSubtype	VariableType	OptionSetType	Defined in 7.19

7.5 ServerVendorCapabilityType

This *VariableType* is an abstract type whose subtypes define capabilities of the *Server*. Vendors may define subtypes of this type. This *VariableType* is formally defined in Table 64.

Table 64 – ServerVendorCapabilityType Definition

Attribute		Value			
BrowseName		ServerVendorCapabilityType			
IsAbstract		True			
ValueRank		−1 (−1 = Scalar)			
DataType		BaseDataType			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.					

7.6 DataTypeDictionaryType

DataTypeDictionaries are defined in IEC 62541-3. This *VariableType* is used as the type for the *DataTypeDictionaries*. There are no *References* specified for this *VariableType*. It is formally defined in Table 65.

Table 65 – DataTypeDictionaryType Definition

Attribute		Value			
BrowseName		DataTypeDictionaryType			
IsAbstract		False			
ValueRank		−1 (−1 = Scalar)			
DataType		ByteString			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasProperty	Variable	DataTypeVersion	String	PropertyType	Optional
HasProperty	Variable	NamespaceUri	String	PropertyType	Optional

The *Property* DataTypeVersion is defined in IEC 62541-3. The NamespaceUri is the URI for the namespace described by the *Value Attribute* of the DataTypeDictionary.

7.7 DataTypeDescriptionType

DataTypeDescriptions are defined in IEC 62541-3. This *VariableType* is used as the type for the *DataTypeDescriptions*. There are no *References* specified for this *VariableType*. It is formally defined in Table 66.

Table 66 – DataTypeDescriptionType Definition

Attribute		Value			
BrowseName		DataTypeDescriptionType			
IsAbstract		False			
ValueRank		-1 (-1 = Scalar)			
DataType		ByteString			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasProperty	Variable	DataTypeVersion	String	PropertyType	Optional
HasProperty	Variable	DictionaryFragment	ByteString	PropertyType	Optional

The *Properties* DataTypeVersion and DictionaryFragment are defined in IEC 62541-3.

7.8 ServerStatusType

This complex *VariableType* is used for information about the *Server* status. Its *DataVariables* reflect its *DataType* having the same semantic defined in 12.10. The *VariableType* is formally defined in Table 67.

Table 67 – ServerStatusType Definition

Attribute		Value			
BrowseName		ServerStatusType			
IsAbstract		False			
ValueRank		−1 (−1 = Scalar)			
DataType		ServerStatusDataType			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasComponent	Variable	StartTime	UtcTime	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentTime	UtcTime	BaseDataVariableType	Mandatory
HasComponent	Variable	State	ServerState	BaseDataVariableType	Mandatory
HasComponent	Variable	BuildInfo[1]	BuildInfo	BuildInfoType	Mandatory
HasComponent	Variable	SecondsTillShutdown	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	ShutdownReason	LocalizedText	BaseDataVariableType	Mandatory
NOTE Containing *Objects* and *Variables* of these *Objects* and *Variables* are defined by their *BrowseName* defined in the corresponding *TypeDefinitionNode*. The *NodeId* is defined by the composed symbolic name described in 4.1.					

7.9 BuildInfoType

This complex *VariableType* is used for information about the *Server* status. Its *DataVariables* reflect its *DataType* having the same semantic defined in 12.4. The *VariableType* is formally defined in Table 68.

Table 68 – BuildInfoType Definition

Attribute		Value			
BrowseName		BuildInfoType			
IsAbstract		False			
ValueRank		−1 (−1 = Scalar)			
DataType		BuildInfo			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasComponent	Variable	ProductUri	String	BaseDataVariableType	Mandatory
HasComponent	Variable	ManufacturerName	String	BaseDataVariableType	Mandatory
HasComponent	Variable	ProductName	String	BaseDataVariableType	Mandatory
HasComponent	Variable	SoftwareVersion	String	BaseDataVariableType	Mandatory
HasComponent	Variable	BuildNumber	String	BaseDataVariableType	Mandatory
HasComponent	Variable	BuildDate	UtcTime	BaseDataVariableType	Mandatory

7.10 ServerDiagnosticsSummaryType

This complex *VariableType* is used for diagnostic information. Its *DataVariables* reflect its *DataType* having the same semantic defined in 12.9. The *VariableType* is formally defined in Table 69.

Table 69 – ServerDiagnosticsSummaryType Definition

Attribute		Value			
BrowseName		ServerDiagnosticsSummaryType			
IsAbstract		False			
ValueRank		−1 (−1 = Scalar)			
DataType		ServerDiagnosticsSummaryDataType			
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasComponent	Variable	ServerViewCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentSessionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CumulatedSessionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	SecurityRejectedSessionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	RejectedSessionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	SessionTimeoutCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	SessionAbortCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	PublishingIntervalCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentSubscriptionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CumulatedSubscriptionCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	SecurityRejectedRequestsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	RejectedRequestsCount	UInt32	BaseDataVariableType	Mandatory

7.11 SamplingIntervalDiagnosticsArrayType

This complex *VariableType* is used for diagnostic information. For each entry of the array, instances of this type will provide a *Variable* of the SamplingIntervalDiagnosticsType *VariableType* having the sampling rate as *BrowseName*. The *VariableType* is formally defined in Table 70.

Table 70 – SamplingIntervalDiagnosticsArrayType Definition

Attribute		Value		
BrowseName		SamplingIntervalDiagnosticsArrayType		
IsAbstract		False		
ValueRank		1 (1 = OneDimension)		
ArrayDimensions		{0} (0 = UnknownSize)		
DataType		SamplingIntervalDiagnosticsDataType		
References	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	VariableType	SamplingIntervalDiagnostics	SamplingIntervalDiagnosticsDataType SamplingIntervalDiagnosticsType	ExposesItsArray

7.12 SamplingIntervalDiagnosticsType

This complex *VariableType* is used for diagnostic information. Its *DataVariables* reflect its *DataType*, having the same semantic defined in 12.8. The *VariableType* is formally defined in Table 71.

Table 71 – SamplingIntervalDiagnosticsType Definition

Attribute			Value		
BrowseName			SamplingIntervalDiagnosticsType		
IsAbstract			False		
ValueRank			−1 (−1 = Scalar)		
DataType			SamplingIntervalDiagnosticsDataType		
References	**Node Class**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasComponent	Variable	SamplingInterval	Duration	BaseDataVariableType	Mandatory
HasComponent	Variable	SampledMonitoredItemsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	MaxSampledMonitoredItemsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	DisabledMonitoredItemsSamplingCount	UInt32	BaseDataVariableType	Mandatory

7.13 SubscriptionDiagnosticsArrayType

This complex *VariableType* is used for diagnostic information. For each entry of the array, instances of this type will provide a *Variable* of the SubscriptionDiagnosticsType *VariableType* having the SubscriptionId as *BrowseName*. The *VariableType* is formally defined in Table 72.

Table 72 – SubscriptionDiagnosticsArrayType Definition

Attribute		Value		
BrowseName		SubscriptionDiagnosticsArrayType		
IsAbstract		False		
ValueRank		1 (1 = OneDimension)		
ArrayDimensions		{0} (0 = UnknownSize)		
DataType		SubscriptionDiagnosticsDataType		
References	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	VariableType	SubscriptionDiagnostics	SubscriptionDiagnosticsDataType SubscriptionDiagnosticsType	ExposesItsArray

7.14 SubscriptionDiagnosticsType

This complex *VariableType* is used for diagnostic information. Its *DataVariables* reflect its *DataType*, having the same semantic defined in 12.15. The *VariableType* is formally defined in Table 73.

Table 73 – SubscriptionDiagnosticsType Definition

Attribute	Value				
BrowseName	SubscriptionDiagnosticsType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	SubscriptionDiagnosticsDataType				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.					
HasComponent	Variable	SessionId	NodeId	BaseDataVariableType	Mandatory
HasComponent	Variable	SubscriptionId	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	Priority	Byte	BaseDataVariableType	Mandatory
HasComponent	Variable	PublishingInterval	Duration	BaseDataVariableType	Mandatory
HasComponent	Variable	MaxKeepAliveCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	MaxLifetimeCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	MaxNotificationsPerPublish	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	PublishingEnabled	Boolean	BaseDataVariableType	Mandatory
HasComponent	Variable	ModifyCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	EnableCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	DisableCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	RepublishRequestCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	RepublishMessageRequestCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	RepublishMessageCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	TransferRequestCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	TransferredToAltClientCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	TransferredToSameClientCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	PublishRequestCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	DataChangeNotificationsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	EventNotificationsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	NotificationsCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	LatePublishRequestCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentKeepAliveCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentLifetimeCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	UnacknowledgedMessageCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	DiscardedMessageCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	MonitoredItemCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	DisabledMonitoredItemCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	MonitoringQueueOverflowCount	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	NextSequenceNumber	UInt32	BaseDataVariableType	Mandatory
HasComponent	Variable	EventQueueOverflowCount	UInt32	BaseDataVariableType	Mandatory

7.15 SessionDiagnosticsArrayType

This complex *VariableType* is used for diagnostic information. For each entry of the array instances of this type will provide a *Variable* of the SessionDiagnosticsVariableType *VariableType*, having the SessionDiagnostics as *BrowseName*. Those *Variables* will also be referenced by the SessionDiagnostics *Objects* defined by their type in 6.3.5. The *VariableType* is formally defined in Table 74.

Table 74 – SessionDiagnosticsArrayType Definition

Attribute	Value			
BrowseName	SessionDiagnosticsArrayType			
IsAbstract	False			
ValueRank	1 (1 = OneDimension)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	SessionDiagnosticsDataType			
References	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	Variable	SessionDiagnostics	SessionDiagnosticsDataType SessionDiagnosticsVariableType	ExposesItsArray

7.16 SessionDiagnosticsVariableType

This complex *VariableType* is used for diagnostic information. Its *DataVariables* reflect its *DataType*, having the same semantic defined in 12.11. The *VariableType* is formally defined in Table 75.

Table 75 – SessionDiagnosticsVariableType Definition

Attribute	Value			
BrowseName	SessionDiagnosticsVariableType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionDiagnosticsDataType			
References	**Node Class**	**BrowseName**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	Variable	SessionId	NodeId BaseDataVariableType	Mandatory
HasComponent	Variable	SessionName	String BaseDataVariableType	Mandatory
HasComponent	Variable	ClientDescription	ApplicationDescription BaseDataVariableType	Mandatory
HasComponent	Variable	ServerUri	String BaseDataVariableType	Mandatory
HasComponent	Variable	EndpointUrl	String BaseDataVariableType	Mandatory
HasComponent	Variable	LocaleIds	LocaleId[] BaseDataVariableType	Mandatory
HasComponent	Variable	MaxResponseMessageSize	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	ActualSessionTimeout	Duration BaseDataVariableType	Mandatory
HasComponent	Variable	ClientConnectionTime	UtcTime BaseDataVariableType	Mandatory
HasComponent	Variable	ClientLastContactTime	UtcTime BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentSubscriptionsCount	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentMonitoredItemsCount	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	CurrentPublishRequestsInQueue	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	TotalRequestsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	UnauthorizedRequestsCount	UInt32 BaseDataVariableType	Mandatory
HasComponent	Variable	ReadCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	HistoryReadCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	WriteCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	HistoryUpdateCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	CallCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	CreateMonitoredItemsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	ModifyMonitoredItemsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	SetMonitoringModeCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	SetTriggeringCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	DeleteMonitoredItemsCount	ServiceCounterDataType BaseDataVariableType	Mandatory

Attribute	Value			
BrowseName	SessionDiagnosticsVariableType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionDiagnosticsDataType			
References	**Node Class**	**BrowseName**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	Variable	CreateSubscriptionCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	ModifySubscriptionCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	SetPublishingModeCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	PublishCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	RepublishCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	TransferSubscriptionsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	DeleteSubscriptionsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	AddNodesCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	AddReferencesCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	DeleteNodesCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	DeleteReferencesCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	BrowseCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	BrowseNextCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	TranslateBrowsePathsToNodeIdsCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	QueryFirstCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	QueryNextCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	RegisterNodesCount	ServiceCounterDataType BaseDataVariableType	Mandatory
HasComponent	Variable	UnregisterNodesCount	ServiceCounterDataType BaseDataVariableType	Mandatory

7.17 SessionSecurityDiagnosticsArrayType

This complex *VariableType* is used for diagnostic information. For each entry of the array instances of this type will provide a *Variable* of the SessionSecurityDiagnosticsType *VariableType*, having the SessionSecurityDiagnostics as *BrowseName*. Those *Variables* will also be referenced by the SessionDiagnostics *Objects* defined by their type in 6.3.5. The *VariableType* is formally defined in Table 76. Since this information is security related, it should not be made accessible to all users, but only to authorised users.

Table 76 – SessionSecurityDiagnosticsArrayType Definition

Attribute	Value			
BrowseName	SessionSecurityDiagnosticsArrayType			
IsAbstract	False			
ValueRank	1 (1 = OneDimension)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	SessionSecurityDiagnosticsDataType			
References	**Node Class**	**Browse Name**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4.				
HasComponent	Variable	SessionSecurityDiagnostics	SessionSecurityDiagnosticsDataType SessionSecurityDiagnosticsType	ExposesItsArray

7.18 SessionSecurityDiagnosticsType

This complex *VariableType* is used for diagnostic information. Its *DataVariables* reflect its *DataType*, having the same semantic defined in 12.12. The *VariableType* is formally defined in Table 77. Since this information is security-related, it should not be made accessible to all users, but only to authorised users.

Table 77 – SessionSecurityDiagnosticsType Definition

Attribute	Value			
BrowseName	SessionSecurityDiagnosticsType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionSecurityDiagnosticsDataType			
References	**Node Class**	**BrowseName**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4				
HasComponent	Variable	SessionId	NodeId BaseDataVariableType	Mandatory
HasComponent	Variable	ClientUserIdOfSession	String BaseDataVariableType	Mandatory
HasComponent	Variable	ClientUserIdHistory	String[] BaseDataVariableType	Mandatory
HasComponent	Variable	AuthenticationMechanism	String BaseDataVariableType	Mandatory
HasComponent	Variable	Encoding	String BaseDataVariableType	Mandatory
HasComponent	Variable	TransportProtocol	String BaseDataVariableType	Mandatory
HasComponent	Variable	SecurityMode	MessageSecurityMode BaseDataVariableType	Mandatory
HasComponent	Variable	SecurityPolicyUri	String BaseDataVariableType	Mandatory
HasComponent	Variable	ClientCertificate	ByteString BaseDataVariableType	Mandatory

7.19 OptionSetType

The *OptionSetType VariableType* is used to represent a bit mask and the *OptionSetValues Property* contains the human-readable representation for each bit of the bit mask set to true. The order of the bits of the bit mask points to a position of the array, i.e. the first bit (least significant bit) points to the first entry in the array, etc.

The *DataType* of this *VariableType* shall be capable of representing a bit mask. It shall be either a numeric *DataType* representing a signed or unsigned integer, or a *ByteString*. For example, it can be the *BitFieldMaskDataType.*

The optional BitMask *Property* provides the bit mask in an array of Booleans. This allows subscribing to individual entries of the bit mask. The order of the bits of the bit mask points to a position of the array, i.e. the first bit points to the first entry in the array, etc.

The OptionSetValues array contains an empty LocalizedText for each bit that has no specific meaning. The *VariableType* is formally defined in Table 76.

Table 78 – OptionSetType Definition

Attribute	**Value**			
BrowseName	OptionSetType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	BaseDataType			
References	**NodeClass**	**Browse Name**	**DataType TypeDefinition**	**Modelling Rule**
Subtype of the BaseDataVariableType defined in 7.4				
HasProperty	Variable	OptionSetValues	LocalizedText[] PropertyType	Mandatory
HasProperty	Variable	BitMask	Boolean[] PropertyType	Optional

8 Standard Objects and their Variables

8.1 General

Objects and *Variables* described in the following subclauses can be extended by additional *Properties* or *References* to other *Nodes*, except where it is stated in the text that it is restricted.

8.2 Objects used to organise the AddressSpace structure

8.2.1 Overview

To promote interoperability of clients and *Servers*, the OPC UA *AddressSpace* is structured as a hierarchy, with the top levels standardised for all *Servers*. Figure 1 illustrates the structure of the *AddressSpace*. All *Objects* in this figure are organised using *Organizes References* and have the *ObjectType FolderType* as type definition.

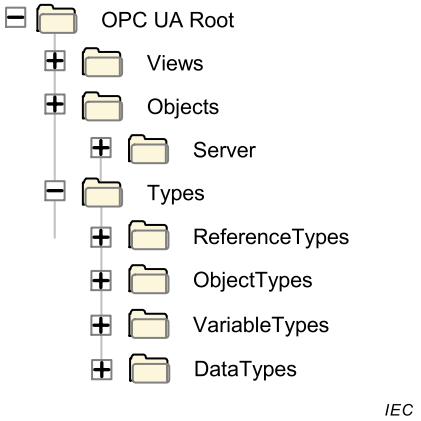


Figure 1 – Standard AddressSpace Structure

The remainder of this provides descriptions of these standard *Nodes* and the organization of *Nodes* beneath them. *Servers* typically implement a subset of these standard *Nodes*, depending on their capabilities.

8.2.2 Root

This standard *Object* is the browse entry point for the *AddressSpace*. It contains a set of *Organizes References* that point to the other standard *Objects*. The "*Root*" *Object* shall not reference any other *NodeClasses*. It is formally defined in Table 79.

Table 79 – Root Definition

Attribute	**Value**		
BrowseName	Root		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	Object	Views	Defined in 8.2.3
Organizes	Object	Objects	Defined in 8.2.4
Organizes	Object	Types	Defined in 8.2.5

8.2.3 Views

This standard *Object* is the browse entry point for *Views*. Only *Organizes References* are used to relate *View Nodes* to the "*Views*" standard *Object*. All *View Nodes* in the *AddressSpace* shall be referenced by this *Node*, either directly or indirectly. That is, the "*Views*" *Object* may reference other *Objects* using *Organizes References*. Those *Objects* may reference additional *Views*. Figure 2 illustrates the Views Organization. The "*Views"* standard *Object* directly references the *Views* "View1" and "View2" and indirectly "View3" by referencing another *Object* called "Engineering".

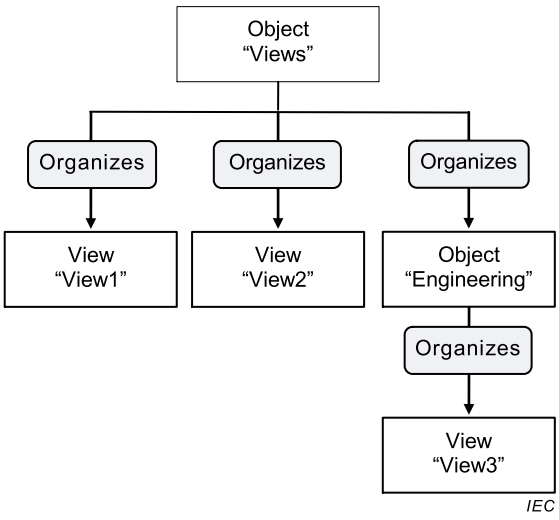


Figure 2 – Views Organization

The "*Views*" *Object* shall not reference any other *NodeClasses*. The "*Views*" *Object* is formally defined in Table 80.

Table 80 – Views Definition

Attribute	Value		
BrowseName	Views		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6

8.2.4 Objects

This standard *Object* is the browse entry point for *Object Nodes*. Figure 3 illustrates the structure beneath this *Node*. Only *Organizes References* are used to relate *Objects* to the "*Objects*" standard *Object*. A *View Node* can be used as entry point into a subset of the *AddressSpace* containing *Objects* and *Variables* and thus the "*Objects*" *Object* can also reference *View Nodes* using *Organizes References*. The intent of the "*Objects*" *Object* is that all *Objects* and *Variables* that are not used for type definitions or other organizational purposes (e.g. organizing the *Views*) are accessible through *hierarchical References* starting from this *Node*. However, this is not a requirement, because not all *Servers* may be able to support this. This *Object* references the standard *Server Object* defined in 8.3.2.

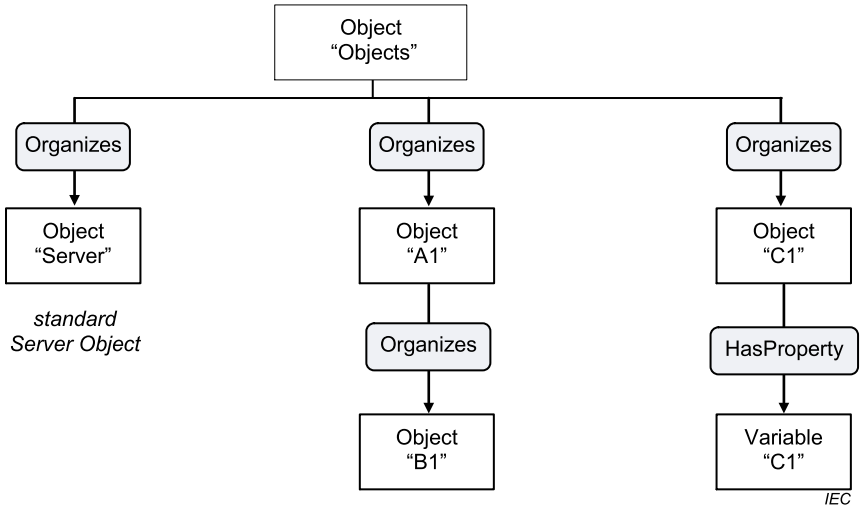


Figure 3 – Objects Organization

The "*Objects*" *Object* shall not reference any other *NodeClasses*. The "*Objects*" *Object* is formally defined in Table 81.

Table 81 – Objects Definition

Attribute	Value		
BrowseName	Objects		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	Object	Server	Defined in 8.3.2

8.2.5 Types

This standard *Object Node* is the browse entry point for type *Nodes*. Figure 1 illustrates the structure beneath this *Node*. Only *Organizes References* are used to relate *Objects* to the "*Types*" standard *Object*. The "*Types*" *Object* shall not reference any other *NodeClasses*. It is formally defined in Table 82.

Table 82 – Types Definition

Attribute	Value		
BrowseName	Types		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	Object	ObjectTypes	Defined in 8.2.6
Organizes	Object	VariableTypes	Defined in 8.2.7
Organizes	Object	ReferenceTypes	Defined in 8.2.8
Organizes	Object	DataTypes	Defined in 8.2.9
Organizes	Object	EventTypes	Defined in 8.2.12

8.2.6 ObjectTypes

This standard *Object Node* is the browse entry point for *ObjectType Nodes*. Figure 4 illustrates the structure beneath this *Node* showing some of the standard *ObjectTypes* defined in 6. Only *Organizes References* are used to relate *Objects* and *ObjectTypes* to the "*ObjectTypes*" standard *Object*. The "*ObjectTypes*" *Object* shall not reference any other *NodeClasses*.

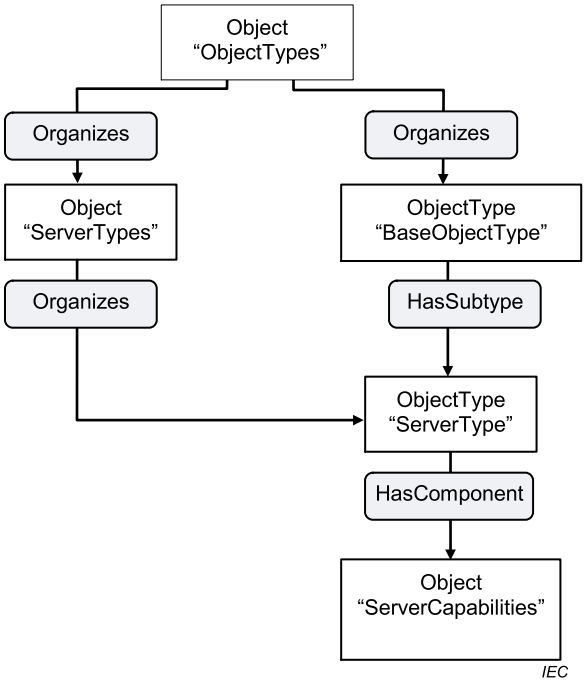


Figure 4 – ObjectTypes Organization

The intention of the "*ObjectTypes*" *Object* is that all *ObjectTypes* of the *Server* are either directly or indirectly accessible browsing *HierarchicalReferences* starting from this *Node*. However, this is not required and *Servers* might not provide some of their *ObjectTypes* because they may be well-known in the industry, such as the *ServerType* defined in 6.3.1.

This *Object* also indirectly references the *BaseEventType* defined in 6.4.2, which is the base type of all *EventTypes*. Thereby it is the entry point for all *EventTypes* provided by the *Server*. It is required that the *Server* expose all its *EventTypes*, so a client can usefully subscribe to *Events*.

The "*ObjectTypes*" *Object* is formally defined in Table 83.

Table 83 – ObjectTypes Definition

Attribute	Value		
BrowseName	ObjectTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	ObjectType	BaseObjectType	Defined in 6.2

8.2.7 VariableTypes

This standard *Object* is the browse entry point for *VariableType Nodes*. Figure 5 illustrates the structure beneath this *Node*. Only *Organizes References* are used to relate *Objects* and *VariableTypes* to the "*VariableTypes*" standard *Object*. The "*VariableTypes*" *Object* shall not reference any other *NodeClasses*.

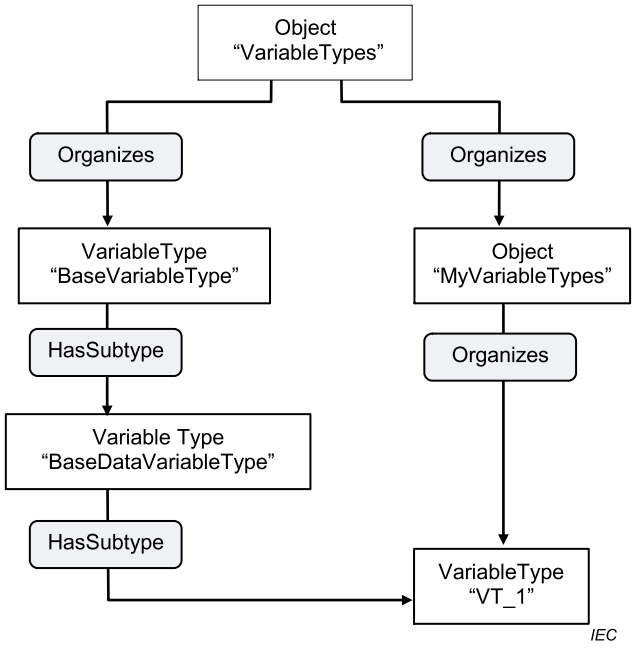


Figure 5 – VariableTypes Organization

The intent of the "*VariableTypes*" *Object* is that all *VariableTypes* of the *Server* are either directly or indirectly accessible browsing *HierarchicalReferences* starting from this *Node*. However, this is not required and *Servers* might not provide some of their *VariableTypes*, because they may be well-known in the industry, such as the "*BaseVariableType*" defined in 7.2.

The "*VariableTypes*" *Object* is formally defined in Table 84.

Table 84 – VariableTypes Definition

Attribute	Value		
BrowseName	VariableTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	VariableType	BaseVariableType	Defined in 7.2

8.2.8 ReferenceTypes

This standard *Object* is the browse entry point for *ReferenceType Nodes*. Figure 6 illustrates the organization of *ReferenceTypes*. *Organizes References* are used to define *ReferenceTypes* and *Objects* referenced by the "*ReferenceTypes*" *Object*. The "*ReferenceTypes*" *Object* shall not reference any other *NodeClasses*. See Clause 11 for a discussion of the standard *ReferenceTypes* that appear beneath the "*ReferenceTypes*" *Object*.

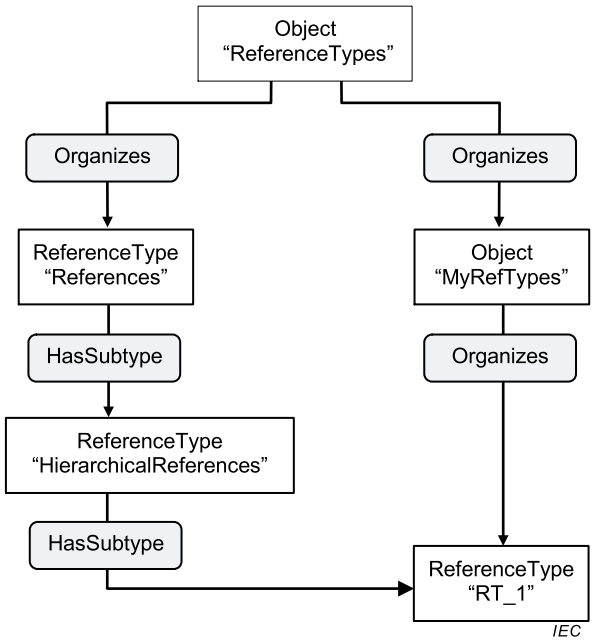


Figure 6 – ReferenceType Definitions

Since *ReferenceTypes* will be used as filters in the browse *Service* and in queries, the *Server* shall provide all its *ReferenceTypes*, directly or indirectly following *hierarchical References* starting from the "*ReferenceTypes*" *Object*. This means that, whenever the client follows a *Reference*, the *Server* shall expose the type of this Reference in the *ReferenceType* hierarchy. It shall provide all *ReferenceTypes* so that the client would be able, following the inverse subtype of *References*, to come to the base *References ReferenceType*. It does not mean that the *Server* shall expose the *ReferenceTypes* that the client has not used any *Reference* of.

The "*ReferenceTypes*" *Object* is formally defined in Table 85.

Table 85 – ReferenceTypes Definition

Attribute	**Value**		
BrowseName	ReferenceTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	ReferenceType	References	Defined in 11.1

8.2.9 DataTypes

This standard *Object* is the browse entry point for *DataTypes* that the *Server* wishes to expose in the *AddressSpace*. The standard *Object* uses *Organizes References* to reference *Objects* of the *DataTypeSystemType* representing *DataTypeSystems*. Referenced by those *Objects* are *DataTypeDictionaries* that refer to their *DataTypeDescriptions*. However, it is not

required to provide the *DataTypeSystem Objects*, and the *DataTypeDictionary* need not to be provided.

Because *DataTypes* are not related to *DataTypeDescriptions* using *hierarchical References*, *DataType Nodes* should be made available using *Organizes References* pointing either directly from the "DataTypes" *Object* to the *DataType Nodes* or using additional *Folder Objects* for grouping purposes. The intent is that all *DataTypes* of the *Server* exposed in the *AddressSpace* are accessible following *hierarchical References* starting from the "DataTypes" *Object*. However, this is not required.

Figure 7 illustrates this hierarchy using the "OPC Binary" and "XML Schema" standard *DataTypeSystems* as examples. Other *DataTypeSystems* may be defined under this *Object*.

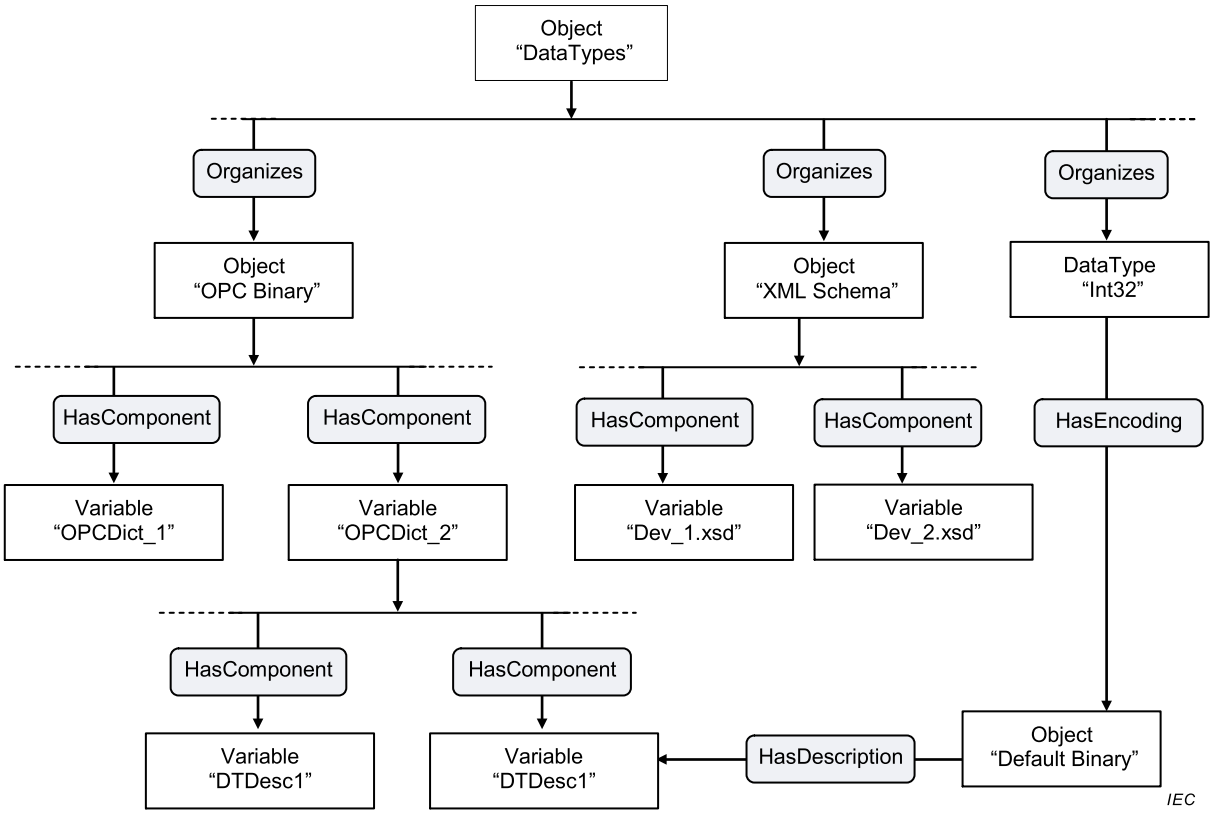


Figure 7 – DataTypes Organization

Each *DataTypeSystem Object* is related to its *DataTypeDictionary Nodes* using *HasComponent References*. Each *DataTypeDictionary Node* is related to its *DataTypeDescription Nodes* using *HasComponent References*. These *References* indicate that the *DataTypeDescriptions* are defined in the dictionary.

In the example, the "DataTypes" *Object* references the *DataType* "Int32" using an *Organizes Reference*. The *DataType* uses the non-hierarchical *HasEncoding Reference* to point to its default encoding, which references a *DataTypeDescription* using the non-hierarchical *HasDescription Reference*.

The "*DataTypes*" *Object* is formally defined in Table 86.

Table 86 – DataTypes Definition

Attribute	Value		
BrowseName	DataTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	Object	OPC Binary	Defined in 8.2.10
Organizes	Object	XML Schema	Defined in 8.2.11
Organizes	DataType	BaseDataType	Defined in 12.2

8.2.10 OPC Binary

OPC Binary is a standard *DataTypeSystem* defined by OPC. It is represented in the *AddressSpace* by an *Object Node*. The OPC Binary *DataTypeSystem* is defined in IEC 62541-3. OPC Binary uses XML to describe complex binary data values. The "*OPC Binary*" *Object* is formally defined in Table 87.

Table 87 – OPC Binary Definition

Attribute	Value		
BrowseName	OPC Binary		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	DataTypeSystemType	Defined in 6.8

8.2.11 XML Schema

XML Schema is a standard *DataTypeSystem* defined by the W3C. It is represented in the *AddressSpace* by an *Object Node*. XML Schema documents are XML documents whose xmlns attribute in the first line is:

schema xmlns =http://www.w3.org/1999/XMLSchema

The "*XML Schema*" *Object* is formally defined in Table 88.

Table 88 – XML Schema Definition

Attribute	Value		
BrowseName	XML Schema		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	DataTypeSystemType	Defined in 6.8

8.2.12 EventTypes

This standard *Object Node* is the browse entry point for *EventType Nodes*. Figure 8 illustrates the structure beneath this *Node* showing some of the standard *EventTypes* defined in Clause 6. Only *Organizes References* are used to relate *Objects* and *ObjectTypes* to the "*EventTypes*" standard *Object*. The "*EventTypes*" *Object* shall not reference any other *NodeClasses*.

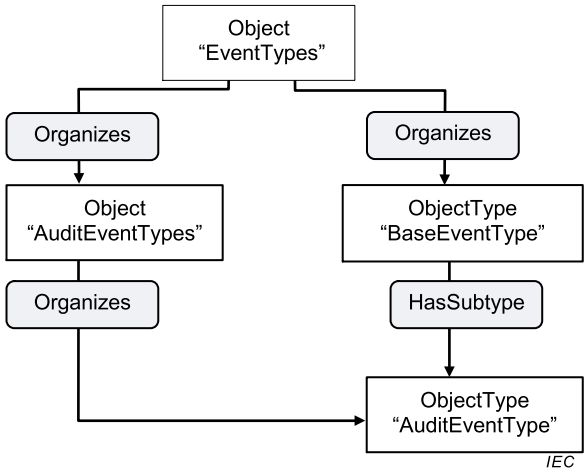


Figure 8 – EventTypes Organization

The intention of the “*EventTypes*” *Object* is that all *EventTypes* of the *Server* are either directly or indirectly accessible browsing *HierarchicalReferences* starting from this *Node*. It is required that the *Server* expose all its *EventTypes*, so a client can usefully subscribe to *Events*.

The “*EventTypes*” *Object* is formally defined in Table 89.

Table 89 – EventTypes Definition

Attribute	**Value**		
BrowseName	ObjectTypes		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Defined in 6.6
Organizes	ObjectType	BaseEventType	Defined in 6.4.2

8.3 Server Object and its containing Objects

8.3.1 General

The *Server Object* and its containing *Objects* and *Variables* are built in a way that the information can be gained in several ways, suitable for different kinds of clients having different requirements. Annex A gives an overview of the design decisions made in providing the information in that way, and discusses the pros and cons of the different approaches. Figure 9 gives an overview of the containing *Objects* and *Variables* of the diagnostic information of the Server *Object* and where the information can be found.

The SessionsDiagnosticsSummary *Object* contains one *Object* per session and a *Variable* with an array with one entry per session. This array is of a complex *DataType* holding the diagnostic information about the session. Each *Object* representing a session references a complex *Variable* containing the information about the session using the same DataType as the array containing information about all sessions. Such a *Variable* also exposes all its information as *Variables* with simple *DataTypes* containing the same information as in the complex *DataType*. Not shown in Figure 9 is the security-related information per session, which follows the same rules.

The *Server* provides an array with an entry per subscription containing diagnostic information about this subscription. Each entry of this array is also exposed as a complex *Variable* with *Variables* for each individual value. Each *Object* representing a session also provides such an array, but providing the subscriptions of the session.

The arrays containing information about the sessions or the subscriptions may be of different length for different connections with different user credentials since not all users may see all entries of the array. That also implies that the length of the array may change if the user is impersonated. Therefore clients that subscribe to a specific index range may get unexpected results.

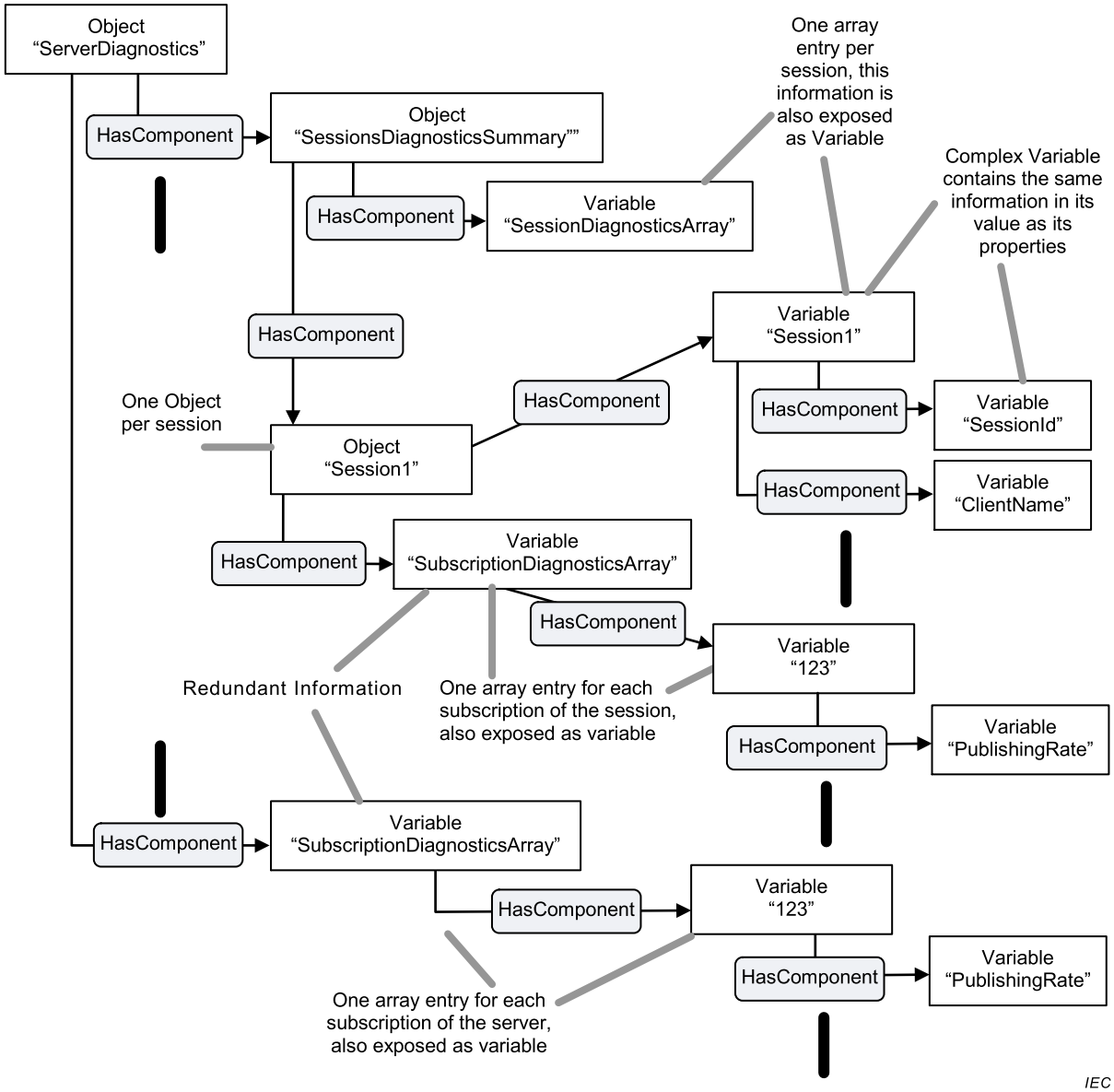


Figure 9 – Excerpt of Diagnostic Information of the Server

8.3.2 Server Object

This *Object* is used as the browse entry point for information about the *Server*. The content of this *Object* is already defined by its type definition in 6.3.1. It is formally defined in Table 90. The *Server Object* serves as root notifier, that is, its *EventNotifier Attribute* shall be set providing *Events*. All *Events* of the *Server* shall be accessible subscribing to the *Events* of the *Server Object*.

Table 90 – Server Definition

Attribute	Value				
BrowseName	Server				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasTypeDefinition	Object Type	ServerType	Defined in 6.3.1		
HasProperty	Variable	ServerArray	String[]	PropertyType	Mandatory
HasProperty	Variable	NamespaceArray	String[]	PropertyType	Mandatory
HasComponent	Variable	ServerStatus[a]	ServerStatusDataType	ServerStatusType	Mandatory
HasProperty	Variable	ServiceLevel	Byte	PropertyType	Mandatory
HasComponent	Object	ServerCapabilities[a]	--	ServerCapabilities	Mandatory
HasComponent	Object	ServerDiagnostics[a]	--	ServerDiagnostics Type	Mandatory
HasComponent	Object	VendorServerInfo	--	vendor-specific[b]	Mandatory
HasComponent	Object	ServerRedundancy[a]	--	depends on supported redundancy[c]	Mandatory

[a] Containing *Objects* and *Variables* of these *Objects* and *Variables* are defined by their *BrowseName* defined in the corresponding *TypeDefinitionNode*. The *NodeId* is defined by the composed symbolic name described in 4.1.

[b] Shall be the *VendorServerInfo ObjectType* or one of its subtypes.

[c] Shall be the *ServerRedundancyType* or one of its subtypes.

8.4 ModellingRule Objects

8.4.1 ExposesItsArray

The *ModellingRule ExposesItsArray* is defined in IEC 62541-3. Its representation in the *AddressSpace,* the "*ExposesItsArray*" *Object,* is formally defined in Table 91.

Table 91 – ExposesItsArray Definition

Attribute	Value		
BrowseName	ExposesItsArray		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Value set to "Constraint"

8.4.2 Mandatory

The *ModellingRule Mandatory* is defined in IEC 62541-3. Its representation in the *AddressSpace,* the "*Mandatory*" *Object,* is formally defined in Table 92.

Table 92 – Mandatory Definition

Attribute	Value		
BrowseName	Mandatory		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Value set to "Mandatory"

8.4.3 Optional

The *ModellingRule Optional* is defined in IEC 62541-3. Its representation in the *AddressSpace*, the "*Optional*" *Object,* is formally defined in Table 93.

Table 93 – Optional Definition

Attribute	Value		
BrowseName	Optional		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Value set to “Optional”

8.4.4 OptionalPlaceholder

The *ModellingRule OptionalPlaceholder* is defined in IEC 62541-3. Its representation in the *AddressSpace*, the “*OptionalPlaceholder*” *Object,* is formally defined in Table 94.

Table 94 – OptionalPlaceholder Definition

Attribute	Value		
BrowseName	OptionalPlaceholder		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Value set to “Constraint”

8.4.5 MandatoryPlaceholder

The *ModellingRule MandatoryPlaceholder* is defined in IEC 62541-3. Its representation in the *AddressSpace*, the “*MandatoryPlaceholder*” *Object,* is formally defined in Table 95.

Table 95 – MandatoryPlaceholder Definition

Attribute	Value		
BrowseName	MandatoryPlaceholder		
References	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Value set to “Constraint”

9 Standard Methods – GetMonitoredItems

GetMonitoredItems is used to get information about monitored items of a subscription.Its intended use is defined in IEC 62541-4.

Signature

```
GetMonitoredItems(
    [in] UInt32 subscriptionId
    [out] UInt32[] serverHandles
    [out] UInt32[] clientHandles
  );
```

Argument	Description
subscriptionId	Identifier of the subscription.
serverHandles	Array of serverHandles for all MonitoredItems of the subscription identified by subscriptionId
clientHandles	Array of clientHandles for all MonitoredItems of the subscription identified by subscriptionId

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_SubscriptionIdInvalid	Defined in IEC 62541-4

Table 96 specifies the *AddressSpace* representation for the *GetMonitoredItems Method*.

Table 96 – GetMonitoredItems Method AddressSpace Definition

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

10 Standard Views

There are no core OPC UA *Views* defined.

11 Standard ReferenceTypes

11.1 References

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 97.

Table 97 – References ReferenceType

Attributes	Value		
BrowseName	References		
InverseName	--		
Symmetric	True		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HierarchicalReferences	Defined in 11.2
HasSubtype	ReferenceType	NonHierarchicalReferences	Defined in 11.3

11.2 HierarchicalReferences

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 98.

Table 98 – HierarchicalReferences ReferenceType

Attributes	Value		
BrowseName	HierarchicalReferences		
InverseName	--		
Symmetric	False		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasChild	Defined in 11.4
HasSubtype	ReferenceType	Organizes	Defined in 11.6
HasSubtype	ReferenceType	HasEventSource	Defined in 11.15

11.3 NonHierarchicalReferences

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 99.

Table 99 – NonHierarchicalReferences ReferenceType

Attributes	Value		
BrowseName	NonHierarchicalReferences		
InverseName	--		
Symmetric	True		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasModellingRule	Defined in 11.11
HasSubtype	ReferenceType	HasTypeDefinition	Defined in 11.12
HasSubtype	ReferenceType	HasEncoding	Defined in 11.13
HasSubtype	ReferenceType	HasDescription	Defined in 11.14
HasSubtype	ReferenceType	GeneratesEvent	Defined in 11.17

11.4 HasChild

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 100.

Table 100 – HasChild ReferenceType

Attributes	Value		
BrowseName	HasChild		
InverseName	--		
Symmetric	False		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	Aggregates	Defined in 11.5
HasSubtype	ReferenceType	HasSubtype	Defined in 11.10

11.5 Aggregates

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 101.

Table 101 – Aggregates ReferenceType

Attributes	Value		
BrowseName	Aggregates		
InverseName	--		
Symmetric	False		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasComponent	Defined in 11.7
HasSubtype	ReferenceType	HasProperty	Defined in 11.9

11.6 Organizes

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 102.

Table 102 – Organizes ReferenceType

Attributes	Value		
BrowseName	Organizes		
InverseName	OrganizedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.7 HasComponent

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 103.

Table 103 – HasComponent ReferenceType

Attributes	Value		
BrowseName	HasComponent		
InverseName	ComponentOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasOrderedComponent	Defined in 11.8

11.8 HasOrderedComponent

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 104.

Table 104 – HasOrderedComponent ReferenceType

Attributes	Value		
BrowseName	HasOrderedComponent		
InverseName	OrderedComponentOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.9 HasProperty

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 105.

Table 105 – HasProperty ReferenceType

Attributes	Value		
BrowseName	HasProperty		
InverseName	PropertyOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.10 HasSubtype

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 106.

Table 106 – HasSubtype ReferenceType

Attributes	Value		
BrowseName	HasSubtype		
InverseName	SubtypeOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.11 HasModellingRule

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 107.

Table 107 – HasModellingRule ReferenceType

Attributes	Value		
BrowseName	HasModellingRule		
InverseName	ModellingRuleOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.12 HasTypeDefinition

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 108.

Table 108 – HasTypeDefinition ReferenceType

Attributes	Value		
BrowseName	HasTypeDefinition		
InverseName	TypeDefinitionOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.13 HasEncoding

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 109.

Table 109 – HasEncoding ReferenceType

Attributes	Value		
BrowseName	HasEncoding		
InverseName	EncodingOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.14 HasDescription

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 110.

Table 110 – HasDescription ReferenceType

Attributes	**Value**		
BrowseName	HasDescription		
InverseName	DescriptionOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.15 HasEventSource

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 111.

Table 111 – HasEventSource ReferenceType

Attributes	**Value**		
BrowseName	HasEventSource		
InverseName	EventSourceOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasNotifier	Defined in 11.16

11.16 HasNotifier

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 112.

Table 112 – HasNotifier ReferenceType

Attributes	**Value**		
BrowseName	HasNotifier		
InverseName	NotifierOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

11.17 GeneratesEvent

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 113.

Table 113 – GeneratesEvent ReferenceType

Attributes	Value		
BrowseName	GeneratesEvent		
InverseName	GeneratedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	AlwaysGeneratesEvent	Defined in 11.18

11.18 AlwaysGeneratesEvent

This standard *ReferenceType* is defined in IEC 62541-3. Its representation in the *AddressSpace* is specified in Table 114.

Table 114 – AlwaysGeneratesEvent ReferenceType

Attributes	Value		
BrowseName	AlwaysGeneratesEvent		
InverseName	AlwaysGeneratedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

12 Standard DataTypes

12.1 Overview

An OPC UA *Server* need not expose its *DataTypes* in its *AddressSpace*. Independent of the exposition of *DataTypes*, it shall support the *DataTypes* as described in the following subclauses.

12.2 DataTypes defined in IEC 62541-3

IEC 62541-3 defines a set of *DataTypes*. Their representation in the *AddressSpace* is defined in Table 115.

Table 115 – IEC 62541-3 DataType Definitions

BrowseName
BaseDataType
Argument
Boolean
Byte
ByteString
DateTime
Double
Duration
Enumeration
Float
Guid
IdType
SByte
Integer
Int16
Int32
Int64
Image
ImageBMP
ImageGIF
ImageJPG
ImagePNG
LocaleId
LocalizedText
NamingRuleType
NodeClass
NodeId
Number
QualifiedName
String
Structure
Time
UInteger
UInt16
UInt32
UInt64
UtcTime
XmlElement
TimeZoneDataType
EnumValueType

Of the *DataTypes* defined in Table 115 only some are the sources of *References* as defined in the following tables.

The *References* of the *BaseDataType* are defined in Table 116.

Table 116 – BaseDataType Definition

Attributes	Value		
BrowseName	BaseDataType		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Boolean	FALSE
HasSubtype	DataType	ByteString	FALSE
HasSubtype	DataType	DateTime	FALSE
HasSubtype	DataType	DataValue	FALSE
HasSubtype	DataType	DiagnosticInfo	FALSE
HasSubtype	DataType	Enumeration	TRUE
HasSubtype	DataType	ExpandedNodeId	FALSE
HasSubtype	DataType	Guid	FALSE
HasSubtype	DataType	LocalizedText	FALSE
HasSubtype	DataType	NodeId	FALSE
HasSubtype	DataType	Number	TRUE
HasSubtype	DataType	QualifiedName	FALSE
HasSubtype	DataType	String	FALSE
HasSubtype	DataType	Structure	TRUE
HasSubtype	DataType	XmlElement	FALSE

The *References* of *Structure* are defined in Table 117.

Table 117 – Structure Definition

Attributes	Value		
BrowseName	Structure		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Argument	FALSE
HasSubtype	DataType	UserIdentityToken	TRUE
HasSubtype	DataType	AddNodesItem	FALSE
HasSubtype	DataType	AddReferencesItem	FALSE
HasSubtype	DataType	DeleteNodesItem	FALSE
HasSubtype	DataType	DeleteReferencesItem	FALSE
HasSubtype	DataType	ApplicationDescription	FALSE
HasSubtype	DataType	BuildInfo	FALSE
HasSubtype	DataType	RedundantServerDataType	FALSE
HasSubtype	DataType	SamplingIntervalDiagnosticsDataType	FALSE
HasSubtype	DataType	ServerDiagnosticsSummaryDataType	FALSE
HasSubtype	DataType	ServerStatusDataType	FALSE
HasSubtype	DataType	SessionDiagnosticsDataType	FALSE
HasSubtype	DataType	SessionSecurityDiagnosticsDataType	FALSE
HasSubtype	DataType	ServiceCounterDataType	FALSE
HasSubtype	DataType	StatusResult	FALSE
HasSubtype	DataType	SubscriptionDiagnosticsDataType	FALSE
HasSubtype	DataTypes	ModelChangeStructureDataType	FALSE
HasSubtype	DataTypes	SemanticChangeStructureDataType	FALSE
HasSubtype	DataType	SignedSoftwareCertificate	FALSE
HasSubtype	DataType	TimeZoneDataType	FALSE
HasSubtype	DataType	EnumValueType	FALSE

The *References* of *Enumeration* are defined in Table 118.

Table 118 – Enumeration Definition

Attributes	Value		
BrowseName	Enumeration		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	IdType	FALSE
HasSubtype	DataType	NamingRuleType	FALSE
HasSubtype	DataType	NodeClass	FALSE
HasSubtype	DataType	SecurityTokenRequestType	FALSE
HasSubtype	DataType	MessageSecurityMode	FALSE
HasSubtype	DataType	RedundancySupport	FALSE
HasSubtype	DataType	ServerState	FALSE

The *References* of *ByteString* are defined in Table 119.

Table 119 – ByteString Definition

Attributes	Value		
BrowseName	ByteString		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Image	TRUE

The *References* of *Number* are defined in Table 120.

Table 120 – Number Definition

Attributes	Value		
BrowseName	Number		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Integer	TRUE
HasSubtype	DataType	UInteger	TRUE
HasSubtype	DataType	Double	FALSE
HasSubtype	DataType	Float	FALSE

The *References* of *Double* are defined in Table 121.

Table 121 – Double Definition

Attributes	Value		
BrowseName	Double		
IsAbstract	FALSE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Duration	FALSE

The *References* of *Integer* are defined in Table 122.

Table 122 – Integer Definition

Attributes	Value		
BrowseName	Integer		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	SByte	FALSE
HasSubtype	DataType	Int16	FALSE
HasSubtype	DataType	Int32	FALSE
HasSubtype	DataType	Int64	FALSE

The *References* of *DateTime* are defined in Table 123.

Table 123 – DateTime Definition

Attributes	Value		
BrowseName	DateTime		
IsAbstract	FALSE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	UtcTime	FALSE

The *References* of *String* are defined in Table 124.

Table 124 – String Definition

Attributes	Value		
BrowseName	String		
IsAbstract	FALSE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	LocaleId	FALSE
HasSubtype	DataType	NumericRange	FALSE

The *References* of UInteger are defined in Table 125.

Table 125 – UInteger Definition

Attributes	Value		
BrowseName	UInteger		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Byte	FALSE
HasSubtype	DataType	UInt16	FALSE
HasSubtype	DataType	UInt32	FALSE
HasSubtype	DataType	UInt64	FALSE

The *References* of Image are defined in Table 126.

Table 126 – Image Definition

Attributes	Value		
BrowseName	Image		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	ImageBMP	FALSE
HasSubtype	DataType	ImageGIF	FALSE
HasSubtype	DataType	ImageJPG	FALSE
HasSubtype	DataType	ImagePNG	FALSE

The *References* of UInt64 are defined in Table 127.

Table 127 – UInt64 Definition

Attributes	**Value**		
BrowseName	UInt64		
IsAbstract	FALSE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	BitFieldMaskDataType	FALSE

12.3 DataTypes defined in IEC 62541-4

IEC 62541-4 defines a set of *DataTypes*. Their representation in the *AddressSpace* is defined in Table 128.

Table 128 – IEC 62541-4 DataType Definitions

BrowseName
AnonymousIdentityToken
DataValue
DiagnosticInfo
ExpandedNodeId
SignedSoftwareCertificate
UserIdentityToken
UserNameIdentityToken
X509IdentityToken
WssIdentityToken
SecurityTokenRequestType
AddNodesItem
AddReferencesItem
DeleteNodesItem
DeleteReferencesItem
NumericRange
MessageSecurityMode
ApplicationDescription

The *SecurityTokenRequestType* is an enumeration that is defined as the type of the requestType parameter of the OpenSecureChannel *Service* in IEC 62541-4.

The *AddNodesItem* is a structure that is defined as the type of the nodesToAdd parameter of the AddNodes *Service* in IEC 62541-4.

The *AddReferencesItem* is a structure that is defined as the type of the referencesToAdd parameter of the AddReferences *Service* in IEC 62541-4.

The *DeleteNodesItem* is a structure that is defined as the type of the nodesToDelete parameter of the DeleteNodes *Service* in IEC 62541-4.

The *DeleteReferencesItem* is a structure that is defined as the type of the referencesToDelete parameter of the DeleteReferences *Service* in IEC 62541-4.

The *References* of *UserIdentityToken* are defined in Table 129.

Table 129 – UserIdentityToken Definition

Attributes	Value		
BrowseName	UserIdentityToken		
IsAbstract	TRUE		
References	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	UserNameIdentityToken	FALSE
HasSubtype	DataType	X509IdentityToken	FALSE
HasSubtype	DataType	WssIdentityToken	FALSE
HasSubtype	DataType	AnonymousIdentityToken	FALSE

12.4 BuildInfo

This structure contains elements that describe the build information of the *Server*. Its elements are defined in Table 130.

Table 130 – BuildInfo Structure

Name	Type	Description
BuildInfo	structure	Information that describes the build of the software.
productUri	String	URI that identifies the software
manufacturerName	String	Name of the software manufacturer.
productName	String	Name of the software.
softwareVersion	String	Software version
buildNumber	String	Build number
buildDate	UtcTime	Date and time of the build.

Its representation in the *AddressSpace* is defined in Table 131.

Table 131 – BuildInfo Definition

Attributes	Value
BrowseName	BuildInfo

12.5 RedundancySupport

This *DataType* is an enumeration that defines the redundancy support of the *Server*. Its values are defined in Table 132.

Table 132 – RedundancySupport Values

Value	Description
NONE_0	None means that there is no redundancy support.
COLD_1	Cold means that the server supports cold redundancy as defined in IEC 62541-4.
WARM_2	Warm means that the server supports warm redundancy as defined in IEC 62541-4.
HOT_3	Hot means that the server supports hot redundancy as defined in IEC 62541-4.
TRANSPARENT_4	Transparent means that the server supports transparent redundancy as defined in IEC 62541-4.
HOT_AND_MIRRORED_5	HotAndMirrored means that the server supports HotAndMirrored redundancy as defined in IEC 62541-4.

See IEC 62541-4 for a more detailed description of the different values.

Its representation in the *AddressSpace* is defined in Table 133.

Table 133 – RedundancySupport Definition

Attributes	Value
BrowseName	RedundancySupport

12.6 ServerState

This *DataType* is an enumeration that defines the execution state of the *Server*. Its values are defined in Table 134.

Table 134 – ServerState Values

Value	Description
RUNNING_0	The server is running normally. This is the usual state for a server.
FAILED_1	A vendor-specific fatal error has occurred within the server. The server is no longer functioning. The recovery procedure from this situation is vendor-specific. Most *Service* requests should be expected to fail.
NO_CONFIGURATION_2	The server is running but has no configuration information loaded and therefore does not transfer data.
SUSPENDED_3	The server has been temporarily suspended by some vendor-specific method and is not receiving or sending data.
SHUTDOWN_4	The server has shut down or is in the process of shutting down. Depending on the implementation, this might or might not be visible to clients.
TEST_5	The server is in Test Mode. The outputs are disconnected from the real hardware, but the server will otherwise behave normally. Inputs may be real or may be simulated depending on the vendor implementation. StatusCode will generally be returned normally.
COMMUNICATION_FAULT_6	The server is running properly, but is having difficulty accessing data from its data sources. This may be due to communication problems or some other problem preventing the underlying device, control system, etc. from returning valid data. It may be a complete failure, meaning that no data is available, or a partial failure, meaning that some data is still available. It is expected that items affected by the fault will individually return with a BAD FAILURE status code indication for the items.
UNKNOWN_7	This state is used only to indicate that the OPC UA server does not know the state of underlying servers.

Its representation in the *AddressSpace* is defined in Table 135.

Table 135 – ServerState Definition

Attributes	Value
BrowseName	ServerState

12.7 RedundantServerDataType

This structure contains elements that describe the status of the *Server*. Its composition is defined in Table 136.

Table 136 – RedundantServerDataType Structure

Name	Type	Description
RedundantServerDataType	structure	
serverId	String	The Id of the server (not the URI).
serviceLevel	Byte	The service level of the server.
serverState	ServerState	The current state of the server.

Its representation in the *AddressSpace* is defined in Table 137.

Table 137 – RedundantServerDataType Definition

Attributes	Value
BrowseName	RedundantServerDataType

12.8 SamplingIntervalDiagnosticsDataType

This structure contains diagnostic information about the sampling rates currently used by the *Server*. Its elements are defined in Table 138.

Table 138 – SamplingIntervalDiagnosticsDataType Structure

Name	Type	Description
SamplingIntervalDiagnosticsDataType	structure	
samplingInterval	Duration	The sampling interval in milliseconds.
sampledMonitoredItemsCount	UInt32	The number of *MonitoredItems* being sampled at this sample rate.
maxSampledMonitoredItemsCount	UInt32	The maximum number of *MonitoredItems* being sampled at this sample rate at the same time since the server was started (restarted).
disabledMonitoredItemsSamplingCount	UInt32	The number of *MonitoredItems* at this sample rate whose sampling currently disabled.

Its representation in the *AddressSpace* is defined in Table 139.

Table 139 – SamplingIntervalDiagnosticsDataType Definition

Attributes	Value
BrowseName	SamplingIntervalDiagnosticsDataType

12.9 ServerDiagnosticsSummaryDataType

This structure contains diagnostic summary information for the *Server*. Its elements are defined in Table 140.

Table 140 – ServerDiagnosticsSummaryDataType Structure

Name	Type	Description
ServerDiagnosticsSummaryDataType	structure	
serverViewCount	UInt32	The number of server-created views in the server.
currentSessionCount	UInt32	The number of client sessions currently established in the server.
cumulatedSessionCount	UInt32	The cumulative number of client sessions that have been established in the server since the server was started (or restarted). This includes the *currentSessionCount*.
securityRejectedSessionCount	UInt32	The number of client session establishment requests that were rejected due to security constraints since the server was started (or restarted).
rejectedSessionCount	UInt32	The number of client session establishment requests that were rejected since the server was started (or restarted). This number includes the *securityRejectedSessionCount*.
sessionTimeoutCount	UInt32	The number of client sessions that were closed due to timeout since the server was started (or restarted).
sessionAbortCount	UInt32	The number of client sessions that were closed due to errors since the server was started (or restarted).
publishingIntervalCount	UInt32	The number of publishing intervals currently supported in the server.
currentSubscriptionCount	UInt32	The number of subscriptions currently established in the server.
cumulatedSubscriptionCount	UInt32	The cumulative number of subscriptions that have been established in the server since the server was started (or restarted). This includes the *currentSubscriptionCount*.
securityRejectedRequestsCount	UInt32	The number of requests that were rejected due to security constraints since the server was started (or restarted). The requests include all *Services* defined in IEC 62541-4, also requests to create sessions.
rejectedRequestsCount	UInt32	The number of requests that were rejected since the server was started (or restarted). The requests include all *Services* defined in IEC 62541-4, also requests to create sessions. This number includes the *securityRejectedRequestsCount*.

Its representation in the *AddressSpace* is defined in Table 141.

Table 141 – ServerDiagnosticsSummaryDataType Definition

Attributes	Value
BrowseName	ServerDiagnosticsSummaryDataType

12.10 ServerStatusDataType

This structure contains elements that describe the status of the *Server*. Its composition is defined in Table 142.

Table 142 – ServerStatusDataType Structure

Name	Type	Description
ServerStatusDataType	structure	
startTime	UtcTime	Time (UTC) the server was started. This is constant for the server instance and is not reset when the server changes state. Each instance of a server should keep the time when the process started.
currentTime	UtcTime	The current time (UTC) as known by the server.
state	ServerState	The current state of the server. Its values are defined in 12.6.
buildInfo	BuildInfo	
secondsTillShutdown	UInt32	Approximate number of seconds until the server will be shut down. The value is only relevant once the state changes into SHUTDOWN.
shutdownReason	LocalizedText	An optional localized text indicating the reason for the shutdown. The value is only relevant once the state changes into SHUTDOWN.

Its representation in the *AddressSpace* is defined in Table 143.

Table 143 – ServerStatusDataType Definition

Attributes	Value
BrowseName	ServerStatusDataType

12.11 SessionDiagnosticsDataType

This structure contains diagnostic information about client sessions. Its elements are defined in Table 144. Most of the values represented in this structure provide information about the number of calls of a *Service*, the number of currently used *MonitoredItems*, etc. Those numbers need not provide the exact value; they need only provide the approximate number, so that the *Server* is not burdened with providing the exact numbers.

Table 144 – SessionDiagnosticsDataType Structure

Name	Type	Description
SessionDiagnosticsDataType	structure	
sessionId	NodeId	Server-assigned identifier of the session.
sessionName	String	The name of the session provided in the CreateSession request.
clientDesription	Application Description	The description provided by the client in the CreateSession request.
serverUri	String	The serverUri request in the CreateSession request.
endpointUrl	String	The endpointUrl passed by the client to the CreateSession request.
localeIds	LocaleId[]	Array of LocaleIds specified by the client in the open session call.
actualSessionTimeout	Duration	The requested session timeout specified by the client in the open session call.
maxResponseMessageSize	UInt32	The maximum size for the response message sent to the client.
clientConnectionTime	UtcTime	The server timestamp when the client opens the session.
clientLastContactTime	UtcTime	The server timestamp of the last request of the client in the context of the session.
currentSubscriptionsCount	UInt32	The number of subscriptions currently used by the session.
currentMonitoredItemsCount	UInt32	The number of *MonitoredItems* currently used by the session
currentPublishRequestsInQueue	UInt32	The number of publish requests currently in the queue for the session.
currentPublishTimerExpirations	UInt32	The number of publish timer expirations when there are data to be sent, but there are no publish requests for this session. The value shall be 0 if there are no data to be sent or publish requests queued.
totalRequestsCount	ServiceCounter DataType	Counter of all*Services*, identifying the number of received requests of any *Services* on the session.
unauthorizedRequestsCount	UInt32	Counter of all *Services*, identifying the number of *Service* requests that were rejected due to authorization failure
readCount	ServiceCounter DataType	Counter of the Read *Service*, identifying the number of received requests of this *Service* on the session.
historyReadCount	ServiceCounter DataType	Counter of the HistoryRead *Service*, identifying the number of received requests of this *Service* on the session.
writeCount	ServiceCounter DataType	Counter of the Write *Service*, identifying the number of received requests of this *Service* on the session.
historyUpdateCount	ServiceCounter DataType	Counter of the HistoryUpdate *Service*, identifying the number of received requests of this *Service* on the session.
callCount	ServiceCounter DataType	Counter of the Call *Service*, identifying the number of received requests of this *Service* on the session.
createMonitoredItemsCount	ServiceCounter DataType	Counter of the CreateMonitoredItems *Service*, identifying the number of received requests of this *Service* on the session.
modifyMonitoredItemsCount	ServiceCounter DataType	Counter of the ModifyMonitoredItems *Service*, identifying the number of received requests of this *Service* on the session.
setMonitoringModeCount	ServiceCounter DataType	Counter of the SetMonitoringMode *Service*, identifying the number of received requests of this *Service* on the session.
setTriggeringCount	ServiceCounter DataType	Counter of the SetTriggering *Service*, identifying the number of received requests of this *Service* on the session.
deleteMonitoredItemsCount	ServiceCounter DataType	Counter of the DeleteMonitoredItems *Service*, identifying the number of received requests of this *Service* on the session.
createSubscriptionCount	ServiceCounter DataType	Counter of the CreateSubscription *Service*, identifying the number of received requests of this *Service* on the session.

Name	Type	Description
modifySubscriptionCount	ServiceCounter DataType	Counter of the ModifySubscription *Service*, identifying the number of received requests of this *Service* on the session.
setPublishingModeCount	ServiceCounter DataType	Counter of the SetPublishingMode *Service*, identifying the number of received requests of this *Service* on the session.
publishCount	ServiceCounter DataType	Counter of the Publish *Service*, identifying the number of received requests of this *Service* on the session.
republishCount	ServiceCounter DataType	Counter of the Republish *Service*, identifying the number of received requests of this *Service* on the session.
transferSubscriptionsCount	ServiceCounter DataType	Counter of the TransferSubscriptions *Service*, identifying the number of received requests of this *Service* on the session.
deleteSubscriptionsCount	ServiceCounter DataType	Counter of the DeleteSubscriptions *Service*, identifying the number of received requests of this *Service* on the session.
addNodesCount	ServiceCounter DataType	Counter of the AddNodes *Service*, identifying the number of received requests of this *Service* on the session.
addReferencesCount	ServiceCounter DataType	Counter of the AddReferences *Service*, identifying the number of received requests of this *Service* on the session.
deleteNodesCount	ServiceCounter DataType	Counter of the DeleteNodes *Service*, identifying the number of received requests of this *Service* on the session.
deleteReferencesCount	ServiceCounter DataType	Counter of the DeleteReferences *Service*, identifying the number of received requests of this *Service* on the session.
browseCount	ServiceCounter DataType	Counter of the Browse *Service*, identifying the number of received requests of this *Service* on the session.
browseNextCount	ServiceCounter DataType	Counter of the BrowseNext *Service*, identifying the number of received requests of this *Service* on the session.
translateBrowsePathsToNodeIdsCount	ServiceCounter DataType	Counter of the TranslateBrowsePathsToNodeIds *Service*, identifying the number of received requests of this *Service* on the session.
queryFirstCount	ServiceCounter DataType	Counter of the QueryFirst *Service*, identifying the number of received requests of this *Service* on the session.
queryNextCount	ServiceCounter DataType	Counter of the QueryNext *Service*, identifying the number of received requests of this *Service* on the session.
registerNodesCount	ServiceCounter DataType	Counter of the RegisterNodes *Service*, identifying the number of received requests of this *Service* on the session.
unregisterNodesCount	ServiceCounter DataType	Counter of the UnregisterNodes*Service*, identifying the number of received requests of this *Service* on the session.

Its representation in the *AddressSpace* is defined in Table 145.

Table 145 – SessionDiagnosticsDataType Definition

Attributes	Value
BrowseName	SessionDiagnosticsDataType

12.12 SessionSecurityDiagnosticsDataType

This structure contains security-related diagnostic information about client sessions. Its elements are defined in Table 146. Because this information is security-related, it should not be made accessible to all users, but only to authorised users.

Table 146 – SessionSecurityDiagnosticsDataType Structure

Name	Type	Description
SessionSecurityDiagnosticsDataType	structure	
sessionId	NodeId	Server-assigned identifier of the session.
clientUserIdOfSession	String	Name of authenticated user when creating the session.
clientUserIdHistory	String[]	Array containing the name of the authenticated user currently active (either from creating the session or from calling the *ActivateSession Service*) and the history of those names. Each time the active user changes, an entry shall be made at the end of the array. The active user is always at the end of the array. Servers may restrict the size of this array, but shall support at least a size of 2. How the name of the authenticated user can be obtained from the system via the information received as part of the session establishment is defined in 6.4.3.
authenticationMechanism	String	Type of authentication (user name and password, X.509, Kerberos).
encoding	String	Which encoding is used on the wire, for example XML or UA Binary.
transportProtocol	String	Which transport protocol is used, for example TCP or HTTP.
securityMode	MessageSecurityMode	The message security mode used for the session.
securityPolicyUri	String	The name of the security policy used for the session.
clientCertificate	ByteString	The application instance certificate provided by the client in the CreateSession request.

Its representation in the *AddressSpace* is defined in Table 147.

Table 147 – SessionSecurityDiagnosticsDataType Definition

Attributes	Value
BrowseName	SessionSecurityDiagnosticsDataType

12.13 ServiceCounterDataType

This structure contains diagnostic information about subscriptions. Its elements are defined in Table 148.

Table 148 – ServiceCounterDataType Structure

Name	Type	Description
ServiceCounterDataType	structure	
totalCount	UInt32	The number of *Service* requests that have been received.
errorCount	UInt32	The total number of *Service* requests that were rejected.

Its representation in the *AddressSpace* is defined in Table 149.

Table 149 – ServiceCounterDataType Definition

Attributes	Value
BrowseName	ServiceCounterDataType

12.14 StatusResult

This structure combines a *StatusCode* and diagnostic information and can, for example, be used by Methods to return several *StatusCodes* and the corresponding diagnostic information that are not handled in the *Call Service* parameters. The elements of this *DataType* are defined in Table 150. Whether the diagnosticInfo is returned depends on the setting of the *Service* calls.

Table 150 – StatusResult Structure

Name	Type	Description
StatusResult	structure	
statusCode	StatusCode	The StatusCode.
diagnosticInfo	DiagnosticInfo	The diagnostic information for the statusCode.

Its representation in the *AddressSpace* is defined in Table 151.

Table 151 – StatusResult Definition

Attributes	Value
BrowseName	StatusResult

12.15 SubscriptionDiagnosticsDataType

This structure contains diagnostic information about subscriptions. Its elements are defined in Table 152.

Table 152 – SubscriptionDiagnosticsDataType Structure

Name	Type	Description
SubscriptionDiagnosticsDataType	structure	
sessionId	NodeId	Server-assigned identifier of the session the subscription belongs to.
subscriptionId	UInt32	Server-assigned identifier of the subscription.
priority	Byte	The priority the client assigned to the subscription.
publishingInterval	Duration	The publishing interval of the subscription in milliseconds
maxKeepAliveCount	UInt32	The maximum keep-alive count of the subscription.
maxLifetimeCount	UInt32	The maximum lifetime count of the subscription.
maxNotificationsPerPublish	UInt32	The maximum number of notifications per publish response.
publishingEnabled	Boolean	Whether publishing is enabled for the subscription.
modifyCount	UInt32	The number of ModifySubscription requests received for the subscription.
enableCount	UInt32	The number of times the subscription has been enabled.
disableCount	UInt32	The number of times the subscription has been disabled.
republishRequestCount	UInt32	The number of Republish *Service* requests that have been received and processed for the subscription.
republishMessageRequestCount	UInt32	The total number of messages that have been requested to be republished for the subscription
republishMessageCount	UInt32	The number of messages that have been successfully republished for the subscription.
transferRequestCount	UInt32	The total number of TransferSubscriptions *Service* requests that have been received for the subscription.
transferredToAltClientCount	UInt32	The number of times the subscription has been transferred to an alternate client.
transferredToSameClientCount	UInt32	The number of times the subscription has been transferred to an alternate session for the same client.
publishRequestCount	UInt32	The number of Publish *Service* requests that have been received and processed for the subscription.
dataChangeNotificationsCount	UInt32	The number of data change Notifications sent by the subscription.
eventNotificationsCount	UInt32	The number of Event Notifications sent by the subscription.
notificationsCount	UInt32	The total number of Notifications sent by the subscription.
latePublishRequestCount	UInt32	The number of times the subscription has entered the LATE State, i.e. the number of times the publish timer expires and there are unsent notifications.
currentKeepAliveCount	UInt32	The number of times the subscription has entered the KEEPALIVE State.
currentLifetimeCount	UInt32	The current lifetime count of the subscription.
unacknowledgedMessageCount	UInt32	The number of unacknowledged messages saved in the republish queue.
discardedMessageCount	UInt32	The number of messages that were discarded before they were acknowledged.
monitoredItemCount	UInt32	The total number of monitored items of the subscription, including the disabled monitored items.
disabledMonitoredItemCount	UInt32	The number of disabled monitored items of the subscription.
monitoringQueueOverflowCount	UInt32	The number of times a monitored item dropped notifications because of a queue overflow.
nextSequenceNumber	UInt32	Sequence number for the next notification message.
eventQueueOverflowCount	UInt32	The number of times a monitored item in the subscription has generated an Event of type EventQueueOverflowEventType.

Its representation in the *AddressSpace* is defined in Table 153.

Table 153 – SubscriptionDiagnosticsDataType Definition

Attributes	Value
BrowseName	SubscriptionDiagnosticsDataType

12.16 ModelChangeStructureDataType

This structure contains elements that describe changes of the model. Its composition is defined in Table 154.

Table 154 – ModelChangeStructureDataType Structure

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>ModelChangeStructure DataType</td><td>structure</td><td></td></tr>
<tr><td>affected</td><td>NodeId</td><td>NodeId of the Node that was changed. The client should assume that the affected Node has been created or deleted, had a Reference added or deleted, or the DataType has changed as described by the verb.</td></tr>
<tr><td>affectedType</td><td>NodeId</td><td>If the affected Node was an Object or Variable, affectedType contains the NodeId of the TypeDefinitionNode of the affected Node. Otherwise it is set to null.</td></tr>
<tr><td>verb</td><td>Byte</td><td>Describes the changes happening to the affected Node.
The verb is an 8-bit unsigned integer used as bit mask with the structure defined in the following table:
<table>
<tr><th>Field</th><th>Bit</th><th>Description</th></tr>
<tr><td>NodeAdded</td><td>0</td><td>Indicates the affected Node has been added.</td></tr>
<tr><td>NodeDeleted</td><td>1</td><td>Indicates the affected Node has been deleted.</td></tr>
<tr><td>ReferenceAdded</td><td>2</td><td>Indicates a Reference has been added. The affected Node may be either a SourceNode or TargetNode. Note that an added bidirectional Reference is reflected by two ChangeStructures.</td></tr>
<tr><td>ReferenceDeleted</td><td>3</td><td>Indicates a Reference has been deleted. The affected Node may be either a SourceNode or TargetNode. Note that a deleted bidirectional Reference is reflected by two ChangeStructures.</td></tr>
<tr><td>DataTypeChanged</td><td>4</td><td>This verb may be used only for affected Nodes that are Variables or VariableTypes. It indicates that the DataType Attribute has changed.</td></tr>
<tr><td>Reserved</td><td>5:7</td><td>Reserved for future use. Shall always be zero.</td></tr>
</table>
A verb may identify several changes on the affected Node at once. This feature should be used if event compression is used (see IEC 62541-3 for details).
Note that all verbs shall always be considered in the context where the ModelChangeStructureDataType is used. A NodeDeleted may indicate that a Node was removed from a view but still exists in other Views.</td></tr>
</table>

Its representation in the *AddressSpace* is defined in Table 155.

Table 155 – ModelChangeStructureDataType Definition

Attributes	Value
BrowseName	ModelChangeStructureDataType

12.17 SemanticChangeStructureDataType

This structure contains elements that describe a change of the model. Its composition is defined in Table 156.

Table 156 – SemanticChangeStructureDataType Structure

Name	Type	Description
SemanticChangeStructureDataType	structure	
affected	NodeId	*NodeId* of the *Node* that owns the *Property* that has changed.
affectedType	NodeId	If the *affected Node* was an *Object* or *Variable*, *affectedType* contains the *NodeId* of the *TypeDefinitionNode* of the *affected Node*. Otherwise it is set to null.

Its representation in the *AddressSpace* is defined in Table 157.

Table 157 – SemanticChangeStructureDataType Definition

Attributes	Value
BrowseName	SemanticChangeStructureDataType

12.18 BitFieldMaskDataType

This simple *DataType* is a subtype of UInt64 and represents a bit mask up to 32 bits where individual bits can be written without modifying the other bits.

The first 32 bits (least significant bits) of the *BitFieldMaskDataType* represent the bit mask and the second 32 bits represent the validity of the bits in the bit mask. When the *Server* returns the value to the client, the validity provides information of which bits in the bit mask have a meaning. When the client passes the value to the *Server*, the validity defines which bits should be written. Only those bits defined in validity are changed in the bit mask, all others stay the same. The *BitFieldMaskDataType* can be used as *DataType* in the *OptionSetType VariableType*

Its representation in the *AddressSpace* is defined in Table 158.

Table 158 – BitFieldMaskDataType Definition

Attributes	Value
BrowseName	BitFieldMaskDataType

12.19 NetworkGroupDataType

This structure contains information on different network paths for one *Server*. Its composition is defined in Table 159.

Table 159 – NetworkGroupDataType Structure

Name	Type	Description
NetworkGroupDataType	structure	
serverUri	String	URI of the Server represented by the network group.
networkPaths	EndpointUrlListDataType[]	Array of different network paths to the server, for example provided by different network cards in a Server node. Each network path can have several Endpoints representing different protocol options for the same path.

Its representation in the *AddressSpace* is defined in Table 160.

Table 160 – NetworkGroupDataType Definition

Attributes	Value
BrowseName	NetworkGroupDataType

12.20 EndpointUrlListDataType

This structure represents a list of URLs of an *Endpoint*. Its composition is defined in Table 161.

Table 161 – EndpointUrlListDataType Structure

Name	Type	Description
EndpointUrlListDataType	structure	
endpointUrlList	String[]	List of URLs of an Endpoint.

Its representation in the *AddressSpace* is defined in Table 162.

Table 162 – EndpointUrlListDataType Definition

Attributes	Value
BrowseName	EndpointUrlListDataType

Annex A
(informative)

Design decisions when modelling the server information

A.1 Overview

This annex describes the design decisions of modelling the information provided by each OPC UA *Server*, exposing its capabilities, diagnostic information, and other data needed to work with the *Server*, such as the *NamespaceArray*.

This annex gives an example of what should be considered when modelling data using the Address Space Model. General considerations for using the Address Space Model can be found in IEC 62541-3.

This annex is for information only, that is, each *Server* vendor can model its data in the appropriate way that fits its needs.

The following subclauses describe the design decisions made while modelling the *Server Object*. General *DataTypes*, *VariableTypes* and *ObjectTypes* such as the *EventTypes* described in this standard are not taken into account.

A.2 ServerType and Server Object

The first decision is to decide at what level types are needed. Typically, each *Server* will provide one *Server Object* with a well known *NodeId*. The *NodeIds* of the containing *Nodes* are also well-known because their symbolic name is specified in this standard and the *NodeId* is based on the symbolic name in IEC 62541-6. Nevertheless, aggregating *Servers* may want to expose the *Server Objects* of the OPC UA *Servers* they are aggregating in their *AddressSpace*. Therefore, it is very helpful to have a type definition for the *Server Object*. The *Server Object* is an *Object*, because it groups a set of *Variables* and *Objects* containing information about the *Server*. The *ServerType* is a complex *ObjectType*, because the basic structure of the *Server Object* should be well-defined. However, the *Server Object* can be extended by adding *Variables* and *Objects* in an appropriate structure of the *Server Object* or its containing *Objects*.

A.3 Typed complex Objects beneath the Server Object

Objects beneath the *Server Object* used to group information, such as *Server* capabilities or diagnostics, are also typed because an aggregating *Server* may want to provide only part of the *Server* information, such as diagnostics information, in its *AddressSpace*. Clients are able to program against these structures if they are typed, because they have its type definition.

A.4 Properties versus DataVariables

Since the general description in IEC 62541-3 about the semantic difference between *Properties* and *DataVariables* are not applicable for the information provided about the *Server* the rules described in IEC 62541-3 are used.

If simple data structures should be provided, *Properties* are used. Examples of *Properties* are the *NamespaceArray* of the *Server Object* and the *MinSupportedSampleRate* of the *ServerCapabilities Object*.

If complex data structures are used, *DataVariables* are used. Examples of *DataVariables* are the *ServerStatus* of the *Server Object* and the *ServerDiagnosticsSummary* of the *ServerDiagnostics Object*.

A.5 Complex Variables using complex DataTypes

DataVariables providing complex data structures expose their information as complex *DataTypes*, as well as components in the *AddressSpace*. This allows access to simple values as well as access to the whole information at once in a transactional context.

For example, the *ServerStatus Variable* of the *Server Object* is modelled as a complex *DataVariable* having the *ServerStatusDataType* providing all information about the *Server* status. But it also exposes the *CurrentTime* as a simple *DataVariable*, because a client may want to read only the current time of the *Server*, and is not interested in the build information, etc.

A.6 Complex Variables having an array

A special case of providing complex data structures is an array of complex data structures. The *SubscriptionDiagnosticsArrayType* is an example of how this is modelled. It is an array of a complex data structure, providing information of a subscription. Because a *Server* typically has several subscriptions, it is an array. Some clients may want to read the diagnostic information about all subscriptions at once; therefore it is modelled as an array in a *Variable*. On the other hand, a client may be interested in only a single entry of the complex structure, such as the *PublishRequestCount*. Therefore, each entry of the array is also exposed individually as a complex *DataVariable*, having each entry exposed as simple data.

Note that it is never necessary to expose the individual entries of an array to access them separately. The *Services* already allow accessing individual entries of an array of a *Variable*. However, if the entries should also be used for other purposes in the *AddressSpace,* such as having *References* or additional *Properties* or exposing their complex structure using *DataVariables,* it is useful to expose them individually.

A.7 Redundant information

Providing redundant information should generally be avoided. But to fulfil the needs of different clients, it may be helpful.

Using complex *DataVariables* automatically leads to providing redundant information, because the information is directly provided in the complex *DataType* of the *Value Attribute* of the complex *Variable*, and also exposed individually in the components of the complex *Variable*.

The diagnostics information about subscriptions is provided in two different locations. One location is the *SubscriptionDiagnosticsArray* of the *ServerDiagnostics Object*, providing the information for all subscriptions of the *Server*. The second location is the *SubscriptionDiagnosticsArray* of each individual *SessionDiagnosticsObject Object*, providing only the subscriptions of the session. This is useful because some clients may be interested in only the subscriptions grouped by sessions, whereas other clients may want to access the diagnostics information of all sessions at once.

The *SessionDiagnosticsArray* and the *SessionSecurityDiagnosticsArray* of the *SessionsDiagnosticsSummary Object* do not expose their individual entries, although they represent an array of complex data structures. But the information of the entries can also be accessed individually as components of the *SessionDiagnostics Objects* provided for each session by the *SessionsDiagnosticsSummary Object*. A client can either access the arrays (or parts of the arrays) directly or browse to the *SessionDiagnostics Objects* to get the

information of the individual entries. Thus, the information provided is redundant, but the *Variables* containing the arrays do not expose their individual entries.

A.8 Usage of the BaseDataVariableType

All *DataVariables* used to expose complex data structures of complex *DataVariables* have the *BaseDataVariableType* as type definition if they are not complex by themselves. The reason for this approach is that the complex *DataVariables* already define the semantic of the containing *DataVariables* and this semantic is not used in another context. It is not expected that they are subtyped, because they should reflect the data structure of the *DataType* of the complex *DataVariable*.

A.9 Subtyping

Subtyping is used for modelling information about the redundancy support of the *Server*. Because the provided information shall differ depending on the supported redundancy of the *Server*, subtypes of the *ServerRedundancyType* will be used for this purpose.

Subtyping is also used as an extensibility mechanism (see A.10).

A.10 Extensibility mechanism

The information of the *Server* will be extended by other parts of this series of standards, by companion specifications or by *Server* vendors. There are preferred ways to provide the additional information.

Do not subtype *DataTypes* to provide additional information about the *Server*. Clients might not be able to read those new defined *DataTypes* and are not able to get the information, including the basic information. If information is added by several sources, the *DataType* hierarchy may be difficult to maintain. Note that this rule applies to the information about the *Server*; in other scenarios this may be a useful way to add information.

Add *Objects* containing *Variables* or add *Variables* to the *Objects* defined in this part. If, for example, additional diagnostic information per subscription is needed, add a new *Variable* containing in array with an entry per subscription in the same places that the *SubscriptionDiagnosticsArray* is used.

Use subtypes of the *ServerVendorCapabilityType* to add information about the server-specific capabilities on the *ServerCapabilities Objects*. Because this extensibility point is already defined in this part, clients will look there for additional information.

Use a subtype of the *VendorServerInfoType* to add server-specific information. Because an *Object* of this type is already defined in this part, clients will look there for server-specific information.

Annex B
(normative)

StateMachines

B.1 General

This annex describes the basic infrastructure to model state machines. It defines *ObjectTypes*, *VariableTypes* and *ReferenceTypes* and explains how they should be used.

This annex is an integral part of this standard, that is, the types defined in this annex have to be used as defined. However, it is not required but strongly recommended that a *Server* uses these types to expose its state machines. The defined types may be subtyped to refine their behaviour.

When a *Server* exposes its state machine using the types defined in this annex, it might only provide a simplified view on its internal state machine, hiding for example substates or putting several internal states into one exposed state.

The scope of the state machines described in this annex is to provide an appropriate foundation for state machines needed for IEC 62541-9 and IEC 62541-10. It does not provide more complex functionality of a state machine like parallel states, forks and joins, history states, choices and junctions, etc. However, the base state machine defined in this annex can be extended to support such concepts.

The following clauses describe examples of state machines, define state machines in the context of this annex and define the representation of state machines in OPC UA. Finally, some examples of state machines, represented in OPC UA, are given.

B.2 Examples of finite state machines

B.2.1 Simple state machine

The following example provides an overview of the base features that the state machines defined in this annex will support. In the following, a more complex example is given, that also supports sub-state machines.

Figure B.1 gives an overview over a simple state machine. It contains the three states “State1”, “State2” and “State3”. There are transitions from “State1” to “State2”, “State2” to “State2”, etc. Some of the transitions provide additional information with regard to what causes (or triggers) the transition, for example the call of “Method1” for the transition from “State1” to “State2”. The effect (or action) of the transition can also be specified, for example the generation of an *Event* of the “EventType1” in the same transition. The notation used to identify the cause is simply listing it on the transition, the effect is prefixed with a “/”. More than one cause or effect are separated by a “,”. Not every transition has to have a cause or effect, for example the transition between “State2” and “State3”.

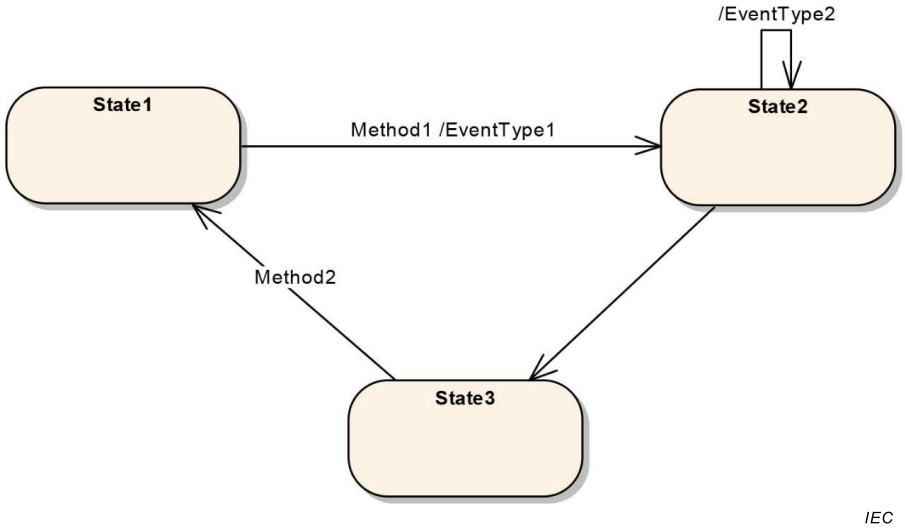


Figure B.1 – Example of a simple state machine

For simplicity, the state machines described in this annex will only support causes in form of specifying *Methods* that have to be called and effects in form of *EventTypes* of *Events* that are generated. However, the defined infrastructure allows extending this to support additional different causes and effects.

B.2.2 State machine containing substates

Figure B.2 shows an example of a state machine where “State6” is a sub-state-machine. This means, that when the overall state machine is in State6, this state can be distinguished to be in the sub-states “State7” or “State8”. Sub-state-machines can be nested, that is, “State7” could be another sub-state-machine.

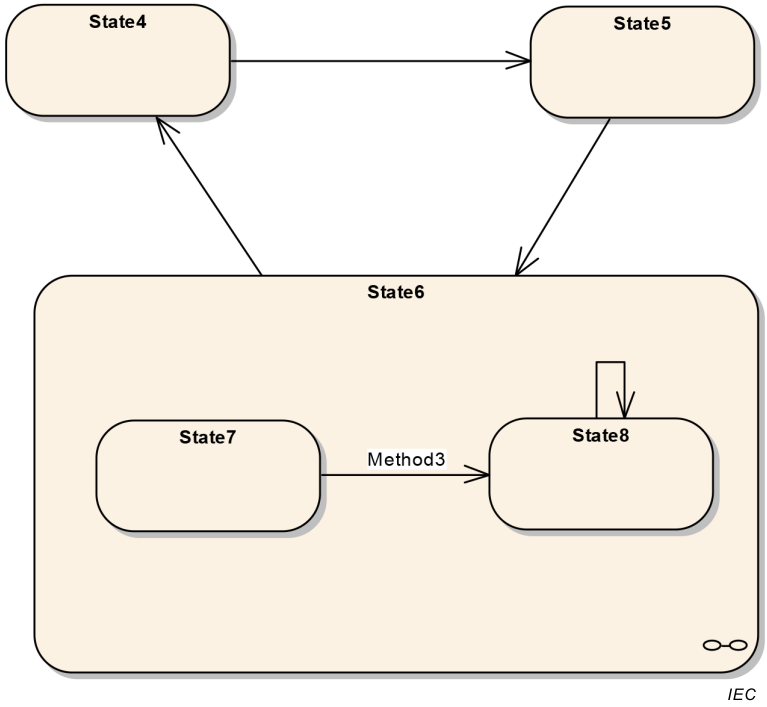


Figure B.2 – Example of a state machine having a sub-machine

B.3 Definition of state machine

The infrastructure of state machines defined in this annex only deals with the basics of state machines needed to support IEC 62541-9 and IEC 62541-10. The intention is to keep the basic simple but extensible.

For the state machines defined in this annex we assume that state machines are typed and instances of a type have their states and semantics specified by the type. For some types, this means that the states and transitions are fixed. For other types the states and transitions may be dynamic or unknown. A state machine where all the states are specified explicitly by the type is called a finite state machine.

Therefore we distinguish between *StateMachineType* and *StateMachine* and their subtypes like *FiniteStateMachineType*. The *StateMachineType* specifies a description of the state machine, that is, its states, transitions, etc., whereas the *StateMachine* is an instance of the *StateMachineType* and only contains the current state.

Each *StateMachine* contains information about the current state. If the *StateMachineType* has *SubStateMachines*, the *StateMachine* also contains information about the current state of the *SubStateMachines*. *StateMachines* which have their states completely defined by the type are instances of a *FiniteStateMachineType*.

Each *FiniteStateMachineType* has one or more *States*. For simplicity, we do not distinguish between different *States* like the start or the end states.

Each *State* can have one or more *SubStateMachines*.

Each *FiniteStateMachineType* may have one or more *Transitions*. A *Transition* is directed and points from one *State* to another *State*.

Each *Transition* can have one or more *Causes*. A *Cause* leads a *FiniteStateMachine* to change its current *State* from the source of the *Transition* to its target. In this annex we only specify *Method* calls to be *Causes* of *Transitions*. *Transitions* do not have to have a *Cause*. A *Transition* can always be caused by some server-internal logic that is not exposed in the *AddressSpace*.

Each *Transition* can have one or more *Effects*. An *Effect* occurs if the *Transition* is used to change the *State* of a *StateMachine*. In this annex we only specify the generation of *Events* to be *Effects* of a *Transition*. A *Transition* is not required to expose any *Effects* in the *AddressSpace.*

Although this annex only specifies simple concepts for state machines, the provided infrastructure is extensible. If needed, special *States* can be defined as well as additional *Causes* or *Effects*.

B.4 Representation of state machines in the AddressSpace

B.4.1 Overview

The types defined in this annex are illustrated in Figure B.3. The *MyFiniteStateMachineType* is a minimal example which illustrates how these *Types* can be used to describe a *StateMachine*. See IEC 62541-9 and IEC 62541-10 for additional examples of *StateMachines*.

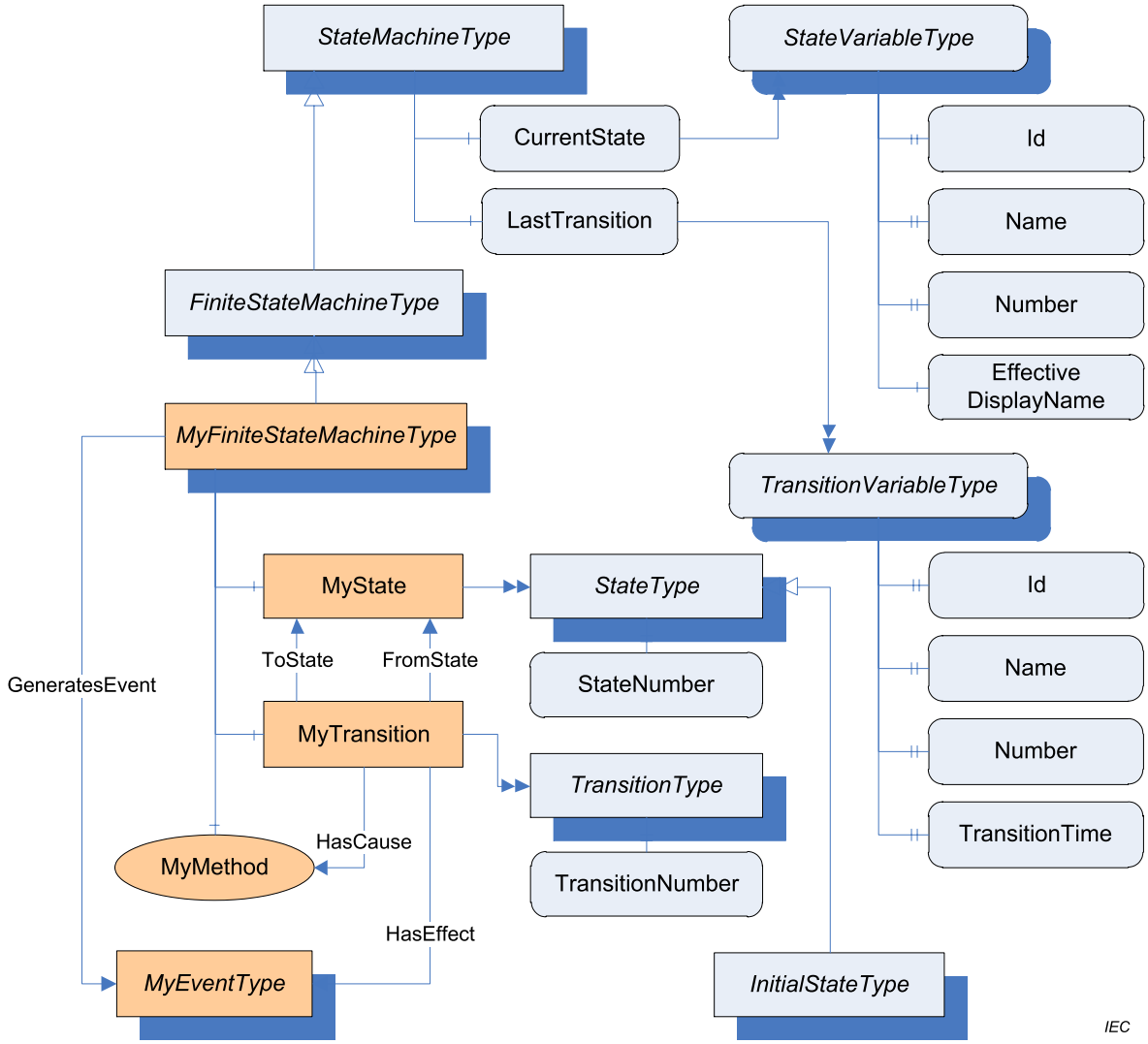


Figure B.3 – The StateMachine Information Model

B.4.2 StateMachineType

The *StateMachineType* is the base *ObjectType* for all *StateMachineTypes*. It defines a single *Variable* which represents the current state of the machine. An instance of this *ObjectType* shall generate an *Event* whenever a significant state change occurs. The *Server* decides which state changes are significant. *Servers* shall use the *GeneratesEvent ReferenceType* to indicate which *Event(s)* could be produced by the *StateMachine*.

Subtypes may add *Methods* which affect the state of the machine. The *Executable Attribute* is used to indicate whether the *Method* is valid given the current state of the machine. The generation of *AuditEvents* for *Methods* is defined in IEC 62541-4. A *StateMachine* may not be active. In this case, the *CurrentState* and *LastTransition Variables* shall have a status equal to *Bad_StateNotActive* (see Table B.17).

Subtypes may add components which are instances of *StateMachineTypes*. These components are considered to be sub-states of the *StateMachine*. *SubStateMachines* are only active when the parent machine is in an appropriate state.

Events produced by *SubStateMachines* may be suppressed by the parent machine. In some cases, the parent machine will produce a single *Event* that reflects changes in multiple *SubStateMachines*.

FiniteStateMachineType is subtype of *StateMachineType* that provides a mechanism to explicitly define the states and transitions. A *Server* should use this mechanism if it knows what the possible states are and the state machine is not trivial. The *FiniteStateMachineType* is defined in B.4.5.

The *StateMachineType* is formally defined in Table B.1.

Table B.1 – StateMachineType Definition

Attribute	Value				
BrowseName	StateMachineType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2. Note that a *Reference* to this subtype is not shown in the definition of the BaseObjectType.					
HasSubtype	ObjectType	FiniteStateMachineType	Defined in B.4.5		
HasComponent	Variable	CurrentState	LocalizedText	StateVariableType	Mandatory
HasComponent	Variable	LastTransition	LocalizedText	TransitionVariableType	Optional

CurrentState stores the current state of an instance of the *StateMachineType*. *CurrentState* provides a human readable name for the current state which may not be suitable for use in application control logic. Applications should use the *Id Property* of *CurrentState* if they need a unique identifier for the state.

LastTransition stores the last transition which occurred in an instance of the *StateMachineType*. *LastTransition* provides a human readable name for the last transition which may not be suitable for use in application control logic. Applications should use the *Id Property* of *LastTransition* if they need a unique identifier for the transition.

B.4.3 StateVariableType

The *StateVariableType* is the base *VariableType* for *Variables* that store the current state of a *StateMachine* as a human readable name.

The *StateVariableType* is formally defined in Table B.2.

Table B.2 – StateVariableType Definition

Attribute	Value				
BrowseName	StateVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseDataVariableType* defined in 7.4. Note that a *Reference* to this subtype is not shown in the definition of the *BaseDataVariableType*.					
HasSubtype	VariableType	FiniteStateVariableType	Defined in B.4.6		
HasProperty	Variable	Id	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	Name	QualifiedName	PropertyType	Optional
HasProperty	Variable	Number	UInt32	PropertyType	Optional
HasProperty	Variable	EffectiveDisplayName	LocalizedText	PropertyType	Optional

Id is a name which uniquely identifies the current state within the *StateMachineType*. A subtype may restrict the *DataType*.

Name is a *QualifiedName* which uniquely identifies the current state within the *StateMachineType*.

Number is an integer which uniquely identifies the current state within the *StateMachineType*.

EffectiveDisplayName contains a human readable name for the current state of the state machine after taking the state of any *SubStateMachines* in account. There is no rule specified for which state or sub-state should be used. It is up to the *Server* and will depend on the semantics of the *StateMachineType*.

StateMachines produce *Events* which may include the current state of a *StateMachine*. In that case *Servers* shall provide all the optional *Properties* of the *StateVariableType* in the *Event*, even if they are not provided on the instances in the *AddressSpace*.

B.4.4 TransitionVariableType

The *TransitionVariableType* is the base *VariableType* for *Variables* that store a *Transition* that occurred within a *StateMachine* as a human readable name.

The *SourceTimestamp* for the value specifies when the *Transition* occurred. This value may also be exposed with the *TransitionTime Property*.

The *TransitionVariableType* is formally defined in Table B.3.

Table B.3 – TransitionVariableType Definition

Attribute	**Value**				
BrowseName	TransitionVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseDataVariableType* defined in 7.4. Note that a *Reference* to this subtype is not shown in the definition of the *BaseDataVariableType*.					
HasSubtype	VariableType	FiniteTransitionVariableType	Defined in B.4.7		
HasProperty	Variable	Id	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	Name	QualifiedName	PropertyType	Optional
HasProperty	Variable	Number	UInt32	PropertyType	Optional
HasProperty	Variable	TransitionTime	UtcTime	PropertyType	Optional
HasProperty	Variable	EffectiveTransitionTime	UtcTime	PropertyType	Optional

Id is a name which uniquely identifies a *Transition* within the *StateMachineType*. A subtype may restrict the *DataType*.

Name is a *QualifiedName* which uniquely identifies a transition within the *StateMachineType*.

Number is an integer which uniquely identifies a transition within the *StateMachineType*.

TransitionTime specifies when the transition occurred.

EffectiveTransitionTime specifies the time when the current state or one of its substates was entered. If, for example, a StateA is active and – while active – switches several times between its substates SubA and SubB, then the *TransitionTime* stays at the point in time where StateA became active whereas the *EffectiveTransitionTime* changes with each change of a substate.

B.4.5 FiniteStateMachineType

The *FiniteStateMachineType* is the base *ObjectType* for *StateMachines* that explicitly define the possible *States* and *Transitions*. Once the *States* are defined subtypes shall not add new *States* (see B.4.18).

The *States* of the machine are represented with instances of the *StateType ObjectType.* Each *State* shall have a *BrowseName* which is unique within the *StateMachine* and shall have a

StateNumber which shall also be unique across all *States* defined in the *StateMachine*. Be aware that *States* in a *SubStateMachine* may have the same *StateNumber* or *BrowseName* as *States* in the parent machine. A concrete subtype of *FiniteStateMachineType* shall define at least one *State*.

A *StateMachine* may define one *State* which is an instance of the *InitialStateType*. This *State* is the *State* that the machine goes into when it is activated.

The *Transitions* that may occur are represented with instances of the *TransitionType*. Each *Transition* shall have a *BrowseName* which is unique within the *StateMachine* and may have a *TransitionNumber* which shall also be unique across all *Transitions* defined in the *StateMachine*.

The initial *State* for a *Transition* is a *StateType Object* which is the target of a *FromState Reference*. The final *State* for a *Transition* is a *StateType Object* which is the target of a *ToState Reference*. The *FromState* and *ToState References* shall always be specified.

A *Transition* may produce an *Event*. The *Event* is indicated by a *HasEffect Reference* to a subtype of *BaseEventType*. The *StateMachineType* shall have *GeneratesEvent References* to the targets of a *HasEffect Reference* for each of its *Transitions*.

A *FiniteStateMachineType* may define *Methods* that cause a transition to occur. These *Methods* are targets of *HasCause References* for each of the *Transitions* that may be triggered by the *Method*. The *Executable Attribute* for a *Method* is used to indicate whether the current *State* of the machine allows the *Method* to be called.

A *FiniteStateMachineType* may have sub-state-machines which are represented as instances of *StateMachineType ObjectTypes*. Each *State* shall have a *HasSubStateMachine Reference* to the *StateMachineType Object* which represents the child *States*. The *SubStateMachine* is not active if the parent *State* is not active. In this case the *CurrentState* and *LastTransition Variables* of the *SubStateMachine* shall have a status equal to *Bad_StateNotActive* (see Table B.17).

The *FiniteStateMachineType* is formally defined in Table B.4.

Table B.4 – FiniteStateMachineType Definition

Attribute	**Value**				
BrowseName	FiniteStateMachineType				
IsAbstract	True				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the StateMachineType defined in 6.2.					
HasComponent	Variable	CurrentState	LocalizedText	FiniteStateVariableType	Mandatory
HasComponent	Variable	LastTransition	LocalizedText	FiniteTransitionVariableType	Optional

B.4.6 FiniteStateVariableType

The *FiniteStateVariableType* is a subtype of *StateVariableType* and is used to store the current state of a Finite*StateMachine* as a human readable name.

The *FiniteStateVariableType* is formally defined in Table B.5.

Table B.5 – FiniteStateVariableType Definition

Attribute	Value				
BrowseName	FiniteStateVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *StateVariableType* defined in B.4.3					
HasProperty	Variable	Id	NodeId	PropertyType	Mandatory

Id is inherited from the *StateVariableType* and overridden to reflect the required *DataType*. This value shall be the *NodeId* of one of the *State Objects* of the *FiniteStateMachineType.*

The *Name Property* is inherited from *StateVariableType.* Its *Value* shall be the *BrowseName* of one of the *State Objects* of the *FiniteStateMachineType*.

The *Number Property* is inherited from *StateVariableType.* Its *Value* shall be the *StateNumber* for one of the *State Objects* of the *FiniteStateMachineType*.

B.4.7 FiniteTransitionVariableType

The *FiniteTransitionVariableType* is a subtype of *TransitionVariableType* and is used to store a *Transition* that occurred within a *FiniteStateMachine* as a human readable name.

The *FiniteTransitionVariableType* is formally defined in Table B.6.

Table B.6 – FiniteTransitionVariableType Definition

Attribute	Value				
BrowseName	FiniteTransitionVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the TransitionVariableType defined in B.4.4. Note that a *Reference* to this subtype is not shown in the definition of the *BaseDataVariableType*.					
HasProperty	Variable	Id	NodeId	PropertyType	Mandatory

Id is inherited from the *TransitionVariableType* and overridden to reflect the required *DataType*. This value shall be the *NodeId* of one of the *Transition Objects* of the *FiniteStateMachineType*.

The *Name Property* is inherited from the *TransitionVariableType.* Its *Value* shall be the *BrowseName* of one of the *Transition Objects* of the *FiniteStateMachineType*.

The *Number Property* is inherited from the *TransitionVariableType.* Its *Value* shall be the *TransitionNumber* for one of the *Transition Objects* of the *FiniteStateMachineType*.

B.4.8 StateType

States of a *FiniteStateMachine* are represented as *Objects* of the *StateType.*

The *StateType* is formally defined in Table B.7.

Table B.7 – StateType Definition

Attribute	Value				
BrowseName	StateType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2. Note that a *Reference* to this subtype is not shown in the definition of the BaseObjectType.					
HasProperty	Variable	StateNumber	UInt32	PropertyType	Mandatory
HasSubtype	ObjectType	InitialStateType	Defined in B.4.9		

B.4.9 InitialStateType

The *InitialStateType* is a subtype of the *StateType* and is formally defined in Table B.8. An *Object* of the *InitialStateType* represents the *State* that a *FiniteStateMachine* enters when it is activated. Each *FiniteStateMachine* can have at most one *State* of type *InitialStateType*, but a *FiniteStateMachine* does not have to have a *State* of this type.

A *SubStateMachine* goes into its initial state whenever the parent state is entered. However, a state machine may define a transition that goes directly to a state of the *SubStateMachine*. In this case the *SubStateMachine* goes into that *State* instead of the initial *State*. The two scenarios are illustrated in Figure B.4. The transition from State5 to State6 causes the *SubStateMachine* to go into the initial *State* (State7), however, the transition from State4 to State8 causes the parent machine to go to State6 and the *SubStateMachine* will go to State8.

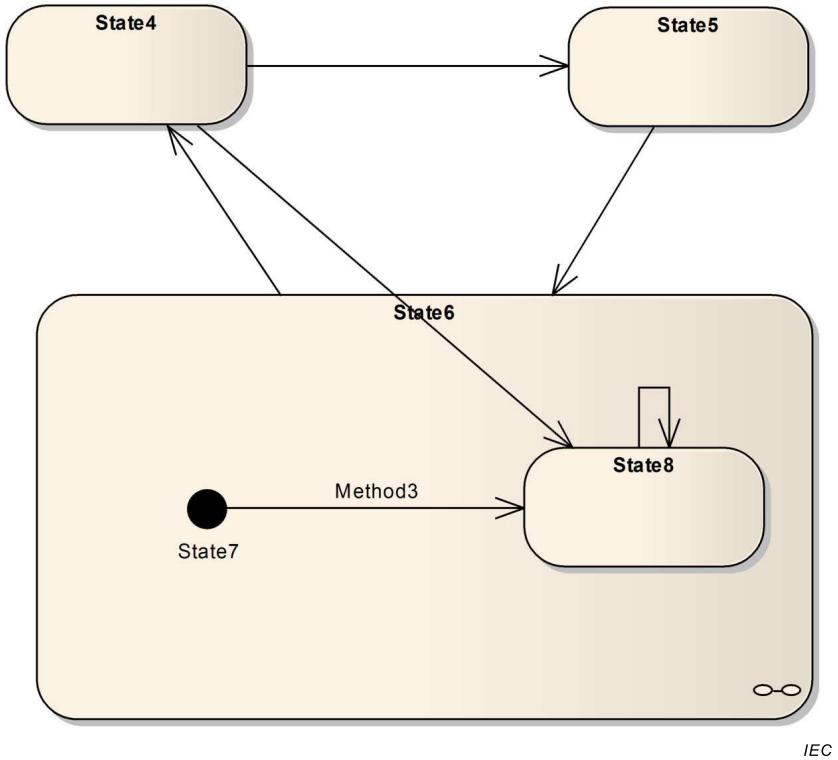


Figure B.4 – Example of an initial State in a sub-machine

If no initial state for a *SubStateMachine* exists and the *State* having the *SubStateMachine* is entered directly, then the *State* of the *SubStateMachine* is server-specific.

Table B.8 – InitialStateType Definition

Attribute	Value				
BrowseName	InitialStateType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *StateType* defined in B.4.8					

B.4.10 TransitionType

Transitions of a *FiniteStateMachine* are represented as *Objects* of the *ObjectType TransitionType* formally defined in Table B.9.

Each valid *Transition* shall have exactly one *FromState Reference* and exactly one *ToState Reference*, each pointing to an *Object* of the *ObjectType StateType*.

Each *Transition* can have one or more *HasCause References* pointing to the cause that triggers the *Transition*.

Each *Transition* can have one or more *HasEffect References* pointing to the effects that occur when the *Transition* was triggered.

Table B.9 – TransitionType Definition

Attribute	Value				
BrowseName	TransitionType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in 6.2. Note that a *Reference* to this subtype is not shown in the definition of the BaseObjectType.					
HasProperty	Variable	TransitionNumber	UInt32	PropertyType	Mandatory

B.4.11 FromState

The *FromState ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to the starting *State* the *Transition* connects.

The *SourceNode* of this *ReferenceType* shall be an *Object* of the *ObjectType TransitionType* or one of its subtypes. The *TargetNode* of this *ReferenceType* shall be an *Object* of the *ObjectType StateType* or one of its subtypes.

The representation of the *FromState ReferenceType* in the *AddressSpace* is specified in Table B.10.

Table B.10 – FromState ReferenceType

Attributes	Value		
BrowseName	FromState		
InverseName	ToTransition		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

B.4.12 ToState

The *ToState ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to the ending *State* the *Transition* connects.

The *SourceNode* of this *ReferenceType* shall be an *Object* of the *ObjectType TransitionType* or one of its subtypes. The *TargetNode* of this *ReferenceType* shall be an *Object* of the *ObjectType StateType* or one of its subtypes.

References of this *ReferenceType* may be only exposed uni-directional. Sometimes this is required, for example, if a *Transition* points to a *State* of a sub-machine.

The representation of the *ToState ReferenceType* in the *AddressSpace* is specified in Table B.11.

Table B.11 – ToState ReferenceType

Attributes	Value		
BrowseName	ToState		
InverseName	FromTransition		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

B.4.13 HasCause

The *HasCause ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to something that causes the *Transition*. In this annex we only define *Methods* as *Causes*. However, the *ReferenceType* is not restricted to point to *Methods*. The referenced Methods can, but do not have to point to a Method of the StateMachineType. For example, it is allowed to point to a server-wide restart Method leading the state machine to go into its initial state.

The *SourceNode* of this *ReferenceType* shall be an *Object* of the *ObjectType TransitionType* or one of its subtypes. The *TargetNode* can be of any *NodeClass*.

The representation of the *HasCause ReferenceType* in the *AddressSpace* is specified in Table B.12.

Table B.12 – HasCause ReferenceType

Attributes	Value		
BrowseName	HasCause		
InverseName	MayBeCausedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

B.4.14 HasEffect

The *HasEffect ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point form a *Transition* to something that will be effected when the *Transition* is triggered. In this annex we only define *EventTypes* as *Effects*. However, the *ReferenceType* is not restricted to point to *EventTypes*.

The *SourceNode* of this *ReferenceType* shall be an *Object* of the *ObjectType TransitionType* or one of its subtypes. The *TargetNode* can be of any *NodeClass*.

The representation of the *HasEffect ReferenceType* in the *AddressSpace* is specified in Table B.13.

Table B.13 – HasEffect ReferenceType

Attributes	**Value**		
BrowseName	HasEffect		
InverseName	MayBeEffectedBy		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

B.4.15 HasSubStateMachine

The *HasSubStateMachine ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to point from a *State* to an instance of a *StateMachineType* which represents the sub-states for the *State*.

The *SourceNode* of this *ReferenceType* shall be an *Object* of the *ObjectType StateType*. The *TargetNode* shall be an *Object* of the *ObjectType StateMachineType* or one of its subtypes. Each *Object* can be the *TargetNode* of at most one *HasSubStateMachine Reference*.

The *SourceNode* (the state) and the *TargetNode* (the *SubStateMachine*) shall belong to the same *StateMachine*, that is, both shall be referenced from the same *Object* of type *StateMachineType* using a *HasComponent Reference* or a subtype of *HasComponent*.

The representation of the *HasSubStateMachine ReferenceType* in the *AddressSpace* is specified in Table B.14.

Table B.14 – HasSubStateMachine ReferenceType

Attributes	**Value**		
BrowseName	HasSubStateMachine		
InverseName	SubStateMachineOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

B.4.16 TransitionEventType

The *TransitionEventType* is a subtype of the *BaseEventType*. It can be used to generate an *Event* identifying that a *Transition* of a *StateMachine* was triggered. It is formally defined in Table B.15.

Table B.15 – TransitionEventType

Attribute	Value				
BrowseName	TransitionEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the base *BaseEventType* defined in 6.4.2					
HasComponent	Variable	Transition	LocalizedText	TransitionVariableType	Mandatory
HasComponent	Variable	FromState	LocalizedText	StateVariableType	Mandatory
HasComponent	Variable	ToState	LocalizedText	StateVariableType	Mandatory

The *TransitionEventType* inherits the *Properties* of the *BaseEventType*.

The inherited *Property SourceNode* shall be filled with the *NodeId* of the *StateMachine* instance where the *Transition* occurs. If the *Transition* occurs in a *SubStateMachine*, then the *NodeId* of the *SubStateMachine* has to be used. If the Transition occurs between a *StateMachine* and a *SubStateMachine*, then the *NodeId* of the *StateMachine* has to be used, independent of the direction of the *Transition*.

Transition identifies the *Transition* that triggered the *Event*.

FromState identifies the *State* before the *Transition*.

ToState identifies the *State* after the *Transition*.

B.4.17 AuditUpdateStateEventType

The *AuditUpdateStateEventType* is a subtype of the *AuditUpdateMethodEventType*. It can be used to generate an *Event* identifying that a *Transition* of a *StateMachine* was triggered. It is formally defined in Table B.16.

Table B.16 – AuditUpdateStateEventType

Attribute	Value				
BrowseName	AuditUpdateStateEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateMethodEventType* defined in 6.4.27					
HasProperty	Variable	OldStateId	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	NewStateId	BaseDataType	PropertyType	Mandatory

The *AuditUpdateStateEventType* inherits the *Properties* of the *AuditUpdateMethodEventType*.

The inherited *Property SourceNode* shall be filled with the *NodeId* of the *StateMachine* instance where the *State* changed. If the *State* changed in a *SubStateMachine*, then the *NodeId* of the *SubStateMachine* has to be used.

The *SourceName* for *Events* of this type should be the effect that generated the event (e.g. the name of a Method). If the effect was generated by a *Method* call, the *SourceName* should be the name of the *Method* prefixed with “Method/”.

OldStateId reflects the *Id* of the state prior the change.

NewStateId reflects the new *Id* of the state after the change.

B.4.18 Special Restrictions on subtyping StateMachines

In general, all rules on subtyping apply for *StateMachine* types as well. Some additional rules apply for *StateMachine* types. If a StateMachine type is not abstract, subtypes of it shall not

change the behaviour of it. That means, that in this case a subtype shall not add *States* and it shall not add *Transitions* between its *States*. However, a subtype may add *SubStateMachines*, it may add *Transitions* from the *States* to the *States* of the *SubStateMachine*, and it may add *Causes* and *Effects* to a *Transition*. In addition, a subtype of a *StateMachine* type shall not remove *States* or *Transitions*.

B.4.19 Specific StatusCodes for StateMachines

In Table B.17 specific *StatusCodes* used for *StateMachines* are defined.

Table B.17 – Specific StatusCodes for StateMachines

Symbolic Id	Description
Bad_StateNotActive	The accessed state is not active.

B.5 Examples of StateMachines in the AddressSpace

B.5.1 StateMachineType using inheritance

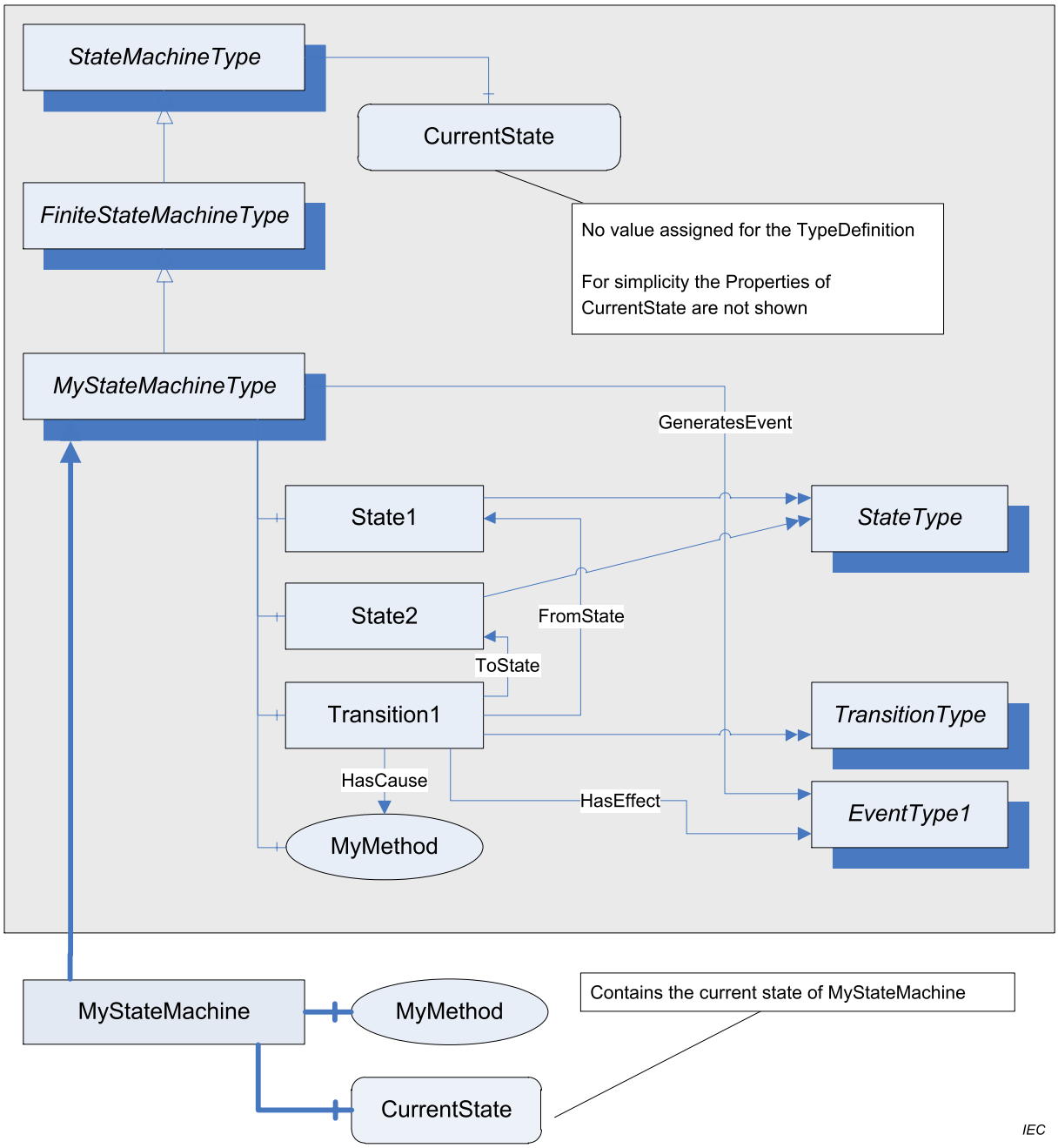


Figure B.5 – Example of a StateMachineType using inheritance

In Figure B.5 an example of a *StateMachine* is given using the Notation defined in IEC 62541-3. First, a new *StateMachineType* is defined, called "MyStateMachineType", inheriting from the base *FiniteStateMachineType*. It contains two *States*, "State1" and "State2" and a *Transition* "Transition1" between them. The *Transition* points to a *Method* "MyMethod" as the *Cause* of the *Transition* and an *EventType* "EventType1" as the *Effect* of the *Transition*.

Instances of "MyStateMachineType" can be created, for example "MyStateMachine". It has a *Variable* "CurrentState" representing the current *State*. The "MyStateMachine" *Object* only includes the *Nodes* which expose information specific to the instance.

B.5.2 StateMachineType with a sub-machine using inheritance

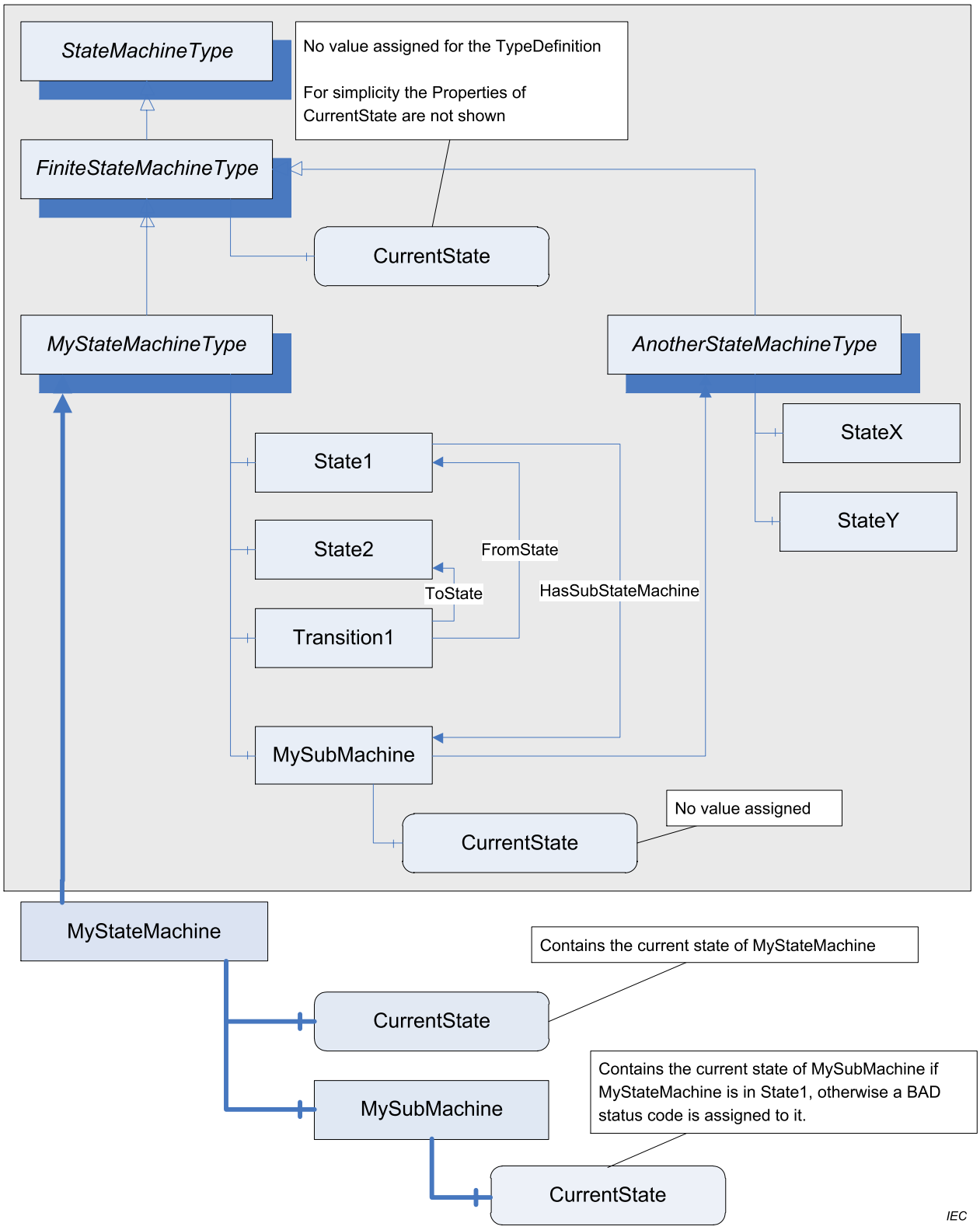


Figure B.6 – Example of a StateMachineType with a SubStateMachine using inheritance

Figure B.6 gives an example of a *StateMachineType* having a *SubStateMachine* for its "State1". For simplicity no effects and causes are shown, as well as type information for the *States* or *ModellingRules*.

The "MyStateMachineType" contains an *Object* "MySubMachine" of type "AnotherStateMachineType" representing a *SubStateMachine*. The "State1" references this *Object* with a *HasSubStateMachine Reference*, thus it is a *SubStateMachine* of "State1". Since "MySubMachine" is an *Object* of type "AnotherStateMachineType" it has a *Variable* representing the current *State*. Since it is used as an *InstanceDeclaration*, no value is assigned to this *Variable*.

An *Object* of "MyStateMachineType", called "MyStateMachine" has *Variables* for the current *State*, but also has an *Object* "MySubMachine" and a *Variable* representing the current state of the *SubStateMachine*. Since the *SubStateMachine* is only used when "MyStateMachine" is in "State1", a client would receive a *Bad_StateNotActive StatusCode* when reading the *SubStateMachine CurrentState Variable* if "MyStateMachine" is in a different *State*.

B.5.3 StateMachineType using containment

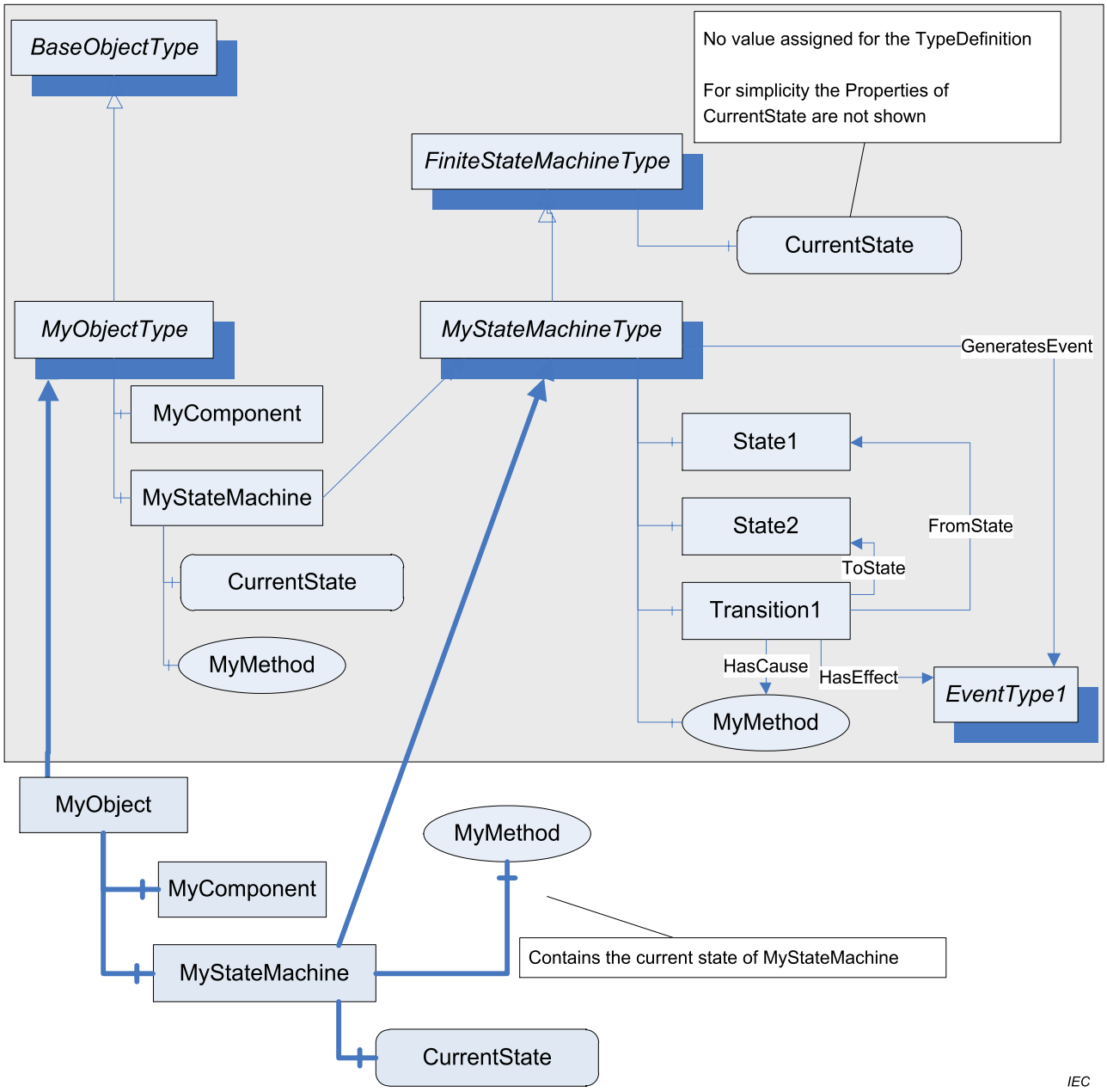


Figure B.7 – Example of a StateMachineType using containment

Figure B.7 gives an example of an *ObjectType* not only representing a *StateMachine* but also having some other functionality. The *ObjectType* "MyObjectType" has an *Object* "MyComponent" representing this other functionality. But it also contains a *StateMachine* "MyStateMachine" of the type "MyStateMachineType". *Objects* of "MyObjectType" also contain such an *Object* representing the StateMachine and a *Variable* containing the current state of the StateMachine, as shown in the Figure.

B.5.4 Example of a StateMachine having Transition to SubStateMachine

The *StateMachines* shown so far only had *Transitions* between *States* on the same level, that is, on the same *StateMachine*. Of cause, it is possible and often required to have *Transitions* between *States* of the *StateMachine* and *States* of its *SubStateMachine*.

Because a *SubStateMachine* can be defined by another *StateMachineType* and this type can be used in several places, it is not possible to add a bi-directional *Reference* from one of the shared *States* of the *SubStateMachine* to another *StateMachine*. In this case it is suitable to expose the *FromState* or *ToState References* uni-directional, that is, only pointing from the *Transition* to the *State* and not being able to browse to the other direction. If a *Transition* points from a *State* of a *SubStateMachine* to a *State* of another sub-machine, both, the *FromState* and the *ToState Reference*, are handled uni-directional.

A Client shall be able to handle the information of a *StateMachine* if the *ToState* and *FromState References* are only exposed as forward *References* and the inverse *References* are omitted.

Figure B.8 gives an example of a state machine having a transition from a sub-state to a state.

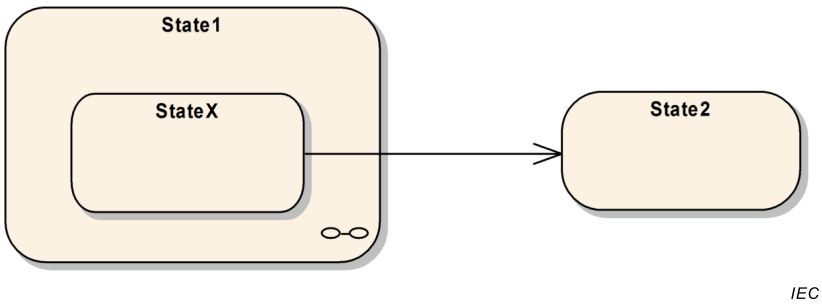


Figure B.8 – Example of a state machine with transitions from sub-states

In Figure B.9, the representation of this example as *StateMachineType* in the *AddressSpace* is given. The "Transition1", part of the definition of "MyStateMachineType", points to the "StateX" of the *StateMachineType* "AnotherStateMachineType". The *Reference* is only exposed as forward *Reference* and the inverse *Reference* is omitted. Thus, there is no *Reference* from the "StateX" of "AnotherStateMachineType" to any part of "MyStateMachineType" and "AnotherStateMachineType" can be used in other places as well.

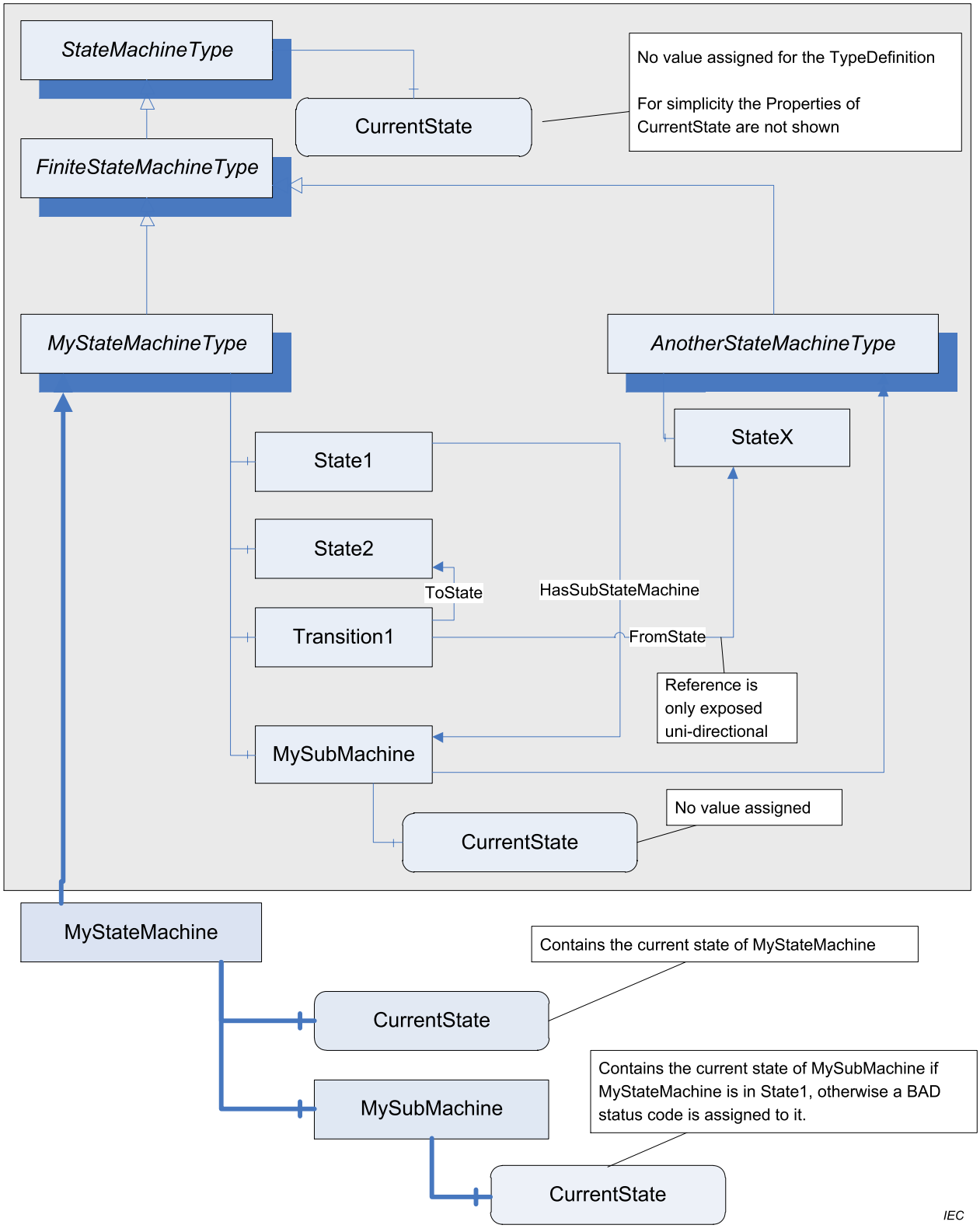


Figure B.9 – Example of a StateMachineType having Transition to SubStateMachine

Annex C
(normative)

File Transfer

C.1 Overview

This annex describes an information model for file transfer. Files could be modelled in OPC UA as simple Variables using ByteStrings. However, the overall message size in OPC UA is limited due to resources and security issues (denial of service attacks). Only accessing parts of the array can lead to concurrency issues if one client is reading the array while others are manipulating it. Therefore an *ObjectType* is defined representing a file with *Methods* to access the file.

The *Services* defined in the NodeManagement Service Set can be used to create or delete files in an *AddressSpace*. The life-cycle of a file stored on a hard disk and an instance of the *FileType* representing the file in an OPC UA *AddressSpace* can be independent. Deleting the OPC UA *Object* does not imply that the file is deleted from disk and a deletion from disk does not imply that the OPC UA *Object* is deleted.

This annex is an integral part of this standard, that is, the types defined in this annex have to be used as defined. However, it is not required but strongly recommended that a *Server* uses these types to expose its files. The defined types may be subtyped to refine their behaviour.

C.2 FileType

This *ObjectType* defines a type for files. It is formally defined in Table C.1.

Table C.1 – FileType

Attribute	**Value**				
BrowseName	FileType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the BaseObjectType defined in 6.2					
HasProperty	Variable	Size	UInt64	PropertyType	Mandatory
HasProperty	Variable	Writable	Boolean	PropertyType	Mandatory
HasProperty	Variable	UserWritable	Boolean	PropertyType	Mandatory
HasProperty	Variable	OpenCount	UInt16	PropertyType	Mandatory
HasComponent	Method	Open	Defined in C.3		Mandatory
HasComponent	Method	Close	Defined in C.4		Mandatory
HasComponent	Method	Read	Defined in C.5		Mandatory
HasComponent	Method	Write	Defined in C.6		Mandatory
HasComponent	Method	GetPosition	Defined in C.7		Mandatory
HasComponent	Method	SetPosition	Defined in C.8		Mandatory

Size defines the size of the file in Byte. When a file is opened for write and the fileHandle is still valid the size might not be accurate.

Writable indicates whether the file is writable. It does not take any user access rights into account, i.e. although the file is writable this may be restricted to a certain user / user group. The *Property* does not take into account whether the file is currently opened for writing by another client and thus currently locked and not writable by others.

UserWriteable indicates whether the file is writable taking user access rights into account. The Property does not take into account whether the file is currently opened for writing by another client and thus currently locked and not writable by others.

OpenCount indicates the number of currently valid file handles on the file.

Note that all *Methods* on a file require a fileHandle, which is returned in the *Open Method*.

C.3 Open

Open is used to open a file represented by an *Object* of FileType. When a client opens a file it gets a file handle that is valid while the session is open. Clients shall use the Close *Method* to release the handle when they do not need access to the file anymore. Clients can open the same file several times for read. A request to open for writing shall return Bad_NotWritable when the file is already opened. A request to open for reading shall return Bad_NotReadable when the file is already opened for writing.

Signature

```
Open(
    [in] Byte mode
    [out] UInt32 fileHandle
   );
```

Argument	Description
mode	Indicates whether the file should be opened only for read operations or for read and write operations and where the initial position is set. The *mode* is an 8-bit unsigned integer used as bit mask with the structure defined in the following table:
fileHandle	A handle for the file used in other method calls indicating not the file (this is done by the Object of the Method call) but the access request and thus the position in the file. The fileHandle is generated by the server and is unique for the Session. Clients cannot transfer the fileHandle to another Session but need to get a new fileHandle by calling the Open Method.

Field	Bit	Description
Read	0	The file is opened for reading. If this bit is not set the Read Method cannot be executed.
Write	1	The file is opened for writing. If this bit is not set the Write Method cannot be executed.
EraseExisiting	2	This bit can only be set if the file is opened for writing (Write bit is set). The existing content of the file is erased and an empty file is provided.
Append	3	When the Append bit is set the file is opened at end of the file, otherwise at begin of the file. The SetPosition Method can be used to change the position.
Reserved	4:7	Reserved for future use. Shall always be zero.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotReadable	See IEC 62541-4 for a general description. File might be locked and thus not readable.
Bad_NotWritable	See IEC 62541-4 for a general description.
Bad_InvalidState	See IEC 62541-4 for a general description. The file is locked and thus not writeable.
Bad_InvalidArguments	See IEC 62541-4 for a general description. Mode setting is invalid.
Bad_NotFound	See IEC 62541-4 for a general description.
Bad_UnexpectedError	See IEC 62541-4 for a general description.

Table C.2 specifies the *AddressSpace* representation for the *Open Method*.

Table C.2 – Open Method AddressSpace Definition

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

C.4 Close

Close is used to close a file represented by a FileType. When a client closes a file the handle becomes invalid.

Signature

```
Close(
    [in] UInt32 fileHandle
  );
```

Argument	Description
fileHandle	A handle indicating the access request and thus indirectly the position inside the file.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See IEC 62541-4 for a general description. Invalid file handle in call.

Table C.3 specifies the *AddressSpace* representation for the *Close Method*.

Table C.3 – Close Method AddressSpace Definition

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

C.5 Read

Read is used to read a part of the file starting from the current file position. The file position is advanced by the number of bytes read.

Signature

```
Read(
    [in] Uint32 fileHandle
    [in] Int32 length
    [out] ByteString data
  );
```

Argument	Description
fileHandle	A handle indicating the access request and thus indirectly the position inside the file.
Length	Defines the length in byte that should be returned in data, starting from the current position of the file handle. If the end of file is reached only all data till the end of the file are returned. If the specified length is longer than the maximum allowed message size of the communication, only those data fitting into the message size are returned. Only positive values are allowed.
Data	Contains the returned data of the file.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See IEC 62541-4 Invalid file handle in call or non-positive length.
Bad_UnexpectedError	See IEC 62541-4 for a general description.
Bad_InvalidState	See IEC 62541-4 for a general description. File was not opened for read access.

Table C.4 specifies the *AddressSpace* representation for the *Read Method*.

Table C.4 – Read Method AddressSpace Definition

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

C.6 Write

Write is used to write a part of the file starting from the current file position. The file position is advanced by the number of bytes written.

Signature

```
Write(
    [in] UInt32 fileHandle
    [in] ByteString data
   );
```

Argument	Description
fileHandle	A handle indicating the access request and thus indirectly the position inside the file.
data	Contains the data to be written at the position of the file. It is server-dependent whether the written data are persistently stored if the session is ended without calling the Close Method with the fileHandle.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See IEC 62541-4 for a general description. Invalid file handle in call.
Bad_NotWritable	See IEC 62541-4 for a general description. File might be locked and thus not writable.
Bad_InvalidState	See IEC 62541-4 for a general description. File was not opened for write access.

Table C.5 specifies the *AddressSpace* representation for the *Write Method*.

Table C.5 – Write Method AddressSpace Definition

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

C.7 GetPosition

GetPosition is used to provide the current position of the file handle.

Signature

```
GetPosition(
    [in] UInt32 fileHandle
    [out] UInt64 position
```

```
);
```

Argument	Description
fileHandle	A handle indicating the access request and thus indirectly the position inside the file.
Position	The position of the fileHandle in the file. If a Read or Write is called it starts at that position.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See IEC 62541-4 for a general description. Invalid file handle in call.

Table C.6 specifies the *AddressSpace* representation for the *GetPosition Method*.

Table C.6 – GetPosition Method AddressSpace Definition

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

C.8 SetPosition

SetPosition is used to set the current position of the file handle.

Signature

```
SetPosition(
    [in] UInt32 fileHandle
    [in] UInt64 position
  );
```

Argument	Description
fileHandle	A handle indicating the access request and thus indirectly the position inside the file.
Position	The position to be set for the fileHandle in the file. If a Read or Write is called it starts at that position. If the position is higher than the file size the position is set to the end of the file.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	See IEC 62541-4 for a general description. Invalid file handle in call.

Table C.7 specifies the *AddressSpace* representation for the *SetPosition Method*.

Table C.7 – SetPosition Method AddressSpace Definition

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

SOMMAIRE

AVANT-PROPOS 130
1 Domaine d'application 133
2 Références normatives 133
3 Termes, définitions et conventions 133
3.1 Termes et définitions 133
3.2 Abréviations et symboles 134
3.3 Conventions pour les descriptions de Nœuds 134
4 NodeIds et BrowseNames 135
4.1 NodeIds 135
4.2 BrowseNames 136
5 Attributs communs 136
5.1 Généralités 136
5.2 Objets 136
5.3 Variables 136
5.4 VariableTypes 137
6 ObjectTypes (Types d'Objet) normalisés 137
6.1 Généralités 137
6.2 BaseObjectType 137
6.3 ObjectTypes pour l'Objet Serveur (Server Object) 138
6.3.1 ServerType 138
6.3.2 ServerCapabilitiesType 140
6.3.3 ServerDiagnosticsType 142
6.3.4 SessionsDiagnosticsSummaryType 143
6.3.5 SessionDiagnosticsObjectType 143
6.3.6 VendorServerInfoType 144
6.3.7 ServerRedundancyType 144
6.3.8 TransparentRedundancyType 144
6.3.9 NonTransparentRedundancyType 145
6.3.10 NonTransparentNetworkRedundancyType 146
6.3.11 OperationLimitsType 147
6.3.12 AddressSpaceFileType 148
6.3.13 NamespaceMetadataType 148
6.3.14 NamespacesType 150
6.4 ObjectTypes utilisés comme EventTypes 150
6.4.1 Généralités 150
6.4.2 BaseEventType 150
6.4.3 AuditEventType 153
6.4.4 AuditSecurityEventType 154
6.4.5 AuditChannelEventType 154
6.4.6 AuditOpenSecureChannelEventType 155
6.4.7 AuditSessionEventType 155
6.4.8 AuditCreateSessionEventType 156
6.4.9 AuditUrlMismatchEventType 157
6.4.10 AuditActivateSessionEventType 157
6.4.11 AuditCancelEventType 158
6.4.12 AuditCertificateEventType 158

6.4.13 AuditCertificateDataMismatchEventType 159
6.4.14 AuditCertificateExpiredEventType 159
6.4.15 AuditCertificateInvalidEventType 159
6.4.16 AuditCertificateUntrustedEventType 160
6.4.17 AuditCertificateRevokedEventType 160
6.4.18 AuditCertificateMismatchEventType 160
6.4.19 AuditNodeManagementEventType 161
6.4.20 AuditAddNodesEventType 161
6.4.21 AuditDeleteNodesEventType 162
6.4.22 AuditAddReferencesEventType 162
6.4.23 AuditDeleteReferencesEventType 162
6.4.24 AuditUpdateEventType 163
6.4.25 AuditWriteUpdateEventType 163
6.4.26 AuditHistoryUpdateEventType 164
6.4.27 AuditUpdateMethodEventType 164
6.4.28 SystemEventType 165
6.4.29 DeviceFailureEventType 165
6.4.30 SystemStatusChangeEventType 166
6.4.31 BaseModelChangeEventType 166
6.4.32 GeneralModelChangeEventType 166
6.4.33 SemanticChangeEventType 167
6.4.34 EventQueueOverflowEventType 167
6.4.35 ProgressEventType 167
6.5 ModellingRuleType 168
6.6 FolderType 168
6.7 DataTypeEncodingType 169
6.8 DataTypeSystemType 169
6.9 AggregateFunctionType 169
7 VariableTypes normalisés 170
7.1 Généralités 170
7.2 BaseVariableType 170
7.3 PropertyType 170
7.4 BaseDataVariableType 171
7.5 ServerVendorCapabilityType 171
7.6 DataTypeDictionaryType 171
7.7 DataTypeDescriptionType 172
7.8 ServerStatusType 172
7.9 BuildInfoType 173
7.10 ServerDiagnosticsSummaryType 173
7.11 SamplingIntervalDiagnosticsArrayType 173
7.12 SamplingIntervalDiagnosticsType 174
7.13 SubscriptionDiagnosticsArrayType 174
7.14 SubscriptionDiagnosticsType 174
7.15 SessionDiagnosticsArrayType 175
7.16 SessionDiagnosticsVariableType 176
7.17 SessionSecurityDiagnosticsArrayType 177
7.18 SessionSecurityDiagnosticsType 178
7.19 OptionSetType 178
8 Objets normalisés et leurs Variables 179

8.1 Généralités 179
8.2 Objets utilisés pour organiser la structure de l'Espace d'Adresses 179
8.2.1 Vue d'ensemble 179
8.2.2 Racine (Root) 180
8.2.3 Views (Vues) 180
8.2.4 Objects (Objets) 181
8.2.5 Types 182
8.2.6 ObjectTypes 182
8.2.7 VariableTypes 183
8.2.8 ReferenceTypes 184
8.2.9 DataTypes 185
8.2.10 OPC Binary (OPC Binaire) 187
8.2.11 XML Schema (Schéma XML) 187
8.2.12 EventTypes 187
8.3 Objet Server et ses objets contenants 188
8.3.1 Généralités 188
8.3.2 Objet Server 189
8.4 Objets ModellingRule 190
8.4.1 ExposesItsArray 190
8.4.2 Mandatory (Obligatoire) 190
8.4.3 Optional (Facultatif) 191
8.4.4 OptionalPlaceholder 191
8.4.5 MandatoryPlaceholder 191
9 Méthodes normalisées 191
9.1 GetMonitoredItems 191
10 Vues normalisées 192
11 ReferenceTypes normalisés 192
11.1 Références 192
11.2 HierarchicalReferences 192
11.3 NonHierarchicalReferences 193
11.4 HasChild 193
11.5 Aggregates 193
11.6 Organizes 194
11.7 HasComponent 194
11.8 HasOrderedComponent 194
11.9 HasProperty 195
11.10 HasSubtype 195
11.11 HasModellingRule 195
11.12 HasTypeDefinition 195
11.13 HasEncoding 196
11.14 HasDescription 196
11.15 HasEventSource 196
11.16 HasNotifier 197
11.17 GeneratesEvent 197
11.18 AlwaysGeneratesEvent 197
12 DataTypes normalisés 197
12.1 Vue d'ensemble 197
12.2 DataTypes définis dans l'IEC 62541-3 197

12.3 DataTypes définis dans l'IEC 62541-4 202
12.4 BuildInfo 203
12.5 RedundancySupport 203
12.6 ServerState 204
12.7 RedundantServerDataType 204
12.8 SamplingIntervalDiagnosticsDataType 205
12.9 ServerDiagnosticsSummaryDataType 205
12.10 ServerStatusDataType 206
12.11 SessionDiagnosticsDataType 207
12.12 SessionSecurityDiagnosticsDataType 208
12.13 ServiceCounterDataType 209
12.14 StatusResult 209
12.15 SubscriptionDiagnosticsDataType 210
12.16 ModelChangeStructureDataType 211
12.17 SemanticChangeStructureDataType 212
12.18 BitFieldMaskDataType 213
12.19 NetworkGroupDataType 213
12.20 EndpointUrlListDataType 213
Annexe A (informative) Décisions de conception pour modéliser les informations du Serveur 214
A.1 Vue d'ensemble 214
A.2 ServerType et Objet Server 214
A.3 Objets complexes typés sous l'Objet Server 214
A.4 Properties par rapport aux DataVariables 214
A.5 Variables complexes utilisant des DataTypes complexes 215
A.6 Variables complexes ayant une matrice 215
A.7 Informations redondantes 215
A.8 Utilisation du BaseDataVariableType 216
A.9 Sous-typage 216
A.10 Mécanisme d'extensibilité 216
Annexe B (normative) Diagrammes d'états (StateMachines) 217
B.1 Généralités 217
B.2 Exemples de diagrammes d'états finis 217
B.2.1 Diagramme d'états simple 217
B.2.2 Diagramme d'états contenant des sous-états 218
B.3 Définition de diagramme d'états 219
B.4 Représentation des diagrammes d'états dans l'Espace d'Adresses 219
B.4.1 Vue d'ensemble 219
B.4.2 StateMachineType 220
B.4.3 StateVariableType 221
B.4.4 TransitionVariableType 222
B.4.5 FiniteStateMachineType 223
B.4.6 FiniteStateVariableType 224
B.4.7 FiniteTransitionVariableType 224
B.4.8 StateType 225
B.4.9 InitialStateType 225
B.4.10 TransitionType 226
B.4.11 FromState 226
B.4.12 ToState 227

B.4.13 HasCause 227
B.4.14 HasEffect 228
B.4.15 HasSubStateMachine 228
B.4.16 TransitionEventType 229
B.4.17 AuditUpdateStateEventType 229
B.4.18 Restrictions spéciales sur le sous-typage des StateMachines 230
B.4.19 StatusCodes spécifiques pour StateMachines 230
B.5 Exemples de StateMachines dans l'Espace d'Adresses 231
B.5.1 StateMachineType utilisant la relation d'héritage 231
B.5.2 StateMachineType avec un sous-diagramme utilisant la relation d'héritage 233
B.5.3 StateMachineType utilisant la hiérarchie d'appartenance (ou emboîtement) 235
B.5.4 Exemple de StateMachine ayant une Transition vers un SubStateMachine 236
Annexe C (normative) Transfert de fichiers 239
C.1 Vue d'ensemble 239
C.2 FileType 239
C.3 Open (Ouvrir) 240
C.4 Close (Fermer) 241
C.5 Read (Lecture) 241
C.6 Write (Ecriture) 242
C.7 GetPosition 243
C.8 SetPosition 243

Figure 1 – Structure normalisée de l'Espace d'Adresses 180
Figure 2 – Organisation des Views 181
Figure 3 – Organisation des Objects 182
Figure 4 – Organisation des ObjectTypes 183
Figure 5 – Organisation des VariableTypes 184
Figure 6 – Définitions de ReferenceType 185
Figure 7 – Organisation des DataTypes 186
Figure 8 – Organisation des EventTypes 188
Figure 9 – Extrait d'informations de diagnostic du Serveur 189
Figure B.1 – Exemple d'un diagramme d'états simple 218
Figure B.2 – Exemple de diagramme d'états ayant un sous-diagramme 218
Figure B.3 – Modèle d'informations StateMachine 220
Figure B.4 – Exemple d'État initial d'un sous-diagramme 225
Figure B.5 – Exemple d'un StateMachineType utilisant la relation d'héritage 231
Figure B.6 – Exemple d'un StateMachineType avec un SubStateMachine utilisant la relation d'héritage 234
Figure B.7 – Exemple d'un StateMachineType utilisant la hiérarchie d'appartenance 235
Figure B.8 – Exemple d'un diagramme d'états avec des transitions partant de sous-états 236
Figure B.9 – Exemple d'un StateMachineType ayant une Transition vers un SubStateMachine 238

Tableau 1 – Exemples de DataTypes 134
Tableau 2 – Tableau de Définition de Type 135
Tableau 3 – Attributs de nœud communs 136
Tableau 4 – Attributs d'objet communs 136
Tableau 5 – Attributs de variable communs 137
Tableau 6 – Attributs de VariableType communs 137
Tableau 7 – Définition de BaseObjectType 138
Tableau 8 – Définition de ServerType 138
Tableau 9 – Définition de ServerCapabilitiesType 140
Tableau 10 – Définition de ServerDiagnosticsType 142
Tableau 11 – Définition de SessionsDiagnosticsSummaryType 143
Tableau 12 – Définition de SessionDiagnosticsObjectType 143
Tableau 13 – Définition de VendorServerInfoType 144
Tableau 14 – Définition de ServerRedundancyType 144
Tableau 15 – Définition de TransparentRedundancyType 145
Tableau 16 – Définition de NonTransparentRedundancyType 145
Tableau 17 – Définition de NonTransparentNetworkRedundancyType 146
Tableau 18 – Définition de OperationLimitsType 147
Tableau 19 – Définition de AddressSpaceFileType 148
Tableau 20 – Définition de NamespaceMetadataType 149
Tableau 21 – Définition de NamespacesType 150
Tableau 22 – Définition de BaseEventType 151
Tableau 23 – Définition de AuditEventType 153
Tableau 24 – Définition de AuditSecurityEventType 154
Tableau 25 – Définition de AuditChannelEventType 154
Tableau 26 – Définition de AuditOpenSecureChannelEventType 155
Tableau 27 – Définition de AuditSessionEventType 156
Tableau 28 – Définition de AuditCreateSessionEventType 156
Tableau 29 – Définition de AuditUrlMismatchEventType 157
Tableau 30 – Définition de AuditActivateSessionEventType 157
Tableau 31 – Définition de AuditCancelEventType 158
Tableau 32 – Définition de AuditCertificateEventType 158
Tableau 33 – Définition de AuditCertificateDataMismatchEventType 159
Tableau 34 – Définition de AuditCertificateExpiredEventType 159
Tableau 35 – Définition de AuditCertificateInvalidEventType 160
Tableau 36 – Définition de AuditCertificateUntrustedEventType 160
Tableau 37 – Définition de AuditCertificateRevokedEventType 160
Tableau 38 – Définition de AuditCertificateMismatchEventType 161
Tableau 39 – Définition de AuditNodeManagementEventType 161
Tableau 40 – Définition de AuditAddNodesEventType 161
Tableau 41 – Définition de AuditDeleteNodesEventType 162
Tableau 42 – Définition de AuditAddReferencesEventType 162
Tableau 43 – Définition de AuditDeleteReferencesEventType 163

Tableau 44 – Définition de AuditUpdateEventType 163
Tableau 45 – Définition de AuditWriteUpdateEventType 163
Tableau 46 – Définition de AuditHistoryUpdateEventType 164
Tableau 47 – Définition de AuditUpdateMethodEventType 165
Tableau 48 – Définition de SystemEventType 165
Tableau 49 – Définition de DeviceFailureEventType 165
Tableau 50 – Définition de SystemStatusChangeEventType 166
Tableau 51 – Définition de BaseModelChangeEventType 166
Tableau 52 – Définition de GeneralModelChangeEventType 166
Tableau 53 – Définition de SemanticChangeEventType 167
Tableau 54 – Définition de EventQueueOverflowEventType 167
Tableau 55 – Définition de ProgressEventType 168
Tableau 56 – Définition de ModellingRuleType 168
Tableau 57 – Définition de FolderType 169
Tableau 58 – Définition de DataTypeEncodingType 169
Tableau 59 – Définition de DataTypeSystemType 169
Tableau 60 – Définition de AggregateFunctionType 169
Tableau 61 – Définition de BaseVariableType 170
Tableau 62 – Définition de PropertyType 170
Tableau 63 – Définition de BaseDataVariableType 171
Tableau 64 – Définition de ServerVendorCapabilityType 171
Tableau 65 – Définition de DataTypeDictionaryType 172
Tableau 66 – Définition de DataTypeDescriptionType 172
Tableau 67 – Définition de ServerStatusType 172
Tableau 68 – Définition de BuildInfoType 173
Tableau 69 – Définition de ServerDiagnosticsSummaryType 173
Tableau 70 – Définition de SamplingIntervalDiagnosticsArrayType 174
Tableau 71 – Définition de SamplingIntervalDiagnosticsType 174
Tableau 72– Définition de SubscriptionDiagnosticsArrayType 174
Tableau 73 – Définition de SubscriptionDiagnosticsType 175
Tableau 74 – Définition de SessionDiagnosticsArrayType 175
Tableau 75 – Définition de SessionDiagnosticsVariableType 176
Tableau 76 – Définition de SessionSecurityDiagnosticsArrayType 178
Tableau 77 – Définition de SessionSecurityDiagnosticsType 178
Tableau 78 – Définition de OptionSetType 179
Tableau 79 – Définition de Root 180
Tableau 80 – Définition de Views 181
Tableau 81 – Définition de Objects 182
Tableau 82 – Définition de Types 182
Tableau 83 – Définition de ObjectTypes 183
Tableau 84 – Définition de VariableTypes 184
Tableau 85 – Définition de ReferenceTypes 185
Tableau 86 – Définition de DataTypes 187

Tableau 87 – Définition de OPC Binary 187
Tableau 88 – Définition de XML Schema 187
Tableau 89 – Définition de EventTypes 188
Tableau 90 – Définition de Server 190
Tableau 91 – Définition de ExposesItsArray 190
Tableau 92 – Définition de Mandatory 190
Tableau 93 – Définition de Optional 191
Tableau 94 – Définition de OptionalPlaceholder 191
Tableau 95 – Définition de MandatoryPlaceholder 191
Tableau 96 – Définition de l'Espace d'Adresses pour la Méthode GetMonitoredItems 192
Tableau 97 – ReferenceType References 192
Tableau 98 – ReferenceType HierarchicalReferences 193
Tableau 99 – ReferenceType NonHierarchicalReferences 193
Tableau 100 – ReferenceType HasChild 193
Tableau 101 – ReferenceType Aggregates 194
Tableau 102 – ReferenceType Organizes 194
Tableau 103 – ReferenceType HasComponent 194
Tableau 104 – ReferenceType HasOrderedComponent 194
Tableau 105 – ReferenceType HasProperty 195
Tableau 106 – ReferenceType HasSubtype 195
Tableau 107 – ReferenceType HasModellingRule 195
Tableau 108 – ReferenceType HasTypeDefinition 196
Tableau 109 – ReferenceType HasEncoding 196
Tableau 110 – ReferenceType HasDescription 196
Tableau 111 – ReferenceType HasEventSource 196
Tableau 112 – ReferenceType HasNotifier 197
Tableau 113 – ReferenceType GeneratesEvent 197
Tableau 114 – ReferenceType AlwaysGeneratesEvent 197
Tableau 115 – Définitions de DataType dans l'IEC 62541-3 198
Tableau 116 – Définition de BaseDataType 199
Tableau 117 – Définition de Structure 199
Tableau 118 – Définition de Enumeration 200
Tableau 119 – Définition de ByteString 200
Tableau 120 – Définition de Number 200
Tableau 121 – Définition de Double 200
Tableau 122 – Définition d'Integer 201
Tableau 123 – Définition de DateTime 201
Tableau 124 – Définition de String 201
Tableau 125 – Définition d'UInteger 201
Tableau 126 – Définition d'Image 201
Tableau 127 – Définition d'UInt64 202
Tableau 128 – Définitions de DataType dans l'IEC 62541-4 202
Tableau 129 – Définition d'UserIdentityToken 203

Tableau 130 – Structure BuildInfo 203
Tableau 131 – Définition de BuildInfo 203
Tableau 132 – Valeurs de RedundancySupport 203
Tableau 133 – Définition de RedundancySupport 204
Tableau 134 – Valeurs de ServerState 204
Tableau 135 – Définition de ServerState 204
Tableau 136 – Structure de RedundantServerDataType 204
Tableau 137 – Définition de RedundantServerDataType 205
Tableau 138 – Structure de SamplingIntervalDiagnosticsDataType 205
Tableau 139 – Définition de SamplingIntervalDiagnosticsDataType 205
Tableau 140 – Structure de ServerDiagnosticsSummaryDataType 206
Tableau 141 – Définition de ServerDiagnosticsSummaryDataType 206
Tableau 142 – Structure de ServerStatusDataType 206
Tableau 143 – Définition de ServerStatusDataType 207
Tableau 144 – Structure de SessionDiagnosticsDataType 207
Tableau 145 – Définition de SessionDiagnosticsDataType 208
Tableau 146 – Structure de SessionSecurityDiagnosticsDataType 209
Tableau 147 – Définition de SessionSecurityDiagnosticsDataType 209
Tableau 148 – Structure de ServiceCounterDataType 209
Tableau 149 – Définition de ServiceCounterDataType 209
Tableau 150 – Structure de StatusResult 210
Tableau 151 – Définition de StatusResult 210
Tableau 152 – Structure de SubscriptionDiagnosticsDataType 211
Tableau 153 – Définition de SubscriptionDiagnosticsDataType 211
Tableau 154 – Structure de ModelChangeStructureDataType 212
Tableau 155 – Définition de ModelChangeStructureDataType 212
Tableau 156 – Structure de SemanticChangeStructureDataType 212
Tableau 157 – Définition de SemanticChangeStructureDataType 212
Tableau 158 – Définition de BitFieldMaskDataType 213
Tableau 159 – Structure de NetworkGroupDataType 213
Tableau 160 – Définition de NetworkGroupDataType 213
Tableau 161 – Structure de EndpointUrlListDataType 213
Tableau 162 – EndpointUrlListDataType Definition 213
Tableau B.1 – Définition de StateMachineType 221
Tableau B.2 – Définition de StateVariableType 221
Tableau B.3 – Définition de TransitionVariableType 222
Tableau B.4 – Définition de FiniteStateMachineType 223
Tableau B.5 – Définition de FiniteStateVariableType 224
Tableau B.6 – Définition de FiniteTransitionVariableType 224
Tableau B.7 – Définition de StateType 225
Tableau B.8 – Définition d'InitialStateType 226
Tableau B.9 – Définition de TransitionType 226
Tableau B.10 – ReferenceType FromState 227

Tableau B.11 – ReferenceType ToState 227
Tableau B.12 – ReferenceType HasCause 228
Tableau B.13 – ReferenceType HasEffect 228
Tableau B.14 – ReferenceType HasSubStateMachine 229
Tableau B.15 – TransitionEventType 229
Tableau B.16 – AuditUpdateStateEventType 229
Tableau B.17 – StatusCodes spécifiques pour StateMachines 230
Tableau C.1 – FileType 239
Tableau C.2 – Définition de l'Espace d'Adresses de la Méthode Open 241
Tableau C.3 – Définition de l'Espace d'Adresses de la Méthode Close 241
Tableau C.4 – Définition de l'Espace d'Adresses de la Méthode Read 242
Tableau C.5 – Définition de l'Espace d'Adresses de la Méthode Write 243
Tableau C.6 – Définition de l'Espace d'Adresses de la Méthode GetPosition 243
Tableau C.7 – Définition de l'Espace d'Adresses de la Méthode SetPosition 244

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

ARCHITECTURE UNIFIÉE OPC –

Partie 5: Modèle d'informations

AVANT-PROPOS

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

La Norme internationale IEC 62541-5 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

Cette deuxième édition annule et remplace la première édition parue en 2011. Cette édition constitue une révision technique.

Cette édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente:

a) ProgressEventType défini en 6.4.35 identifiant l'avancement d'une opération telle que l'appel d'un service (numéro d'édition 0057);

b) DataType défini appelé BitFieldMaskDataType en 12.18 représentant un champ de bits où des champs individuels peuvent être saisis sans redéfinir d'autres champs (numéro d'édition 0188);

c) Propriété SamplingRateCount supprimée dans le ServerDiagnosticSummaryDataType (12.9) car elle n'était pas nécessaire (numéro d'édition 0635);

d) Ajout de la propriété "EffectiveTransitionTime" au TransitionVariableType en B.4.4 (numéro d'édition 0728);

e) Introduction du VariableType OptionSetType en 7.19 représentant un masque de bits et d'un texte qui définit la sémantique des bits individuels (numéro d'édition 0983);

f) Ajout d'un nouveau EventType appelé SystemStatusChangeEventType en 6.4.30 qui peut être utilisé pour indiquer la perte de la connexion avec le système sous-jacent (numéro d'édition 1416);

g) Ajout de propriétés à ServerCapabilitiesType (6.3.2) décrivant la longueur maximale de matrice et de chaîne pour les variables, de même qu'ajout d'un objet pour les limites de fonctionnement (taille maximale des matrices lors de l'appel de services (par exemple, Lecture). Ajout du type OperationLimitsType (6.3.11) contenant ces informations (numéro d'édition 1451);

h) Ajout de SecureChannelId à AuditActivateSession-EventType (6.4.10) et adaptation du texte en divers emplacements (numéro d'édition 1492);

i) Ajout d'une Annexe C normative définissant le FileType et les Méthodes utilisés pour le transfert de fichiers (numéro d'édition 1502);

j) Ajout d'une Méthode GetMonitoredItems à ServerType (6.3.1) afin de recevoir des informations sur les éléments surveillés (numéro d'édition 1543);

k) Suppression du concept de *ModelParent* du document du fait qu'il n'est plus d'usage. Le *NodeId* du *ReferenceType* est conservé pour ne pas interrompre les applications existantes (numéros d'édition 1555 et 1556).

l) Ajout de métadonnées pour les espaces de noms dans ServerType (6.3.1) et création de types en vue de leur gestion (numéro d'édition 1702).

m) Ajout des représentations pour les ModellingRules OptionalPlaceholder en 8.4.4 et MandatoryPlaceholder en 8.4.5 (numéro d'édition 1831);

n) Ajout des nouveaux types NonTransparentNetworkRedundancyType (6.3.10), NetworkGroupDataType (12.19) et EndpointUrlListDataType (12.20) afin de gérer la redondance HotAndMiorrored. Ajout d'une description supplémentaire de la redondance et mise à jour de l'énumération RedundancySupport en 12.5 (numéro d'édition 2031);

Le texte de cette norme est issu des documents suivants:

CDV	Rapport de vote
65E/376/CDV	65E/404/RVC

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/IEC, Partie 2.

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

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. À 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.

ARCHITECTURE UNIFIÉE OPC –

Partie 5: Modèle d'informations

1 Domaine d'application

La présente partie de l'IEC 62541 définit le Modèle d'informations de l'Architecture unifiée OPC. Le Modèle d'informations décrit des *Nœuds (Nodes)* normalisés d'un *Espace d'Adresses (AddressSpace*) d'un *Serveur (Server)*. Ces *Nœuds* sont des types normalisés ainsi que des instances normalisées utilisés pour le diagnostic ou comme des points d'entrée à des *Nœuds* spécifiques au serveur. Ainsi, le Modèle d'informations définit l'*Espace d'Adresses* d'un *Serveur* OPC UA vide. Il n'est cependant pas prévu que tous les *Serveurs* fournissent la totalité de ces *Nœuds*.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC TR 62541-1, *OPC unified architecture – Part 1: Concepts* (disponible en anglais seulement)

IEC 62541-3, *OPC unified architecture – Part 3: Address Space Model* (disponible en anglais seulement)

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

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

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

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

IEC 62541-10, *Architecture unifiée OPC – Partie 10: Programmes*

IEC 62541-11, *OPC unified architecture – Part 11: Historical Access* (disponible en anglais seulement)

3 Termes, définitions et conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans l'IEC TR 62541-1 et l'IEC 62541-3 ainsi que le suivant s'appliquent.

3.1.1
ClientUserId (Identificateur d'Utilisateur Client)
chaîne qui identifie l'utilisateur du client demandant une action

Note 1 à l'article: Le *ClientUserId* est obtenu, directement ou indirectement, du *UserIdentityToken* (*Jeton d'Identité d'Utilisateur*) passé par le *Client* dans l'appel de *Service* de l'*ActivateSession (Activer Session)*. Voir 6.4.3 pour les détails.

3.2 Abréviations et symboles

UA Unified Architecture (Architecture unifiée)

XML Extensible Markup Language (Langage de balisage extensible)

3.3 Conventions pour les descriptions de Nœuds

Les définitions des *Nœuds* sont spécifiées à l'aide de tableaux (voir Tableau 2).

Les *Attributs* sont définis par la fourniture du nom de l'*Attribut* et d'une valeur, ou une description de la valeur.

Les *Références* sont définies par la fourniture du nom de *ReferenceType* (*Type de Référence*), du *BrowseName* (*Nom de Navigation*), du *TargetNode* (*Nœud Cible*) et de sa *NodeClass* (*Classe de Nœuds*).

- Si le *TargetNode* est une composante du *Nœud* défini dans le tableau, les *Attributs* du *Nœud* composé sont définis dans la même rangée du tableau.
- Le *DataType* (*Type de Données*) n'est spécifié que pour les *Variables*; “[<number>]” indique une matrice unidimensionnelle, alors que pour les matrices multidimensionnelles, l'expression est répétée pour chaque dimension (par exemple [2][3] pour une matrice à deux dimensions). Pour toutes les matrices, *ArrayDimensions* (*Dimensions de la Matrice*) est établi comme identifié par des valeurs <number>. Si aucun <number> n'est défini, la dimension correspondante est mise à 0, indiquant une taille inconnue. Si aucun nombre n'est fourni, *ArrayDimensions* peut être omis. L'absence de crochets identifie un *DataType* scalaire et le *ValueRank* (*Rang de valeur*) est mis à la valeur correspondante (voir IEC 62541-3). De plus, *ArrayDimensions* est mis à null ou est omis. Si elle peut être Any (Quelconque) ou ScalarOrOneDimension (Scalaire ou Unidimensionnelle), la valeur est placée dans “{<value>}” et, ainsi, “{Any}” ou “{ScalarOrOneDimension}” et le *ValueRank* sont mis à la valeur correspondante (voir IEC 62541-3) et *ArrayDimensions* est mis à null ou est omis. Le Tableau 1 présente des exemples.

Tableau 1 – Exemples de DataTypes

Notation	Type de Données	Rang de Valeur	Dimensions de la matrice	Description
Int32	Int32	−1	omis ou null	Un Int32 scalaire.
Int32[]	Int32	1	omis ou {0}	Matrice unidimensionnelle de Int32 de taille inconnue.
Int32[][]	Int32	2	omis ou {0,0}	Matrice bidimensionnelle de Int32 de tailles inconnues pour les deux dimensions.
Int32[3][]	Int32	2	{3,0}	Matrice bidimensionnelle de Int32 de taille 3 pour la première dimension et de taille inconnue pour la seconde dimension.
Int32[5][3]	Int32	2	{5,3}	Matrice bidimensionnelle de Int32 de taille 5 pour la première dimension et de taille 3 pour la seconde dimension.
Int32{Any}	Int32	−2	omis ou null	Int32 où l'on ne sait pas s'il s'agit d'un scalaire ou d'une matrice de dimension quelconque.
Int32{ScalarOrOneDimension}	Int32	−3	omis ou null	Int32 où il s'agit d'une matrice unidimensionnelle ou d'un scalaire.

- La TypeDefinition (Définition de Type) est spécifiée pour les *Objets* (*Objects*) et les *Variables*.
- La colonne TypeDefinition spécifie un nom symbolique pour un *NodeId* (*Identificateur de Nœud*), c'est-à-dire les points *Nœud* spécifiés avec une *Référence HasTypeDefinition* (*Dispose d'une Définition de Type*) au *Nœud* correspondant.
- La *ModellingRule* (*Règle de Modélisation*) de la composante référencée est fournie par la spécification du nom symbolique de la règle dans la colonne *ModellingRule*. Dans l'*Espace d'Adresses*, le *Nœud* doit utiliser une *Référence HasModellingRule* (*Dispose d'une Règle de Modélisation*) pour pointer vers l'*Objet ModellingRule* correspondant.

Si le *NodeId* d'un *DataType* est fourni, le nom symbolique du *Nœud* représentant le *DataType* doit être utilisé.

Les *Nœuds* de toutes les autres *NodeClasses* ne peuvent pas être définis dans le même tableau; par conséquent, seul le *ReferenceType* utilisé, sa *NodeClass* et son *BrowseName* sont spécifiés. Une référence à une autre partie de ce document pointe sur leur définition.

Le Tableau 2 illustre le Tableau correspondant. Si aucune composante n'est fournie, les colonnes DataType, TypeDefinition et ModellingRule peuvent être omises et seule la colonne Comment (Commentaires) est introduite pour pointer vers la définition du *Nœud*.

Tableau 2 – Tableau de Définition de Type

Attribut	**Valeur**				
Nom d'attribut	Valeur d'attribut. S'il s'agit d'un attribut facultatif qui n'est pas mis, "--" sera utilisé.				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Nom du *ReferenceType*	*NodeClass* du Target*Node*.	*BrowseName* du *Nœud* cible. Si la *Référence* est à instancier par le serveur, la valeur de BrowseName du Nœud cible est alors "--".	*Les Attributs* du *Nœud* référencé, applicables seulement aux *Variables* et *Objets*.		*ModellingRule* référencée de l'*Objet* référencé.
NOTE Notes référençant les notes de bas de tableau du contenu du tableau					

Les composantes de *Nœuds* peuvent être complexes, c'est-à-dire contenant elles-mêmes des composantes. Les TypeDefinition, NodeClass, DataType et ModellingRule peuvent être dérivés des définitions de type, et le nom symbolique peut être créé comme défini en 4.1. Par conséquent, celles qui contiennent des composantes ne sont pas spécifiées de manière explicite; elles sont implicitement spécifiées par les définitions de type.

4 NodeIds et BrowseNames

4.1 NodeIds

Les *NodeIds* de tous les *Nœuds* décrits dans la présente norme sont uniquement des noms symboliques. L'IEC 62541-6 définit les *NodeIds* réels.

Le nom symbolique de chaque *Nœud* défini dans la présente norme est son *BrowseName*, ou, lorsqu'il fait partie d'un autre *Nœud*, le *BrowseName* de l'autre *Nœud*, un ".", et son propre *BrowseName*. Dans ce cas, "fait partie de" signifie que l'ensemble a une *Référence HasProperty* (*Dispose d'une propriété*) ou *HasComponent* (*Dispose d'une composante*) à cette partie. Sachant que tous les *Nœuds* ne faisant pas partie d'un autre *Nœud* ont un nom unique dans la présente norme, le nom symbolique est unique. Par exemple, le *ServerType* (*Type de Serveur*) défini en 6.3.1 a le nom symbolique "ServerType". Une de ses *InstanceDeclarations* (*Déclarations d'Instance*) serait identifiée comme étant "ServerType.ServerCapabilities". Sachant que cet *Objet* est complexe, une autre *InstanceDeclaration* du *ServerType* est "ServerType.ServerCapabilities.MinSupportedSampleRate". L'*Objet Server* (*Serveur*) défini en 8.3.2 est basé sur le ServerType et a le nom symbolique "Server". Par conséquent, l'instance basée sur l'*InstanceDeclaration* décrite ci-dessus a le nom symbolique "Server.ServerCapabilities.MinSupportedSampleRate".

Le NamespaceIndex (Indice d'espace de Nom) pour tous les *NodeIds* définis dans la présente norme est 0. L'espace de nom pour ce NamespaceIndex est spécifié dans l'IEC 62541-3.

Noter que la présente norme définit non seulement des *Nœuds* concrets, mais exige aussi que certains *Nœuds* soient créés, par exemple une fois pour chaque Session exécutée sur le S*erveur*. Les *NodeIds* de ces *Nœuds* sont spécifiques au serveur, y compris le Namespace.

Toutefois, le NamespaceIndex de ces *Nœuds* ne peut pas être le NamespaceIndex 0, car ils ne sont pas définis par la Fondation OPC, mais créés par le *Serveur*.

4.2 BrowseNames

La partie texte des *BrowseNames* pour tous les *Nœuds* définis dans la présente norme est spécifiée dans les tableaux définissant les *Nœuds*. Le NamespaceIndex pour tous les *BrowseNames* définis dans la présente norme est 0.

5 Attributs communs

5.1 Généralités

Pour tous les *Nœuds* spécifiés dans la présente norme, les *Attributs* énumérés dans le Tableau 3 doivent être définis comme spécifié dans le Tableau 3.

Tableau 3 – Attributs de nœud communs

Attribut	Valeur
DisplayName	Le *DisplayName* (nom d'affichage) est un *LocalizedText* (texte localisé). Chaque serveur doit fournir le *DisplayName* identique au *BrowseName* du *Nœud* pour le LocaleId “en”. La question de décider si le serveur fournit ou non des noms traduits pour d'autres LocaleId est spécifique au fournisseur.
Description	Facultativement, une description spécifique au fournisseur est donnée.
NodeClass	Doit refléter la *NodeClass* du *Nœud.*
NodeId	Le *NodeId* est décrit par des *BrowseNames* comme défini en 4.1 et défini dans l'IEC 62541-6.
WriteMask	De manière facultative, l'*Attribut WriteMask* (Masque d'écriture) peut être fourni. Si l'*Attribut WriteMask* est fourni, il doit mettre tous les *Attributs* à "non inscriptible" qui ne sont pas dits spécifiques au fournisseur. Par exemple, l'*Attribut Description* peut être mis à inscriptible, car un Serveur peut fournir une description spécifique au serveur pour le *Nœud*. Le *NodeId* ne doit pas être inscriptible, car il est défini pour chaque *Nœud* dans la présente norme.
UserWriteMask	De manière facultative, l'*Attribut UserWriteMask* (*Masque d'écriture utilisateur*) peut être fourni. Les mêmes règles que dans le cas de l'*Attribut WriteMask* s'appliquent.

5.2 Objets

Pour tous les *Objets* spécifiés dans la présente norme, les *Attributs* énumérés dans le Tableau 4 doivent être définis comme spécifié dans le Tableau 4.

Tableau 4 – Attributs d'objet communs

Attribut	Valeur
EventNotifier	La question est de décider si le *Nœud* peut être utilisé pour s'abonner à des *Événements (Events*) ou s’il est spécifique au fournisseur.

5.3 Variables

Pour toutes les *Variables* spécifiées dans la présente norme, les *Attributs* énumérés dans le Tableau 5 doivent être définis comme spécifié dans le Tableau 5.

Tableau 5 – Attributs de variable communs

Attribut	Valeur
MinimumSamplingInterval	De manière facultative, un intervalle d'échantillonnage minimal spécifique au fournisseur est donné.
AccessLevel	Le niveau d'accès pour les *Variables* utilisées dans les définitions de types est spécifique au fournisseur; pour toutes les autres *Variables* définies dans la présente norme, le niveau d'accès doit permettre une lecture de la valeur courante; d'autres réglages sont spécifiques au fournisseur.
UserAccessLevel	La valeur pour l'*Attribut UserAccessLevel* est spécifique au fournisseur. Il est admis que toutes les *Variables* soient accessibles par au moins un utilisateur.
Valeur	Pour les *Variables* utilisées comme *InstanceDeclarations,* la valeur est spécifique au fournisseur; autrement, elle doit représenter la valeur décrite dans le texte.
ArrayDimensions	Si le *ValueRank* n'identifie pas une matrice d'une dimension spécifique (c'est-à-dire *ValueRank* <= 0), *ArrayDimensions* peut être mis à null ou bien l'*Attribut* est absent. Ce comportement est spécifique au fournisseur. Si le *ValueRank* spécifie une matrice d'une dimension spécifique (c'est-à-dire *ValueRank* > 0), l'*Attribut ArrayDimensions* doit alors être spécifié dans le tableau définissant la *Variable*.

5.4 VariableTypes

Pour tous les *VariableTypes* spécifiés dans la présente norme, les *Attributs* énumérés dans le Tableau 6 doivent être définis comme spécifié dans le Tableau 6.

Tableau 6 – Attributs de VariableType communs

Attributs	Valeur
Valeur	De manière facultative, une valeur par défaut spécifique au fournisseur peut être donnée.
ArrayDimensions	Si le *ValueRank* n'identifie pas une matrice d'une dimension spécifique (c'est-à-dire *ValueRank* <= 0), *ArrayDimensions* peut être mis à null ou bien l'*Attribut* est absent. Ce comportement est spécifique au fournisseur. Si le *ValueRank* spécifie une matrice d'une dimension spécifique (c'est-à-dire *ValueRank* > 0), l'*Attribut ArrayDimensions* doit alors être spécifié dans le tableau définissant le *VariableType*.

6 ObjectTypes (Types d'Objet) normalisés

6.1 Généralités

Typiquement, les composantes d'un *ObjectType* sont fixes et peuvent être étendues par sous-typage. Cependant, sachant que chaque *Objet* d'un *ObjectType* peut être étendu avec des composantes supplémentaires, la présente norme permet d'étendre les *ObjectTypes* normalisés définis dans ce document avec des composantes supplémentaires. Il est ainsi possible d'exprimer les informations supplémentaires dans la définition de type qui est déjà contenue dans chaque *Objet*. Certains *ObjectTypes* fournissent déjà des points d'entrée pour les extensions spécifiques au serveur. Cependant, il n'est pas permis de limiter les composantes des *ObjectTypes* normalisés définis dans la présente norme. Un exemple d'extension des *ObjectTypes* consiste à placer la *Propriété* normalisée *NodeVersion (Version de Nœud)* définie dans l'IEC 62541-3 dans le *BaseObjectType* (*Type d'Objet de Base*), énonçant que chaque *Objet* du *Serveur* fournira une *NodeVersion*.

6.2 BaseObjectType

Le *BaseObjectType* est utilisé comme définition de type chaque fois qu'il existe un *Objet* n'ayant plus de définitions disponibles pour les types concrets. Il convient que les *Serveurs* évitent d'utiliser ce*t ObjectType,* mais utilisent si c'est possible un type plus spécifique. Cet *ObjectType* est l'*ObjectType* de base et tous les autres *ObjectTypes* doivent hériter de lui, directement ou indirectement. Cependant, des *Serveurs* peuvent ne pas fournir toutes les *Références HasSubtype* de cet *ObjectType* à ses sous-types et, donc, il n'est pas exigé de fournir cette information.

Pour cet *ObjectType*, il n'est pas spécifié d'autres *Références* que les *Références HasSubtype*. Ceci est défini de façon formelle dans le Tableau 7.

Tableau 7 – Définition de BaseObjectType

Attribut	Valeur				
BrowseName	BaseObjectType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasSubtype	ObjectType	ServerType	Défini en 6.3.1		
HasSubtype	ObjectType	ServerCapabilitiesType	Défini en 6.3.2		
HasSubtype	ObjectType	ServerDiagnosticsType	Défini en 6.3.3		
HasSubtype	ObjectType	SessionsDiagnosticsSummaryType	Défini en 6.3.4		
HasSubtype	ObjectType	SessionDiagnosticsObjectType	Défini en 6.3.5		
HasSubtype	ObjectType	VendorServerInfoType	Défini en 6.3.6		
HasSubtype	ObjectType	ServerRedundancyType	Défini en 6.3.7		
HasSubtype	ObjectType	BaseEventType	Défini en 6.4.2		
HasSubtype	ObjectType	ModellingRuleType	Défini en 6.5		
HasSubtype	ObjectType	FolderType	Défini en 6.6		
HasSubtype	ObjectType	DataTypeEncodingType	Défini en 6.7		
HasSubtype	ObjectType	DataTypeSystemType	Défini en 6.8		

6.3 ObjectTypes pour l'Objet Serveur (Server Object)

6.3.1 ServerType

Cet *ObjectType* définit les fonctions prises en charge par le *Serveur* OPC UA. Ceci est défini de façon formelle dans le Tableau 8.

Tableau 8 – Définition de ServerType

Attribut	Valeur				
BrowseName	ServerType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					
HasProperty	Variable	ServerArray	String[]	PropertyType	Obligatoire
HasProperty	Variable	NamespaceArray	String[]	PropertyType	Obligatoire
HasComponent	Variable	ServerStatus[1]	ServerStatusDataType	ServerStatusType	Obligatoire
HasProperty	Variable	ServiceLevel	Byte	PropertyType	Obligatoire
HasProperty	Variable	Auditing	Boolean	PropertyType	Obligatoire
HasComponent	Object	ServerCapabilities[1]	-	ServerCapabilitiesType	Obligatoire
HasComponent	Object	ServerDiagnostics[1]	-	ServerDiagnosticsType	Obligatoire
HasComponent	Object	VendorServerInfo	-	VendorServerInfoType	Obligatoire
HasComponent	Object	ServerRedundancy[1]	-	ServerRedundancyType	Obligatoire
HasComponent	Object	Namespaces	-	NamespacesType	Facultatif
HasComponent	Method	GetMonitoredItems	Défini en 9		Facultatif
NOTE Les *Objets* et *Variables* conteneurs de ces *Objets* et *Variables* sont définis par leur *BrowseName* défini dans le *TypeDefinitionNode* correspondant. Le *NodeId* est défini par le nom symbolique composé décrit en 4.1.					

ServerArray (*Matrice du Serveur*) définit une matrice des URI (Uniform Resource Identifier, Identificateur uniforme de ressource) du *Serveur*. Cette *Variable* est également appelée *server table (tableau du serveur*). Chaque URI dans cette matrice représente un nom logique unique au niveau global pour un *Serveur* dans le domaine d'application du réseau dans lequel il est installé. Chaque instance de *Serveur* OPC UA a un seul URI qui est utilisé dans le *server table* d'autres *Serveurs* OPC UA. L'indice 0 est réservé à l'URI du *serveur* local. Les valeurs au-dessus de 0 sont utilisées pour identifier des *Serveurs* distants et sont spécifiques à un S*erveur*. L'IEC 62541-4 décrit un mécanisme de découverte qui peut être utilisé pour résoudre les URI en des URL (Uniform Resource Locator, localisateur uniforme de ressource). L'URI du *Serveur* est sensible à la casse.

L'URI de *ServerArray* avec l'indice 0 doit être identique à l'URI de la *NamespaceArray* avec l'indice 1, car ils représentent tous deux le *Serveur* local.

Les indices dans le *server table* sont appelés *server indexes* (*indices de serveur*) ou *server names* (*noms de serveur*). Les *Services* OPC UA les utilisent pour identifier des *TargetNodes* de *Références* qui résident dans des *Serveurs* distants. Les clients peuvent lire le tableau en entier ou ils peuvent lire individuellement les entrées dans le tableau. Le *Serveur* ne doit pas modifier ou supprimer des entrées de ce tableau lorsqu'un client a une session ouverte avec le *Serveur*, car les clients peuvent mettre le *server table* en cache. Un *Serveur* peut ajouter des entrées au *server table* même si des clients sont connectés au *Serveur*.

NamespaceArray (Matrice d'Espace de nom) définit une matrice des URI d'espace de nom. Cette *Variable* est également appelée *namespace table* (*tableau d'espace de nom*). Les indices dans le *namespace table* sont appelés *NamespaceIndexes*. Les *NamespaceIndexes* sont utilisés dans des *NodeIds* dans des *Services* OPC UA, en lieu et place des URI d'espace de nom plus longs. L'indice 0 est réservé à l'espace de nom OPC UA, et l'indice 1 est réservé au *Serveur* local. Les clients peuvent lire le *namespace table* en entier ou ils peuvent lire individuellement les entrées dans le *namespace table*. Le *Serveur* ne doit pas modifier ou supprimer des entrées du *namespace table* lorsqu'un client a une session ouverte avec le *Serveur*, car les clients peuvent mettre le *namespace table* en cache. Un *Serveur* peut ajouter des entrées au *namespace table* même si des clients sont connectés au *Serveur*. Il est recommandé aux *Serveurs* de ne pas changer les indices du *namespace table,* mais d'ajouter seulement des entrées, car le client peut mettre des *NodeIds* en cache en utilisant les indices. Néanmoins, il est parfois impossible aux *Serveurs* d'éviter de changer des indices dans le *namespace table*. Il convient pour les clients qui mettent en cache des *NamespaceIndexes* de *NodeIds* de toujours vérifier lors de l'ouverture d'une session que les *NamespaceIndexes* mis en cache n'ont pas changé.

ServerStatus (Statut du Serveur) contient des éléments qui décrivent le statut du *Serveur*. Voir 12.10 pour une description de ces éléments.

ServiceLevel (Niveau du Service) décrit la capacité du *Server* à fournir ses données au client. La plage de valeurs va de 0 à 255, dans laquelle 0 indique le plus mauvais et 255 le meilleur. Les valeurs concrètes sont spécifiques au fournisseur. L'intention est de fournir aux clients une indication de disponibilité parmi des *Serveurs* redondants.

Auditing (*Exécution d'un audit*) est un Booléen spécifiant si le *Serveur* génère actuellement des événements d'audit. Il est mis à TRUE (VRAI) si le *Serveur* génère des événements d'audit, autrement il est mis à FALSE (FAUX). Les *Profils* définis dans l'IEC 62541-7 spécifient le type d'événements d'audit qui sont générés par le *Serveur*.

ServerCapabilities (Fonctions du Serveur) définit les fonctions prises en charge par le *Serveur* OPC UA. Voir 6.3.2 pour sa description.

ServerDiagnostics (Diagnostic du Serveur) définit les informations de diagnostic relatives au *Serveur* OPC UA. Voir 6.3.3 pour sa description.

VendorServerInfo (Informations de Serveur du Fournisseur) représente le point d'entrée de navigation pour les informations de *Serveur* définies par le fournisseur. La présence de cet *Objet* est requise même s'il n'y a pas d'*Objets* définis par le fournisseur en dessous. Voir 6.3.6 pour sa description.

ServerRedundancy (Redondance du Serveur) décrit les fonctions de redondance fournies par le *Serveur*. Cet *Objet* est requis même si le *Serveur* ne fournit aucun support de redondance. Si le *Serveur* prend en charge la redondance, un sous-type de *ServerRedundancyType* (*Type de redondance du Serveur*) est alors utilisé pour décrire ses fonctions. Autrement, il fournit un *Objet* de type *ServerRedundancyType* avec la *Propriété* RedundancySupport mise à zéro. Voir 6.3.7 pour la description de *ServerRedundancyType*.

Namespaces fournit une liste d'*Objets NamespaceMetadataType* (*Type de Métadonnées d'Espace de Nom*) comprenant des informations supplémentaires concernant les espaces de noms utilisés dans le *Serveur*. Voir 6.3.14 pour la description de *NamespaceMetadataType.*

La *Méthode GetMonitoredItems* (*Obtenir des Éléments Surveillés*) sert à identifier les *MonitoredItems* d'un abonnement. Elle est définie en 9; l'utilisation prévue est définie dans l'IEC 62541-4.

6.3.2 ServerCapabilitiesType

Cet *ObjectType* définit les fonctions prises en charge par le *Serveur* OPC UA. Il est défini de façon formelle dans le Tableau 9.

Tableau 9 – Définition de ServerCapabilitiesType

Attribut	**Valeur**			
BrowseName	ServerCapabilitiesType			
IsAbstract	False			
Références	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2				
HasProperty	Variable	ServerProfileArray	String[] PropertyType	Obligatoire
HasProperty	Variable	LocaleIdArray	LocaleId[] PropertyType	Obligatoire
HasProperty	Variable	MinSupportedSampleRate	Duration PropertyType	Obligatoire
HasProperty	Variable	MaxBrowseContinuationPoints	UInt16 PropertyType	Obligatoire
HasProperty	Variable	MaxQueryContinuationPoints	UInt16 PropertyType	Obligatoire
HasProperty	Variable	MaxHistoryContinuationPoints	UInt16 PropertyType	Obligatoire
HasProperty	Variable	SoftwareCertificates	SignedSoftwareCertificate[] PropertyType	Obligatoire
HasProperty	Variable	MaxArrayLength	UInt32 PropertyType	Facultatif
HasProperty	Variable	MaxStringLength	UInt32 PropertyType	Facultatif
HasComponent	Object	OperationLimits	-- OperationLimitsType	Facultatif
HasComponent	Object	ModellingRules	FolderType	Obligatoire
HasComponent	Object	AggregateFunctions	FolderType	Obligatoire
HasComponent	Variable	*Variables* spécifiques au fournisseur d'un sous-type du ServerVendorCapabilityType défini en 7.5		--

ServerProfileArray (Matrice des Profils du Serveur) énumère les *Profils* pris en charge par le *Serveur*. Voir l'IEC 62541-7 pour les définitions des *Profils* de *Serveur*. Il convient de limiter cette liste aux *Profils* pris en charge par le *Serveur* dans sa configuration courante.

LocaleIdArray (Matrice des Identificateurs de Paramètres de Lieu) est une matrice de LocaleIds qui sont pris en charge par le *Serveur*. Le *Serveur* peut ne pas connaître tous les LocaleIds qu'il prend en charge, car il peut fournir un accès à des serveurs, systèmes ou appareils sous-jacents qui ne communiquent pas les LocaleIds qu'ils prennent en charge.

MinSupportedSampleRate (Fréquence Minimale d'Échantillonnage prise en charge) définit la fréquence minimale d'échantillonnage prise en charge par le *Serveur*, y compris 0.

MaxBrowseContinuationPoints (*Points de continuation maximum de Navigation*) est un entier spécifiant le nombre maximal de points de continuation parallèles du *Service* Browse que le *Serveur* peut prendre en charge par session. La valeur spécifie le maximum que le *Serveur* peut prendre en charge dans des conditions normales et, donc, il n'est pas garanti que le *Serveur* puisse toujours prendre en charge le maximum. Il convient que le client n'ouvre pas plus d'appels Browse avec points de continuation ouverts que cette *Variable* présente. La valeur 0 indique que le *Serveur* ne limite pas le nombre de points de continuation parallèles qu'il convient que le client utilise.

MaxQueryContinuationPoints (*Points de continuation maximum de Requête*) est un entier spécifiant le nombre maximal de points de continuation parallèles des *Services* QueryFirst que le *Serveur* peut prendre en charge par session. La valeur spécifie le maximum que le *Serveur* peut prendre en charge dans des conditions normales et, donc, il n'est pas garanti que le *Serveur* puisse toujours prendre en charge le maximum. Il convient que le client n'ouvre pas plus d'appels QueryFirst avec points de continuation ouverts que cette *Variable* présente. La valeur 0 indique que le *Serveur* ne limite pas le nombre de points de continuation parallèles qu'il convient que le client utilise.

MaxHistoryContinuationPoints (Points de continuation maximum d'Historique) est un entier spécifiant le nombre maximal de points de continuation parallèles des *Services* HistoryRead que le *Serveur* peut prendre en charge par session. La valeur spécifie le maximum que le *Serveur* peut prendre en charge dans des conditions normales et, donc, il n'est pas garanti que le *Serveur* prenne toujours en charge le maximum. Il convient que le client n'ouvre pas plus d'appels HistoryRead avec points de continuation ouverts que cette *Variable* présente. La valeur 0 indique que le *Serveur* ne limite pas le nombre de points de continuation parallèles qu'il convient que le client utilise.

SoftwareCertificates (*Certificats de Logiciels*) est une matrice de *SignedSoftwareCertificates* (*Certificats de Logiciels Signés)* contenant tous les *SoftwareCertificates* pris en charge par le *Serveur*. Un *SoftwareCertificate* identifie les fonctions du *Serveur.* Il contient la liste des *Profils* pris en charge par le *Serveur*. Les *Profils* sont décrits dans l'IEC 62541-7.

La *Propriété MaxArrayLength* indique la longueur maximale d'une matrice uni- ou multidimensionnelle prise en charge par les Variables du *Serveur*. Dans une matrice multidimensionnelle, elle indique la longueur totale. Par exemple, une matrice tridimensionnelle de 2x3x10 a une longueur de 60. Le *Serveur* peut limiter davantage la longueur pour les Variables individuelles sans en informer le client. Les *Serveurs* peuvent utiliser la *Propriété MaxArrayLength* définie dans l'IEC 62541-3 concernant les *DataVariables* individuelles afin de spécifier la taille propre aux valeurs individuelles. La *Propriété* individuelle peut avoir une valeur supérieure ou inférieure à *MaxArrayLength*.

La *Propriété MaxStringLength* indique la longueur maximale des Chaînes prise en charge par les *Variables* du *Serveur*. Le *Serveur* peut limiter davantage la longueur des Chaînes pour les *Variables* individuelles sans en informer le client. Les *Serveurs* peuvent utiliser la *Propriété MaxStringLength* définie dans l'IEC 62541-3 concernant les *DataVariables* individuelles afin de spécifier la longueur propre aux valeurs individuelles. La *Propriété* individuelle peut avoir des valeurs supérieures ou inférieures à *MaxStringLength*.

OperationLimits est un point d'entrée pour accéder aux informations sur les limites de fonctionnement du *Serveur*, par exemple, la longueur maximale d'une matrice dans un appel de *Service* de lecture.

ModellingRules est un point d'entrée pour parcourir toutes les *ModellingRules* prises en charge par le *Serveur*. Il convient que toutes les *ModellingRules* prises en charge par le *Serveur* puissent être parcourues à partir de cet *Objet*.

AggregateFunctions est un point d'entrée pour parcourir toutes les *AggregateFunctions* prises en charge par le *Serveur*. Il convient que toutes les *AggregateFunctions* prises en charge par le *Serveur* puissent être parcourues à partir de cet *Object*. Les AggregateFunctions sont des Objets d'*AggregateFunctionType*.

Les composantes restantes du *ServerCapabilitiesType* (*Type de fonctions du Serveur*) définissent les fonctions spécifiques au *Serveur*. Chacune est définie en utilisant une *Référence HasComponent* dont la cible est une instance d'un sous-type défini par le fournisseur de *ServerVendorCapabilityType* (*Type de Fonctions du serveur du Fournisseur*) abstrait (voir 7.5). Chaque sous-type de ce type définit une fonction *Serveur* spécifique. Les *NodeIds* pour ces *Variables* et leurs *VariableTypes* sont définis par le serveur.

6.3.3 ServerDiagnosticsType

Cet *ObjectType* définit les informations de diagnostic relatives au *Serveur* OPC UA. Cet *ObjectType* est défini de façon formelle dans le Tableau 10.

Tableau 10 – Définition de ServerDiagnosticsType

Attribut	Valeur			
BrowseName	ServerDiagnosticsType			
IsAbstract	False			
Références	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2				
HasComponent	Variable	ServerDiagnosticsSummary	ServerDiagnosticsSummaryDataType ServerDiagnosticsSummaryType	Obligatoire
HasComponent	Variable	SamplingIntervalDiagnosticsArray	SamplingIntervalDiagnosticsDataType[] SamplingIntervalDiagnosticsArrayType	Facultatif
HasComponent	Variable	SubscriptionDiagnosticsArray	SubscriptionDiagnosticsDataType[] SubscriptionDiagnosticsArrayType	Obligatoire
HasComponent	Object	SessionsDiagnosticsSummary	-- SessionsDiagnosticsSummaryType	Obligatoire
HasProperty	Variable	EnabledFlag	Boolean PropertyType	Obligatoire

ServerDiagnosticsSummary (*Récapitulatif des diagnostics du Serveur)* contient des informations récapitulatives de diagnostic pour le *Serveur*, comme défini en 12.9.

SamplingIntervalDiagnosticsArray (*Matrice des diagnostics d'intervalles d'échantillonnage*) est une matrice d'informations de diagnostic par fréquence d'échantillonnage comme défini en 12.8. Il y a une entrée pour chaque fréquence d'échantillonnage actuellement utilisée par le *Serveur*. Son *TypeDefinitionNode* est le *VariableType SamplingIntervalDiagnosticsArrayType (Type de Matrice des diagnostics d'intervalles d'échantillonnage)*, fournissant une *Variable* pour chaque entrée de la matrice, comme défini en 7.11.

Les diagnostics d'intervalles d'échantillonnage sont uniquement collectés par les *Serveurs* qui utilisent un jeu fixe d'intervalles d'échantillonnage. Dans ces cas, la longueur de la matrice et le jeu de *Variables* contenues sont déterminés par la configuration du *Serveur*. Le *NodeId* affecté à une variable donnée de diagnostic d'intervalles d'échantillonnage ne doit pas changer tant que la configuration du *Serveur* ne change pas. Un *Serveur* peut ne pas présenter la SamplingIntervalDiagnosticsArray s'il n'utilise pas des fréquences d'échantillonnage fixes.

SubscriptionDiagnosticsArray (Matrice des diagnostics d'abonnement) est une matrice d'informations de diagnostic d'abonnement pour chaque abonnement comme défini en 12.15. Il y a une entrée pour chaque canal de Notification effectivement établi dans le *Serveur*. Son *TypeDefinitionNode* est le *VariableType* SubscriptionDiagnosticsArrayType (Type de Matrice des diagnostics d'abonnement*)*, fournissant une *Variable* pour chaque entrée de la matrice, comme défini en 7.13. Ces *Variables* sont aussi utilisées comme *Variables* référencées par d'autres *Variables*.

SessionsDiagnosticsSummary (*Récapitulatif des diagnostics des sessions*) contient des informations de diagnostic par session, comme défini en 6.3.4.

EnabledFlag (*Fanion Activé*) identifie si les informations de diagnostic sont recueillies ou non par le *Serveur*. Il peut également être utilisé par un client pour activer ou désactiver la collecte d'informations de diagnostic du *Serveur*. Les réglages suivants de la valeur booléenne s'appliquent: TRUE indique que le *Serveur* recueille des informations de diagnostic; le fait de mettre la valeur à TRUE conduit à la réinitialisation et à l'activation de la collecte. FALSE indique qu'aucune information statistique n'est recueillie; le fait de mettre la valeur à FALSE désactive la collecte sans réinitialiser les valeurs statistiques.

Les *Nœuds* de diagnostic statique qui apparaissent toujours dans l'*Espace d'Adresses* retournent Bad_NotReadable lorsque l'*Attribut Value* d'un tel *Nœud* est lu ou fait l'objet d'un abonnement et le diagnostic est désactivé. Les *Nœuds* de diagnostic dynamique (tels que les *Nœuds Session*) n'apparaissent pas dans l'*Espace d'Adresses* lorsque les diagnostics sont désactivés.

6.3.4 SessionsDiagnosticsSummaryType

Cet *ObjectType* définit les informations de diagnostic relatives aux sessions du *Serveur* OPC UA. Ce*t ObjectType* est défini de façon formelle dans le Tableau 11.

Tableau 11 – Définition de SessionsDiagnosticsSummaryType

Attribut	**Valeur**			
BrowseName	SessionsDiagnosticsSummaryType			
IsAbstract	False			
Références	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**Modelling Rule**
Sous-type du BaseObjectType défini en 6.2				
HasComponent	Variable	SessionDiagnosticsArray	SessionDiagnosticsDataType[] SessionDiagnosticsArrayType	Obligatoire
HasComponent	Variable	SessionSecurityDiagnosticsArray	SessionSecurityDiagnosticsDataType[] SessionSecurityDiagnosticsArrayType	Obligatoire
HasComponent	Object	<ClientName>	-- SessionDiagnosticsObjectType	Paramètre fictif facultatif
NOTE Cette rangée représente l'absence de *Nœud* dans l'*Espace d'Adresses*. Il s'agit d'un paramètre fictif signalant que les instances d'*ObjectType* auront ces *Objets*.				

SessionDiagnosticsArray (*Matrice de Diagnostic de Session*) fournit une entrée pour chaque session dans le *Serveur* ayant des informations générales de diagnostic relatives à une session.

SessionSecurityDiagnosticsArray (*Matrice de Diagnostic Sécurité de Session*) fournit une matrice avec une entrée pour chaque session active dans le *Serveur* ayant des informations de diagnostic liées à la sécurité d'une session. Ces informations étant relatives à la sécurité, il convient de les rendre accessibles seulement aux utilisateurs autorisés et non à tous.

Pour chaque session du *Serveur*, cet *Objet* fournit également un *Objet* représentant la session, indiqué par <*ClientName*>. Le BrowseName peut être déduit du *sessionName* défini dans le *Service CreateSession* (IEC 62541-4) ou d'autres mécanismes spécifiques au serveur. Il est du *Type d'Objet* SessionDiagnosticsObjectType, comme défini en 6.3.5.

6.3.5 SessionDiagnosticsObjectType

Ce*t ObjectType* définit les informations de diagnostic relatives à une session du *Serveur* OPC UA. Ce*t ObjectType* est défini de façon formelle dans le Tableau 12.

Tableau 12 – Définition de SessionDiagnosticsObjectType

Attribut	**Valeur**			
BrowseName	SessionDiagnosticsObjectType			
IsAbstract	False			
Références	**NodeClass**	**BrowseName**	**DataType / TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2				
HasComponent	Variable	SessionDiagnostics	SessionDiagnosticsDataType SessionDiagnosticsVariableType	Obligatoire
HasComponent	Variable	SessionSecurityDiagnostics	SessionSecurityDiagnosticsDataType SessionSecurityDiagnosticsType	Obligatoire
HasComponent	Variable	SubscriptionDiagnosticsArray	SubscriptionDiagnosticsDataType[] SubscriptionDiagnosticsArrayType	Obligatoire

SessionDiagnostics contient les informations générales de diagnostic relatives à la session; la *Variable SessionSecurityDiagnostics* contient les informations de diagnostic relatives à la sécurité. Les informations de la seconde *Variable* étant relatives à la sécurité, il convient de les rendre accessibles seulement aux utilisateurs autorisés et non à tous.

SubscriptionDiagnosticsArray est une matrice d'informations de diagnostic d'Abonnement pour chaque abonnement ouvert, comme défini en 12.15. Son *TypeDefinitionNode* est le *VariableType* SubscriptionDiagnosticsArrayType, qui fournit une *Variable* pour chaque entrée de la matrice, comme défini en 7.13.

6.3.6 VendorServerInfoType

Cet *ObjectType* définit un paramètre fictif *Objet* pour les informations spécifiques au fournisseur relatives au *Serveur* OPC UA. Cet *ObjectType* définit un *ObjectType* vide qui ne comporte pas de composantes. Il doit être sous-typé par les fournisseurs pour définir les informations qui leur sont spécifiques. Cet *ObjectType* est défini de façon formelle dans le Tableau 13.

Tableau 13 – Définition de VendorServerInfoType

Attribut	**Valeur**				
BrowseName	VendorServerInfoType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					

6.3.7 ServerRedundancyType

Cet *ObjectType* définit les fonctions de redondance prises en charge par le *Serveur* OPC UA. Il est défini de façon formelle dans le Tableau 14.

Tableau 14 – Définition de ServerRedundancyType

Attribut	**Valeur**				
BrowseName	ServerRedundancyType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					
HasProperty	Variable	RedundancySupport	RedundancySupport	PropertyType	Obligatoire
HasSubtype	ObjectType	TransparentRedundancyType	Défini en 6.3.8		
HasSubtype	ObjectType	NonTransparentRedundancyType	Défini en 6.3.9		

RedundancySupport (*Prise en charge d'une redondanc*e) indique la redondance qui est prise en charge par le *Serveur*. Ses valeurs sont définies en 12.5. Elle doit être mise sur NONE_0 pour toutes les instances du *ServerRedundancyType* utilisant directement l'*ObjectType* (pas de sous-type).

6.3.8 TransparentRedundancyType

Cet ObjectType est un sous-type de *ServerRedundancyType* et sert à identifier les fonctions du *Serveur* OPC UA pour la redondance commandée par serveur avec une commutation transparente pour le client. Il est défini de façon formelle dans le Tableau 15.

Tableau 15 – Définition de TransparentRedundancyType

Attribut	Valeur				
BrowseName	TransparentRedundancyType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du ServerRedundancyType défini en 6.3.7, c'est-à-dire héritant des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	CurrentServerId	String	PropertyType	Obligatoire
HasProperty	Variable	RedundantServerArray	RedundantServerDataType[]	PropertyType	Obligatoire

RedundancySupport est hérité de *ServerRedundancyType*. Il doit être mis à TRANSPARENT_4 pour toutes les instances du *TransparentRedundancyType.*

Bien que dans un scénario de commutation transparente tous les *Serveurs* redondants sont vus sous le même URI par le client, il peut toutefois être exigé de suivre la source exacte de données sur le client. Par conséquent, *CurrentServerId (Identificateur du Serveur Courant*) contient un identificateur du *Serveur* actuellement utilisé dans l'ensemble redondant. Ce *Serveur* est valide uniquement dans une session; si un client ouvre plusieurs sessions, des *Serveurs* différents parmi les serveurs de l'ensemble redondant de *Serveurs* peuvent le desservir dans des sessions différentes. La valeur du *CurrentServerId* peut changer en raison d'une reprise sur défaillance ou d'un équilibrage de charge et, donc, un client ayant besoin de suivre la source de données doit s'abonner à cette *Variable*.

En tant qu'informations relatives au diagnostic, *RedundantServerArray* (*Matrice de Serveurs Redondants*) contient une matrice de *Serveurs* disponibles dans l'ensemble redondant; y compris leurs niveaux de service (voir 12.7). Cette matrice peut changer au cours d'une session.

6.3.9 NonTransparentRedundancyType

Cet *ObjectType* est un sous-type de *ServerRedundancyType* et sert à identifier les fonctions du *Serveur* OPC UA pour la redondance non transparente. Il est défini de façon formelle dans le Tableau 16.

Tableau 16 – Définition de NonTransparentRedundancyType

Attribut	Valeur				
BrowseName	NonTransparentRedundancyType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du ServerRedundancyType défini en 6.3.7, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	ServerUriArray	String[]	PropertyType	Obligatoire
HasSubtype	ObjectType	NonTransparentNetworkRedundancyType	Défini en 6.3.10		

ServerUriArray (Matrice de l'Uri des Serveurs) est une matrice avec l'URI de tous les *Serveurs* redondants du *Serveur* OPC UA. Voir l'IEC 62541-4 pour la définition de la redondance dans la présente norme. Dans un environnement de redondance non transparente, le client a la responsabilité de s'abonner aux *Serveurs* redondants. Par conséquent, il peut ouvrir une session sur un ou plusieurs *Serveurs* redondants de cette matrice. La *ServerUriArray* doit contenir le *Serveur local*.

RedundancySupport est héritée du *ServerRedundancyType*. Elle doit être mise à COLD_1, WARM_2, HOT_3 ou HOT_AND_MIRRORED_5 pour toutes les instances du *NonTransparentRedundancyType*. Elle définit la prise en charge d'une redondance fournie par le *Serveur*. Le *Client* n'est autorisé à accéder au *Serveur* redondant seulement de la manière décrite. Cependant, la commutation «à chaud» (Hot) implique la prise en charge de la commutation «tiède» (Warm) et celle-ci implique à son tour la prise en charge de la commutation «à froid» (Cold). La prise en charge de la redondance HotAndMirrored implique la prise en charge de la commutation «à chaud»; toutefois, pour les Serveurs qui prennent en

charge la redondance HotandMirrored, il est fortement recommandé aux *Clients* d'utiliser les mécanismes HotAndMirrored.

Si le *Serveur* prend en charge la commutation «à froid» uniquement, il convient de considérer la *Variable ServiceLevel* de l'*Obje*t *Server* pour identifier le *Serveur* primaire. Dans ce scénario, seul le *Serveur* primaire peut accéder au système sous-jacent, car ce dernier ne peut prendre en charge l'accès qu'à partir d'un seul *Serveur*. Dans ce cas, tous les autres *Serveurs* sont identifiés avec un *ServiceLevel* de zéro.

6.3.10 NonTransparentNetworkRedundancyType

Cet *ObjectType* est un sous-type du *NonTransparentRedundancyType* et sert à identifier les fonctions du *Serveur* OPC UA pour la redondance de réseaux non transparente. Il est défini de façon formelle dans le Tableau 16.

Tableau 17 – Définition de NonTransparentNetworkRedundancyType

Attribut	**Valeur**				
BrowseName	NonTransparentNetworkRedundancyType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du NonTransparentRedundancyType défini en 6.3.9, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question					
HasProperty	Variable	ServerNetworkGroups	NetworkGroupDataType[]	PropertyType	Obligatoire

Les clients qui effectuent des commutations entre des chemins de réseaux reliés au même *Serveur* ont le même comportement que la redondance HotAndMirrored. Les redondances du *Serveur* et du réseau peuvent être combinées. Dans l'approche combinée, il est important que le *Client* sache quels Uri de Serveur appartiennent au même *Serveur* représentant différents chemins de réseaux et quels Uri de Serveur représentent différents Serveurs. Par conséquent, un *Serveur* qui met en œuvre une redondance de réseaux non transparente doit utiliser le *NonTransparentNetworkRedundancyType* pour identifier sa prise en charge d'une redondance.

RedundancySupport est héritée du *ServerRedundancyType*. Elle doit être mise à COLD_1, WARM_2, HOT_3 ou HOT_AND_MIRRORED_5 pour toutes les instances du *NonTransparentNetworkRedundancyType*. Lorsqu'aucune redondance du serveur n'est prise en charge (la *ServerUriArray* ne contient qu'une entrée), la *RedundancySupport* doit être mise à HOT_AND_MIRRORED_5.

ServerNetworkGroups (*Groupes de Réseaux de Serveurs*) contient une matrice de *NetworkGroupDataType (Type de Données de Groupes de Réseaux)*. Les URI des *Serveurs* dans cette matrice (dans le *ServerUri* de la structure) doivent être exactement identiques à ceux fournis dans la *ServerUriArray*. L'ordre peut toutefois être différent. Ainsi, la matrice représente une liste des *Serveurs* redondants HotAndMirrored. Si un serveur prend uniquement en charge la redondance de réseaux, il ne comporte qu'une seule entrée dans les *ServerNetworkGroups*. Les *networkPaths (Chemins de Réseaux)* dans la structure représentent les chemins de réseaux redondants pour chacun des *Serveurs*. Les *networkPaths* décrivent les différents chemins (une entrée pour chaque chemin) ordonnés par priorité. Chaque chemin de réseau contient une *endpointUrlList (Liste des Url de points d'extrémité)* comportant une matrice de Chaînes, chacune contenant une URL d'un *Point d'extrémité*. Ceci permet d'utiliser différentes options de protocole pour le même chemin de réseau.

Les *Points d'extrémité* fournis doivent correspondre aux *Points d'extrémité* fournis par le Service *GetEndpoints* du *Serveur* correspondant.

6.3.11 OperationLimitsType

Cet *ObjectType* est un sous-type de *FolderType (Type de Dossier)* et sert à identifier les limites de fonctionnement du *Serveur* OPC UA. Il est défini de façon formelle dans le Tableau 18.

Tableau 18 – Définition de OperationLimitsType

Attribut	Valeur				
BrowseName	OperationLimitsType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du FolderType défini en 6.6, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	MaxNodesPerRead	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxNodesPerHistoryReadData	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxNodesPerHistoryReadEvents	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxNodesPerWrite	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxNodesPerHistoryUpdateData	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxNodesPerHistoryUpdateEvents	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxNodesPerMethodCall	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxNodesPerBrowse	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxNodesPerRegisterNodes	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxNodesPerTranslateBrowsePaths ToNodeIds	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxNodesPerNodeManagement	UInt32	PropertyType	Facultatif
HasProperty	Variable	MaxMonitoredItemsPerCall	UInt32	PropertyType	Facultatif

La *Propriété MaxNodesPerRead* indique la taille maximale de la matrice nodesToRead lorsqu'un *Client* appelle le *Service* Read.

La *Propriété MaxNodesPerHistoryReadData* indique la taille maximale de la matrice nodesToRead lorsqu'un *Client* appelle le *Service* HistoryRead en utilisant les historyReadDetails RAW, PROCESSED, MODIFIED ou ATTIME.

La *Propriété MaxNodesPerHistoryReadEvents* indique la taille maximale de la matrice nodesToRead lorsqu'un *Client* appelle le *Service* HistoryRead en utilisant les historyReadDetails EVENTS.

La *Propriété MaxNodesPerWrite* indique la taille maximale de la matrice nodesToWrite lorsqu'un *Client* appelle le *Service* Write.

La *Propriété MaxNodesPerHistoryUpdateData* indique la taille maximale de la matrice historyUpdateDetails prise en charge par le *Serveur* lorsqu'un *Client* appelle le *Service* HistoryUpdate en utilisant les historyReadDetails RAW, PROCESSED, MODIFIED ou ATTIME.

La *Propriété MaxNodesPerHistoryUpdateEvents* indique la taille maximale de la matrice historyUpdateDetails lorsqu'un *Client* appelle le *Service* HistoryUpdate en utilisant les historyReadDetails EVENTS.

La *Propriété MaxNodesPerMethodCall* indique la taille maximale de la matrice methodsToCall lorsqu'un *Client* appelle le *Service* Call.

La *Propriété MaxNodesPerBrowse* indique la taille maximale de la matrice nodesToBrowse lors de l'appel du *Service* Browse ou de la matrice continuationPoints (Points de continuation) lorsqu'un *Client* appelle le *Service* BrowseNext.

La *Propriété MaxNodesPerRegisterNodes* indique la taille maximale de la matrice nodesToRegister lorsqu'un *Client* appelle le *Service* RegisterNodes et la taille maximale de la matrice nodesToUnregister lors de l'appel du *Service* UnregisterNodes.

La *Propriété MaxNodesPerTranslateBrowsePathsToNodeIds* indique la taille maximale de la matrice browsePaths lorsqu'un *Client* appelle le *Service* TranslateBrowsePathsToNodeIds.

La *Propriété MaxNodesPerNodeManagement* indique la taille maximale de la matrice nodesToAdd lorsqu'un *Client* appelle le *Service* AddNodes, la taille maximale de la matrice referencesToAdd lorsqu'un *Client* appelle le *Service* AddReferences, la taille maximale de la matrice nodesToDelete lorsqu'un *Client* appelle le *Service* DeleteNodes, et la taille maximale de la matrice referencesToDelete lorsqu'un *Client* appelle le *Service* DeleteReferences.

La *Propriété MaxMonitoredItemsPerCall* indique la taille maximale de la matrice itemsToCreate lorsqu'un *Client* appelle le *Service* CreateMonitoredItems, la taille maximale de la matrice itemsToModify lorsqu'un *Client* appelle le *Service* ModifyMonitoredItems, la taille maximale de la matrice monitoredItemIds lorsqu'un *Client* appelle le *Service* SetMonitoringMode, et la taille maximale des matrices linksToAdd et linksToRemove lorsqu'un *Client* appelle le *Service* SetTriggering.

6.3.12 AddressSpaceFileType

Cet *ObjectType* définit le fichier propre à un espace de nom fourni par le *Serveur* OPC UA. Il est défini de façon formelle dans le Tableau 19. Il représente fichier d'Espace d'Adresses XML utilisant le schéma XML défini dans l'IEC 62541-6.

Tableau 19 – Définition de AddressSpaceFileType

Attribut	**Valeur**				
BrowseName	AddressSpaceFileType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du FileType défini en C.2					
HasComponent	Method	ExportNamespace	La méthode n'a pas de paramètres.		Facultatif

La *Méthode ExportNamespace* fournit un moyen d'exportation de l'espace de nom de l'*Espace d'Adresses* du *Serveur* au fichier XML représenté par l'*AddressSpaceFileType*. Les *Attributs Value* sont exportés uniquement s'ils représentent des informations de configuration statiques. Le client est censé appeler tout d'abord la *Méthode ExportNamespace* afin de mettre à jour le fichier XML, puis accéder au fichier avec les *Méthodes* définies dans le *FileType*.

Les *Serveurs* peuvent fournir certains mécanismes spécifiques au fournisseur qui importent des parties d'un Espace d'Adresses comme sous-type de cet *ObjectType*, par exemple, en définissant des *Méthodes* appropriées.

6.3.13 NamespaceMetadataType

Cet *ObjectType* définit les métadonnées applicables à un espace de nom fourni par le *Serveur*. Il est défini de façon formelle dans le Tableau 20.

Les instances de cet *Objet* permettent aux *Serveurs* de fournir davantage d'informations telles que des informations concernant la version outre l'URI de l'espace de nom. Des informations importantes pour les *Serveurs* d'agrégation sont fournies par les *Propriétés StaticNodeIdTypes, StaticNumericNodeIdRange et StaticStringNodeIdPattern*.

Tableau 20 – Définition de NamespaceMetadataType

Attribut	Valeur				
BrowseName	NamespaceMetadataType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					
HasProperty	Variable	NamespaceUri	String	PropertyType	Obligatoire
HasProperty	Variable	NamespaceVersion	String	PropertyType	Obligatoire
HasProperty	Variable	NamespacePublicationDate	DateTime	PropertyType	Obligatoire
HasProperty	Variable	IsNamespaceSubset	Boolean	PropertyType	Obligatoire
HasProperty	Variable	StaticNodeIdTypes	IdType[]	PropertyType	Obligatoire
HasProperty	Variable	StaticNumericNodeIdRange	NumericRange[]	PropertyType	Obligatoire
HasProperty	Variable	StaticStringNodeIdPattern	String	PropertyType	Obligatoire
HasComponent	Object	NamespaceFile	-	AddressSpaceFileType	Facultatif

Le *BrowseName* des instances de ce type doit être déduit de l'espace de nom représenté. Ceci peut par exemple être réalisé au moyen de l'indice de l'espace de nom dans la *NamespaceArray* en tant que *namespaceIndex* du *QualifiedName* (*Nom Qualifié*) et de l'URI de l'espace de nom en tant que *nom* du *QualifiedName*.

La *Propriété NamespaceUri* contient l'espace de nom représenté par une instance du MetaDataType.

La *Propriété NamespaceVersion* fournit les informations de version applicables à l'espace de nom. Elle est destinée à des fins d'affichage et ne doit pas être utilisée pour identifier de manière programmatique la toute dernière version.

La *Propriété NamespacePublicationDate* fournit la date de publication de la version de l'espace de nom. Cette valeur de *Propriété* peut être utilisée par les *Clients* pour déterminer la toute dernière version lorsque différentes versions sont fournies par différents *Serveurs*.

La *Propriété IsNamespaceSubset* définit si tous les *Nœuds* de l'espace de nom sont accessibles dans le *Serveur* ou uniquement un sous-ensemble. Elle est mise à FALSE si l'espace de nom complet est fourni et à TRUE dans le cas contraire.

Les *Nœuds* statiques sont identiques pour tous les *Attributs* dans tous les *Serveurs*, y compris l'*Attribut Valeur*. Pour les*TypeDefinitionNodes*, les *InstanceDeclarations* doivent également être identiques. Cela signifie que la sémantique est toujours la même pour les *Nœuds* statiques. Les espaces de noms avec des *Nœuds* statiques sont, par exemple, des espaces de noms définis par les organismes de normalisation tels que la Fondation OPC. Ceci constitue des informations importantes pour les *Serveurs* d'agrégation. Lorsque l'espace de nom est dynamique et utilisé dans plusieurs *Serveurs,* il faut que le *Serveur* d'agrégation différencie l'espace de nom pour chaque *Serveur* agrégé. Il est nécessaire de traiter les *Nœuds* d'un espace de nom une fois uniquement, même s'ils sont utilisés par plusieurs *Serveurs* agrégés.

La *Propriété StaticNodeIdTypes* fournit une liste des *IdTypes (Types d'Identificateur)* utilisés pour les *Nœuds* statiques. Tous les *Nœuds* dans l'*Espace d'Adresses* de l'espace de nom utilisant l'un des *IdTypes* dans la matrice doivent être des *Nœuds* statiques.

La *Propriété StaticNumericNodeIdRange* fournit une liste des *NumericRanges* (*Intervalles Numériques*) utilisés pour les *NodeIds* des *Nœuds* statiques. Si la *Propriété StaticNodeIdTypes* contient une entrée propre aux NodeIds numériques, il n'est alors pas tenu compte de cette Propriété.

La *Propriété StaticStringNodeIdPattern* fournit une expression régulière telle que spécifiée pour l'*Opérateur Like* défini dans l'IEC 62541-4 pour le filtrage des *NodeIds* de chaînes des

Nœuds statiques. Lorsque la *Propriété StaticNodeIdTypes* contient une entrée propre aux *NodeIds* de chaînes, il n'est alors pas tenu compte de cette *Propriété*.

L'*Objet NamespaceFile* contient tous les *Nœuds* et *Références* de l'espace de nom dans un fichier XML où le schéma XML est défini dans l'IEC 62541-6. Le fichier XML est fourni par un *Objet AddressSpaceFileType*.

6.3.14 NamespacesType

Cet *ObjectType* définit une liste des *Objets NamespaceMetadataType* fournis par le *Serveur*. Il est défini de façon formelle dans le Tableau 21.

Tableau 21 – Définition de NamespacesType

Attribut	**Valeur**				
BrowseName	NamespacesType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					
HasComponent	Object	<NamespaceIdentifier>	-	NamespaceMetadataType	OptionalPlaceholder

L'*ObjectType* contient une liste des *Objets NamespaceMetadataType* qui représentent les espaces de nom dans le *Serveur*. Le *BrowseName* d'un *Objet* doit être déduit de l'espace de nom représenté par l'*Objet*. Ceci peut par exemple être réalisé au moyen de l'indice de l'espace de nom dans la *NamespaceArray* en tant que *namespaceIndex* du *QualifiedName* et de l'URI de l'espace de nom en tant que *nom* du *QualifiedName*.

6.4 ObjectTypes utilisés comme EventTypes

6.4.1 Généralités

La présente norme internationale définit les *EventTypes* (*Types d'Événement*) normalisés. Ils sont représentés dans l'*Espace d'Adresses* comme des *ObjectTypes*. Les *EventTypes* sont déjà définis dans l'IEC 62541-3. Les paragraphes ci-après spécifient leur représentation dans l'*Espace d'Adresses*.

6.4.2 BaseEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 22.

Tableau 22 – Définition de BaseEventType

Attribut	Valeur				
BrowseName	BaseEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseObjectType* défini en 6.2					
HasSubtype	ObjectType	AuditEventType	Défini en 6.4.3		
HasSubtype	ObjectType	SystemEventType	Défini en 6.4.28		
HasSubtype	ObjectType	BaseModelChangeEventType	Défini en 6.4.31		
HasSubtype	ObjectType	SemanticChangeEventType	Défini en 6.4.33		
HasSubtype	ObjectType	EventQueueOverflowEventType	Défini en 6.4.34		
HasSubtype	ObjectType	ProgressEventType	Défini en 6.4.35		
HasProperty	Variable	EventId	ByteString	PropertyType	Obligatoire
HasProperty	Variable	EventType	NodeId	PropertyType	Obligatoire
HasProperty	Variable	SourceNode	NodeId	PropertyType	Obligatoire
HasProperty	Variable	SourceName	String	PropertyType	Obligatoire
HasProperty	Variable	Time	UTCTime	PropertyType	Obligatoire
HasProperty	Variable	ReceiveTime	UTCTime	PropertyType	Obligatoire
HasProperty	Variable	LocalTime	TimeZoneDataType	PropertyType	Facultatif
HasProperty	Variable	Message	LocalizedText	PropertyType	Obligatoire
HasProperty	Variable	Severity	UInt16	PropertyType	Obligatoire

EventId est généré par le *Serveur* pour identifier de façon unique une *Event Notification* (*Notification d'Evénement*) particulière. Le *Serveur* est chargé d'assurer que chaque *Événement* a un *EventId* unique. Il peut le faire, par exemple, en plaçant des GUID dans la ByteString. Les clients peuvent utiliser l'*EventId* pour contribuer à réduire au maximum, voire éliminer les trous et recouvrements qui peuvent se produire au cours de la reprise sur défaillance de redondance. L'*EventId* doit toujours être retourné comme valeur et le *Serveur* n'est pas autorisé à retourner un *StatusCode* (*Code de Statut*) pour l'*EventId* qui indique une erreur.

EventType décrit le type spécifique de l'*Événement*. *EventType* doit toujours être retourné comme valeur et le *Serveur* n'est pas autorisé à retourner un *StatusCode* pour l'*EventType* qui indique une erreur.

SourceNode identifie le *Nœud* qui est l'origine de l'*Événement*. Si l'*Événement* n'est pas spécifique à un *Nœud*, le *NodeId* est mis à null. Certains sous-types de ce *BaseEventType* peuvent définir des règles complémentaires pour le *SourceNode*.

SourceName fournit une description de la source de l'*Événement*. Il peut constituer la partie chaîne du *DisplayName* de la source de l'*Événement* utilisant le paramètre de lieu par défaut du serveur, si l'*Événement* est spécifique à un *Nœud,* ou quelque autre notation spécifique au serveur.

Time fournit l'heure à laquelle l'*Événement* a eu lieu. Cette valeur est mise à disposition du générateur d'événement dès que possible. Elle provient souvent de l'appareil ou système sous-jacent. La valeur étant établie, les *Serveurs* OPC UA intermédiaires ne doivent pas modifier la valeur.

ReceiveTime fournit l'heure à laquelle le *Serveur* OPC UA a reçu l'*Événement* de la part de l'appareil sous-jacent d'un autre *Serveur*. *ReceiveTime* est analogue à *ServerTimestamp* défini dans l'IEC 62541-4, c'est-à-dire que dans le cas où le *Serveur* OPC UA reçoit un *Événement* d'un autre *Serveur* OPC UA, chaque *Serveur* applique son propre *ReceiveTime*. Cela implique qu'un *Client* peut obtenir le même *Événement,* ayant le même *EventId*, de la part de *Serveurs* différents ayant des valeurs différentes pour *ReceiveTime*. *ReceiveTime* doit toujours être retourné comme valeur et le *Serveur* n'est pas autorisé à retourner un *StatusCode* pour le *ReceiveTime* qui indique une erreur.

LocalTime est une structure contenant l'Offset (Décalage) et le fanion DaylightSavingInOffset (heure d'été/hiver incluse dans le décalage). L'Offset spécifie la différence temporelle (en minutes) entre la *Propriété Time* et l'heure à l'emplacement où l'événement a été émis. Si DaylightSavingInOffset est TRUE, l'heure standard/l'heure été-hiver (DST) à l'emplacement d'origine est en vigueur et l'Offset inclut la correction de DST. S'il est FALSE, l'Offset n'inclut alors pas la correction de DST et l'heure DST peut ou peut ne pas avoir été en vigueur.

Message fournit une description textuelle en clair et localisable de l'*Événement*. Le *Serveur* peut retourner n'importe quel texte approprié pour décrire l'*Événement*. Une chaîne null n'est pas une valeur valide; si le *Serveur* n'a pas de description, il doit retourner la partie chaîne texte du *BrowseName* du *Nœud* associé à l'*Événement*.

Severity (Sévérité) est une indication de l'urgence de l'*Événement*. Cela s'appelle également communément «priorité». Les valeurs s'étendent entre 1 (la sévérité la plus faible) et 1 000 (la sévérité la plus élevée). En général, une sévérité de 1 indiquerait un *Événement* de nature informative alors qu'une valeur de 1 000 indiquerait un *Événement* de nature catastrophique, qui se traduirait probablement par de sévères pertes financières ou des pertes humaines.

Très peu de mises en œuvre de *Serveur* sont censées prendre en charge 1 000 niveaux de sévérité distincts. Par conséquent, les développeurs de *Serveur* ont la responsabilité de répartir leurs niveaux de sévérité dans la plage de 1 à 1 000 de manière à ce que les clients puissent mettre en place une distribution linéaire. Par exemple, si un client souhaite présenter cinq niveaux de sévérité à un utilisateur, il convient qu'il soit capable d'effectuer la correspondance suivante:

Sévérité client	**Sévérité OPC**
HIGH (élevée)	801 – 1 000
MEDIUM HIGH (moyennement élevée)	601 – 800
MEDIUM (moyenne)	401 – 600
MEDIUM LOW (moyennement faible)	201 – 400
LOW (faible)	1 – 200

Dans de nombreux cas, une correspondance strictement linéaire avec la plage des Sévérités OPC n'est pas responsable des sévérités de sources sous-jacentes. Au contraire, le développeur de *Serveur* met de façon intelligente les sévérités des sources sous-jacentes en correspondance avec la plage de Sévérités OPC de 1 à 1 000. En particulier, il est recommandé que les développeurs de *Serveur* mettent les *Événements* de haute urgence en correspondance avec la plage de sévérités OPC de 667 à 1 000, les *Événements* de moyenne urgence en correspondance avec la plage de sévérités OPC de 334 à 666 et les *Événements* de faible urgence en correspondance avec la plage de sévérités OPC de 1 à 333.

Par exemple, si une source prend en charge 16 niveaux de sévérité qui sont regroupés afin que les sévérités 0 à 2 soient considérées comme LOW, 3 à 7 comme MEDIUM et 8 à 15 comme HIGH, une correspondance appropriée peut alors être comme suit:

Plage OPC	Sévérité de la source	Sévérité OPC
HIGH (667 – 1 000)	15	1 000
	14	955
	13	910
	12	865
	11	820
	10	775
	9	730
	8	685
MEDIUM (334 – 666)	7	650
	6	575
	5	500
	4	425
	3	350
LOW (1 – 333)	2	300
	1	150
	0	1

Certains *Serveurs* peuvent ne prendre en charge aucun *Événement* de nature catastrophique et ils peuvent choisir de mettre toutes leurs sévérités en correspondance avec un sous-ensemble de la plage de 1 à 1 000 (par exemple, 1 à 666). D'autres *Serveurs* peuvent ne prendre en charge aucun *Événement* de nature purement informationnelle et ils peuvent ainsi choisir de mettre toutes leurs sévérités en correspondance avec un autre sous-ensemble de la plage de 1 à 1 000 (par exemple, 334 à 1 000).

Cette approche a pour but de permettre à des clients d'utiliser en toute cohérence des valeurs de sévérité issues de plusieurs *Serveurs* provenant de fournisseurs différents. Des informations complémentaires sur la sévérité peuvent être consultées dans l'IEC 62541-9.

6.4.3 AuditEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 23.

Tableau 23 – Définition de AuditEventType

Attribut	Valeur				
BrowseName	AuditEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasSubtype	ObjectType	AuditSecurityEventType	Défini en 6.4.4		
HasSubtype	ObjectType	AuditNodeManagementEventType	Défini en 6.4.17		
HasSubtype	ObjectType	AuditUpdateEventType	Défini en 6.4.24		
HasSubtype	ObjectType	AuditUpdateMethodEventType	Défini en 6.4.27		
HasProperty	Variable	ActionTimeStamp	UTCTime	PropertyType	Obligatoire
HasProperty	Variable	Status	Boolean	PropertyType	Obligatoire
HasProperty	Variable	ServerId	String	PropertyType	Obligatoire
HasProperty	Variable	ClientAuditEntryId	String	PropertyType	Obligatoire
HasProperty	Variable	ClientUserId	String	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2.

ActionTimeStamp (*Horodatage de l'action*) identifie l'heure à laquelle l'utilisateur a lancé l'action qui a abouti à la création de l'*AuditEvent*. Il diffère de la *Propriété Time,* car il s'agit de l'heure à laquelle le serveur a généré l'*AuditEvent* documentant l'action.

Status (Statut) identifie si l'action demandée peut être exécutée (mettre *Status* à TRUE) ou non (mettre *Status* à FALSE).

ServerId identifie de façon unique le *Serveur* créant l'*Événement*. Il identifie le *Serveur* de façon unique même dans un scénario de redondance transparente commandée par *serveur* lorsque plusieurs *Serveurs* peuvent utiliser le même URI.

ClientAuditEntryId contient l'*AuditEntryId* lisible en clair et défini dans l'IEC 62541-3.

Le *ClientUserId* identifie l'utilisateur du client demandant une action. Le *ClientUserId* peut être obtenu à partir du *UserIdentityToken* passé dans l'appel d'*ActivateSession*. Si l'*UserIdentityToken* est un *UserNameIdentityToken*, le *ClientUserId* est alors l'*UserName.* Si l'*UserIdentityToken* est un *X509IdentityToken,* le *ClientUserId* est alors le Nom de Sujet X509 du *Certificat*. Si l'*UserIdentityToken* est un *IssuedIdentityToken*, il convient alors que le *ClientUserId* soit une chaîne qui représente le propriétaire du jeton. Le meilleur choix de la chaîne dépend du type d'*IssuedIdentityToken.* Si un *AnonymousIdentityToken* a été utilisé, la valeur est null.

6.4.4 AuditSecurityEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 24.

Tableau 24 – Définition de AuditSecurityEventType

Attribut	**Valeur**				
BrowseName	AuditSecurityEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'AuditEventType défini en 6.4.3, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasSubtype	ObjectType	AuditChannelEventType	Défini en 6.4.5		
HasSubtype	ObjectType	AuditSessionEventType	Défini en 6.4.7		
HasSubtype	ObjectType	AuditCertificateEventType	Défini en 6.4.12		

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditEventType*. Leur sémantique est définie en 6.4.3. Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.5 AuditChannelEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 25.

Tableau 25 – Définition de AuditChannelEventType

Attribut	**Valeur**				
BrowseName	AuditChannelEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'AuditSecurityEventType défini en 6.4.4, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasSubtype	ObjectType	AuditOpenSecureChannelEventType	Défini en 6.4.6		
HasProperty	Variable	SecureChannelId	String	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSecurityEventType*. Leur sémantique est définie en 6.4.4. Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*. Il convient que le *SourceNode* pour les *Événements* de ce type soit affecté à l'*Objet Server*. Il convient que le *SourceName* pour les *Événements* de ce type soit "SecureChannel/" et le *Service* qui génère l'*Événement* (par exemple SecureChannel/*OpenSecureChannel* ou

SecureChannel/*CloseSecureChannel*). Si le *ClientUserId* n'est pas disponible pour un appel *CloseSecureChannel*, ce paramètre doit être mis à ”System/CloseSecureChannel”.

Le *SecureChannelId* doit identifier de façon unique le SecureChannel (*Canal Sécurisé*). L'application doit utiliser le même identificateur dans tous les *AuditEvents* (*Événements d'audit*) relatifs au Jeu de Services Session (AuditCreateSessionEventType, AuditActivateSessionEventType et leurs sous-types) et au Jeu de Services SecureChannel (AuditChannelEventType et ses sous-types).

6.4.6 AuditOpenSecureChannelEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 26.

Tableau 26 – Définition de AuditOpenSecureChannelEventType

Attribut	**Valeur**				
BrowseName	AuditOpenSecureChannelEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditChannelEventType* défini en 6.4.5, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	ClientCertificate	ByteString	PropertyType	Obligatoire
HasProperty	Variable	ClientCertificateThumbprint	String	PropertyType	Obligatoire
HasProperty	Variable	RequestType	SecurityTokenRequestType	PropertyType	Obligatoire
HasProperty	Variable	SecurityPolicyUri	String	PropertyType	Obligatoire
HasProperty	Variable	SecurityMode	MessageSecurityMode	PropertyType	Obligatoire
HasProperty	Variable	RequestedLifetime	Duration	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditChannelEventType*. Leur sémantique est définie en 6.4.5. Il convient que le *SourceName* pour les *Événements* de ce type soit “SecureChannel/OpenSecureChannel”. Le *ClientUserId* n'est pas disponible pour cet appel et, donc, ce paramètre doit être mis à ”System/OpenSecureChannel”.

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres du *Service* Call qui a déclenché l'*Événement*.

ClientCertificate est le paramètre clientCertificate d'appel du *Service* OpenSecureChannel.

ClientCertificateThumbprint est une empreinte caractéristique du *ClientCertificate. Voir* l'IEC 62541-6 pour les détails concernant les empreintes.

RequestType est le paramètre requestType d'appel du *Service* OpenSecureChannel.

SecurityPolicyUri est le paramètre securityPolicyUri d'appel du *Service* OpenSecureChannel.

SecurityMode est le paramètre securityMode d'appel de *Service* OpenSecureChannel.

RequestedLifetime est le paramètre requestedLifetime d'appel du *Service* OpenSecureChannel.

6.4.7 AuditSessionEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 27.

Tableau 27 – Définition de AuditSessionEventType

Attribut	Valeur				
BrowseName	AuditSessionEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditEventType* défini en 6.4.4, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasSubtype	ObjectType	AuditCreateSessionEventType	Défini en 6.4.8		
HasSubtype	ObjectType	AuditActivateSessionEventType	Défini en 6.4.10		
HasSubtype	ObjectType	AuditCancelEventType	Défini en 6.4.11		
HasProperty	Variable	SessionId	NodeId	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditEventType*. Leur sémantique est définie en 6.4.4.

Si l'*Événement* est généré par un appel de *Service TransferSubscriptions*, il convient que le *SourceNode* soit affecté à l'*Objet SessionDiagnostics* qui représente la session. Il convient que le *SourceName* pour les *Événements* de ce type soit "Session/TransferSubscriptions".

Autrement, il convient que le *SourceNode* pour les *Événements* de ce type soit affecté à l'*Objet Server*. Il convient que le *SourceName* pour les *Événements* de ce type soit "Session/" et le *Service* qui génère l'*Événement* (par exemple *CreateSession*, *ActivateSession* ou *CloseSession*).

Il convient que le *SessionId* contienne le *SessionId* de la session sur laquelle le *Service* Call a été mis. Dans le *Service CreateSession*, il doit être mis au *SessionId* nouvellement créé. En l'absence de contexte de session (par exemple pour un appel de *Service CreateSession* non abouti), le *SessionId* est mis à null.

6.4.8 AuditCreateSessionEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 28.

Tableau 28 – Définition de AuditCreateSessionEventType

Attribut	Valeur				
BrowseName	AuditCreateSessionEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditSessionEventType* défini en 6.4.7, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasSubtype	ObjectType	AuditUrlMismatchEventType	Défini en 6.4.9		
HasProperty	Variable	SecureChannelId	String	PropertyType	Obligatoire
HasProperty	Variable	ClientCertificate	ByteString	PropertyType	Obligatoire
HasProperty	Variable	ClientCertificateThumbprint	String	PropertyType	Obligatoire
HasProperty	Variable	RevisedSessionTimeout	Duration	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSessionEventType*. Leur sémantique est définie en 6.4.7. Il convient que le *SourceName* pour les *Événements* de ce type soit "Session/CreateSession". Le *ClientUserId* n'est pas disponible pour cet appel et, donc, ce paramètre doit être mis à "System/CreateSession".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres du *Service* Call qui déclenche l'*Événement*.

Le *SecureChannelId* doit identifier de façon unique le SecureChannel. L'application doit utiliser le même identificateur dans tous les *AuditEvents* relatifs à Jeu de Services Session (AuditCreateSessionEventType, AuditActivateSessionEventType et leurs sous-types) et au Jeu de Services SecureChannel (AuditChannelEventType et ses sous-types).

ClientCertificate est le paramètre clientCertificate de l'appel de *Service* CreateSession.

ClientCertificateThumbprint est une empreinte caractéristique du *ClientCertificate.Voir* l'IEC 62541-6 pour les détails concernant les empreintes.

RevisedSessionTimeout est le paramètre revisedSessionTimeout qui est retourné lors de l'appel de *Service* CreateSession.

6.4.9 AuditUrlMismatchEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 29.

Tableau 29 – Définition de AuditUrlMismatchEventType

Attribut	**Valeur**				
BrowseName	AuditUrlMismatchEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'AuditCreateSessionEventType défini en 6.4.8 ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	EndpointUrl	String	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSessionEventType*. Leur sémantique est définie en 6.4.8.

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres du *Service* Call qui déclenche l'*Événement*.

EndpointUrl est le paramètre endpointUrl de l'appel de *Service* CreateSession.

6.4.10 AuditActivateSessionEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 30.

Tableau 30 – Définition de AuditActivateSessionEventType

Attribut	**Valeur**				
BrowseName	AuditActivateSessionEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditSessionEventType* défini en 6.4.7, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	ClientSoftwareCertificates	SignedSoftwareCertificate[]	PropertyType	Obligatoire
HasProperty	Variable	UserIdentityToken	UserIdentityToken	PropertyType	Obligatoire
HasProperty	Variable	SecureChannelId	String	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSessionEventType*. Leur sémantique est définie en 6.4.7. Il convient que le *SourceName* pour les *Événements* de ce type soit "Session/ActivateSession".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres du *Service* Call qui déclenche l'*Événement*.

ClientSoftwareCertificates est le paramètre clientSoftwareCertificates de l'appel de *Service* ActivateSession.

UserIdentityToken reflète le paramètre userIdentityToken de l'appel de *Service* ActivateSession. Pour les jetons Username/Password (Nom d'utilisateur/Mot de passe), il convient de NE PAS inclure le mot de passe.

Le *SecureChannelId* doit identifier de façon unique le SecureChannel. L'application doit utiliser le même identificateur dans tous les *AuditEvents* relatifs au Jeu de Services Session (AuditCreateSessionEventType, AuditActivateSessionEventType et leurs sous-types) et au Jeu de Services SecureChannel (AuditChannelEventType et ses sous-types).

6.4.11 AuditCancelEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 31.

Tableau 31 – Définition de AuditCancelEventType

Attribut	**Valeur**				
BrowseName	AuditCancelEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditSessionEventType* défini en 6.4.7, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	RequestHandle	UInt32	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSessionEventType*. Leur sémantique est définie en 6.4.7. Il convient que le *SourceName* pour les *Événements* de ce type soit "Session/Cancel".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres du *Service* Call qui déclenche l'*Événement*.

RequestHandle est le paramètre requestHandle de l'appel de *Service* Cancel.

6.4.12 AuditCertificateEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 32.

Tableau 32 – Définition de AuditCertificateEventType

Attribut	**Valeur**				
BrowseName	AuditCertificateEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditSecurityEventType* défini en 6.4.7, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasSubtype	ObjectType	AuditCertificateDataMismatchEventType	Défini en 6.4.13		
HasSubtype	ObjectType	AuditCertificateExpiredEventType	Défini en 6.4.14		
HasSubtype	ObjectType	AuditCertificateInvalidEventType	Défini en 6.4.15		
HasSubtype	ObjectType	AuditCertificateUntrustedEventType	Défini en 6.4.16		
HasSubtype	ObjectType	AuditCertificateRevokedEventType	Défini en 6.4.17		
HasSubtype	ObjectType	AuditCertificateMismatchEventType	Défini en 6.4.18		
HasProperty	Variable	Certificate	ByteString	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditSecurityEventType*. Leur sémantique est définie en 6.4.4. Il convient que le *SourceName* pour les *Événements* de ce type soit "Security/Certificate".

Certificate est le certificat qui a rencontré un problème de validation. Des sous-types complémentaires de cet *EventType* sont définis et représentent les erreurs individuelles de

validation. Ce certificat peut être mis en correspondance avec le *Service* qui l'a passé (Jeu de Services Session ou SecureChannel), car les *AuditEvents* de ces *Services* incluaient aussi le certificat.

6.4.13 AuditCertificateDataMismatchEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 33.

Tableau 33 – Définition de AuditCertificateDataMismatchEventType

Attribut	Valeur				
BrowseName	AuditCertificateDataMismatchEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'AuditCertificateEventType défini en 6.4.12, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	InvalidHostname	String	PropertyType	Obligatoire
HasProperty	Variable	InvalidUri	String	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Il convient que le *SourceName* pour les *Événements* de ce type soit "Security/Certificate".

InvalidHostname est la chaîne qui représente le nom d'hôte transmis comme partie de l'URL qui s'est avéré non valide. Si le nom d'hôte n'était pas non valide, il peut être null.

InvalidUri est l'URI qui a été transmis et s'est avéré ne pas correspondre au contenu du certificat. Si l'URI n'était pas non valide, il peut être null.

L'*InvalidHostname* ou bien l'*InvalidUri* doit être fourni.

6.4.14 AuditCertificateExpiredEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 34.

Tableau 34 – Définition de AuditCertificateExpiredEventType

Attribut	Valeur				
BrowseName	AuditCertificateExpiredEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCertificateEventType* défini en 6.4.12, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Il convient que le *SourceName* pour les *Événements* de ce type soit "Security/Certificate". La *Variable Message* doit inclure une description des raisons pour lesquelles le certificat a expiré (c'est-à-dire le temps avant le début ou le temps après la fin). Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.15 AuditCertificateInvalidEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 35.

Tableau 35 – Définition de AuditCertificateInvalidEventType

Attribut	Valeur				
BrowseName	AuditCertificateInvalidEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCertificateEventType* défini en 6.4.12, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Il convient que le *SourceName* pour les *Événements* de ce type soit "Security/Certificate". Le *Message* doit inclure une description de la non-validité du certificat. Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.16 AuditCertificateUntrustedEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 36.

Tableau 36 – Définition de AuditCertificateUntrustedEventType

Attribut	Valeur				
BrowseName	AuditCertificateUntrustedEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCertificateEventType* défini en 6.4.12, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Il convient que le *SourceName* pour les *Événements* de ce type soit "Security/Certificate". La *Variable Message* doit inclure une description des raisons pour lesquelles il n'est pas accordé de confiance au certificat. Si une chaîne de confiance est impliquée, il convient de décrire le certificat qui a échoué dans la chaîne de confiance. Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.17 AuditCertificateRevokedEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est formellement définie dans le Tableau 37.

Tableau 37 – Définition de AuditCertificateRevokedEventType

Attribut	Valeur				
BrowseName	AuditCertificateRevokedEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCertificateEventType* défini en 6.4.12, ce qui signifie qu'il hérite des InstanceDeclarations du Noeud en question.					

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Il convient que le *SourceName* pour les *Événements* de ce type soit "Security/Certificate". La *Variable Message* doit inclure une description des raisons de la révocation du certificat (une liste de révocation était-elle disponible ou le certificat figurait-il sur la liste?). Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.18 AuditCertificateMismatchEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est formellement définie dans le Tableau 38.

Tableau 38 – Définition de AuditCertificateMismatchEventType

Attribut	Valeur				
BrowseName	AuditCertificateMismatchEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCertificateEventType* défini en 6.4.12, ce qui signifie qu'il hérite des InstanceDeclarations du Noeud en question.					

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditCertificateEventType*. Leur sémantique est définie en 6.4.12. Il convient que le *SourceName* pour les *Événements* de ce type soit "Security/Certificate". La *Variable Message* doit inclure une description de l'utilisation impropre du certificat. Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.19 AuditNodeManagementEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 39.

Tableau 39 – Définition de AuditNodeManagementEventType

Attribut	Valeur				
BrowseName	AuditNodeManagementEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditCertificateEventType* défini en 6.4.3, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasSubtype	ObjectType	AuditAddNodesEventType			
HasSubtype	ObjectType	AuditDeleteNodesEventType			
HasSubtype	ObjectType	AuditAddReferencesEventType			
HasSubtype	ObjectType	AuditDeleteReferencesEventType			

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditEventType*. Leur sémantique est définie en 6.4.3. Il n'y a pas d'autres *Properties* définies pour cet *EventType*. Il convient que le *SourceNode* pour les *Événements* de ce type soit affecté à l'*Objet Server*. Il convient que le *SourceName* pour les *Événements* de ce type soit "NodeManagement/" et le *Service* qui génère l'*Événement* (par exemple *AddNodes*, *AddReferences*, *DeleteNodes*, *DeleteReferences*).

6.4.20 AuditAddNodesEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 40.

Tableau 40 – Définition de AuditAddNodesEventType

Attribut	Valeur				
BrowseName	AuditAddNodesEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditNodeManagementEventType* défini en 6.4.17, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	NodesToAdd	AddNodesItem[]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditNodeManagementEventType*. Leur sémantique est définie en 6.4.17. Il convient que le *SourceName* pour les *Événements* de ce type soit "NodeManagement/AddNodes".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres du *Service* Call qui déclenche l'*Événement*.

NodesToAdd est le paramètre NodesToAdd de l'appel de *Service* AddNodes.

6.4.21 AuditDeleteNodesEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 41.

Tableau 41 – Définition de AuditDeleteNodesEventType

Attribut	**Valeur**				
BrowseName	AuditDeleteNodesEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditNodeManagementEventType* défini en 6.4.17, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	NodesToDelete	DeleteNodesItem[]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditNodeManagementEventType*. Leur sémantique est définie en 6.4.17. Il convient que le *SourceName* pour les *Événements* de ce type soit "NodeManagement/DeleteNodes".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres du *Service* Call qui déclenche l'*Événement*.

NodesToDelete est le paramètre nodesToDelete de l'appel de *Service* DeleteNodes.

6.4.22 AuditAddReferencesEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 42.

Tableau 42 – Définition de AuditAddReferencesEventType

Attribut	**Valeur**				
BrowseName	AuditAddReferencesEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditNodeManagementEventType* défini en 6.4.19, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	ReferencesToAdd	AddReferencesItem[]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditNodeManagementEventType*. Leur sémantique est définie en 6.4.19. Il convient que le *SourceName* pour les *Événements* de ce type soit "NodeManagement/AddReferences".

Les *Propriétés* complémentaires définies pour ce Type d'Événement reflètent les paramètres du *Service* Call qui déclenche l'*Événement*.

ReferencesToAdd est le paramètre referencesToAdd de l'appel de *Service* AddReferences.

6.4.23 AuditDeleteReferencesEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 43.

Tableau 43 – Définition de AuditDeleteReferencesEventType

Attribut	Valeur				
BrowseName	AuditDeleteReferencesEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditNodeManagementEventType* défini en 6.4.17, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	ReferencesToDelete	DeleteReferencesItem[]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditNodeManagementEventType*. Leur sémantique est définie en 6.4.17. Il convient que le *SourceName* pour les *Événements* de ce type soit "NodeManagement/DeleteReferences".

Les *Propriétés* complémentaires définies pour cet *EventType* reflètent les paramètres du *Service* Call qui déclenche l'*Événement*.

ReferencesToDelete est le paramètre referencesToDelete de l'appel de *Service* DeleteReferences.

6.4.24 AuditUpdateEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 44.

Tableau 44 – Définition de AuditUpdateEventType

Attribut	Valeur				
BrowseName	AuditUpdateEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditEventType* défini en 6.4.3, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasSubtype	ObjectType	AuditWriteUpdateEventType	Défini en 6.4.25		
HasSubtype	ObjectType	AuditHistoryUpdateEventType	Défini en 6.4.26		

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditEventType*. Leur sémantique est définie en 6.4.3. Il convient d'affecter le *SourceNode* pour les *Événements* de ce type au *NodeId* qui a été modifié. Il convient que le *SourceName* pour les *Événements* de ce type soit "Attribute/" et le *Service* qui a généré l'événement (par exemple *Write*, *HistoryUpdate*). Noter qu'un même *Service* Call peut générer plusieurs *Événements* de ce type, un par valeur modifiée.

6.4.25 AuditWriteUpdateEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 45.

Tableau 45 – Définition de AuditWriteUpdateEventType

Attribut	Valeur				
BrowseName	AuditWriteUpdateEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditUpdateEventType* défini en 6.4.24, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	AttributeID	UInt32	PropertyType	Obligatoire
HasProperty	Variable	IndexRange	NumericRange	PropertyType	Obligatoire
HasProperty	Variable	NewValue	BaseDataType	PropertyType	Obligatoire
HasProperty	Variable	OldValue	BaseDataType	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditUpdateEventType*. Il convient que le *SourceName* pour les *Événements* de ce type soit “Attribute/Write”. Leur sémantique est définie en 6.4.24.

AttributeId identifie l'*Attribut* qui a été écrit sur le *SourceNode*.

IndexRange identifie la plage d'indices de l'*Attribut* écrit si l'*Attribut* est une matrice. Si l'*Attribut* n'est pas une matrice ou la matrice complète a été inscrite, *IndexRange* est mise à null.

NewValue identifie la valeur qui a été écrite au *SourceNode*. Si *IndexRange* est fournie, seules les valeurs de la plage fournie sont montrées.

OldValue identifie la valeur que contenait le *SourceNode* avant l'écriture. Si *IndexRange* est fournie, seule la valeur de la plage en question est montrée. Il est acceptable qu'un *Serveur* qui n'a pas cette information rapporte une valeur null.

La *NewValue* et aussi l'*OldValue* contiendront une valeur dans le *DataType* et le codage utilisé pour écrire la valeur.

6.4.26 AuditHistoryUpdateEventType

Cet *EventType* est défini dans l’IEC 62541-3. Sa représentation dans l'*Espace d’Adresses* est définie de façon formelle dans le Tableau 46.

Tableau 46 – Définition de AuditHistoryUpdateEventType

Attribut	**Valeur**				
BrowseName	AuditHistoryUpdateEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditUpdateEventType* défini en 6.4.24, ce qui signifie qu’il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	ParameterDataTypeId	NodeId	PropertyType	New

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditUpdateEventType*. Leur sémantique est définie en 6.4.24.

Le *ParameterDataTypeId* identifie le *DataTypeId* pour le paramètre extensible utilisé par l'HistoryUpdate. Ce paramètre indique le type de HistoryUpdate qui est en cours d'exécution.

Les sous-types de cet *EventType* sont définis dans l’IEC 62541-11 représentant les différentes possibilités de manipulation de données historiques.

6.4.27 AuditUpdateMethodEventType

Cet *EventType* est défini dans l’IEC 62541-3. Sa représentation dans l'*Espace d’Adresses* est définie de façon formelle dans le Tableau 47.

Tableau 47 – Définition de AuditUpdateMethodEventType

Attribut	Valeur				
BrowseName	AuditUpdateMethodEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditEventType* défini en 6.4.3, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	MethodId	NodeId	PropertyType	Obligatoire
HasProperty	Variable	InputArguments	BaseDataType[]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditEventType*. Leur sémantique est définie en 6.4.3. Il convient d'affecter le *SourceNode* pour les *Événements* de type au *NodeId* de l'objet sur lequel la méthode réside. Il convient que le *SourceName* pour les *Événements* de ce type soit "Attribute/Call". Noter qu'un même *Service* Call peut générer plusieurs *Événements* de ce type, un par méthode appelée. Il convient de sous-typer cet *EventType* pour mieux refléter la fonctionnalité de la méthode et refléter les modifications apportées à l'Espace d'Adresses ou les valeurs mises à jour déclenchées par la méthode.

MethodId identifie la méthode qui a été appelée.

InputArguments identifie les arguments d'entrée pour la méthode. Ce paramètre peut être null si aucun argument d'entrée n'a été fourni.

6.4.28 SystemEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 48.

Tableau 48 – Définition de SystemEventType

Attribut	Valeur				
BrowseName	SystemEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasSubtype	ObjectType	DeviceFailureEventType	Défini en 6.4.29		
HasSubtype	ObjectType	SystemStatusChangeEventType	Défini en 6.4.30		
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2. Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.29 DeviceFailureEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 49.

Tableau 49 – Définition de DeviceFailureEventType

Attribut	Valeur				
BrowseName	DeviceFailureEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *SystemEventType* défini en 6.4.28, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					

Cet *EventType* hérite de toutes les *Propriétés* du *SystemEventType*. Leur sémantique est définie en 6.4.2. Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*.

6.4.30 SystemStatusChangeEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 50.

Tableau 50 – Définition de SystemStatusChangeEventType

Attribut	**Valeur**				
BrowseName	SystemStatusChangeEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *SystemEventType* défini en 6.4.28, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	SystemState	ServerState	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* du *SystemEventType*. Leur sémantique est définie en 6.4.28. Le *SourceNode* et le *SourceName* doivent identifier le système. Le système peut être le *Serveur* lui-même ou un système sous-jacent.

Le *SystemState* spécifie l'état courant du système. Les changements apportés au *ServerState* du système doivent déclencher un *SystemStatusChangeEvent*, lorsque l'événement est pris en charge par le système.

6.4.31 BaseModelChangeEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 51.

Tableau 51 – Définition de BaseModelChangeEventType

Attribut	**Valeur**				
BrowseName	BaseModelChangeEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasSubtype	ObjectType	GeneralModelChangeEventType	Défini en 6.4.32		

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2. Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*. Il convient que le *SourceNode* pour les événements de ce type soit le *Nœud* de la *View* qui donne le contexte des modifications. Si l'*Espace d'Adresses* tout entier est le contexte, le *SourceNode* est mis au *NodeId* de l'*Objet Server*. Il convient que le *SourceName* pour les *Événements* de ce type soit la partie *String* du *BrowseName* de la *View*; pour l'*Espace d'Adresses* tout entier, il convient que ce soit "Server".

6.4.32 GeneralModelChangeEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 52.

Tableau 52 – Définition de GeneralModelChangeEventType

Attribut	**Valeur**				
BrowseName	GeneralModelChangeEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseModelChangeEventType* défini en 6.4.31, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	Changes	ModelChangeStructureDataType[]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* du *BaseModelChangeEventType*. Leur sémantique est définie en 6.4.31.

La *Propriété* supplémentaire définie pour cet *EventType* reflète les changements qui ont émis le *ModelChangeEvent*. Elle doit contenir au moins une entrée dans sa matrice. Sa structure est définie en 12.16.

6.4.33 SemanticChangeEventType

Cet *EventType* est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est définie de façon formelle dans le Tableau 53.

Tableau 53 – Définition de SemanticChangeEventType

Attribut	**Valeur**				
BrowseName	SemanticChangeEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	Changes	SemanticChangeStructureDataType[]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2. Il n'y a pas d'autres *Propriétés* définies pour cet *EventType*. Il convient que le *SourceNode* pour les Événements de ce type soit le *Nœud* de la *View* qui donne le contexte des modifications. Si l'*Espace d'Adresses* tout entier est le contexte, le *SourceNode* est mis au *NodeId* de l'*Objet Server*. Il convient que le *SourceName* pour les *Événements* de ce type soit la partie *String* du *BrowseName* de la *View*; pour l'*Espace d'Adresses* tout entier, il convient que ce soit "Server".

La *Propriété* supplémentaire définie pour cet *EventType* reflète les changements qui ont émis le *SemanticChangeEvent*. Sa structure est définie en 12.17.

6.4.34 EventQueueOverflowEventType

Les *Événements EventQueueOverflow* sont créés lorsqu'une file d'attente interne d'un MonitoredItem s'abonnant aux *Événements* dans le *Serveur* déborde. L'IEC 62541-4 définit le moment auquel les *Événements EventQueueOverflow* internes doivent être générés.

L'*EventType* pour les *Événements EventQueueOverflow* est défini de façon formelle dans le Tableau 54.

Tableau 54 – Définition de EventQueueOverflowEventType

Attribut	**Valeur**				
BrowseName	EventQueueOverflowEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2. Le *SourceNode* pour les *Événements* de ce type doit être affecté au *NodeId* de l'*Objet Server*. Le *SourceName* pour les *Événements* de ce type doit être "Internal/EventQueueOverflow".

6.4.35 ProgressEventType

Les ProgressEvents sont créés pour identifier l'avancement d'une opération. Une opération peut être un *Service* Call ou toute action spécifique à l'application telle que l'exécution d'un programme.

L'*EventType* pour les *Événements Progress* est défini de façon formelle dans le Tableau 55.

Tableau 55 – Définition de ProgressEventType

Attribut	**Valeur**				
BrowseName	ProgressEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseEventType* défini en 6.4.2, ce qui signifie qu'il hérite des InstanceDeclarations du Nœud en question.					
HasProperty	Variable	Context	BaseDataType	PropertyType	Obligatoire
HasProperty	Variable	Progress	UInt16	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* du *BaseEventType*. Leur sémantique est définie en 6.4.2. Le *SourceNode* pour les *Événements* de ce type doit être affecté au *NodeId* de l'*Objet Session* où l'opération a été déclenchée. Le *SourceName* pour les *Événements* de ce type doit être "Service/<Service Name as defined in IEC 62541-4>" lorsque l'avancement d'un *Service* Call est présenté.

La *Propriété* supplémentaire *Context* contient les informations de contexte concernant le type d'avancement d'opération consigné. Dans le cas des appels de *Service*, ce doit être un UInt32 contenant la *requestHandle* du *RequestHeader* du *Service* Call.

La *Propriété* supplémentaire *Progress* contient le pourcentage d'achèvement de l'avancement. La valeur doit être comprise entre 0 et 100, où 100 indique la fin de l'opération.

Il est recommandé que les *Serveurs* présentent uniquement les *ProgressEvents* pour les appels de *Service* à la *Session* qui a appelé le *Service*.

6.5 ModellingRuleType

Les *ModellingRules* sont définies dans l'IEC 62541-3. Cet *ObjectType* est utilisé comme étant le type pour les *ModellingRules*. Il est défini de façon formelle dans le Tableau 56.

Tableau 56 – Définition de ModellingRuleType

Attribut	**Valeur**				
BrowseName	ModellingRuleType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					
HasProperty	Variable	NamingRule	NamingRuleType	PropertyType	Obligatoire

La *Propriété NamingRule* identifie la *NamingRule* d'une *ModellingRule* telle que définie dans l'IEC 62541-3.

6.6 FolderType

Des Instances de cet *ObjectType* sont utilisées pour organiser l'*Espace d'Adresses* en une hiérarchie de *Nœuds*. Elles représentent le *Nœud root* (*racine*) d'un sous-arbre et n'ont aucune autre sémantique associée. Cependant, il convient que le *DisplayName* d'une instance du *FolderType*, tel que "ObjectTypes", implique la sémantique associée à son utilisation. Il n'y a pas de Références spécifiées pour cet *ObjectType*. Il est défini de façon formelle dans le Tableau 57.

Tableau 57 – Définition de FolderType

Attribut	Valeur				
BrowseName	FolderType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					

6.7 DataTypeEncodingType

Les *DataTypeEncodings* (*Codages de Types de Données*) sont définis dans l'IEC 62541-3. Cet *ObjetType* est utilisé comme type pour les *DataTypeEncodings*. Il n'y a pas de *Références* spécifiées pour cet *ObjectType*. Il est défini de façon formelle dans le Tableau 58.

Tableau 58 – Définition de DataTypeEncodingType

Attribut	Valeur				
BrowseName	DataTypeEncodingType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					

6.8 DataTypeSystemType

Les *DataTypeSystems* (*Systèmes de Types de Données*) sont définis dans l'IEC 62541-3. Cet *ObjectType* est utilisé comme type pour les *DataTypeSystems*. Il n'y a pas de *Références* spécifiées pour cet ObjectType. Il est défini de façon formelle dans le Tableau 59.

Tableau 59 – Définition de DataTypeSystemType

Attribut	Valeur				
BrowseName	DataTypeSystemType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					

6.9 AggregateFunctionType

Cet *ObjectType* définit une *AggregateFunction* prise en charge par un *Serveur* UA. Il est défini de façon formelle dans le Tableau 60.

Tableau 60 – Définition de AggregateFunctionType

Attribut	Valeur				
BrowseName	AggregateFunctionType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					

Pour l'*AggregateFunctionType*, l'*Attribut Description* est obligatoire. L'*Attribut Description* fournit une description localisée de l'*AggregateFunction.* Des *AggregateFunctions* spécifiques peuvent être définies dans d'autres parties de cette série de normes.

7 VariableTypes normalisés

7.1 Généralités

Typiquement, les composantes d'un *VariableType* complexe sont fixes et peuvent être étendues par sous-typage. Cependant, sachant que chaque *Variable* d'un *VariableType* peut être étendue avec des composantes supplémentaires, la présente norme permet d'étendre les *VariableTypes* normalisés définis dans le présent document avec des composantes supplémentaires. Cela permet l'expression d'informations complémentaires dans la TypeDefinition qui seraient de toute façon contenues dans chaque *Variable*. Cependant, il n'est pas permis de limiter les composantes des *VariableTypes* normalisés définis dans la présente Norme internationale. Un exemple d'extension de *VariableTypes* est de placer la *Propriété* normalisée *NodeVersion*, définie dans l'IEC 62541-3, dans le *BaseDataVariableType*, énonçant que chaque *DataVariable* du *Serveur* fournira une *NodeVersion*.

7.2 BaseVariableType

Le *BaseVariableType* est le type de base abstrait pour tous les autres *VariableTypes*. Cependant, seuls le *PropertyType* et le *BaseDataVariableType* héritent directement de ce type.

Pour ce *VariableType*, il n'est pas spécifié d'autres *Références* que les *Références HasSubtype*. Il est défini de façon formelle dans le Tableau 61.

Tableau 61 – Définition de BaseVariableType

Attribut	**Valeur**				
BrowseName	BaseVariableType				
IsAbstract	True				
ValueRank	−2 (−2 = Any)				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasSubtype	VariableType	PropertyType	Défini en 7.3		
HasSubtype	VariableType	BaseDataVariableType	Défini en 7.4		

7.3 PropertyType

Le *PropertyType* est un sous-type du *BaseVariableType*. Il est utilisé comme la définition de type de toutes les *Propriétés*. Les *Propriétés* sont définies par leur *BrowseName* et, donc, elles n'ont pas besoin d'une définition de type spécialisée. Il n'est pas permis de sous-typer ce *VariableType*.

Il n'y a pas de *Références* spécifiées pour ce *VariableType*. Il est défini de façon formelle dans le Tableau 62.

Tableau 62 – Définition de PropertyType

Attribut	**Valeur**				
BrowseName	PropertyType				
IsAbstract	False				
ValueRank	−2 (−2 = Any)				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseVariableType défini en 7.2					

7.4 BaseDataVariableType

Le *BaseDataVariableType* est un sous-type du *BaseVariableType*. Il est utilisé comme définition de type chaque fois qu'il existe une *DataVariable* n'ayant plus de définition de type concret disponible. Ce *VariableType* est le *VariableType* de base pour les *VariableTypes* des *DataVariables*, et tous les autres *VariableTypes* des *DataVariables* doivent hériter, directement ou indirectement, de lui. Mais, il peut ne pas être possible à des *Serveurs* de fournir toutes les *Références HasSubtype* de ce *VariableType* à ses sous-types et, donc, il n'est pas exigé de fournir cette information.

Pour ce *VariableType*, il n'est pas spécifié d'autres *Références* que les *Références HasSubtype*. Il est défini de façon formelle dans le Tableau 63.

Tableau 63 – Définition de BaseDataVariableType

Attribut	Valeur		
BrowseName	BaseDataVariableType		
IsAbstract	False		
ValueRank	−2 (−2 = Any)		
DataType	BaseDataType		
Références	**NodeClass**	**BrowseName**	**Comment**
Sous-type du BaseVariableType défini en 7.2			
HasSubtype	VariableType	ServerVendorCapabilityType	Défini en 7.5
HasSubtype	VariableType	DataTypeDictionaryType	Défini en 7.6
HasSubtype	VariableType	DataTypeDescriptionType	Défini en 7.7
HasSubtype	VariableType	ServerStatusType	Défini en 7.8
HasSubtype	VariableType	BuildInfoType	Défini en 7.9
HasSubtype	VariableType	ServerDiagnosticsSummaryType	Défini en 7.10
HasSubtype	VariableType	SamplingIntervalDiagnosticsArrayType	Défini en 7.11
HasSubtype	VariableType	SamplingIntervalDiagnosticsType	Défini en 7.12
HasSubtype	VariableType	SubscriptionDiagnosticsArrayType	Défini en 7.13
HasSubtype	VariableType	SubscriptionDiagnosticsType	Défini en 7.14
HasSubtype	VariableType	SessionDiagnosticsArrayType	Défini en 7.15
HasSubtype	VariableType	SessionDiagnosticsVariableType	Défini en 7.16
HasSubtype	VariableType	SessionSecurityDiagnosticsArrayType	Défini en 7.17
HasSubtype	VariableType	SessionSecurityDiagnosticsType	Défini en 7.18
HasSubtype	VariableType	OptionSetType	Défini en 7.19

7.5 ServerVendorCapabilityType

Ce *VariableType* est un type abstrait dont les sous-types définissent des fonctions du *Serveur*. Les fournisseurs peuvent définir des sous-types de ce type. Ce *VariableType* est défini de façon formelle dans le Tableau 64.

Tableau 64 – Définition de ServerVendorCapabilityType

Attribut	Valeur				
BrowseName	ServerVendorCapabilityType				
IsAbstract	True				
ValueRank	−1 (−1 = Scalar)				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4					

7.6 DataTypeDictionaryType

Les *DataTypeDictionaries* sont définis dans l'IEC 62541-3. Ce *VariableType* est utilisé comme étant le type pour les *DataTypeDictionaries*. Il n'y a pas de *Références* spécifiées pour ce *VariableType*. Il est défini de façon formelle dans le Tableau 65.

Tableau 65 – Définition de DataTypeDictionaryType

Attribut	Valeur				
BrowseName	DataTypeDictionaryType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	ByteString				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.					
HasProperty	Variable	DataTypeVersion	String	PropertyType	Facultatif
HasProperty	Variable	NamespaceUri	String	PropertyType	Facultatif

La *Propriété* DataTypeVersion est définie dans l'IEC 62541-3. Le NamespaceUri est l'URI pour l'espace de nom décrit par l'*Attribut Value* du DataTypeDictionary.

7.7 DataTypeDescriptionType

Les *DataTypeDescriptions* sont définies dans l'IEC 62541-3. Ce *VariableType* est utilisé comme étant le type pour les *DataTypeDescriptions*. Il n'y a pas de *Références* spécifiées pour ce *VariableType*. Il est défini de façon formelle dans le Tableau 66.

Tableau 66 – Définition de DataTypeDescriptionType

Attribut	Valeur				
BrowseName	DataTypeDescriptionType				
IsAbstract	False				
ValueRank	-1 (-1 = Scalar)				
DataType	ByteString				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.					
HasProperty	Variable	DataTypeVersion	String	PropertyType	Facultatif
HasProperty	Variable	DictionaryFragment	ByteString	PropertyType	Facultatif

Les *Propriétés* DataTypeVersion et DictionaryFragment sont définies dans l'IEC 62541-3.

7.8 ServerStatusType

Ce *VariableType* complexe est utilisé pour des informations relatives au statut du *Serveur*. Ses *DataVariables* reflètent son *DataType* ayant la même sémantique définie en 12.10. Le *VariableType* est défini de façon formelle dans le Tableau 67.

Tableau 67 – Définition de ServerStatusType

Attribut	Valeur				
BrowseName	ServerStatusType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	ServerStatusDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.					
HasComponent	Variable	StartTime	UTCTime	BaseDataVariableType	Obligatoire
HasComponent	Variable	CurrentTime	UTCTime	BaseDataVariableType	Obligatoire
HasComponent	Variable	State	ServerState	BaseDataVariableType	Obligatoire
HasComponent	Variable	BuildInfo[1]	BuildInfo	BuildInfoType	Obligatoire
HasComponent	Variable	SecondsTillShutdown	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	ShutdownReason	LocalizedText	BaseDataVariableType	Obligatoire
NOTE Les *Objets* et *Variables* conteneurs de ces *Objets* et *Variables* sont définis par leur *BrowseName* défini dans le *TypeDefinitionNode* correspondant. Le *NodeId* est défini par le nom symbolique composé décrit en 4.1.					

7.9 BuildInfoType

Ce *VariableType* complexe est utilisé pour des informations relatives au statut du *Serveur*. Ses *DataVariables* reflètent son *DataType* ayant la même sémantique définie en 12.4. Le *VariableType* est défini de façon formelle dans le Tableau 68.

Tableau 68 – Définition de BuildInfoType

Attribut	**Valeur**				
BrowseName	BuildInfoType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	BuildInfo				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.					
HasComponent	Variable	ProductUri	String	BaseDataVariableType	Obligatoire
HasComponent	Variable	ManufacturerName	String	BaseDataVariableType	Obligatoire
HasComponent	Variable	ProductName	String	BaseDataVariableType	Obligatoire
HasComponent	Variable	SoftwareVersion	String	BaseDataVariableType	Obligatoire
HasComponent	Variable	BuildNumber	String	BaseDataVariableType	Obligatoire
HasComponent	Variable	BuildDate	UTCTime	BaseDataVariableType	Obligatoire

7.10 ServerDiagnosticsSummaryType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Ses *DataVariables* reflètent son *DataType* ayant la même sémantique définie en 12.9. Le *VariableType* est défini de façon formelle dans le Tableau 69.

Tableau 69 – Définition de ServerDiagnosticsSummaryType

Attribut	**Valeur**				
BrowseName	ServerDiagnosticsSummaryType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	ServerDiagnosticsSummaryDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.					
HasComponent	Variable	ServerViewCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	CurrentSessionCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	CumulatedSessionCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	SecurityRejectedSessionCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	RejectedSessionCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	SessionTimeoutCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	SessionAbortCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	PublishingIntervalCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	CurrentSubscriptionCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	CumulatedSubscriptionCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	SecurityRejectedRequestsCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	RejectedRequestsCount	UInt32	BaseDataVariableType	Obligatoire

7.11 SamplingIntervalDiagnosticsArrayType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Pour chaque entrée de la matrice, les instances de ce type fournissent une *Variable* du *VariableType* SamplingIntervalDiagnosticsType ayant la fréquence d'échantillonnage comme *BrowseName*. Le *VariableType* est défini de façon formelle dans le Tableau 70.

Tableau 70 – Définition de SamplingIntervalDiagnosticsArrayType

Attribut		Valeur		
BrowseName		SamplingIntervalDiagnosticsArrayType		
IsAbstract		False		
ValueRank		1 (1 = OneDimension)		
ArrayDimensions		{0} (0 = UnknownSize)		
DataType		SamplingIntervalDiagnosticsDataType		
Références	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 7.4.				
HasComponent	VariableType	SamplingIntervalDiagnostics	SamplingIntervalDiagnosticsDataType SamplingIntervalDiagnosticsType	ExposesIt sArray

7.12 SamplingIntervalDiagnosticsType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Ses *DataVariables* reflètent son *DataType* ayant la même sémantique définie en 12.8. Le *VariableType* est défini de façon formelle dans le Tableau 71.

Tableau 71 – Définition de SamplingIntervalDiagnosticsType

Attribut	Valeur				
BrowseName	SamplingIntervalDiagnosticsType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	SamplingIntervalDiagnosticsDataType				
Références	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Sous-type du BaseDataVariableType défini en 7.4.					
HasComponent	Variable	SamplingInterval	Duration	BaseDataVariableType	Obligatoire
HasComponent	Variable	SampledMonitoredItemsCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	MaxSampledMonitoredItemsCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	DisabledMonitoredItemsSamplingCount	UInt32	BaseDataVariableType	Obligatoire

7.13 SubscriptionDiagnosticsArrayType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Pour chaque entrée de la matrice, les instances de ce type fournissent une *Variable* du *VariableType* SubscriptionDiagnosticsType ayant le SubscriptionId comme *BrowseName*. Le *VariableType* est défini de façon formelle dans le Tableau 72.

Tableau 72– Définition de SubscriptionDiagnosticsArrayType

Attribut		Valeur		
BrowseName		SubscriptionDiagnosticsArrayType		
IsAbstract		False		
ValueRank		1 (1 = OneDimension)		
ArrayDimensions		{0} (0 = UnknownSize)		
DataType		SubscriptionDiagnosticsDataType		
Références	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.				
HasComponent	VariableType	SubscriptionDiagnostics	SubscriptionDiagnosticsDataType SubscriptionDiagnosticsType	ExposesItsArray

7.14 SubscriptionDiagnosticsType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Ses *DataVariables* reflètent son *DataType* ayant la même sémantique définie en 12.15. Le *VariableType* est défini de façon formelle dans le Tableau 73.

Tableau 73 – Définition de SubscriptionDiagnosticsType

Attribut	Valeur				
BrowseName	SubscriptionDiagnosticsType				
IsAbstract	False				
ValueRank	−1 (−1 = Scalar)				
DataType	SubscriptionDiagnosticsDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.					
HasComponent	Variable	SessionId	NodeId	BaseDataVariableType	Obligatoire
HasComponent	Variable	SubscriptionId	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	Priority	Byte	BaseDataVariableType	Obligatoire
HasComponent	Variable	PublishingInterval	Duration	BaseDataVariableType	Obligatoire
HasComponent	Variable	MaxKeepAliveCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	MaxLifetimeCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	MaxNotificationsPerPublish	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	PublishingEnabled	Boolean	BaseDataVariableType	Obligatoire
HasComponent	Variable	ModifyCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	EnableCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	DisableCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	RepublishRequestCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	RepublishMessageRequestCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	RepublishMessageCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	TransferRequestCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	TransferredToAltClientCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	TransferredToSameClientCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	PublishRequestCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	DataChangeNotificationsCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	EventNotificationsCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	NotificationsCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	LatePublishRequestCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	CurrentKeepAliveCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	CurrentLifetimeCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	UnacknowledgedMessageCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	DiscardedMessageCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	MonitoredItemCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	DisabledMonitoredItemCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	MonitoringQueueOverflowCount	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	NextSequenceNumber	UInt32	BaseDataVariableType	Obligatoire
HasComponent	Variable	EventQueueOverflowCount	UInt32	BaseDataVariableType	Obligatoire

7.15 SessionDiagnosticsArrayType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Pour chaque entrée de la matrice, les instances de ce type fournissent une *Variable* du *VariableType* SessionDiagnosticsVariableType ayant le SessionDiagnostics comme *BrowseName*. Ces *Variables* sont également référencées par les *Objets* SessionDiagnostics définis par leur type en 6.3.5. Le *VariableType* est défini de façon formelle dans le Tableau 74.

Tableau 74 – Définition de SessionDiagnosticsArrayType

Attribut		Valeur		
BrowseName		SessionDiagnosticsArrayType		
IsAbstract		False		
ValueRank		1 (1 = OneDimension)		
ArrayDimensions		{0} (0 = UnknownSize)		
DataType		SessionDiagnosticsDataType		
Références	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.				
HasComponent	Variable	SessionDiagnostics	SessionDiagnosticsDataType SessionDiagnosticsVariableType	ExposesItsArray

7.16 SessionDiagnosticsVariableType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Ses *DataVariables* reflètent son *DataType* ayant la même sémantique définie en 12.11. Le *VariableType* est défini de façon formelle dans le Tableau 75.

Tableau 75 – Définition de SessionDiagnosticsVariableType

Attribut	**Valeur**			
BrowseName	SessionDiagnosticsVariableType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionDiagnosticsDataType			
Références	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.				
HasComponent	Variable	SessionId	NodeId BaseDataVariableType	Obligatoire
HasComponent	Variable	SessionName	String BaseDataVariableType	Obligatoire
HasComponent	Variable	ClientDescription	ApplicationDescription BaseDataVariableType	Obligatoire
HasComponent	Variable	ServerUri	String BaseDataVariableType	Obligatoire
HasComponent	Variable	EndpointUrl	String BaseDataVariableType	Obligatoire
HasComponent	Variable	LocaleIds	LocaleId[] BaseDataVariableType	Obligatoire
HasComponent	Variable	MaxResponseMessageSize	UInt32 BaseDataVariableType	Obligatoire
HasComponent	Variable	ActualSessionTimeout	Duration BaseDataVariableType	Obligatoire
HasComponent	Variable	ClientConnectionTime	UTCTime BaseDataVariableType	Obligatoire
HasComponent	Variable	ClientLastContactTime	UTCTime BaseDataVariableType	Obligatoire
HasComponent	Variable	CurrentSubscriptionsCount	UInt32 BaseDataVariableType	Obligatoire
HasComponent	Variable	CurrentMonitoredItemsCount	UInt32 BaseDataVariableType	Obligatoire
HasComponent	Variable	CurrentPublishRequestsInQueue	UInt32 BaseDataVariableType	Obligatoire
HasComponent	Variable	TotalRequestsCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	UnauthorizedRequestsCount	UInt32 BaseDataVariableType	Obligatoire
HasComponent	Variable	ReadCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	HistoryReadCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	WriteCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	HistoryUpdateCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	CallCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	CreateMonitoredItemsCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	ModifyMonitoredItemsCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	SetMonitoringModeCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	SetTriggeringCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	DeleteMonitoredItemsCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	CreateSubscriptionCount	ServiceCounterDataType	Obligatoire

Attribut	Valeur			
BrowseName	SessionDiagnosticsVariableType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionDiagnosticsDataType			
Références	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.				
			BaseDataVariableType	
HasComponent	Variable	ModifySubscriptionCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	SetPublishingModeCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	PublishCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	RepublishCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	TransferSubscriptionsCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	DeleteSubscriptionsCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	AddNodesCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	AddReferencesCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	DeleteNodesCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	DeleteReferencesCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	BrowseCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	BrowseNextCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	TranslateBrowsePathsToNodeIds Count	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	QueryFirstCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	QueryNextCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	RegisterNodesCount	ServiceCounterDataType BaseDataVariableType	Obligatoire
HasComponent	Variable	UnregisterNodesCount	ServiceCounterDataType BaseDataVariableType	Obligatoire

7.17 SessionSecurityDiagnosticsArrayType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Pour chaque entrée de la matrice, les instances de ce type fournissent une *Variable* du *VariableType* SessionSecurityDiagnosticsType ayant le SessionSecurityDiagnosticsType comme *BrowseName*. Ces *Variables* sont également référencées par les *Objets* SessionDiagnostics définis par leur type en 6.3.5. Le *VariableType* est défini de façon formelle dans le Tableau 76. Ces informations étant relatives à la sécurité, il convient de les rendre accessibles seulement aux utilisateurs autorisés et non à tous.

Tableau 76 – Définition de SessionSecurityDiagnosticsArrayType

Attribut	Valeur			
BrowseName	SessionSecurityDiagnosticsArrayType			
IsAbstract	False			
ValueRank	1 (1 = OneDimension)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	SessionSecurityDiagnosticsDataType			
Références	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4.				
HasComponent	Variable	SessionSecurityDiagnostics	SessionSecurityDiagnosticsDataType SessionSecurityDiagnosticsType	ExposesItsArray

7.18 SessionSecurityDiagnosticsType

Ce *VariableType* complexe est utilisé pour les informations de diagnostic. Ses *DataVariables* reflètent son *DataType* ayant la même sémantique définie en 12.12. Le *VariableType* est défini de façon formelle dans le Tableau 77. Ces informations étant relatives à la sécurité, il convient de les rendre accessibles seulement aux utilisateurs autorisés et non à tous.

Tableau 77 – Définition de SessionSecurityDiagnosticsType

Attribut	Valeur			
BrowseName	SessionSecurityDiagnosticsType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
DataType	SessionSecurityDiagnosticsDataType			
Références	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4				
HasComponent	Variable	SessionId	NodeId BaseDataVariableType	Obligatoire
HasComponent	Variable	ClientUserIdOfSession	String BaseDataVariableType	Obligatoire
HasComponent	Variable	ClientUserIdHistory	String[] BaseDataVariableType	Obligatoire
HasComponent	Variable	AuthenticationMechanism	String BaseDataVariableType	Obligatoire
HasComponent	Variable	Encoding	String BaseDataVariableType	Obligatoire
HasComponent	Variable	TransportProtocol	String BaseDataVariableType	Obligatoire
HasComponent	Variable	SecurityMode	MessageSecurityMode BaseDataVariableType	Obligatoire
HasComponent	Variable	SecurityPolicyUri	String BaseDataVariableType	Obligatoire
HasComponent	Variable	ClientCertificate	ByteString BaseDataVariableType	Obligatoire

7.19 OptionSetType

Le *VariableType OptionSetType* sert à représenter un masque de bits et la *Propriété OptionSetValues* contient la représentation lisible en clair pour chaque bit du masque de bits mis à true (vrai). L'ordre des bits du masque de bits pointe vers une position de la matrice, c'est-à-dire que le premier bit (bit de poids faible) est dirigé vers la première entrée dans la matrice, etc.

Le *DataType* de ce *VariableType* doit être capable de représenter un masque de bits. Il doit être soit un *DataType* représentant un entier signé ou non signé, soit une *ByteString*. Par exemple, il peut être le *BitFieldMaskDataType.*

La *Propriété* BitMask *Property* facultative fournit le masque de bits dans une matrice de Booléens. Ceci permet de s'abonner à des entrées individuelles du masque de bits. L'ordre des bits du masque de bits pointe vers une position de la matrice, c'est-à-dire que le premier bit est dirigé vers la première entrée dans la matrice, etc.

La matrice OptionSetValues contient un LocalizedText vide pour chaque bit n'ayant pas de signification particulière. Le *VariableType* est défini de façon formelle dans le Tableau 76.

Tableau 78 – Définition de OptionSetType

Attribut	**Valeur**			
BrowseName	OptionSetType			
IsAbstract	False			
ValueRank	−1 (−1 = Scalar)			
ArrayDimensions	{0} (0 = UnknownSize)			
DataType	BaseDataType			
Références	**NodeClass**	**BrowseName**	**DataType TypeDefinition**	**ModellingRule**
Sous-type du BaseDataVariableType défini en 7.4				
HasProperty	Variable	OptionSetValues	LocalizedText[] PropertyType	Obligatoire
HasProperty	Variable	BitMask	Boolean[] PropertyType	Facultatif

8 Objets normalisés et leurs Variables

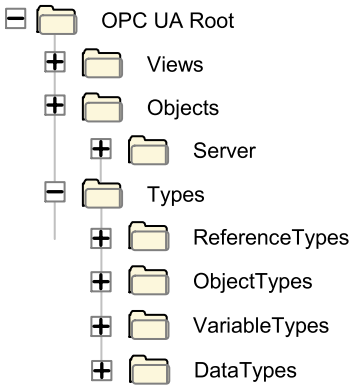
8.1 Généralités

Les *Objets* et *Variables* décrits dans les paragraphes ci-après peuvent être étendus par des *Propriétés* supplémentaires ou des *Références* à d'autres *Nœuds*, sauf lorsqu'il est énoncé dans le texte que ceci est limité.

8.2 Objets utilisés pour organiser la structure de l'Espace d'Adresses

8.2.1 Vue d'ensemble

Aux fins de favoriser l'interopérabilité des clients et des *Serveurs*, l'*Espace d'Adresses* OPC UA est structuré sous la forme d'une hiérarchie, les niveaux supérieurs étant normalisés pour tous les *Serveurs*. La Figure 1 illustre la structure de l'*Espace d'Adresses*. Tous les *Objets* dans cette figure sont organisés à l'aide de *Références Organizes* et ont l'*ObjectType FolderType* comme définition de type.



Anglais	Français
OPC UA Root	Racine OPC UA
Views	Vues
Objects	Objets
Server	Serveur

Figure 1 – Structure normalisée de l'Espace d'Adresses

Le reste fournit des descriptions de ces *Nœuds* normalisés et l'organisation des *Nœuds* en dessous d'eux. Typiquement, les *Serveurs* mettent en œuvre un sous-ensemble de ces *Nœuds* normalisés, en fonction de leurs capacités.

8.2.2 Racine (Root)

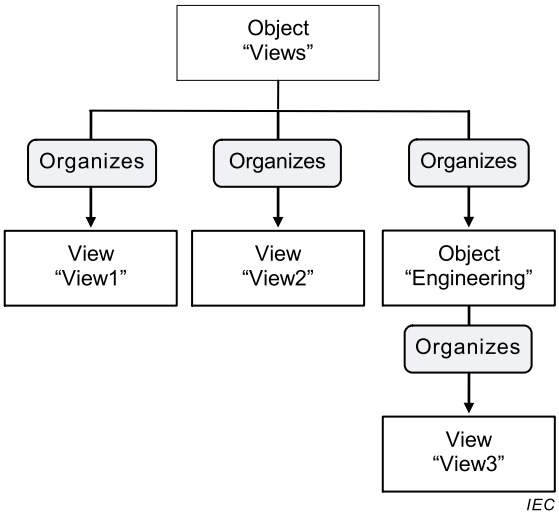
Cet *Objet* normalisé est le point d'entrée de navigation pour l'*Espace d'Adresses*. Il contient un jeu de *Références Organizes* qui pointent vers d'autres *Objets* normalisés. L'*Objet* "*Root*" ne doit pas référencer d'autres *NodeClasses*. Il est défini de façon formelle dans le Tableau 79.

Tableau 79 – Définition de Root

Attribut	**Valeur**		
BrowseName	Root		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	Object	Views	Défini en 8.2.3
Organizes	Object	Objets	Défini en 8.2.4
Organizes	Object	Types	Défini en 8.2.5

8.2.3 Views (Vues)

Cet *Objet* normalisé est le point d'entrée de navigation pour les *Views*. Seules les *Références Organizes* sont utilisées pour relier des *Nœuds View* à l'*Objet* normalisé "*Views*". Tous les *Nœuds View* dans l'*Espace d'Adresses* doivent être référencés par ce *Nœud*, directement ou indirectement. C'est-à-dire que l'*Objet* "*Views*" peut référencer d'autres *Objets* en utilisant les *Références Organizes*. Ces *Objets* peuvent référencer des *Views* supplémentaires. La Figure 2 illustre l'organisation des Views. L'*Objet* normalisé "*Views"* référence directement les *Views* "View1" et "View2" et indirectement "View3" en référençant un autre *Objet* appelé "Engineering" (Etude).



Anglais	Français
Object	Objet
View	Vue
Organizes	Organise

Figure 2 – Organisation des Views

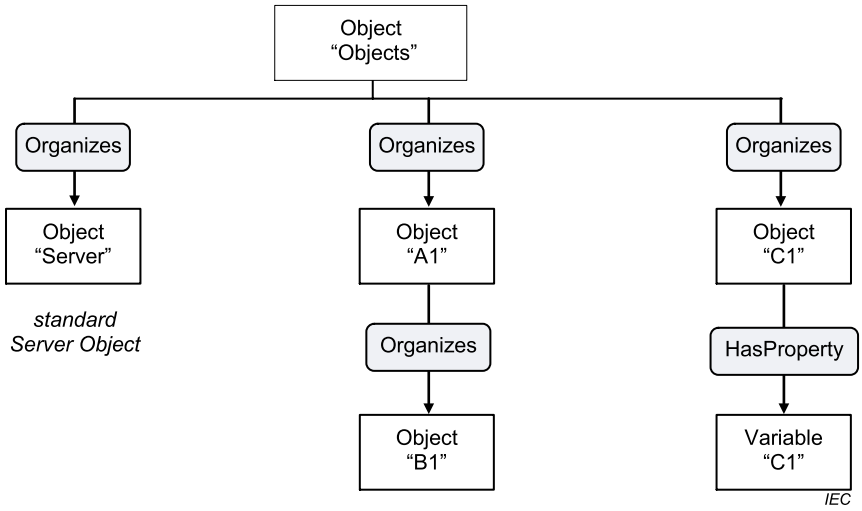
L'*Objet* “*Views*” ne doit pas référencer d'autres *NodeClasses*. L'*Objet* “*Views*” est défini de façon formelle dans le Tableau 80.

Tableau 80 – Définition de Views

Attribut	**Valeur**		
BrowseName	Views		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6

8.2.4 Objects (Objets)

Cet *Objet* normalisé est le point d'entrée de navigation pour le *Nœud Object*. La Figure 3 illustre la structure sous ce *Nœud*. Seules les *Références Organizes* sont utilisées pour relier des *Objets* à l'*Objet* normalisé “*Objects*”. Un *Nœud View* peut être utilisé comme point d'entrée dans un sous-ensemble de l'*Espace d'Adresses* contenant des *Objets* et des *Variables* et, ainsi, l'*Objet* “*Objects*“ peut aussi référencer des *Nœuds View* en utilisant des *Références Organizes*. L'intention de l'*Objet* “*Objects*” est que tous les *Objets* et *Variables* qui ne sont pas utilisés pour des définitions de type ou à d'autres fins organisationnelles (par exemple, organisation des *Views*) soient accessibles par le biais de *Références hiérarchiques* en commençant à partir de ce *Nœud*. Cependant, elle ne constitue pas une exigence, car tous les *Serveurs* peuvent ne pas être forcément capables de la prendre en charge. Cet *Objet* référence l'*Objet Server* normalisé défini en 8.3.2.



Anglais	Français
Object	Objet
Organizes	Organise
Standard Server Object	Objet Server normalisé

Figure 3 – Organisation des Objects

L'*Objet* “*Objects*” ne doit pas référencer d'autres *NodeClasses*. L'*Objet* “*Objects* est défini de façon formelle dans le Tableau 81.

Tableau 81 – Définition de Objects

Attribut	**Valeur**		
BrowseName	Objets		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	Object	Server;Server	Défini en 8.3.2

8.2.5 Types

Cet *Objet Node* normalisé est le point d'entrée de navigation pour le type *Nodes (Nœuds)*. La Figure 1 illustre la structure sous ce *Nœud*. Seules les *Références Organizes* sont utilisées pour relier des *Objets* à l'*Objet* normalisé “*Types*”. L'*Objet* “*Types*” ne doit pas référencer d'autres *NodeClasses*. Il est défini de façon formelle dans le Tableau 82.

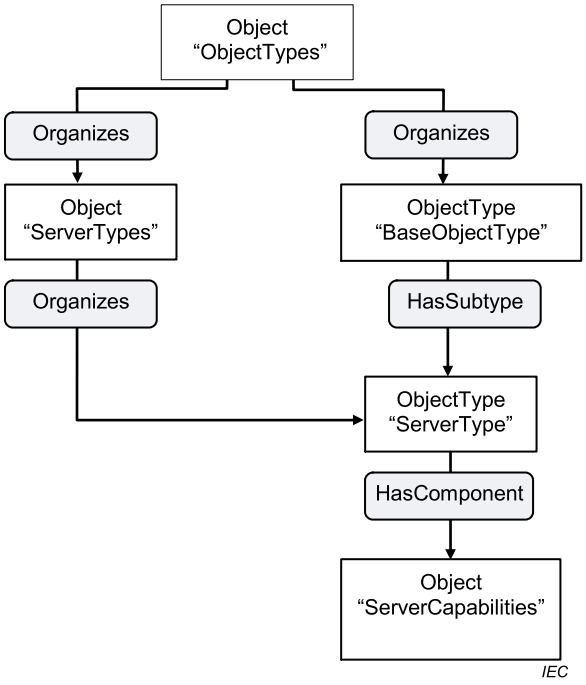
Tableau 82 – Définition de Types

Attribut	**Valeur**		
BrowseName	Types		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	Object	ObjectTypes	Défini en 8.2.6
Organizes	Object	VariableTypes	Défini en 8.2.7
Organizes	Object	ReferenceTypes	Défini en 8.2.8
Organizes	Object	DataTypes	Défini en 8.2.9
Organizes	Object	EventType	Défini en 8.2.9

8.2.6 ObjectTypes

Cet *Objet Node* normalisé est le point d'entrée de navigation pour l'*ObjectType Nodes*. La Figure 4 illustre la structure sous ce *Nœud* en montrant certains des *ObjectTypes* normalisés

définis à l'Article 6. Seules les *Références Organizes* sont utilisées pour relier des *Objets* et *ObjectTypes* à l'*Objet* normalisé "*ObjectTypes*". L'*Objet* "*ObjectTypes*" ne doit pas référencer d'autres *NodeClasses*.



Anglais	Français
Object	Objet
Organizes	Organise

Figure 4 – Organisation des ObjectTypes

L'intention de l'*Objet* "*ObjectTypes*" est que tous les *ObjectTypes* du *Serveur* soient directement ou indirectement accessibles en parcourant les *HierarchicalReferences* en commençant par ce *Nœud*. Cependant, cela n'est pas exigé et des *Serveurs* peuvent ne pas fournir certains de leurs *ObjectTypes,* car ils peuvent être notoirement connus dans la profession, par exemple le *ServerType* défini en 6.3.1.

Cet *Objet* référence également indirectement le *BaseEventType* défini en 6.4.2, qui est le type de base de tous les *EventTypes*. De ce fait, il est le point d'entrée pour tous les *EventTypes* fournis par le *Serveur*. Il est exigé que le *Serveur* présente tous ses *EventTypes* et un client peut ainsi s'abonner utilement à des *Événements*.

L'*Objet* "*ObjectTypes*" est défini de façon formelle dans le Tableau 83.

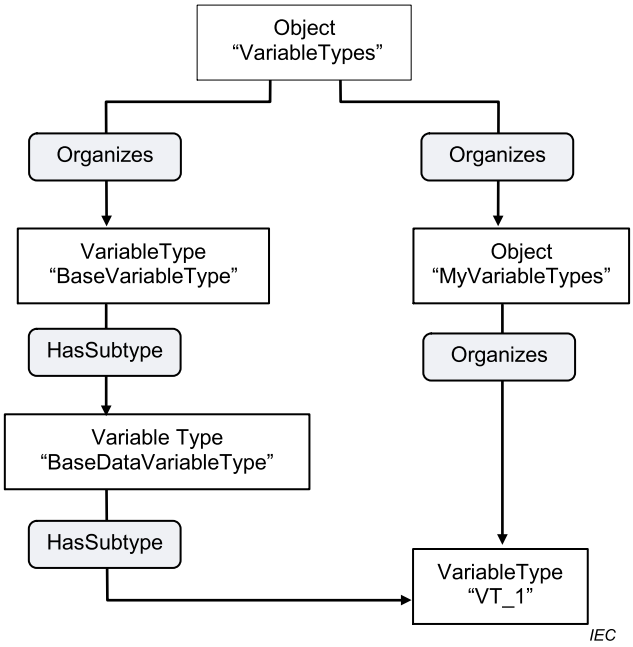
Tableau 83 – Définition de ObjectTypes

Attribut	**Valeur**		
BrowseName	ObjectTypes		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	ObjectType	BaseObjectType	Défini en 6.2

8.2.7 VariableTypes

Cet *Objet* normalisé est le point d'entrée de navigation pour les *Nodes VariableType*. La Figure 5 illustre la structure sous ce *Nœud*. Seules les *Références Organizes* sont utilisées

pour relier des *Objets* et *VariableTypes* à l'*Objet* normalisé "*VariableTypes*". L'*Objet* "*VariableTypes*" ne doit pas référencer d'autres *NodeClasses*.



Anglais	Français
Object	Objet
Organizes	Organise

Figure 5 – Organisation des VariableTypes

L'intention de l'*Objet* "*VariableTypes*" est que tous les *VariableTypes* du *Serveur* soient directement ou indirectement accessibles en parcourant les *HierarchicalReferences* en commençant par ce *Nœud*. Cependant, cela n'est pas exigé et des *Serveurs* peuvent ne pas fournir certains de leurs *VariableTypes,* car ils peuvent être notoirement connus dans la profession, par exemple le "*BaseVariableType"* défini en 7.2.

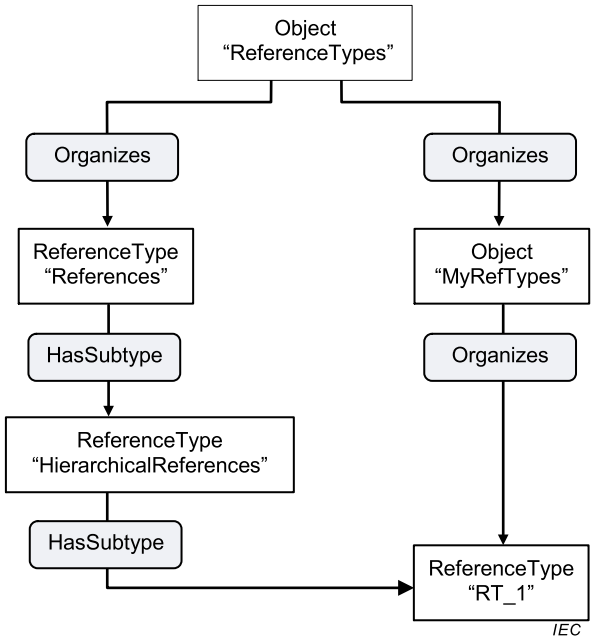
L'*Objet* "*VariableTypes*" est défini de façon formelle dans le Tableau 84.

Tableau 84 – Définition de VariableTypes

Attribut	**Valeur**		
BrowseName	VariableTypes		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	VariableType	BaseVariableType	Défini en 7.2

8.2.8 ReferenceTypes

Cet *Objet* normalisé est le point d'entrée de navigation pour les *Nodes ReferenceType*. La Figure 6 illustre l'organisation des *ReferenceTypes*. Les *Références Organizes* sont utilisées pour définir des *ReferenceTypes* et des *Objects* référencés par l'*Objet* "*ReferenceTypes*". L'*Objet* "*ReferenceTypes*" ne doit pas référencer d'autres *NodeClasses*. Voir l'Article 11 pour une discussion portant sur les *ReferenceTypes* normalisés qui apparaissent sous l'*Objet* "*ReferenceTypes*".



Anglais	Français
Object	Objet
Organizes	Organise

Figure 6 – Définitions de ReferenceType

Sachant que des *ReferenceTypes* sont utilisés comme filtres dans le *Service* de navigation et dans des interrogations, le *Serveur* doit fournir tous ses *ReferenceTypes*, directement ou indirectement en suivant les *Références hiérarchiques* en commençant à partir de l'*Objet* “*ReferenceTypes*”. Cela signifie que, chaque fois qu'un client suit une *Référence*, le *Serveur* doit présenter le type de cette Référence dans la hiérarchie des *ReferenceTypes*. Il doit fournir tous les *ReferenceTypes* afin que le client puisse, en suivant le sous-type inverse de *Références*, arriver aux *Références* de base *ReferenceType*. Cela ne signifie pas que le *Serveur* doit présenter les *ReferenceTypes* dont le client n'a utilisé aucune *Référence*.

L'*Objet* “*ReferenceTypes*” est défini de façon formelle dans le Tableau 85.

Tableau 85 – Définition de ReferenceTypes

Attribut	**Valeur**		
BrowseName	ReferenceTypes		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	ReferenceType	References	Défini en 11.1

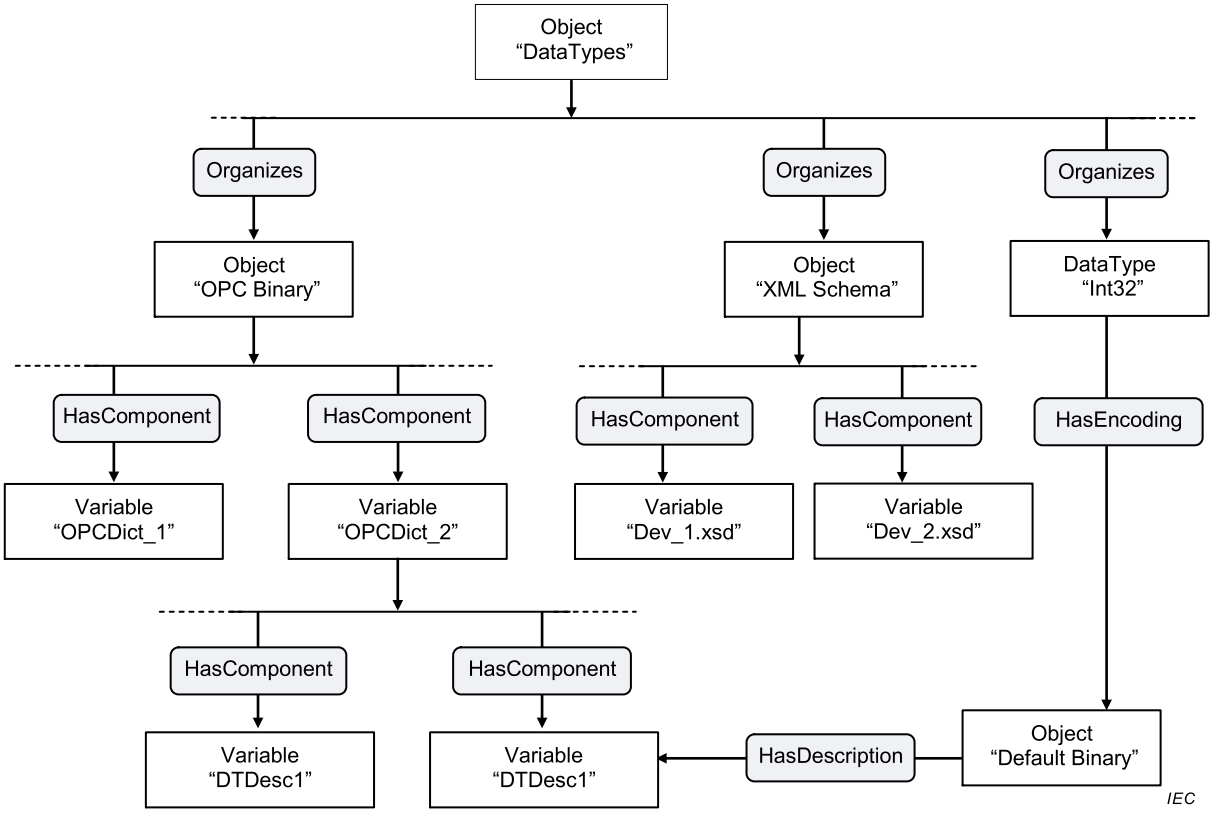
8.2.9 DataTypes

Cet *Objet* normalisé est le point d'entrée de navigation pour les *DataTypes* que le *Serveur* souhaite présenter dans l'*Espace d'Adresses*. L'*Objet* normalisé utilise des *Références Organizes* pour référencer des *Objets* du *DataTypeSystemType* représentant les *DataTypeSystems*. Sont référencés par ces *Objets* les *DataTypeDictionaries* qui se réfèrent à leurs *DataTypeDescriptions*. Cependant, il n'est pas exigé de fournir les *Objets DataTypeSystem* et il n'est également pas nécessaire de fournir le *DataTypeDictionary*.

Sachant que les *DataTypes* ne sont pas reliés aux *DataTypeDescriptions* en utilisant des *Références hiérarchiques*, il convient de rendre les *Nœuds DataType* disponibles en utilisant des *Références Organizes* pointant directement de l'*Objet* “DataTypes” vers les *Nœuds DataType* ou en utilisant des *Objets Folder* supplémentaires à des fins de regroupement.

L'intention est que tous les *DataTypes* du *Serveur* présentés dans l'*Espace d'Adresses* soient accessibles en suivant des *Références hiérarchiques* en commençant à partir de l'*Objet* "DataTypes". Cependant, cela n'est pas exigé.

La Figure 7 illustre la hiérarchie utilisant les *DataTypeSystems* normalisés "OPC Binary" et "XML Schema" comme exemples. D'autres *DataTypeSystems* peuvent être définis sous cet *Objet*.



Anglais	Français
Object	Objet
Organizes	Organise

Figure 7 – Organisation des DataTypes

Chaque *Objet DataTypeSystem* est relié à ses *Nœuds DataTypeDictionary* en utilisant des *Références HasComponent*. Chaque *Nœud DataTypeDictionary* est relié à ses *Nœuds DataTypeDescription* en utilisant des *Références HasComponent*. Ces *Références* indiquent que les *DataTypeDescriptions* sont définies dans le dictionnaire.

Dans l'exemple, l'*Objet* "DataTypes" référence le *DataType* "Int32" en utilisant une *Référence Organizes*. Le *DataType* utilise la *Référence* non hiérarchique *HasEncoding* pour pointer vers son codage par défaut, qui référence une *DataTypeDescription* en utilisant la *Référence* non hiérarchique *HasDescription*.

L'*Objet* "*DataTypes*" est défini de façon formelle dans le Tableau 86.

Tableau 86 – Définition de DataTypes

Attribut	Valeur		
BrowseName	DataTypes		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	Object	OPC Binary	Défini en 8.2.10
Organizes	Object	XML Schema	Défini en 8.2.11
Organizes	DataType	BaseDataType	Défini en 12.2

8.2.10 OPC Binary (OPC Binaire)

OPC Binary est un *DataTypeSystem* normalisé défini par OPC. Il est représenté dans l'*Espace d'Adresses* par un *Objet Node*. Le *DataTypeSystem* OPC Binary est défini dans l'IEC 62541-3. OPC Binary utilise XML pour décrire des valeurs de données binaires complexes. L'*Objet* "*OPC Binary*" est défini de façon formelle dans le Tableau 87.

Tableau 87 – Définition de OPC Binary

Attribut	Valeur		
BrowseName	OPC Binary		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	DataTypeSystemType	Défini en 6.8

8.2.11 XML Schema (Schéma XML)

XML Schema est un *DataTypeSystem* normalisé défini par le W3C. Il est représenté dans l'*Espace d'Adresses* par un *Objet Node*. Les documents du XML Schema sont des documents XML dont l'attribut xmlns dans la première ligne est:

schema xmlns =http://www.w3.org/1999/XMLSchema

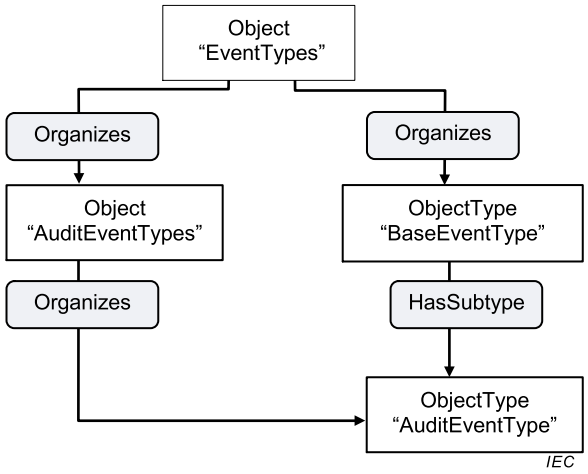
L'*Objet* "*XML Schema*" est défini de façon formelle dans le Tableau 88.

Tableau 88 – Définition de XML Schema

Attribut	Valeur		
BrowseName	XML Schema		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	DataTypeSystemType	Défini en 6.8

8.2.12 EventTypes

Cet *Objet Node* normalisé est le point d'entrée de navigation pour les *Nodes EventType* La Figure 8 illustre la structure sous ce *Nœud* en montrant certains des *EventTypes* normalisés définis à l'Article 6. Seules les *Références Organizes* sont utilisées pour relier des *Objets* et *ObjectTypes* à l'*Objet* normalisé "*EventTypes*". L'*Objet* "*EventTypes*" ne doit pas référencer d'autres *NodeClasses*.



Anglais	Français
Object	Objet
Organizes	Organise

Figure 8 – Organisation des EventTypes

L'intention de l'*Objet* “*EventTypes*” est que tous les *EventTypes* du *Serveur* soient directement ou indirectement accessibles en parcourant les *HierarchicalReferences* en commençant par ce *Nœud*. Il est exigé que le *Serveur* présente tous ses *EventTypes* et un client peut ainsi s'abonner utilement à des *Événements*.

L'*Objet* “*EventTypes*” est défini de façon formelle dans le Tableau 89.

Tableau 89 – Définition de EventTypes

Attribut	**Valeur**		
BrowseName	ObjectTypes		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	FolderType	Défini en 6.6
Organizes	ObjectType	BaseEventType	Défini en 6.4.2

8.3 Objet Server et ses objets contenants

8.3.1 Généralités

L'*Objet Server* et ses *Objets* et *Variables* contenants sont construits de manière à obtenir de l'information de plusieurs façons adaptées à des catégories de clients ayant des exigences différentes. L'Annexe A donne une vue d'ensemble des décisions de conception prises pour fournir de l'information de cette manière, en présentant les avantages et les inconvénients des différentes approches. La Figure 9 donne une vue d'ensemble des *Objets* et *Variables* contenants des informations de diagnostic de l'*Objet* Server où l'information peut être trouvée.

L'*Objet* SessionsDiagnosticsSummary contient un *Objet* par session et une *Variable* avec une matrice à une entrée par session. Cette matrice est d'un *DataType* complexe contenant les informations de diagnostic relatives à la session. Chaque *Objet* représentant une session référence une *Variable* complexe contenant de l'information relative à la session en utilisant le même DataType que la matrice contenant de l'information relative à toutes les sessions. Une telle *Variable* présente également toutes ses informations comme *Variables* avec des *DataTypes* simples contenant les mêmes informations que le *DataType* complexe. Ne sont pas représentées sur la Figure 9, les informations liées à la sécurité de la session qui suivent les mêmes règles.

Le *Serveur* fournit une matrice avec une entrée par abonnement contenant des informations de diagnostic relatives à cet abonnement. Chaque entrée de cette matrice est également présentée comme *Variable* complexe avec des *Variables* pour chaque valeur individuelle. Chaque *Objet* représentant une session fournit également une telle matrice, qui fournit les abonnements de la session.

Les matrices contenant de l'information relative aux sessions ou aux abonnements peuvent avoir des longueurs différentes pour des connexions différentes avec des authentifiants d'utilisateur différents, car tous les utilisateurs ne peuvent pas voir toutes les entrées de la matrice. Cela signifie aussi que la longueur de la matrice peut changer si l'utilisateur est masqué. Par conséquent, les clients qui s'abonnent à une plage d'indices spécifique peuvent avoir des résultats inattendus.

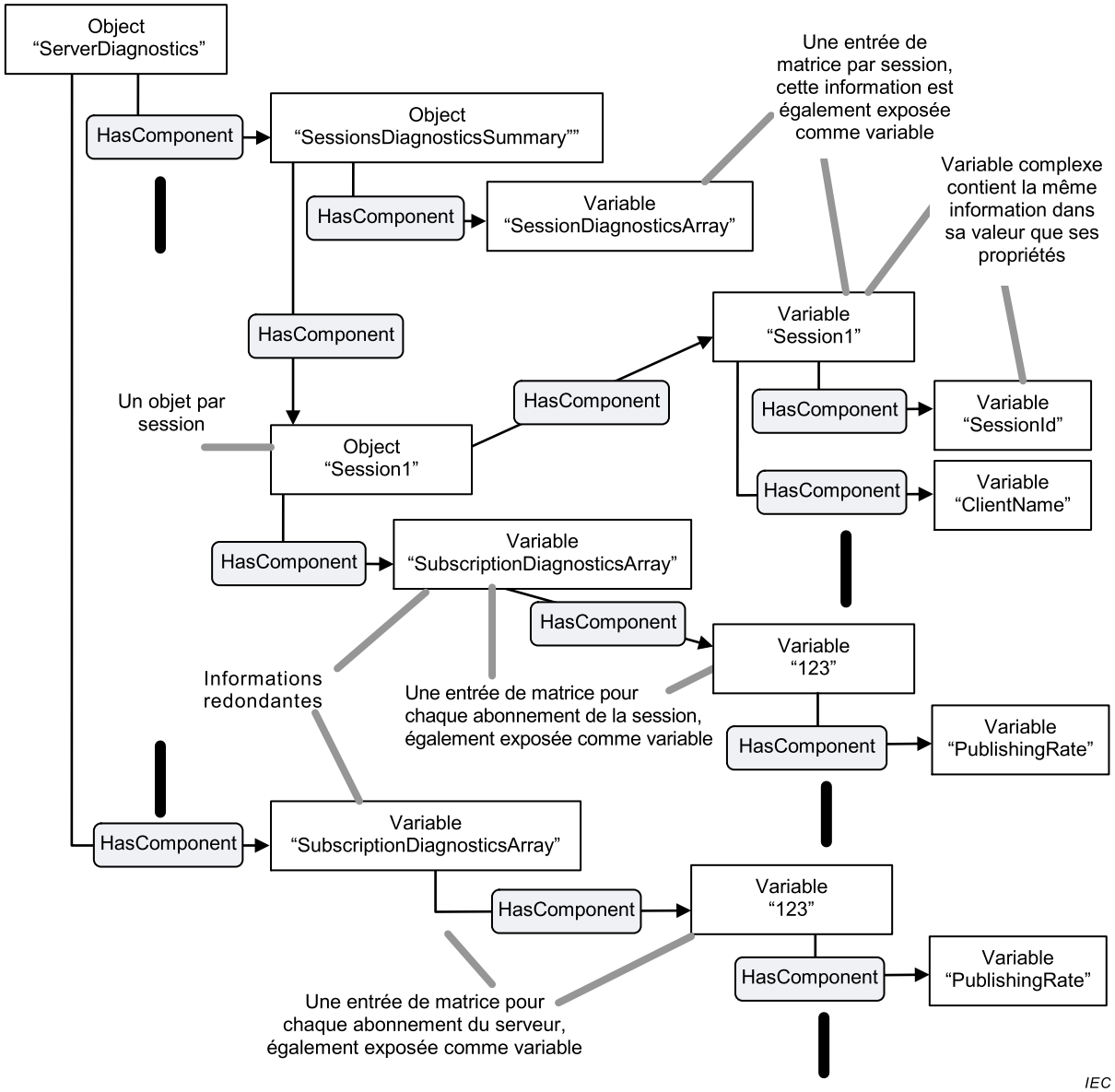


Figure 9 – Extrait d'informations de diagnostic du Serveur

8.3.2 Objet Server

Cet *Objet* est utilisé comme le point d'entrée de navigation pour les informations relatives au *Serveur*. Le contenu de cet *Objet* est déjà défini par sa définition de type en 6.3.1. Il est défini de façon formelle dans le Tableau 90. L'*Objet Server* sert de notificateur de racine, c'est-à-dire que l'*Attribut EventNotifier* doit être positionné pour fournir des *Événements*. Tous les

Événements du *Serveur* doivent être accessibles en s'abonnant aux *Événements* de l'*Objet Server.*

Tableau 90 – Définition de Server

Attribut	**Valeur**				
BrowseName	Server				
Références	**NodeClass**	**BrowseName**	**DataType**	**Type de définition**	**Modelling Rule**
HasTypeDefinition	Object Type	ServerType	Défini en 6.3.1		
HasProperty	Variable	ServerArray	String[]	PropertyType	Obligatoire
HasProperty	Variable	NamespaceArray	String[]	PropertyType	Obligatoire
HasComponent	Variable	ServerStatus[a]	ServerStatusDataType	ServerStatusType	Obligatoire
HasProperty	Variable	ServiceLevel	Byte	PropertyType	Obligatoire
HasComponent	Object	ServerCapabilities[a]	--	ServerCapabilities	Obligatoire
HasComponent	Object	ServerDiagnostics[a]	--	ServerDiagnosticsType	Obligatoire
HasComponent	Object	VendorServerInfo	--	spécifique au fournisseur[b]	Obligatoire
HasComponent	Object	ServerRedundancy[a]	--	dépend de la redondance prise en charge[c]	Obligatoire

[a] Les *Objets* et *Variables* contenants de ces *Objets* et *Variables* sont définis par leur *BrowseName* défini dans le *TypeDefinitionNode* correspondant. Le *NodeId* est défini par le nom symbolique composé décrit en 4.1.

[b] Doit être l'*ObjectType VendorServerInfo* ou l'un de ses sous-types.

[c] Doit être le *ServerRedundancyType* ou l'un de ses sous-types.

8.4 Objets ModellingRule

8.4.1 ExposesItsArray

La *ModellingRule ExposesItsArray* est définie dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses,* l'*Objet* "*ExposesItsArray*", est définie de façon formelle dans le Tableau 91.

Tableau 91 – Définition de ExposesItsArray

Attribut	**Valeur**		
BrowseName	ExposesItsArray		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Défini en 6.5
HasProperty	Variable	NamingRule	Valeur mise à "Constraint"

8.4.2 Mandatory (Obligatoire)

La *ModellingRule Mandatory* (*Obligatoire*) est définie dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses,* l'*Objet* "*Mandatory*", est définie de façon formelle dans le Tableau 92.

Tableau 92 – Définition de Mandatory

Attribut	**Valeur**		
BrowseName	Mandatory		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Défini en 6.5
HasProperty	Variable	NamingRule	Valeur mise à "Mandatory"

8.4.3 Optional (Facultatif)

La *ModellingRule Optional* (*Facultatif*) est définie dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses*, l'*Objet* "*Optional*", est définie de façon formelle dans le Tableau 93.

Tableau 93 – Définition de Optional

Attribut	**Valeur**		
BrowseName	Optional		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Défini en 6.5
HasProperty	Variable	NamingRule	Valeur mise à "Optional" (facultatif)

8.4.4 OptionalPlaceholder

La *ModellingRule OptionalPlaceholder* (*Paramètre fictif facultatif*) est définie dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses*, l'*Objet* "*OptionalPlaceholder*", est définie de façon formelle dans le Tableau 94.

Tableau 94 – Définition de OptionalPlaceholder

Attribut	**Valeur**		
BrowseName	OptionalPlaceholder		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Défini en 6.5
HasProperty	Variable	NamingRule	Valeur mise à "Constraint"

8.4.5 MandatoryPlaceholder

La *ModellingRule MandatoryPlaceholder* (*Paramètre fictif obligatoire)* est définie dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses*, l'*Objet* "*MandatoryPlaceholder*", est définie de façon formelle dans le Tableau 95.

Tableau 95 – Définition de MandatoryPlaceholder

Attribut	**Valeur**		
BrowseName	MandatoryPlaceholder		
Références	**NodeClass**	**BrowseName**	**Comment**
HasTypeDefinition	ObjectType	ModellingRuleType	Defined in 6.5
HasProperty	Variable	NamingRule	Valeur mise à "Constraint"

9 Méthodes normalisées – GetMonitoredItems

GetMonitoredItems sert à obtenir des informations sur les éléments surveillés d'un abonnement. Son utilisation normale est définie dans l'IEC 62541-4.

Signature

```
GetMonitoredItems(
    [in] UInt32 subscriptionId
    [out] UInt32[] serverHandles
    [out] UInt32[] clientHandles
  );
```

Argument	Description
subscriptionId	Identificateur de l'abonnement.
serverHandles	Matrice des serverHandles pour tous les MonitoredItems de l'abonnement identifiés par subscriptionId
clientHandles	Matrice des clientHandles pour tous les MonitoredItems de l'abonnement identifiés par subscriptionId

Codes de résultat de la méthode (définis dans le Service Call)

Code de Résultat	Description
Bad_SubscriptionIdInvalid	Défini dans l'IEC 62541-4

Le Tableau 96 spécifie la représentation de l'*Espace d'Adresses* pour la *Méthode GetMonitoredItems*.

Tableau 96 – Définition de l'Espace d'Adresses pour la Méthode GetMonitoredItems

Attribut	Valeur				
BrowseName	GetMonitoredItems				
Références	**NodeClass**	**BrowseName**	**DataType**	**Type de définition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

10 Vues normalisées

Il n'y a pas de *Vues* OPC UA principales de définies.

11 ReferenceTypes normalisés

11.1 Références

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 97.

Tableau 97 – ReferenceType References

Attributs	Valeur		
BrowseName	References		
InverseName	--		
Symmetric	True		
IsAbstract	True		
Références	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HierarchicalReferences	Défini en 11.2
HasSubtype	ReferenceType	NonHierarchicalReferences	Défini en 11.3

11.2 HierarchicalReferences

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 98.

Tableau 98 – ReferenceType HierarchicalReferences

Attributs	Valeur		
BrowseName	HierarchicalReferences		
InverseName	--		
Symmetric	False		
IsAbstract	True		
Références	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasChild	Défini en 11.4
HasSubtype	ReferenceType	Organizes	Défini en 11.6
HasSubtype	ReferenceType	HasEventSource	Défini en 11.15

11.3 NonHierarchicalReferences

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 99.

Tableau 99 – ReferenceType NonHierarchicalReferences

Attributs	Valeur		
BrowseName	NonHierarchicalReferences		
InverseName	--		
Symmetric	True		
IsAbstract	True		
Références	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasModellingRule	Défini en 11.11
HasSubtype	ReferenceType	HasTypeDefinition	Défini en 11.12
HasSubtype	ReferenceType	HasEncoding	Défini en 11.13
HasSubtype	ReferenceType	HasDescription	Défini en 11.14
HasSubtype	ReferenceType	GeneratesEvent	Défini en 11.17

11.4 HasChild

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 100.

Tableau 100 – ReferenceType HasChild

Attributs	Valeur		
BrowseName	HasChild		
InverseName	--		
Symmetric	False		
IsAbstract	True		
Références	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	Aggregates	Défini en 11.5
HasSubtype	ReferenceType	HasSubtype	Défini en 11.10

11.5 Aggregates

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 101.

Tableau 101 – ReferenceType Aggregates

Attributs	**Valeur**		
BrowseName	Aggregates		
InverseName	--		
Symmetric	False		
IsAbstract	True		
Références	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasComponent	Défini en 11.7
HasSubtype	ReferenceType	HasProperty	Défini en 11.9

11.6 Organizes

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 102.

Tableau 102 – ReferenceType Organizes

Attributs	**Valeur**		
BrowseName	Organizes		
InverseName	OrganizedBy		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

11.7 HasComponent

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 103.

Tableau 103 – ReferenceType HasComponent

Attributs	**Valeur**		
BrowseName	HasComponent		
InverseName	ComponentOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasOrderedComponent	Défini en 11.8

11.8 HasOrderedComponent

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 104.

Tableau 104 – ReferenceType HasOrderedComponent

Attributs	**Valeur**		
BrowseName	HasOrderedComponent		
InverseName	OrderedComponentOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

11.9 HasProperty

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée au Tableau 105.

Tableau 105 – ReferenceType HasProperty

Attributs	**Valeur**		
BrowseName	HasProperty		
InverseName	PropertyOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

11.10 HasSubtype

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 106.

Tableau 106 – ReferenceType HasSubtype

Attributs	**Valeur**		
BrowseName	HasSubtype		
InverseName	SubtypeOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

11.11 HasModellingRule

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 107.

Tableau 107 – ReferenceType HasModellingRule

Attributs	**Valeur**		
BrowseName	HasModellingRule		
InverseName	ModellingRuleOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

11.12 HasTypeDefinition

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 108.

Tableau 108 – ReferenceType HasTypeDefinition

Attributs	**Valeur**		
BrowseName	HasTypeDefinition		
InverseName	TypeDefinitionOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

11.13 HasEncoding

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 109.

Tableau 109 – ReferenceType HasEncoding

Attributs	**Valeur**		
BrowseName	HasEncoding		
InverseName	EncodingOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

11.14 HasDescription

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 110.

Tableau 110 – ReferenceType HasDescription

Attributs	**Valeur**		
BrowseName	HasDescription		
InverseName	DescriptionOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

11.15 HasEventSource

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 111.

Tableau 111 – ReferenceType HasEventSource

Attributs	**Valeur**		
BrowseName	HasEventSource		
InverseName	EventSourceOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	HasNotifier	Défini en 11.16

11.16 HasNotifier

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 112.

Tableau 112 – ReferenceType HasNotifier

Attributs	**Valeur**		
BrowseName	HasNotifier		
InverseName	NotifierOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

11.17 GeneratesEvent

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 113.

Tableau 113 – ReferenceType GeneratesEvent

Attributs	**Valeur**		
BrowseName	GeneratesEvent		
InverseName	GeneratedBy		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**
HasSubtype	ReferenceType	AlwaysGeneratesEvent	Défini en 11.18

11.18 AlwaysGeneratesEvent

Ce *ReferenceType* normalisé est défini dans l'IEC 62541-3. Sa représentation dans l'*Espace d'Adresses* est spécifiée dans le Tableau 114.

Tableau 114 – ReferenceType AlwaysGeneratesEvent

Attributs	**Valeur**		
BrowseName	AlwaysGeneratesEvent		
InverseName	AlwaysGeneratedBy		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

12 DataTypes normalisés

12.1 Vue d'ensemble

Il n'est pas nécessaire qu'un *Serveur* OPC UA présente ses *DataTypes* dans son *Espace d'Adresses*. Indépendant de la présentation des *DataTypes*, il doit prendre en charge les *DataTypes* comme décrit dans les paragraphes ci-après.

12.2 DataTypes définis dans l'IEC 62541-3

L'IEC 62541-3 définit un ensemble de *DataTypes*. Leur représentation dans l'*Espace d'Adresses* est définie dans le Tableau 115.

Tableau 115 – Définitions de DataType dans l'IEC 62541-3

BrowseName
BaseDataType
Argument
Boolean
Byte
ByteString
DateTime
Double
Duration
Enumeration
Float
Guid
IdType
SByte
Integer
Int16
Int32
Int64
Image
ImageBMP
ImageGIF
ImageJPG
ImagePNG
LocaleId
LocalizedText
NamingRuleType
NodeClass
NodeId
Number
QualifiedName
String
structure
Time
UInteger
UInt16
UInt32
UInt64
UTCTime
XmlElement
TimeZoneDataType
EnumValueType

Parmi les *DataTypes* définis au Tableau 115, seuls certains sont les sources de *Références* comme défini dans le tableau ci-après.

Les *Références* du *BaseDataType* sont définies dans le Tableau 116.

Tableau 116 – Définition de BaseDataType

Attributs	**Valeur**		
BrowseName	BaseDataType		
IsAbstract	TRUE (vrai)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Boolean	FALSE (faux)
HasSubtype	DataType	ByteString	FALSE (faux)
HasSubtype	DataType	DateTime	FALSE (faux)
HasSubtype	DataType	DataValue	FALSE (faux)
HasSubtype	DataType	DiagnosticInfo	FALSE (faux)
HasSubtype	DataType	Enumeration	TRUE (vrai)
HasSubtype	DataType	ExpandedNodeId	FALSE (faux)
HasSubtype	DataType	Guid	FALSE (faux)
HasSubtype	DataType	LocalizedText	FALSE (faux)
HasSubtype	DataType	NodeId	FALSE (faux)
HasSubtype	DataType	Number	TRUE (vrai)
HasSubtype	DataType	QualifiedName	FALSE (faux)
HasSubtype	DataType	String	FALSE (faux)
HasSubtype	DataType	structure	TRUE (vrai)
HasSubtype	DataType	XmlElement	FALSE (faux)

Les *Références* de *Structure* sont définies dans le Tableau 117.

Tableau 117 – Définition de Structure

Attributs	**Valeur**		
BrowseName	structure		
IsAbstract	TRUE (vrai)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Argument	FALSE (faux)
HasSubtype	DataType	UserIdentityToken	TRUE (vrai)
HasSubtype	DataType	AddNodesItem	FALSE (faux)
HasSubtype	DataType	AddReferencesItem	FALSE (faux)
HasSubtype	DataType	DeleteNodesItem	FALSE (faux)
HasSubtype	DataType	DeleteReferencesItem	FALSE (faux)
HasSubtype	DataType	ApplicationDescription	FALSE (faux)
HasSubtype	DataType	BuildInfo	FALSE (faux)
HasSubtype	DataType	RedundantServerDataType	FALSE (faux)
HasSubtype	DataType	SamplingIntervalDiagnosticsDataType	FALSE (faux)
HasSubtype	DataType	ServerDiagnosticsSummaryDataType	FALSE (faux)
HasSubtype	DataType	ServerStatusDataType	FALSE (faux)
HasSubtype	DataType	SessionDiagnosticsDataType	FALSE (faux)
HasSubtype	DataType	SessionSecurityDiagnosticsDataType	FALSE (faux)
HasSubtype	DataType	ServiceCounterDataType	FALSE (faux)
HasSubtype	DataType	StatusResult	FALSE (faux)
HasSubtype	DataType	SubscriptionDiagnosticsDataType	FALSE (faux)
HasSubtype	DataTypes	ModelChangeStructureDataType	FALSE (faux)
HasSubtype	DataTypes	SemanticChangeStructureDataType	FALSE (faux)
HasSubtype	DataType	SignedSoftwareCertificate	FALSE (faux)
HasSubtype	DataType	TimeZoneDataType	FALSE (faux)
HasSubtype	DataType	EnumValueType	FALSE (faux)

Les *Références* d'*Enumeration* sont définies dans le Tableau 118.

Tableau 118 – Définition de Enumeration

Attributs	**Valeur**		
BrowseName	Enumeration		
IsAbstract	TRUE (vrai)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	IdType	FALSE (faux)
HasSubtype	DataType	NamingRuleType	FALSE (faux)
HasSubtype	DataType	NodeClass	FALSE (faux)
HasSubtype	DataType	SecurityTokenRequestType	FALSE (faux)
HasSubtype	DataType	MessageSecurityMode	FALSE (faux)
HasSubtype	DataType	RedundancySupport	FALSE (faux)
HasSubtype	DataType	ServerState	FALSE (faux)

Les *Références* de *ByteString* sont définies dans le Tableau 119.

Tableau 119 – Définition de ByteString

Attributs	**Valeur**		
BrowseName	ByteString		
IsAbstract	TRUE (vrai)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Image	TRUE (vrai)

Les *Références* de *Number* sont définies dans le Tableau 120.

Tableau 120 – Définition de Number

Attributs	**Valeur**		
BrowseName	Number		
IsAbstract	TRUE (vrai)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Integer	TRUE (vrai)
HasSubtype	DataType	UInteger	TRUE (vrai)
HasSubtype	DataType	Double	FALSE (faux)
HasSubtype	DataType	Float	FALSE (faux)

Les *Références* de *Double* sont définies dans le Tableau 121.

Tableau 121 – Définition de Double

Attributs	**Valeur**		
BrowseName	Double		
IsAbstract	FALSE (faux)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Duration	FALSE (faux)

Les *Références* d'*Integer* sont définies dans le Tableau 122.

Tableau 122 – Définition d'Integer

Attributs	Valeur		
BrowseName	Integer		
IsAbstract	TRUE (vrai)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	SByte	FALSE (faux)
HasSubtype	DataType	Int16	FALSE (faux)
HasSubtype	DataType	Int32	FALSE (faux)
HasSubtype	DataType	Int64	FALSE (faux)

Les *Références* de *DateTime* sont définies dans le Tableau 123.

Tableau 123 – Définition de DateTime

Attributs	Valeur		
BrowseName	DateTime		
IsAbstract	FALSE (faux)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	UTCTime	FALSE (faux)

Les *Références* de *String* sont définies dans le Tableau 122.

Tableau 124 – Définition de String

Attributs	Valeur		
BrowseName	String		
IsAbstract	FALSE (faux)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	LocaleId	FALSE (faux)
HasSubtype	DataType	NumericRange	FALSE (faux)

Les *Références* d'UInteger sont définies dans le Tableau 125.

Tableau 125 – Définition d'UInteger

Attributs	Valeur		
BrowseName	UInteger		
IsAbstract	TRUE (vrai)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	Byte	FALSE (faux)
HasSubtype	DataType	UInt16	FALSE (faux)
HasSubtype	DataType	UInt32	FALSE (faux)
HasSubtype	DataType	UInt64	FALSE (faux)

Les *Références* d'Image sont définies dans le Tableau 126.

Tableau 126 – Définition d'Image

Attributs	Valeur		
BrowseName	Image		
IsAbstract	TRUE (vrai)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	ImageBMP	FALSE (faux)
HasSubtype	DataType	ImageGIF	FALSE (faux)
HasSubtype	DataType	ImageJPG	FALSE (faux)
HasSubtype	DataType	ImagePNG	FALSE (faux)

Les *Références* d'UInt64 sont définies dans le Tableau 127.

Tableau 127 – Définition d'UInt64

Attributs	**Valeur**		
BrowseName	UInt64		
IsAbstract	FALSE (faux)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	BitFieldMaskDataType	FALSE (Faux)

12.3 DataTypes définis dans l'IEC 62541-4

L'IEC 62541-4 définit un ensemble de *DataTypes*. Leur représentation dans l'*Espace d'Adresses* est définie dans le Tableau 128.

Tableau 128 – Définitions de DataType dans l'IEC 62541-4

BrowseName
AnonymousIdentityToken
DataValue
DiagnosticInfo
ExpandedNodeId
SignedSoftwareCertificate
UserIdentityToken
UserNameIdentityToken
X509IdentityToken
WssIdentityToken
SecurityTokenRequestType
AddNodesItem
AddReferencesItem
DeleteNodesItem
DeleteReferencesItem
NumericRange
MessageSecurityMode
ApplicationDescription

Le *SecurityTokenRequestType* est une énumération qui est définie comme étant le type du paramètre requestType du *Service* OpenSecureChannel dans l'IEC 62541-4.

L'*AddNodesItem* est une structure qui est définie comme étant le type du paramètre nodesToAdd du *Service* AddNodes dans l'IEC 62541-4.

L'*AddReferencesItem* est une structure qui est définie comme étant le type du paramètre referencesToAdd du *Service* AddReferences dans l'IEC 62541-4.

Le *DeleteNodesItem* est une structure qui est définie comme étant le type du paramètre nodesToDelete du *Service* DeleteNodes dans l'IEC 62541-4.

Le *DeleteReferencesItem* est une structure qui est définie comme étant le type du paramètre referencesToDelete du *Service* DeleteReferences dans l'IEC 62541-4.

Les *Références* d'*UserIdentityToken* sont définies dans le Tableau 129.

Tableau 129 – Définition d'UserIdentityToken

Attributs	Valeur		
BrowseName	UserIdentityToken		
IsAbstract	TRUE (vrai)		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
HasSubtype	DataType	UserNameIdentityToken	FALSE (faux)
HasSubtype	DataType	X509IdentityToken	FALSE (faux)
HasSubtype	DataType	WssIdentityToken	FALSE (faux)
HasSubtype	DataType	AnonymousIdentityToken	FALSE (faux)

12.4 BuildInfo

Cette structure contient des éléments qui décrivent les informations d'organisation du *Serveur*. Ses éléments sont définis dans le Tableau 130.

Tableau 130 – Structure BuildInfo

Nom	Type	Description
BuildInfo	structure	Informations qui décrivent la mouture du logiciel.
productUri	String	URI qui identifie le logiciel
manufacturerName	String	Nom du fabricant du logiciel.
productName	String	Nom du logiciel.
softwareVersion	String	Version du logiciel
buildNumber	String	Numéro de mouture
buildDate	UTCTime	Date et heure de la mouture.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 131.

Tableau 131 – Définition de BuildInfo

Attributs	Valeur
BrowseName	BuildInfo

12.5 RedundancySupport

Ce *DataType* est une énumération qui définit la prise en charge de la redondance par le *Serveur*. Ses valeurs sont définies dans le Tableau 132.

Tableau 132 – Valeurs de RedundancySupport

Valeur	Description
NONE_0	None (aucune) signifie qu'il n'y a pas de prise en charge de la redondance.
COLD_1	Cold (froide) signifie que le serveur prend en charge la redondance froide telle que définie dans l'IEC 62541-4 .
WARM_2	Warm (tiède) signifie que le serveur prend en charge la redondance tiède telle que définie dans l'IEC 62541-4 .
HOT_3	Hot (chaude) signifie que le serveur prend en charge la redondance chaude telle que définie dans l'IEC 62541-4 .
TRANSPARENT_4	Transparent signifie que le serveur prend en charge la redondance transparente telle que définie dans l'IEC 62541-4.
HOT_AND_MIRRORED_5	HotAndMirrored (Chaude et reflétée) signifie que le serveur prend en charge la redondance HotAndMirrored telle que définie dans l'IEC 62541-4.

Voir l'IEC 62541-4 pour une description plus détaillée des différentes valeurs.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 133.

Tableau 133 – Définition de RedundancySupport

Attributs	Valeur
BrowseName	RedundancySupport

12.6 ServerState

Ce *DataType* est une énumération qui définit l'état d'exécution du *Serveur*. Ses valeurs sont définies dans le Tableau 134.

Tableau 134 – Valeurs de ServerState

Valeur	Description
RUNNING_0	Le serveur fonctionne normalement. Il s'agit de l'état usuel pour un serveur.
FAILED_1	Une erreur fatale spécifique au fournisseur s'est produite dans le serveur. Le serveur ne fonctionne plus. La procédure de récupération de cette situation est spécifique au fournisseur. Il convient de s'attendre à l'échec de la plupart des demandes de *Service*.
NO_CONFIGURATION_2	Le serveur fonctionne, mais n'a pas d'informations de configuration chargées et ne transfère donc pas de données.
SUSPENDED_3	Le serveur a été suspendu temporairement par quelque méthode spécifique au fournisseur et ne reçoit ni n'envoie de données.
SHUTDOWN_4	Le serveur s'est arrêté ou est dans le processus de s'arrêter. En fonction de la mise en œuvre cela peut ou peut ne pas être visible aux clients.
TEST_5	Le serveur est en mode Essai. Les sorties sont débranchées du matériel réel, mais autrement le serveur se comporte normalement. Les entrées peuvent être lues ou peuvent être simulées en fonction de la mise en œuvre du fournisseur. Le StatusCode est en général retourné normalement.
COMMUNICATION_FAULT_6	Le serveur fonctionne correctement, mais a des difficultés à accéder aux données issues de ses sources de données. Cela peut être dû à des problèmes de communication ou quelque autre problème empêchant l'appareil, le système de commande sous-jacents, etc. de retourner des données valides. Il peut s'agir d'une défaillance complète, ce qui signifie qu'aucune donnée n'est disponible, ou d'une défaillance partielle, signifiant que certaines données restent disponibles. Il est escompté que des éléments altérés par la panne retourneront individuellement avec une indication de code de statut BAD FAILURE pour les éléments.
UNKNOWN_7	Cet état est utilisé uniquement pour indiquer que le serveur OPC UA ne connaît pas l'état des serveurs sous-jacents.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 135.

Tableau 135 – Définition de ServerState

Attributs	Valeur
BrowseName	ServerState

12.7 RedundantServerDataType

Cette structure contient des éléments qui décrivent le statut du *Serveur*. Sa composition est définie dans le Tableau 136.

Tableau 136 – Structure de RedundantServerDataType

Nom	Type	Description
RedundantServerDataType	structure	
serverId	String	L'Id du serveur (pas l'URI).
serviceLevel	Byte	Le niveau de service du serveur.
serverState	ServerState	L'état courant du serveur.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 137.

Tableau 137 – Définition de RedundantServerDataType

Attributs	Valeur
BrowseName	RedundantServerDataType

12.8 SamplingIntervalDiagnosticsDataType

Cette structure contient les informations de diagnostic relatives aux fréquences d'échantillonnage actuellement utilisées par le *Serveur*. Ses éléments sont définis dans le Tableau 138.

Tableau 138 – Structure de SamplingIntervalDiagnosticsDataType

Nom	Type	Description
SamplingIntervalDiagnosticsDataType	structure	
samplingInterval	Duration	L'intervalle d'échantillonnage en millisecondes.
sampledMonitoredItemsCount	UInt32	Le nombre de *MonitoredItems* échantillonnés à cette fréquence d'échantillonnage.
maxSampledMonitoredItemsCount	UInt32	Le nombre maximal de *MonitoredItems* échantillonnés à cette fréquence d'échantillonnage en même temps depuis le démarrage (ou redémarrage) du serveur.
disabledMonitoredItemsSamplingCount	UInt32	Le nombre de *MonitoredItems* à cette fréquence d'échantillonnage dont l'échantillonnage est actuellement désactivé.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 139.

Tableau 139 – Définition de SamplingIntervalDiagnosticsDataType

Attributs	Valeur
BrowseName	SamplingIntervalDiagnosticsDataType

12.9 ServerDiagnosticsSummaryDataType

Cette structure contient des informations récapitulatives de diagnostic pour le *Serveur*. Ses éléments sont définis dans le Tableau 140.

Tableau 140 – Structure de ServerDiagnosticsSummaryDataType

Nom	Type	Description
ServerDiagnosticsSummaryDataType	structure	
serverViewCount	UInt32	Le nombre de vues créées par serveur dans le serveur.
currentSessionCount	UInt32	Le nombre de sessions client actuellement établies dans le serveur.
cumulatedSessionCount	UInt32	Le nombre cumulé de sessions client qui ont été établies dans le serveur depuis le démarrage (ou redémarrage) du serveur. Le *currentSessionCount* y est inclus.
securityRejectedSessionCount	UInt32	Le nombre de demandes de sessions client qui ont été rejetées en raison de contraintes de sécurité depuis le démarrage (ou redémarrage) du serveur.
rejectedSessionCount	UInt32	Le nombre de demandes d'établissement de sessions client qui ont été rejetées depuis le démarrage (ou redémarrage) du serveur. Ce nombre inclut le *securityRejectedSessionCount*.
sessionTimeoutCount	UInt32	Le nombre de sessions client qui ont été fermées en raison de dépassement de délai depuis le démarrage (ou redémarrage) du serveur.
sessionAbortCount	UInt32	Le nombre de sessions client qui ont été fermées en raison d'erreurs depuis le démarrage (ou redémarrage) du serveur.
publishingIntervalCount	UInt32	Le nombre d'intervalles d'édition actuellement pris en charge dans le serveur.
currentSubscriptionCount	UInt32	Le nombre d'abonnements actuellement établis dans le serveur.
cumulatedSubscriptionCount	UInt32	Le nombre cumulé d'abonnements qui ont été établis dans le serveur depuis le démarrage (ou redémarrage) du serveur. Le *currentSubscriptionCount* y est inclus.
securityRejectedRequestsCount	UInt32	Le nombre de demandes qui ont été rejetées en raison de contraintes de sécurité depuis le démarrage (ou redémarrage) du serveur. Les demandes incluent tous les *Services* définis dans l'IEC 62541-4, également les demandes de création de sessions.
rejectedRequestsCount	UInt32	Le nombre de demandes qui ont été rejetées depuis le démarrage (ou redémarrage) du serveur. Les demandes incluent tous les *Services* définis dans l'IEC 62541-4, également les demandes de création de sessions. Ce nombre inclut le *securityRejectedRequestsCount*.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 141.

Tableau 141 – Définition de ServerDiagnosticsSummaryDataType

Attributs	Valeur
BrowseName	ServerDiagnosticsSummaryDataType

12.10 ServerStatusDataType

Cette structure contient des éléments qui décrivent le statut du *Serveur*. Sa composition est définie dans le Tableau 142.

Tableau 142 – Structure de ServerStatusDataType

Nom	Type	Description
ServerStatusDataType	structure	
startTime	UTCTime	Heure (TUC) à laquelle le serveur a démarré. Elle est constante pour l'instance de serveur et n'est pas réinitialisée lorsque le serveur change d'état. Il convient que chaque instance d'un serveur conserve l'heure à laquelle le processus a démarré.
currentTime	UTCTime	L'heure (TUC) courante telle qu'elle est connue par le serveur.
state	ServerState	L'état courant du serveur. Ses valeurs sont définies au 12.6.
buildInfo	BuildInfo	
secondsTillShutdown	UInt32	Nombre approximatif de secondes jusqu'à ce que le serveur soit arrêté. Cette valeur est pertinente uniquement une fois que l'état est passé à SHUTDOWN.
shutdownReason	LocalizedText	Un texte localisé facultatif indiquant la raison de l'arrêt. Cette valeur est pertinente uniquement une fois que l'état est passé à SHUTDOWN.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 143.

Tableau 143 – Définition de ServerStatusDataType

Attributs	Valeur
BrowseName	ServerStatusDataType

12.11 SessionDiagnosticsDataType

Cette structure contient des informations de diagnostic relatives aux sessions client. Ses éléments sont définis dans le Tableau 144. La plupart des valeurs représentées dans la structure fournissent des informations relatives au nombre d'appels d'un *Service*, le nombre de *MonitoredItems* actuellement utilisés, etc. Il n'est pas nécessaire que ces nombres fournissent la valeur exacte, mais seulement une valeur approximative et, de ce fait, le *Serveur* n'est pas chargé de donner des valeurs exactes.

Tableau 144 – Structure de SessionDiagnosticsDataType

Nom	Type	Description
SessionDiagnosticsDataType	structure	
sessionId	NodeId	Identificateur de session affecté par le serveur.
sessionName	String	Le nom de la session fourni dans la demande CreateSession
clientDesription	Application Description	La description fournie par le client dans la demande CreateSession.
serverUri	String	La demande de serverUri dans la demande CreateSession.
endpointUrl	String	L'endpointUrl transmis par le client à la demande CreateSession.
localeIds	LocaleId[]	Matrice de LocaleIds spécifiée par le client dans l'appel de session ouverte.
actualSessionTimeout	Duration	La temporisation de session demandée spécifiée par le client dans l'appel de session ouverte.
maxResponseMessageSize	UInt32	La taille maximale pour le message de réponse envoyé au client.
clientConnectionTime	UTCTime	L'horodatage du serveur lorsque le client ouvre la session.
clientLastContactTime	UTCTime	L'horodatage du serveur de la dernière demande du client dans le contexte de la session.
currentSubscriptionsCount	UInt32	Le nombre d'abonnements actuellement utilisés par la session.
currentMonitoredItemsCount	UInt32	Le nombre de *MonitoredItems* actuellement utilisés par la session.
currentPublishRequestsInQueue	UInt32	Le nombre de demandes d'édition actuellement dans la file d'attente pour la session.
currentPublishTimerExpirations	UInt32	Le nombre d'expirations de minuterie d'édition lorsqu'il y a des données à envoyer, mais il n'y a pas de demandes d'édition pour cette session. La valeur doit être 0 s'il n'y a pas de données à envoyer ou de demandes d'édition mises en file d'attente.
totalRequestsCount	ServiceCounter DataType	Compteur de tous les *Services*, identifiant le nombre de demandes reçues de n'importe quels *Services* sur la session.
unauthorizedRequestsCount	UInt32	Compteur de tous les *Services*, identifiant le nombre de demandes de *Service* qui ont été rejetées en raison d'un échec d'autorisation.
readCount	ServiceCounter DataType	Compteur du *Service* Read (lecture), identifiant le nombre de demandes de ce *Service* reçues sur la session.
historyReadCount	ServiceCounter DataType	Compteur du *Service* HistoryRead (lecture d'historique), identifiant le nombre de demandes de ce *Service* reçues sur la session.
writeCount	ServiceCounter DataType	Compteur du *Service* Write (écriture), identifiant le nombre de demandes de ce *Service* reçues sur la session.
historyUpdateCount	ServiceCounter DataType	Compteur du *Service* HistoryUpdate (mise à jour d'historique), identifiant le nombre de demandes de ce *Service* reçues sur la session.
callCount	ServiceCounter DataType	Compteur du *Service* Call (appel), identifiant le nombre de demandes de ce *Service* reçues sur la session.
createMonitoredItemsCount	ServiceCounter DataType	Compteur du *Service* CreateMonitoredItem (créer un élément surveillé), identifiant le nombre de demandes de ce *Service* reçues sur la session.
modifyMonitoredItemsCount	ServiceCounter DataType	Compteur du *Service* ModifyMonitoredItem (modifier un élément surveillé), identifiant le nombre de demandes de ce *Service* reçues sur la session.
setMonitoringModeCount	ServiceCounter DataType	Compteur du *Service* SetMonitoringMode (établir le mode de surveillance), identifiant le nombre de demandes de ce *Service* reçues sur la session.
setTriggeringCount	ServiceCounter DataType	Compteur du *Service* SetTriggering (établir le déclenchement), identifiant le nombre de demandes de ce *Service* reçues sur la session.

Nom	Type	Description
deleteMonitoredItemsCount	ServiceCounter DataType	Compteur du *Service* DeleteMonitoredItems (supprimer des éléments surveillés), identifiant le nombre de demandes de ce *Service* reçues sur la session.
createSubscriptionCount	ServiceCounter DataType	Compteur du *Service* CreateSubscription (créer un abonnement), identifiant le nombre de demandes de ce *Service* reçues sur la session.
modifySubscriptionCount	ServiceCounter DataType	Compteur du *Service* ModifySubscription (modifier un abonnement), identifiant le nombre de demandes de ce *Service* reçues sur la session.
setPublishingModeCount	ServiceCounter DataType	Compteur du *Service* SetPublishingMode (établir le mode d'édition), identifiant le nombre de demandes de ce *Service* reçues sur la session.
publishCount	ServiceCounter DataType	Compteur du *Service* Publish (éditer), identifiant le nombre de demandes de ce *Service* reçues sur la session.
republishCount	ServiceCounter DataType	Compteur du *Service* Republish (rééditer), identifiant le nombre de demandes de ce *Service* reçues sur la session.
transferSubscriptionsCount	ServiceCounter DataType	Compteur du *Service* TransferSubscriptions (transférer des abonnements), identifiant le nombre de demandes de ce *Service* reçues sur la session.
deleteSubscriptionsCount	ServiceCounter DataType	Compteur du *Service* DeleteSubscriptions (supprimer des abonnements), identifiant le nombre de demandes de ce *Service* reçues sur la session.
addNodesCount	ServiceCounter DataType	Compteur du *Service* AddNodes (ajouter des nœuds), identifiant le nombre de demandes de ce *Service* reçues sur la session.
addReferencesCount	ServiceCounter DataType	Compteur du *Service* AddReferences (ajouter des références), identifiant le nombre de demandes de ce *Service* reçues sur la session.
deleteNodesCount	ServiceCounter DataType	Compteur du *Service* DeleteNodes (supprimer des nœuds), identifiant le nombre de demandes de ce *Service* reçues sur la session.
deleteReferencesCount	ServiceCounter DataType	Compteur du *Service* DeleteReferences (supprimer des références), identifiant le nombre de demandes de ce *Service* reçues sur la session.
browseCount	ServiceCounter DataType	Compteur du *Service* Browse (parcourir), identifiant le nombre de demandes de ce *Service* reçues sur la session.
browseNextCount	ServiceCounter DataType	Compteur du *Service* BrowseNext (parcourir le suivant), identifiant le nombre de demandes de ce *Service* reçues sur la session.
translateBrowsePathsToNodeIds Count	ServiceCounter DataType	Compteur du *Service* TranslateBrowsePathsToNodeIds (traduire les chemins de navigation en identificateurs de nœuds), identifiant le nombre de demandes de ce *Service* reçues sur la session.
queryFirstCount	ServiceCounter DataType	Compteur du *Service* QueryFirst (interroger le premier), identifiant le nombre de demandes de ce *Service* reçues sur la session.
queryNextCount	ServiceCounter DataType	Compteur du *Service* QueryNext (interroger le suivant), identifiant le nombre de demandes de ce *Service* reçues sur la session.
registerNodesCount	ServiceCounter DataType	Compteur du *Service* RegisterNodes (enregistrer des nœuds), identifiant le nombre de demandes de ce *Service* reçues sur la session.
unregisterNodesCount	ServiceCounter DataType	Compteur du *Service* UnregisterNodes (désenregistrer des nœuds), identifiant le nombre de demandes de ce *Service* reçues sur la session.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 145.

Tableau 145 – Définition de SessionDiagnosticsDataType

Attributs	Valeur
BrowseName	SessionDiagnosticsDataType

12.12 SessionSecurityDiagnosticsDataType

Cette structure contient des informations de diagnostic liées à la sécurité relatives aux sessions client. Ses éléments sont définis dans le Tableau 146. Ces informations étant relatives à la sécurité, il convient de les rendre accessibles seulement aux utilisateurs autorisés et non à tous.

Tableau 146 – Structure de SessionSecurityDiagnosticsDataType

Nom	Type	Description
SessionSecurityDiagnosticsDataType	structure	
sessionId	NodeId	Identificateur de session affecté par le serveur.
clientUserIdOfSession	String	Nom d'utilisateur authentifié lors de la création de la session.
clientUserIdHistory	String[]	Matrice contenant le nom de l'utilisateur authentifié actuellement actif (soit à partir de la création de la session, soit à partir de l'appel du *Service ActivateSession*) et l'historique de ces noms. Chaque fois que l'utilisateur actif change, une entrée doit être faite à la fin de la matrice. L'utilisateur actif est toujours à la fin de la matrice. Des Serveurs peuvent limiter la taille de cette matrice, mais doivent prendre en charge au moins la taille 2. La façon dont le nom de l'utilisateur authentifié peut être obtenu du système via les informations reçues en tant qu'élément de l'établissement de session est définie en 6.4.3.
authenticationMechanism	String	Type d'authentification (nom d'utilisateur et mot de passe, X.509, Kerberos).
encoding	String	Le codage utilisé à la volée, par exemple XML ou UA Binary.
transportProtocol	String	Le protocole de transport qui est utilisé, par exemple TCP ou HTTP.
securityMode	MessageSecurityMode	Le mode de sécurité des messages utilisé pour la session.
securityPolicyUri	String	Le nom de la politique de sécurité utilisée pour cette session.
clientCertificate	ByteString	Le certificat d'instance d'application fourni par le client dans la demande CreateSession.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 147.

Tableau 147 – Définition de SessionSecurityDiagnosticsDataType

Attributs	Valeur
BrowseName	SessionSecurityDiagnosticsDataType

12.13 ServiceCounterDataType

Cette structure contient des informations de diagnostic relatives aux abonnements. Ses éléments sont définis dans le Tableau 148.

Tableau 148 – Structure de ServiceCounterDataType

Nom	Type	Description
ServiceCounterDataType	structure	
totalCount	UInt32	Le nombre de demandes de *Service* qui ont été reçues.
errorCount	UInt32	Le nombre total de demandes de *Service* qui ont été rejetées.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 149.

Tableau 149 – Définition de ServiceCounterDataType

Attributs	Valeur
BrowseName	ServiceCounterDataType

12.14 StatusResult

Cette structure combine un *StatusCode* et des informations de diagnostic et peut, par exemple, être utilisée par les Méthodes pour retourner plusieurs *StatusCodes* et les informations de diagnostic correspondantes qui ne sont pas traitées dans les paramètres du

Service Call. Les éléments de ce *DataType* sont définis dans le Tableau 150. La question de retourner les DiagnosticInfo ou non dépend du réglage des appels du Service.

Tableau 150 – Structure de StatusResult

Nom	Type	Description
StatusResult	structure	
statusCode	StatusCode	Le StatusCode.
diagnosticInfo	DiagnosticInfo	Les informations de diagnostic pour le statusCode.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 151.

Tableau 151 – Définition de StatusResult

Attributs	Valeur
BrowseName	StatusResult

12.15 SubscriptionDiagnosticsDataType

Cette structure contient des informations de diagnostic relatives aux abonnements. Ses éléments sont définis dans le Tableau 152.

Tableau 152 – Structure de SubscriptionDiagnosticsDataType

Nom	Type	Description
SubscriptionDiagnosticsDataType	structure	
sessionId	NodeId	Identificateur affecté par le serveur à la session à laquelle l'abonnement appartient.
subscriptionId	UInt32	Identificateur de session affecté par le serveur à l'abonnement.
Priority	Byte	La priorité affectée par le client à l'abonnement.
publishingInterval	Duration	L'intervalle d'édition de l'abonnement en millisecondes.
maxKeepAliveCount	UInt32	Le compte d'entretien maximal de l'abonnement.
maxLifetimeCount	UInt32	Le compte de durée de vie maximale de l'abonnement.
maxNotificationsPerPublish	UInt32	Le nombre maximal de notifications par réponse d'édition.
publishingEnabled	Boolean	Si l'édition est activée pour l'abonnement ou non.
modifyCount	UInt32	Le nombre de demandes ModifySubscription reçues pour l'abonnement.
enableCount	UInt32	Le nombre de fois que l'abonnement a été activé.
disableCount	UInt32	Le nombre de fois que l'abonnement a été désactivé.
republishRequestCount	UInt32	Le nombre de demandes de *Service* Republish (réédition) qui ont été reçues et traitées pour l'abonnement.
republishMessageRequestCount	UInt32	Le nombre total de messages dont la réédition a été demandée pour l'abonnement.
republishMessageCount	UInt32	Le nombre total de messages qui ont été réédités avec succès pour l'abonnement.
transferRequestCount	UInt32	Le nombre total de demandes de *Service* TransferSubscriptions qui ont été reçues pour l'abonnement.
transferredToAltClientCount	UInt32	Le nombre de fois que l'abonnement a été transféré à un autre client.
transferredToSameClientCount	UInt32	Le nombre de fois que l'abonnement a été transféré à une autre session pour le même client.
publishRequestCount	UInt32	Le nombre de demandes de *Service* Publish (édition) qui ont été reçues et traitées pour l'abonnement.
dataChangeNotificationsCount	UInt32	Le nombre de Notifications de modifications des données envoyées par l'abonnement.
eventNotificationsCount	UInt32	Le nombre de Notifications d'événements envoyées par l'abonnement.
notificationsCount	UInt32	Le nombre total de Notifications envoyées par l'abonnement.
latePublishRequestCount	UInt32	Le nombre de fois que l'abonnement s'est mis à l'état LATE (tardif), c'est-à-dire le nombre de fois que la minuterie d'édition expire et il y a des notifications non envoyées.
currentKeepAliveCount	UInt32	Le nombre de fois que l'abonnement s'est mis à l'état KEEPALIVE (entretien).
currentLifetimeCount	UInt32	Le compte de durée de vie actuel de l'abonnement.
unacknowledgedMessageCount	UInt32	Le nombre de messages non acquittés sauvegardés dans la file d'attente de réédition.
discardedMessageCount	UInt32	Le nombre de messages qui ont été rejetés avant leur acquittement.
monitoredItemCount	UInt32	Le nombre total d'éléments surveillés de l'abonnement, y compris les éléments surveillés désactivés.
disabledMonitoredItemCount	UInt32	Le nombre d'éléments surveillés désactivés de l'abonnement.
monitoringQueueOverflowCount	UInt32	Le nombre de fois qu'un élément surveillé a abandonné les notifications en raison d'un débordement de file d'attente.
nextSequenceNumber	UInt32	Numéro de séquence pour le prochain message de notification.
eventQueueOverflowCount	UInt32	Le nombre de fois qu'un élément surveillé dans l'abonnement a généré un Événement du type EventQueueOverflowEventType.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 153.

Tableau 153 – Définition de SubscriptionDiagnosticsDataType

Attributs	Valeur
BrowseName	SubscriptionDiagnosticsDataType

12.16 ModelChangeStructureDataType

Cette structure contient des éléments qui décrivent les modifications du modèle. Sa composition est définie dans le Tableau 154.

Tableau 154 – Structure de ModelChangeStructureDataType

<table>
<tr><th>Nom</th><th>Type</th><th>Description</th></tr>
<tr><td>ModelChangeStructureDataType</td><td>structure</td><td></td></tr>
<tr><td>Affected</td><td>NodeId</td><td>NodeId du Nœud qui a été changé. Il convient que le client suppose que le Nœud affected (nœud altéré) a été créé ou supprimé, avait une Reference ajoutée ou supprimée, ou que le DataType a changé comme décrit par le verbe.</td></tr>
<tr><td>affectedType</td><td>NodeId</td><td>Si le Nœud affected était un Objet ou une Variable, affectedType contient le NodeId du TypeDefinitionNode du Nœud affected. Autrement il est mis à null.</td></tr>
<tr><td>verb</td><td>Byte</td><td>Décrit les changements apportés au Nœud altéré.
verb est un entier non signé de huit bits utilisé comme masque de bits avec la structure définie dans le tableau suivant:
<table>
<tr><th>Field (champ)</th><th>Bit</th><th>Description</th></tr>
<tr><td>NodeAdded</td><td>0</td><td>Indique que le Node affected a été ajouté.</td></tr>
<tr><td>NodeDeleted</td><td>1</td><td>Indique que le Node affected a été supprimé.</td></tr>
<tr><td>ReferenceAdded</td><td>2</td><td>Indique qu'une Référence a été ajoutée. Le Nœud altéré peut être un SourceNode ou un TargetNode. Noter qu'une Référence bidirectionnelle ajoutée est reflétée par deux ChangeStructures.</td></tr>
<tr><td>ReferenceDeleted</td><td>3</td><td>Indique qu'une Référence a été supprimée. Le Nœud altéré peut être un SourceNode ou un TargetNode. Noter qu'une Référence bidirectionnelle supprimée est reflétée par deux ChangeStructures.</td></tr>
<tr><td>DataTypeChanged</td><td>4</td><td>Ce verbe peut être utilisé uniquement pour des Nœuds altérés qui sont des Variables ou des VariableTypes. Il indique que l'Attribut DataType a changé.</td></tr>
<tr><td>Reserved</td><td>5:7</td><td>Réservé pour une utilisation ultérieure. Doit toujours être zéro.</td></tr>
</table>
Un verbe peut identifier plusieurs changements apportés immédiatement à un Nœud altéré. Il convient d'utiliser cette caractéristique si la compression d'événement est utilisée (voir IEC 62541-3 pour les détails). Noter que tous les verbes (verbs) doivent toujours être considérés dans le contexte où le ModelChangeStructureDataType est utilisé. Un NodeDeleted peut indiquer qu'un Nœud a été retiré d'une vue, mais existe encore dans d'autres Vues.</td></tr>
</table>

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 143.

Tableau 155 – Définition de ModelChangeStructureDataType

Attributs	Valeur
BrowseName	ModelChangeStructureDataType

12.17 SemanticChangeStructureDataType

Cette structure contient des éléments qui décrivent une modification du modèle. Sa composition est définie dans le Tableau 156.

Tableau 156 – Structure de SemanticChangeStructureDataType

Nom	Type	Description
SemanticChangeStructureDataType	structure	
Affected	NodeId	*NodeId* du *Nœud* qui est le propriétaire de la *Propriété* qui a changé.
affectedType	NodeId	Si le *Node affected* était un *Objet* ou une *Variable*, *affectedType* contient le *NodeId* du *TypeDefinitionNode* du *Node affected*. Autrement il est mis à null.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 143.

Tableau 157 – Définition de SemanticChangeStructureDataType

Attributs	Valeur
BrowseName	SemanticChangeStructureDataType

12.18 BitFieldMaskDataType

Ce *DataType* simple est un sous-type d'UInt64 et représente un masque de bits jusqu'à 32 bits au plus, où des bits individuels peuvent être saisis sans modifier les autres bits.

Les 32 premiers bits (bits de poids faible) du *BitFieldMaskDataType* représentent le masque de bits et les 32 bits restants représentent la validité des bits dans le masque de bits. Lorsque le *Serveur* retourne la valeur au client, la validité indique quels bits du masque de bits ont une signification. Lorsque le client transmet la valeur au *Serveur*, la validité définit quels bits il convient de saisir. Seuls les bits définis dans la validité sont modifiés dans le masque de bits, tous les autres bits restant inchangés. Le *BitFieldMaskDataType* peut être utilisé comme *DataType* dans le *VariableType OptionSetType.*

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 158.

Tableau 158 – Définition de BitFieldMaskDataType

Attributs	Valeur
BrowseName	BitFieldMaskDataType

12.19 NetworkGroupDataType

Cette structure contient les informations relatives à différents chemins de réseaux pour un *Serveur*. Sa composition est définie dans le Tableau 159.

Tableau 159 – Structure de NetworkGroupDataType

Nom	Type	Description
NetworkGroupDataType	structure	
serverUri	String	URI du Serveur représenté par le groupe de réseaux.
networkPaths	EndpointUrlListDataType[]	Matrice de différents chemins de réseaux vers le serveur, par exemple fournie par différentes cartes réseaux dans un Nœud de Serveur. Chaque chemin de réseau peut avoir plusieurs Points d'extrémité qui représentent différentes options de protocole pour le même chemin.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 160.

Tableau 160 – Définition de NetworkGroupDataType

Attributs	Valeur
BrowseName	NetworkGroupDataType

12.20 EndpointUrlListDataType

Cette structure représente une liste des URL d'un *Point d'extrémité*. Sa composition est définie dans le Tableau 161.

Tableau 161 – Structure de EndpointUrlListDataType

Nom	Type	Description
EndpointUrlListDataType	structure	
endpointUrlList	String[]	Liste des URL d'un *Point d'extrémité*.

Sa représentation dans l'*Espace d'Adresses* est définie dans le Tableau 162.

Tableau 162 – EndpointUrlListDataType Definition

Attributs	Valeur
BrowseName	EndpointUrlListDataType

Annexe A
(informative)

Décisions de conception pour modéliser les informations du Serveur

A.1 Vue d'ensemble

Cette annexe décrit les décisions de conception relatives à la modélisation des informations fournies par chaque *Serveur* OPC UA, présentant ses fonctions, des informations de diagnostic et d'autres données pour travailler avec lui, telles que le *NamespaceArray*.

Cette annexe donne un exemple de ce qu'il convient de prendre en considération pour modéliser des données en utilisant le modèle "Address Space Model". Les considérations générales pour l'utilisation du modèle "Address Space Model" peuvent être consultées dans l'IEC 62541-3.

Cette annexe est donnée uniquement à titre d'information, c'est-à-dire que chaque fournisseur de *Serveur* peut modéliser ses données de la manière appropriée adaptée à ses besoins.

Les paragraphes ci-après décrivent les décisions de conception prises pour modéliser l'*Objet Server*. Les *DataTypes*, *VariableTypes* et *ObjectTypes* généraux tels que les *EventTypes* décrits dans la présente norme ne sont pas pris en compte.

A.2 ServerType et Objet Server

Il faut d'abord décider du niveau de nécessité des types. Typiquement, chaque *Serveur* fournit un *Objet Server* avec un *NodeId* bien connu. Les *NodeIds* des *Nœuds* contenants sont également bien connus, car leur nom symbolique est spécifié dans la présente norme et le *NodeId* est basé sur le nom symbolique donné dans l'IEC 62541-6. Néanmoins, l'ensemble des *Serveurs* peuvent souhaiter présenter les *Objets Server* des *Serveurs* OPC UA qu'ils agrègent dans leur *Espace d'Adresses*. Par conséquent, il est très utile d'avoir une définition de type pour l'*Objet Server*. L'*Objet Server* est un *Objet,* car il regroupe un jeu de *Variables* et d'*Objets* contenant des informations relatives au *Serveur*. Le *ServerType* est un *ObjectType* complexe, car il convient que la structure de base de l'*Objet Server* soit bien définie. Cependant, l'*Objet Server* peut être étendu en ajoutant des *Variables* et des *Objets* dans une structure appropriée de l'*Objet Server* ou de ses *Objets* contenants.

A.3 Objets complexes typés sous l'Objet Server

Les *Objets* sous l'*Objet Server* utilisés pour regrouper des informations, telles que les fonctions *Serveur* ou le diagnostic, sont également typés, car un ensemble de *Serveurs* peut souhaiter fournir une partie seulement des informations *Serveur*, telles que les informations de diagnostic, dans son *Espace d'Adresses*. Les clients peuvent écrire des programmes en fonction de ces structures si elles sont typées, car elles ont leur définition de type.

A.4 Properties par rapport aux DataVariables

La description générale donnée dans l'IEC 62541-3 relative à la différence sémantique entre *Propriétés* et *DataVariables* n'étant pas applicable pour les informations fournies concernant le *Serveur*, les règles décrites dans l'IEC 62541-3 sont utilisées.

S'il convient de fournir des structures de données simples, ce sont les *Propriétés* qui sont utilisées. Des exemples de *Propriétés* sont le *NamespaceArray* de l'*Objet Server* et le *MinSupportedSampleRate* de l'*Objet ServerCapabilities*.

Si les structures de données utilisées sont complexes, ce sont les *DataVariables* qui sont utilisées. Des exemples de *DataVariables* sont le *ServerStatus* de l'*Objet Server* et le *ServerDiagnosticsSummary* de l'*Objet ServerDiagnostics*.

A.5 Variables complexes utilisant des DataTypes complexes

Les *DataVariables* fournissant des structures de données complexes présentent leurs informations sous forme de *DataTypes* complexes, ainsi que des composantes dans l'*Espace d'Adresses*. Cela permet un accès à des valeurs simples ainsi qu'un accès immédiat à la totalité des informations dans un contexte transactionnel.

Par exemple, la *Variable ServerStatus* de l'*Objet Server* est modélisée comme une *DataVariable* complexe ayant le *ServerStatusDataType* fournissant toutes les informations relatives au statut du *Serveur*. Mais elle présente aussi le *CurrentTime* comme une *DataVariable* simple, car un client peut souhaiter lire uniquement l'heure courante du *Serveur* et n'est pas intéressé par les informations d'organisation, etc.

A.6 Variables complexes ayant une matrice

Un cas spécial de fourniture de structures de données complexe est une matrice de structures de données complexes. Le *SubscriptionDiagnosticsArrayType* est un exemple de la manière de modéliser cela. Il s'agit d'une matrice d'une structure de données complexe, fournissant les informations d'un abonnement. Étant donné qu'un *Serveur* a généralement plusieurs abonnements, il s'agit d'une matrice. Certains clients peuvent souhaiter lire immédiatement les informations de diagnostic relatives à tous les abonnements; par conséquent, elles sont modélisées sous la forme d'une matrice dans une *Variable*. D'autre part, un client peut être intéressé par une seule entrée de la structure complexe, telle que le *PublishRequestCount*. Par conséquent, chaque entrée de la matrice est également présentée individuellement comme une *DataVariable* complexe, ayant chaque entrée présentée comme une donnée simple.

Noter qu'il n'est jamais nécessaire de présenter les entrées individuelles d'une matrice pour accéder séparément à celles-ci. Les *Services* permettent déjà d'accéder à des entrées individuelles d'une matrice d'une *Variable*. Cependant, s'il convient d'utiliser également les entrées à d'autres fins dans l'*Espace d'Adresses,* par exemple avoir des *Références* ou des *Propriétés* supplémentaires ou présenter leur structure complexe à l'aide de *DataVariables,* il est utile de les présenter individuellement.

A.7 Informations redondantes

D'une manière générale, il convient d'éviter de fournir des informations redondantes. Mais pour répondre aux besoins de différents clients, cela peut être utile.

L'utilisation automatique de *DataVariables* complexes conduit à fournir des informations redondantes, car les informations sont fournies directement dans le *DataType* complexe de l'*Attribut Value* de la *Variable* complexe et aussi elles sont présentées individuellement dans les composantes de la *Variable* complexe.

Les informations de diagnostic relatives aux abonnements sont fournies en deux emplacements différents. Le premier emplacement est la *SubscriptionDiagnosticsArray* de l'*Objet ServerDiagnostics*, fournissant les informations pour tous les abonnements du *Serveur*. Le second emplacement est la *SubscriptionDiagnosticsArray* de chaque *Objet SessionDiagnosticsObject* individuel, fournissant seulement les abonnements de la session. Ceci est utile, car certains clients peuvent être intéressés uniquement par les abonnements regroupés par sessions, alors que d'autres clients peuvent souhaiter accéder immédiatement aux informations de diagnostic de toutes les sessions.

La *SessionDiagnosticsArray* et la *SessionSecurityDiagnosticsArray* de l'*Objet SessionsDiagnosticsSummary* ne présentent pas leurs entrées individuelles, bien qu'elles représentent une matrice de structures de données complexes. Les informations des entrées peuvent toutefois être accessibles individuellement en tant que composantes des *Objets SessionDiagnostics* fournis pour chaque session par l'*Objet SessionsDiagnosticsSummary*. Un client peut soit accéder directement aux matrices (ou à des parties des matrices), soit parcourir les *Objets SessionDiagnostics* pour obtenir des informations des entrées individuelles. Les informations ainsi fournies sont donc redondantes, mais les *Variables* contenant les matrices ne présentent pas leurs entrées individuelles.

A.8 Utilisation du BaseDataVariableType

Toutes les *DataVariables* utilisées pour présenter des structures de données complexes de *DataVariables* complexes ont le *BaseDataVariableType* comme définition de type si elles ne sont pas complexes elles-mêmes. La raison de cette approche est que les *DataVariables* complexes définissent déjà la sémantique des *DataVariables* contenantes et leur sémantique n'est pas utilisée dans un autre contexte. Elles ne sont pas censées être sous-typées, car il convient qu'elles reflètent la structure de données du *DataType* de la *DataVariable* complexe.

A.9 Sous-typage

Le sous-typage est utilisé pour modéliser des informations relatives à la prise en charge de la redondance par le *Serveur*. Sachant que les informations fournies doivent être différentes selon la redondance prise en charge par le *Serveur*, des sous-types du *ServerRedundancyType* sont utilisés à cet effet.

Le sous-typage est également utilisé comme un mécanisme d'extensibilité (voir A.10).

A.10 Mécanisme d'extensibilité

Les informations du *Serveur* sont étendues par d'autres parties de cette série de normes, par des spécifications d'accompagnement ou par des fournisseurs de *Serveurs*. Il existe des manières préférentielles pour la fourniture des informations supplémentaires.

Ne pas sous-typer des *DataTypes* pour fournir des informations supplémentaires relatives au *Serveur*. Les clients peuvent ne pas être en mesure de lire ces nouveaux *DataTypes* définis et ne peuvent pas récupérer les informations, y compris les informations de base. Si des informations sont ajoutées par plusieurs sources, la hiérarchie des *DataTypes* peut être difficile à maintenir. Noter que cette règle s'applique aux informations relatives au *Serveur*; dans d'autres scénarios, cela peut être un moyen utile d'ajouter de l'information.

Ajouter des *Objets* contenant des *Variables* ou ajouter des *Variables* à des *Objets* définis dans cette partie. Si, par exemple, des informations de diagnostic supplémentaires pour chaque abonnement sont nécessaires, ajouter une nouvelle *Variable* contenant une matrice avec une entrée pour chaque abonnement dans les mêmes emplacements où la *SubscriptionDiagnosticsArray* est utilisée.

Utiliser des sous-types du *ServerVendorCapabilityType* pour ajouter des informations relatives aux fonctions spécifiques au Serveur sur les *Objets ServerCapabilities*. Sachant que ce point de l'extensibilité est déjà défini dans cette partie, c'est là que les clients rechercheront des informations supplémentaires.

Utiliser un sous-type du *VendorServerInfoType* pour ajouter des informations spécifiques au serveur. Sachant qu'un *Objet* de ce type est déjà défini dans cette partie, c'est là que les clients rechercheront des informations spécifiques au serveur.

Annexe B
(normative)

Diagrammes d'états (StateMachines)

B.1 Généralités

Cette annexe décrit l'infrastructure de base pour modéliser les diagrammes d'états. Elle définit les *ObjectTypes*, *VariableTypes* et *ReferenceTypes* et explique comment il convient de les utiliser.

Cette annexe fait partie intégrante de la présente norme, c'est-à-dire que les types définis dans cette annexe sont à utiliser tels qu'ils sont définis. Cependant, il n'est pas exigé, mais il est fortement recommandé qu'un *Serveur* utilise ces types pour présenter ses diagrammes d'états. Les types définis peuvent être sous-typés pour affiner leur comportement.

Lorsqu'un *Serveur* présente son diagramme d'états en utilisant les types définis dans cette annexe, il peut fournir uniquement une vue simplifiée de son diagramme d'états interne, en masquant par exemple les sous-états ou en plaçant plusieurs états internes dans un état présenté.

Le domaine d'application des diagrammes d'états décrits dans cette annexe est la fourniture d'une fondation appropriée pour les diagrammes d'états nécessaires à l'IEC 62541-9 et à l'IEC 62541-10. Ne sont pas décrites les fonctionnalités plus complexes d'un diagramme d'états tel que les états parallèles, les bifurcations et réunions, les états d'historique, les choix et jonctions, etc. Cependant, le diagramme d'états de base défini dans cette annexe peut être étendu pour prendre en charge ces concepts.

Les articles ci-après décrivent des exemples de diagrammes d'états, définissent des diagrammes d'états dans le contexte de cette annexe et définissent la représentation de diagrammes d'états d'OPC UA. Enfin, certains exemples sont donnés pour les diagrammes d'états représentés dans OPC UA,

B.2 Exemples de diagrammes d'états finis

B.2.1 Diagramme d'états simple

L'exemple suivant fournit une vue d'ensemble des caractéristiques de base que prendront en charge les diagrammes d'états définis dans cette annexe. Dans le texte ci-dessous, il est donné un exemple plus complexe, qui prend également en charge des diagrammes de sous-états.

La Figure B.1 donne une vue d'ensemble sur un diagramme d'états simple. Il contient trois états "State1", "State2" et "State3". Il existe des transitions de "State1" à "State2", de "State2"à "State2", etc. Certaines de ces transitions fournissent des informations complémentaires quant aux causes (ou déclencheurs) de la transition, par exemple l'appel de la "Method1" pour la transition de "State1" à "State2". L'effet (ou action) de la transition peut aussi être spécifié, par exemple, la création d'un *Événement* du type "EventType1" dans la même transition. La notation utilisée pour identifier la cause consiste à l'énumérer tout simplement sur la transition, l'effet étant préfixé par un "/". Plusieurs causes ou effets sont séparés par une ",". Toutes les transitions n'ont pas nécessairement de cause ou d'effet, par exemple la transition entre "State2" et "State3".

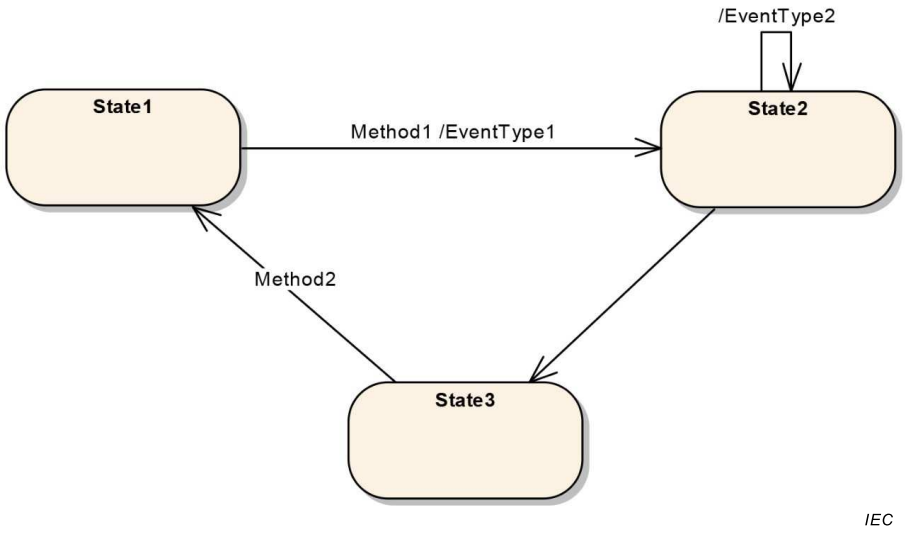


Figure B.1 – Exemple d'un diagramme d'états simple

Pour des raisons de simplicité, les diagrammes d'états décrits dans cette annexe prendront uniquement en charge les causes sous la forme des *Méthodes* de spécification à appeler et les effets sous la forme d'*EventTypes* des *Événements* générés. Cependant, l'infrastructure définie permet d'étendre cela afin de prendre en charge différentes causes et différents effets supplémentaires.

B.2.2 Diagramme d'états contenant des sous-états

La Figure B.2 montre un exemple de diagramme d'états où "State6" est un diagramme de sous-états. Cela signifie que, si le diagramme d'états global est dans l'état State6, cet état peut être distingué comme étant les sous-états "State7" ou "State8". Les diagrammes de sous-états peuvent être imbriqués, c'est-à-dire que "State7" peut être un autre diagramme de sous-état.

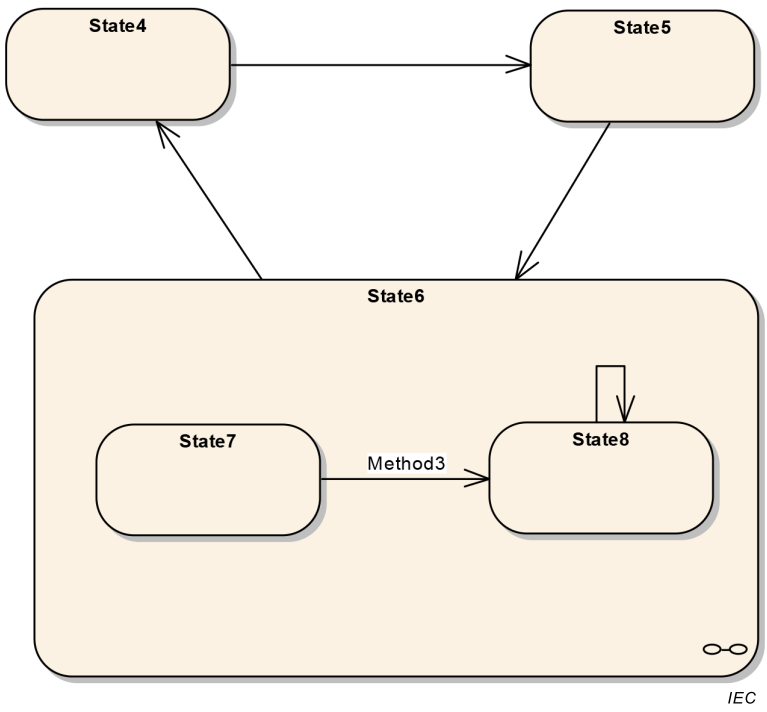


Figure B.2 – Exemple de diagramme d'états ayant un sous-diagramme

B.3 Définition de diagramme d'états

L'infrastructure des diagrammes d'états définis dans cette annexe traite seulement des fondamentaux des diagrammes d'états nécessaires pour venir à l'appui de l'IEC 62541-9 et de l'IEC 62541-10. L'intention est de maintenir les fondamentaux simples, mais extensibles.

Pour les diagrammes d'états définis dans cette annexe, l'hypothèse posée est que les diagrammes d'états sont typés et les instances d'un type ont leurs états et leur sémantique spécifiés par le type. Pour certains types, cela signifie que les états et les transitions sont fixes. Pour d'autres types, les états et les transitions peuvent être dynamiques ou inconnus. Un diagramme d'états où tous les états sont spécifiés explicitement par le type est appelé un diagramme d'états finis.

Par conséquent, une distinction est faite entre *StateMachineType* et *StateMachine,* et leurs sous-types tels que *FiniteStateMachineType*. Le *StateMachineType* spécifie une description du diagramme d'états, c'est-à-dire, ses états, transitions, etc. tandis que le *StateMachine* est une instance de *StateMachineType* et contient seulement l'état courant.

Chaque *StateMachine* contient des informations relatives à l'état courant. Si le *StateMachineType* a des *SubStateMachines*, le *StateMachine* contient aussi des informations relatives à l'état courant des *SubStateMachines*. Les *StateMachines* qui ont leurs états complètement définis par le type sont des instances d'un *FiniteStateMachineType*.

Chaque *FiniteStateMachineType* à un ou plusieurs *États*. Pour des raisons de simplicité, il ne sera pas fait de distinction entre des *États* différents comme les états de départ ou de fin.

Chaque *État* peut avoir un ou plusieurs *SubStateMachines*.

Chaque *FiniteStateMachineType* peut avoir une ou plusieurs *Transitions*. Une *Transition* est orientée et pointe d'un *État* vers un autre *État*.

Chaque *Transition* peut avoir une ou plusieurs *Causes*. Une *Cause* conduit un *FiniteStateMachine* à changer son *État* courant d'une source de *Transition* vers sa cible. Dans cette annexe, seuls les appels de *Method* sont spécifiés comme étant les *Causes* de *Transitions*. Il n'est pas nécessaire que les *Transitions* aient une *Cause*. Une *Transition* peut toujours être provoquée par quelque logique interne au serveur qui n'est pas présentée dans l'*Espace d'Adresses*.

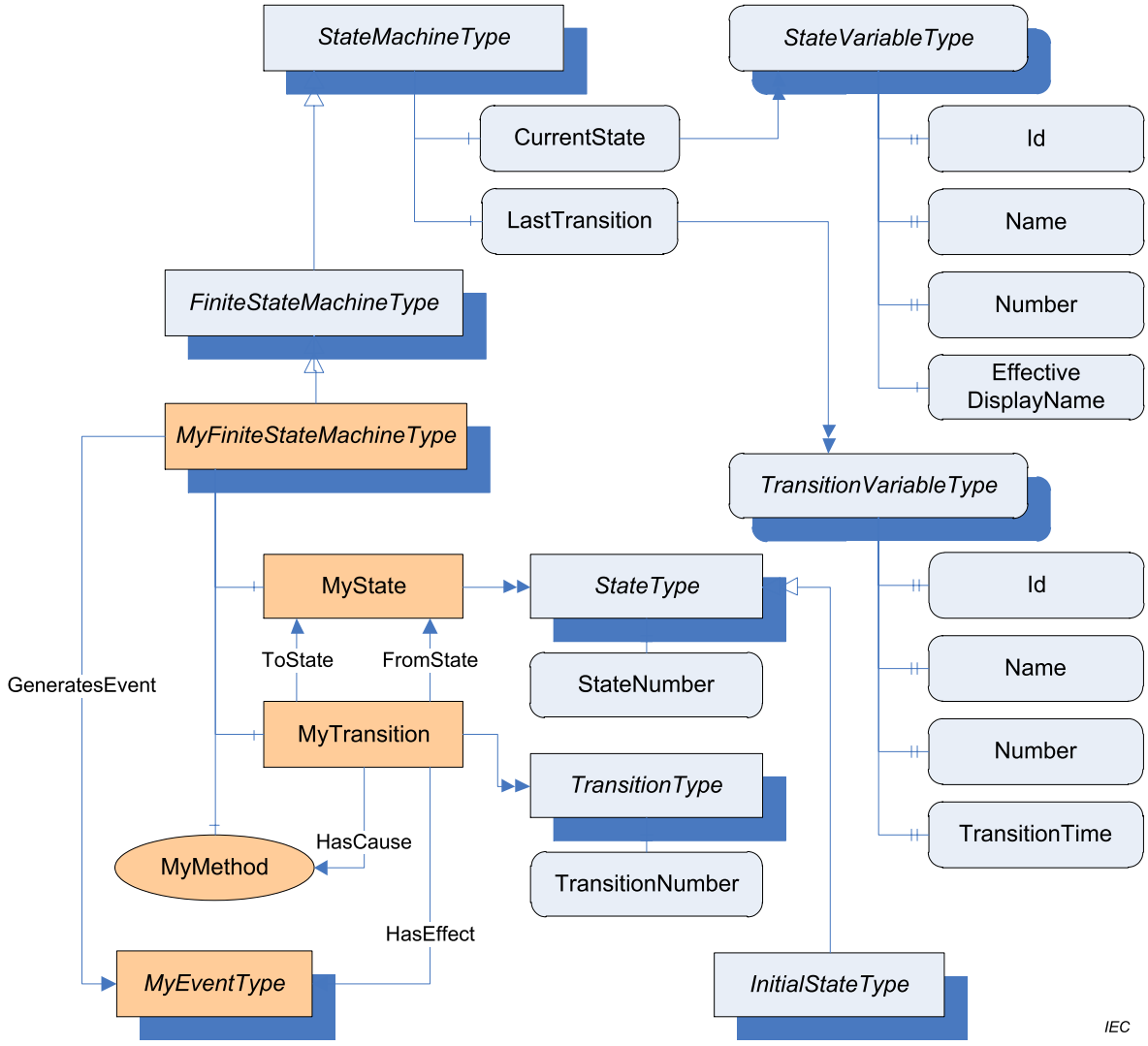
Chaque *Transition* peut avoir une ou plusieurs *Actions*. Une *Action* a lieu si la *Transition* est utilisée pour changer l'*État* d'un *StateMachine*. Dans cette Annexe, seule la création d'*Événements* est spécifiée comme étant des *Actions* d'une *Transition*. Une *Transition* n'est pas tenue de présenter d'*Actions* dans l'*Espace d'Adresses.*

Bien que cette annexe ne spécifie que des concepts simples pour les diagrammes d'états, l'infrastructure fournie est extensible. Au besoin, des *États* spéciaux peuvent être définis ainsi que des *Causes* ou des *Actions* supplémentaires.

B.4 Représentation des diagrammes d'états dans l'Espace d'Adresses

B.4.1 Vue d'ensemble

Les types définis dans cette Annexe sont illustrés à la Figure B.3. Le *MyFiniteStateMachineType* est un exemple minimal qui illustre comment ces *Types* peuvent être utilisés pour décrire un *StateMachine*. Voir l'IEC 62541-9 et l'IEC 62541-10 pour d'autres exemples de *StateMachines*.



Anglais	Français
Name	Nom
Number	Nombre

Figure B.3 – Modèle d'informations StateMachine

B.4.2 StateMachineType

Le *StateMachineType* est l'*ObjectType* de base pour tous les *StateMachineTypes*. Il définit une seule *Variable* qui représente l'état courant du diagramme. Une instance de cet *ObjectType* doit générer un *Événement* chaque fois qu'il se produit un changement d'état significatif. Le *Serveur* décide des changements d'état qui sont significatifs. Les *Serveurs* doivent utiliser le *ReferenceType GeneratesEvent* pour indiquer le ou les *Événements* qui peuvent être produits par le *StateMachine*.

Les sous-types peuvent ajouter des *Méthodes* qui altèrent l'état du diagramme. L'*Attribut Executable* est utilisé pour indiquer si, oui ou non, la *Méthode* est valide pour un état courant donné du diagramme. La création d'*AuditEvents* pour les *Méthodes* est définie dans l'IEC 62541-4. Un *StateMachine* peut ne pas être actif. Dans ce cas, les *Variables CurrentState* et *LastTransition* doivent avoir un statut égal à *Bad_StateNotActive* (voir Tableau B.17).

Les sous-types peuvent ajouter des composantes qui sont des instances des *StateMachineTypes*. Ces composantes sont considérées être des sous-états du

StateMachine. Les *SubStateMachines* ne sont actifs que lorsque le diagramme parent est dans un état approprié.

Les *Événements* produits par des *SubStateMachines* peuvent être supprimés par le diagramme parent. Dans certains cas, le diagramme parent produit un seul *Événement* qui reflète les changements dans plusieurs *SubStateMachines*.

Le *FiniteStateMachineType* est un sous-type de *StateMachineType* qui fournit un mécanisme pour définir explicitement les états et les transitions. Il convient qu'un *Serveur* utilise ce mécanisme s'il connaît les états possibles et si le diagramme d'états n'est pas trivial. Le *FiniteStateMachineType* est défini en B.4.5.

Le *StateMachineType* est défini de façon formelle dans le Tableau B.1.

Tableau B.1 – Définition de StateMachineType

Attribut	**Valeur**				
BrowseName	StateMachineType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2. Noter qu'une *Reference* à ce sous-type n'apparaît pas dans la définition du BaseObjectType.					
HasSubtype	ObjectType	FiniteStateMachineType	Défini en B.4.5		
HasComponent	Variable	CurrentState	LocalizedText	StateVariableType	Obligatoire
HasComponent	Variable	LastTransition	LocalizedText	TransitionVariableType	Facultatif

CurrentState stocke l'état courant d'une instance du *StateMachineType*. *CurrentState* fournit un nom lisible en clair pour l'état courant qui peut ne pas être adapté à une utilisation dans la logique de commande d'application. Il convient que les applications utilisent l'*Id Property* de *CurrentState* si elles ont besoin d'un identificateur unique pour l'état.

LastTransition stocke la dernière transition qui a eu lieu dans une instance du *StateMachineType*. *LastTransition* fournit un nom lisible en clair pour la dernière transition qui peut ne pas être adaptée à une utilisation dans la logique de commande d'application. Il convient que les applications utilisent l'*Id Property* de *LastTransition* si elles ont besoin d'un identificateur unique pour la transition.

B.4.3 StateVariableType

Le *StateVariableType* est le *VariableType* de base pour les *Variables* qui stockent l'état courant d'un *StateMachine* sous la forme d'un nom lisible en clair.

Le *StateVariableType* est défini de façon formelle dans le Tableau B.2.

Tableau B.2 – Définition de StateVariableType

Attribut	**Valeur**				
BrowseName	StateVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseDataVariableType* défini en 7.4. Noter qu'une *Référence* à ce sous-type n'apparaît pas dans la définition du *BaseDataVariableType*.					
HasSubtype	VariableType	FiniteStateVariableType	Défini en B.4.6		
HasProperty	Variable	ID	BaseDataType	PropertyType	Obligatoire
HasProperty	Variable	Nom	QualifiedName	PropertyType	Facultatif
HasProperty	Variable	Number	UInt32	PropertyType	Facultatif
HasProperty	Variable	EffectiveDisplayName	LocalizedText	PropertyType	Facultatif

Id est un nom qui identifie de façon unique l'état courant dans le *StateMachineType*. Un sous-type peut limiter le *DataType*.

Name (Nom) est un *QualifiedName* qui identifie de façon unique l'état courant du *StateMachineType*.

Number (Nombre) est un nombre entier qui identifie de façon unique l'état courant du *StateMachineType*.

EffectiveDisplayName contient un nom lisible en clair pour l'état courant du diagramme d'états après avoir pris en compte l'état de tous les éventuels *SubStateMachines*. Il n'est spécifié aucune règle concernant l'état ou le sous-état qu'il convient d'utiliser. Cela est du ressort du *Serveur* et dépend de la sémantique du *StateMachineType*.

Les *StateMachines* produisent des *Événements* qui peuvent inclure l'état courant d'un *StateMachine*. Dans ce cas, les *Serveurs* doivent fournir toutes les *Propriétés* facultatives du *StateVariableType* dans l'*Événement*, même si elles ne sont pas fournies sur les instances dans l'*Espace d'Adresses*.

B.4.4 TransitionVariableType

Le *TransitionVariableType* est le *VariableType* de base pour les *Variables* qui stockent une *Transition* qui a eu lieu dans un *StateMachine* sous la forme d'un nom lisible en clair.

Le *SourceTimestamp* pour la valeur spécifie le moment où la *Transition* a eu lieu. Cette valeur peut également être présentée avec la *Propriété TransitionTime*.

Le *TransitionVariableType* est défini de façon formelle dans le Tableau B.3.

Tableau B.3 – Définition de TransitionVariableType

Attribut	**Valeur**				
BrowseName	TransitionVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	FALSE (faux)				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du *BaseDataVariableType* défini au 7.4. A noter qu'une *Référence* à ce sous-type n'apparaît pas dans la définition du *BaseDataVariableType*.					
HasSubtype	VariableType	FiniteTransitionVariableType	Défini en B.4.7		
HasProperty	Variable	ID	BaseDataType	PropertyType	Obligatoire
HasProperty	Variable	Nom	QualifiedName	PropertyType	Facultatif
HasProperty	Variable	Number	UInt32	PropertyType	Facultatif
HasProperty	Variable	TransitionTime	UTCTime	PropertyType	Facultatif
HasProperty	Variable	EffectiveTransitionTime	UtcTime	PropertyType	Facultatif

Id est un nom qui identifie de façon unique une *Transition* dans le *StateMachineType*. Un sous-type peut limiter le *DataType*.

Name est un *QualifiedName* qui identifie de façon unique une transition du *StateMachineType*.

Number est un nombre entier qui identifie de façon unique une transition du *StateMachineType*.

TransitionTime spécifie le moment où la transition a eu lieu.

EffectiveTransitionTime spécifie le moment auquel l'état courant ou un de ses sous-états a été saisi. Si, par exemple, un StateA est actif et – tout en étant actif – qu'il commute plusieurs fois entre ses sous-états SubA et SubB, le *TransitionTime* reste au moment où StateA est devenu actif tandis que *EffectiveTransitionTime* change à chaque changement de sous-état.

B.4.5 FiniteStateMachineType

Le *FiniteStateMachineType* est l'*ObjectType* de base pour les *StateMachines* qui définissent explicitement les *États* et *Transitions* possibles. Une fois que les *États* sont définis, les sous-types ne doivent pas ajouter de nouveaux *États* (voir B.4.18).

Les *États* du diagramme sont représentés avec des instances de l'*ObjectType StateType.* Chaque *État* doit avoir un *BrowseName* qui est unique dans le *StateMachine* et doit avoir un *StateNumber* qui doit aussi être unique parmi tous les *États* définis dans le *StateMachine*. Garder à l'esprit que les *États* dans un *SubStateMachine* peuvent avoir le même *StateNumber* ou *BrowseName* que les *États* dans le diagramme parent. Un sous-type concret de *FiniteStateMachineType* doit définir au moins un *État*.

Un *StateMachine* peut définir un seul *État* qui est une instance d'*InitialStateType*. Cet *État* est l'*État* dans lequel entre le diagramme lorsqu'il est activé.

Les *Transitions* qui peuvent se produire sont représentées avec des instances du *TransitionType*. Chaque *Transition* doit avoir un *BrowseName* qui est unique dans le *StateMachine* et peut avoir un *TransitionNumber* qui doit aussi être unique parmi toutes les *Transitions* définies dans le *StateMachine*.

L'*État* initial pour une *Transition* est un *Objet StateType* qui est la cible d'une *Référence FromState*. L'*État* final pour une *Transition* est un *Objet StateType* qui est la cible d'une *Référence ToState*. Les *Références FromState* et *ToState* doivent toujours être spécifiées.

Une *Transition* peut produire un *Événement*. L'*Événement* est indiqué par une *Référence HasEffect* à un sous-type de *BaseEventType*. Le *StateMachineType* doit avoir des *Références GeneratesEvent* aux cibles d'une *Référence HasEffect* pour chacune de ses *Transitions*.

Un *FiniteStateMachineType* peut définir des *Méthodes* qui font qu'une transition se produise. Ces *Méthodes* sont des cibles des *Références HasCause* pour chacune des *Transitions* qui peuvent être enclenchées par la *Méthode*. L'*Attribut Executable* pour une *Méthode* est utilisé pour indiquer si, oui ou non, l'*État* courant du diagramme permet d'appeler la *Méthode*.

Un *FiniteStateMachineType* peut avoir des diagrammes de sous-états qui sont représentés comme des instances des *ObjectTypes StateMachineType*. Chaque *État* doit avoir une *Référence HasSubStateMachine* à l'*Objet StateMachineType* qui représente les *États* enfants. Le *SubStateMachine* n'est pas actif si l'*État* parent n'est pas actif. Dans ce cas, les *Variables CurrentState* et *LastTransition* du *SubStateMachine* doivent avoir un statut égal à *Bad_StateNotActive* (voir Tableau B.17).

Le *FiniteStateMachineType* est défini de façon formelle dans le Tableau B.4.

Tableau B.4 – Définition de FiniteStateMachineType

Attribut	**Valeur**				
BrowseName	FiniteStateMachineType				
IsAbstract	TRUE (vrai)				
Références	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du StateMachineType défini dans en 6.2.					
HasComponent	Variable	CurrentState	LocalizedText	FiniteStateVariableType	Obligatoire
HasComponent	Variable	LastTransition	LocalizedText	FiniteTransitionVariableType	Facultatif

B.4.6 FiniteStateVariableType

Le *FiniteStateVariableType* est un sous-type de *StateVariableType* et il est utilisé pour stocker l'état courant d'un Finite*StateMachine* sous la forme d'un nom lisible en clair.

Le *FiniteStateVariableType* est défini de façon formelle dans le Tableau B.5.

Tableau B.5 – Définition de FiniteStateVariableType

Attribut	**Valeur**				
BrowseName	FiniteStateVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	FALSE (faux)				
Références	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du *StateVariableType* défini en B.4.3					
HasProperty	Variable	ID	NodeId	PropertyType	Obligatoire

Id est hérité du *StateVariableType* et neutralisé pour refléter le *DataType* requis. Cette valeur doit être le *NodeId* de l'un des *Objets State* du *FiniteStateMachineType.*

La *Propriété Name* est héritée de *StateVariableType*. Sa valeur doit être le *BrowseName* de l'un des *Objets State* du *FiniteStateMachineType*.

La *Propriété Number* est héritée de *StateVariableType*. Sa valeur doit être le *StateNumber* de l'un des *Objets State* du *FiniteStateMachineType*.

B.4.7 FiniteTransitionVariableType

Le *FiniteTransitionVariableType* est un sous-type de *TransitionVariableType* et il est utilisé pour stocker une *Transition* qui a eu lieu dans un *FiniteStateMachine* sous la forme d'un nom lisible en clair.

Le *FiniteTransitionVariableType* est défini de façon formelle dans le Tableau B.6.

Tableau B.6 – Définition de FiniteTransitionVariableType

Attribut	**Valeur**				
BrowseName	FiniteTransitionVariableType				
DataType	LocalizedText				
ValueRank	−1 (−1 = Scalar)				
IsAbstract	FALSE (faux)				
Références	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du *TransitionVariableType* défini en B.4.4. A noter qu'une *Référence* à ce sous-type n'apparaît pas dans la définition du *BaseDataVariableType*.					
HasProperty	Variable	ID	NodeId	PropertyType	Obligatoire

Id est hérité du *TransitionVariableType* et neutralisé pour refléter le *DataType* requis. Cette valeur doit être le *NodeId* de l'un des *Objets Transition* du *FiniteStateMachineType*.

La *Propriété Name* est héritée du *TransitionVariableType*. Sa valeur doit être le *BrowseName* de l'un des *Objets Transition* du *FiniteStateMachineType*.

La *Propriété Number* est héritée du *TransitionVariableType*. Sa valeur doit être le *TransitionNumber* de l'un des *Objets Transition* du *FiniteStateMachineType*.

B.4.8 StateType

Les *États* d'un *FiniteStateMachine* sont représentés comme des *Objets* du *StateType.*

Le *StateType* est défini de façon formelle dans le Tableau B.7.

Tableau B.7 – Définition de StateType

Attribut	**Valeur**				
BrowseName	StateType				
IsAbstract	False (faux)				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2. À noter qu'une *Référence* à ce sous-type n'apparaît pas dans la définition du BaseObjectType.					
HasProperty	Variable	StateNumber	UInt32	PropertyType	Obligatoire
HasSubtype	ObjectType	InitialStateType	Défini en B.4.9		

B.4.9 InitialStateType

L'*InitialStateType* est un sous-type du *StateType* et il est défini de façon formelle dans le Tableau B.8. Un *Objet* d'*InitialStateType* représente l'*État* dans lequel entre un *FiniteStateMachine* lorsqu'il est activé. Chaque *FiniteStateMachine* peut avoir au maximum un *État* d'*InitialStateType*, mais un *FiniteStateMachine* peut ne pas avoir d'*État* de ce type.

Un *SubStateMachine* entre dans son état initial chaque fois que l'état parent est accédé. Cependant, un diagramme d'états peut définir une transition qui passe directement à un état du *SubStateMachine*. Dans ce cas, le *SubStateMachine* entre dans l'*État* en question au lieu de l'*État* initial. Les deux scénarios sont illustrés à la Figure B.4. La transition de State5 à State6 amène le *SubStateMachine* à entrer dans l'*État* initial (State7); cependant, la transition de State4 à State8 amène le diagramme parent à passer à State6 et le *SubStateMachine* passera à State8.

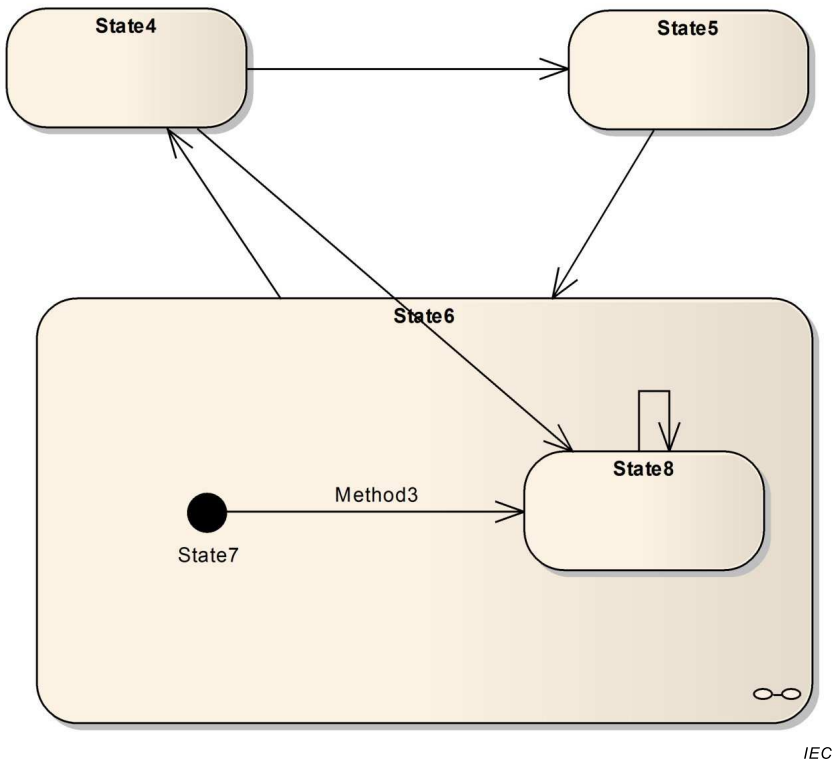


Figure B.4 – Exemple d'État initial d'un sous-diagramme

S'il n'existe pas d'état initial d'un *SubStateMachine* et l'*État* ayant le *SubStateMachine* est accédé directement, l'*État* du *SubStateMachine* est spécifique au serveur.

Tableau B.8 – Définition d'InitialStateType

Attribut	**Valeur**				
BrowseName	InitialStateType				
IsAbstract	FALSE (faux)				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *StateType* défini en B.4.8					

B.4.10 TransitionType

Les *Transitions* d'un *FiniteStateMachine* sont représentées comme des *Objets* de l'*ObjectType TransitionType* défini de façon formelle dans le Tableau B.9.

Chaque *Transition* valide doit avoir exactement une seule *Référence FromState* et exactement une seule *Référence ToState*, chacune pointant vers un *Objet* de l'*ObjectType StateType*.

Chaque *Transition* peut avoir une ou plusieurs *Références HasCause* pointant vers la cause qui déclenche la *Transition*.

Chaque *Transition* peut avoir une ou plusieurs *Références HasEffect* pointant sur les effets qui apparaissent lorsque la *Transition* a été déclenchée.

Tableau B.9 – Définition de TransitionType

Attribut	**Valeur**				
BrowseName	TransitionType				
IsAbstract	FALSE (faux)				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2. À noter qu'une *Référence* à ce sous-type n'apparaît pas dans la définition du BaseObjectType.					
HasProperty	Variable	TransitionNumber	UInt32	PropertyType	Obligatoire

B.4.11 FromState

Le *ReferenceType FromState* est un *ReferenceType* concret et peut être directement utilisé. Il est un sous-type des *NonHierarchicalReferences*.

La sémantique de ce *ReferenceType* est de pointer d'une *Transition* vers l'*État* de départ que la *Transition* relie.

Le *SourceNode* de ce *ReferenceType* doit être un *Objet* de l'*ObjectType TransitionType* ou de l'un de ses sous-types. Le *TargetNode* de ce *ReferenceType* doit être un *Objet* de l'*ObjectType StateType* ou de l'un de ses sous-types.

La représentation du *ReferenceType FromState* dans l'*Espace d'Adresses* est spécifiée dans le Tableau B.10.

Tableau B.10 – ReferenceType FromState

Attributs	Valeur		
BrowseName	FromState		
InverseName	ToTransition		
Symmetric	False (faux)		
IsAbstract	False (faux)		
Références	**NodeClass**	**BrowseName**	**Comment**

B.4.12 ToState

Le *ReferenceType ToState* est un *ReferenceType* concret et peut être directement utilisé. Il est un sous-type des *NonHierarchicalReferences*.

La sémantique de ce *ReferenceType* est de pointer d'une *Transition* vers l'*État* de fin relié par la *Transition*.

Le *SourceNode* de ce *ReferenceType* doit être un *Objet* de l'*ObjectType TransitionType* ou de l'un de ses sous-types. Le *TargetNode* de ce *ReferenceType* doit être un *Objet* de l'*ObjectType StateType* ou de l'un de ses sous-types.

Les *Références* de ce *ReferenceType* peuvent seulement être présentées comme unidirectionnelles. Parfois, cela est exigé, par exemple, si une *Transition* pointe vers un *État* d'un sous-diagramme.

La représentation du *ReferenceType ToState* dans l'*Espace d'Adresses* est spécifiée dans le Tableau B.11.

Tableau B.11 – ReferenceType ToState

Attributs	Valeur		
BrowseName	ToState		
InverseName	FromTransition		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

B.4.13 HasCause

Le *ReferenceType HasCause* est un *ReferenceType* concret et peut être directement utilisé. C'est un sous-type des *NonHierarchicalReferences*.

La sémantique de ce *ReferenceType* est de pointer d'une *Transition* vers quelque chose qui provoque la *Transition*. Dans cette annexe, les *Méthodes* sont seulement définies comme étant des *Causes*. Cependant, le *ReferenceType* ne se limite pas à pointer sur des *Méthodes*. Les Méthodes référencées peuvent pointer sur une Méthode du StateMachineType, mais n'y sont toutefois pas tenues. Par exemple, il est admis de pointer sur une Méthode de redémarrage à l'échelle du serveur, ce qui conduit le diagramme d'états à entrer dans son état initial.

Le *SourceNode* de ce *ReferenceType* doit être un *Objet* de l'*ObjectType TransitionType* ou de l'un de ses sous-types. Le *TargetNode* peut être de n'importe quelle *NodeClass*.

La représentation du *ReferenceType HasCause* dans l'*Espace d'Adresses* est spécifiée dans le Tableau B.12.

Tableau B.12 – ReferenceType HasCause

Attributs	Valeur		
BrowseName	HasCause		
InverseName	MayBeCausedBy		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

B.4.14 HasEffect

Le *ReferenceType HasEffect* est un *ReferenceType* concret et peut être directement utilisé. Il est un sous-type des *NonHierarchicalReferences*.

La sémantique de ce *ReferenceType* est de pointer d'une *Transition* vers quelque chose qui sera effectué lorsque la *Transition* sera déclenchée. Dans cette annexe, les *EventTypes* sont seulement définis comme étant des *Actions*. Cependant, le *ReferenceType* ne se limite pas à pointer sur des *EventTypes*.

Le *SourceNode* de ce *ReferenceType* doit être un *Objet* de l'*ObjectType TransitionType* ou de l'un de ses sous-types. Le *TargetNode* peut être de n'importe quelle *NodeClass*.

La représentation du *ReferenceType HasEffect* dans l'*Espace d'Adresses* est spécifiée dans le Tableau B.13.

Tableau B.13 – ReferenceType HasEffect

Attributs	Valeur		
BrowseName	HasEffect		
InverseName	MayBeEffectedBy		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

B.4.15 HasSubStateMachine

Le *ReferenceType HasSubStateMachine* est un *ReferenceType* concret et peut être directement utilisé. Il est un sous-type des *NonHierarchicalReferences*.

La sémantique de ce *ReferenceType* est de pointer d'un *État* vers une instance d'un *StateMachineType* qui représente les sous-états pour l'*État*.

Le *SourceNode* de ce *ReferenceType* doit être un *Objet* de l'*ObjectType StateType*. Le *TargetNode* doit être un *Objet* de l'*ObjectType StateMachineType* ou de l'un de ses sous-types. Chaque *Objet* peut être le *TargetNode* d'une *Référence HasSubStateMachine* au maximum.

Le *SourceNode* (l'état) et le *TargetNode* (le *SubStateMachine*) doivent appartenir au même *StateMachine*, c'est-à-dire que tous deux doivent être référencés à partir du même *Objet* du type *StateMachineType* en utilisant une *Référence HasComponent* ou un sous-type de *HasComponent*.

La représentation du *ReferenceType HasSubStateMachine* dans l'*Espace d'Adresses* est spécifiée dans le Tableau B.14.

Tableau B.14 – ReferenceType HasSubStateMachine

Attributs	Valeur		
BrowseName	HasSubStateMachine		
InverseName	SubStateMachineOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Comment**

B.4.16 TransitionEventType

Le *TransitionEventType* un sous-type du *BaseEventType*. Il peut être utilisé pour générer un *Événement* signalant qu'une *Transition* d'un *StateMachine* a été déclenchée. Il est défini de façon formelle dans le Tableau B.15.

Tableau B.15 – TransitionEventType

Attribut	Valeur				
BrowseName	TransitionEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseEventType* de base défini en 6.4.2					
HasComponent	Variable	Transition	LocalizedText	TransitionVariableType	Obligatoire
HasComponent	Variable	FromState	LocalizedText	StateVariableType	Obligatoire
HasComponent	Variable	ToState	LocalizedText	StateVariableType	Obligatoire

Le *TransitionEventType* hérite des *Propriétés* du *BaseEventType*.

La *Propriété* héritée du *SourceNode* doit être remplie avec le *NodeId* de l'instance du *StateMachine* où la *Transition* a lieu. Si la *Transition* a lieu dans un *SubStateMachine*, le *NodeId* du *SubStateMachine* est alors à utiliser. Si la Transition a lieu entre un *StateMachine* et un *SubStateMachine*, le *NodeId* du *StateMachine* est à utiliser, indépendamment du sens de la *Transition*.

Transition identifie la *Transition* qui déclenche l'*Événement*.

FromState identifie l'*État* avant la *Transition*.

ToState identifie l'*État* après la *Transition*.

B.4.17 AuditUpdateStateEventType

L'*AuditUpdateStateEventType* est un sous-type d'*AuditUpdateMethodEventType*. Il peut être utilisé pour générer un *Événement* identifiant qu'une *Transition* d'un *StateMachine* a été déclenchée. Il est défini de façon formelle dans le Tableau B.16.

Tableau B.16 – AuditUpdateStateEventType

Attribut	Valeur				
BrowseName	AuditUpdateStateEventType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditUpdateMethodEventType* défini en 6.4.27					
HasProperty	Variable	OldStateId	BaseDataType	PropertyType	Obligatoire
HasProperty	Variable	NewStateId	BaseDataType	PropertyType	Obligatoire

L'*AuditUpdateStateEventType* hérite des *Propriétés* de l'*AuditUpdateMethodEventType*.

La *Propriété* héritée *SourceNode* doit être remplie avec le *NodeId* de l'instance *StateMachine* où l'*État* a changé. Si l'*État* a changé dans un *SubStateMachine*, le *NodeId* du *SubStateMachine* est alors à utiliser.

Il convient que le *SourceName* pour les *Événements* de ce type soit l'action qui a généré l'événement (par exemple le nom d'une méthode). Si l'action a été générée par un appel de *Méthode*, il convient que le *SourceName* soit le nom de la *Méthode* préfixée par “Method/”.

OldStateId reflète l'*Id* de l'état avant le changement.

NewStateId reflète le nouveau *Id* de l'état après le changement.

B.4.18 Restrictions spéciales sur le sous-typage des StateMachines

En général, toutes les règles du sous-typage s'appliquent pour les types *StateMachine* également. Certaines règles supplémentaires s'appliquent aux types *StateMachine*. Si un type StateMachine n'est pas abstrait, ses sous-types ne doivent pas changer son comportement. Cela signifie que dans ce cas, un sous-type ne doit pas ajouter d'*États* et il ne doit pas ajouter de *Transitions* entre ses *États*. Cependant, un sous-type peut ajouter des *SubStateMachines*, peut ajouter des *Transitions* d'*États* vers les *États* du *SubStateMachine* et peut ajouter des *Causes* et des *Actions* à une *Transition*. En outre, un sous-type d'un type *StateMachine* ne doit pas enlever des *États* ou des *Transitions*.

B.4.19 StatusCodes spécifiques pour StateMachines

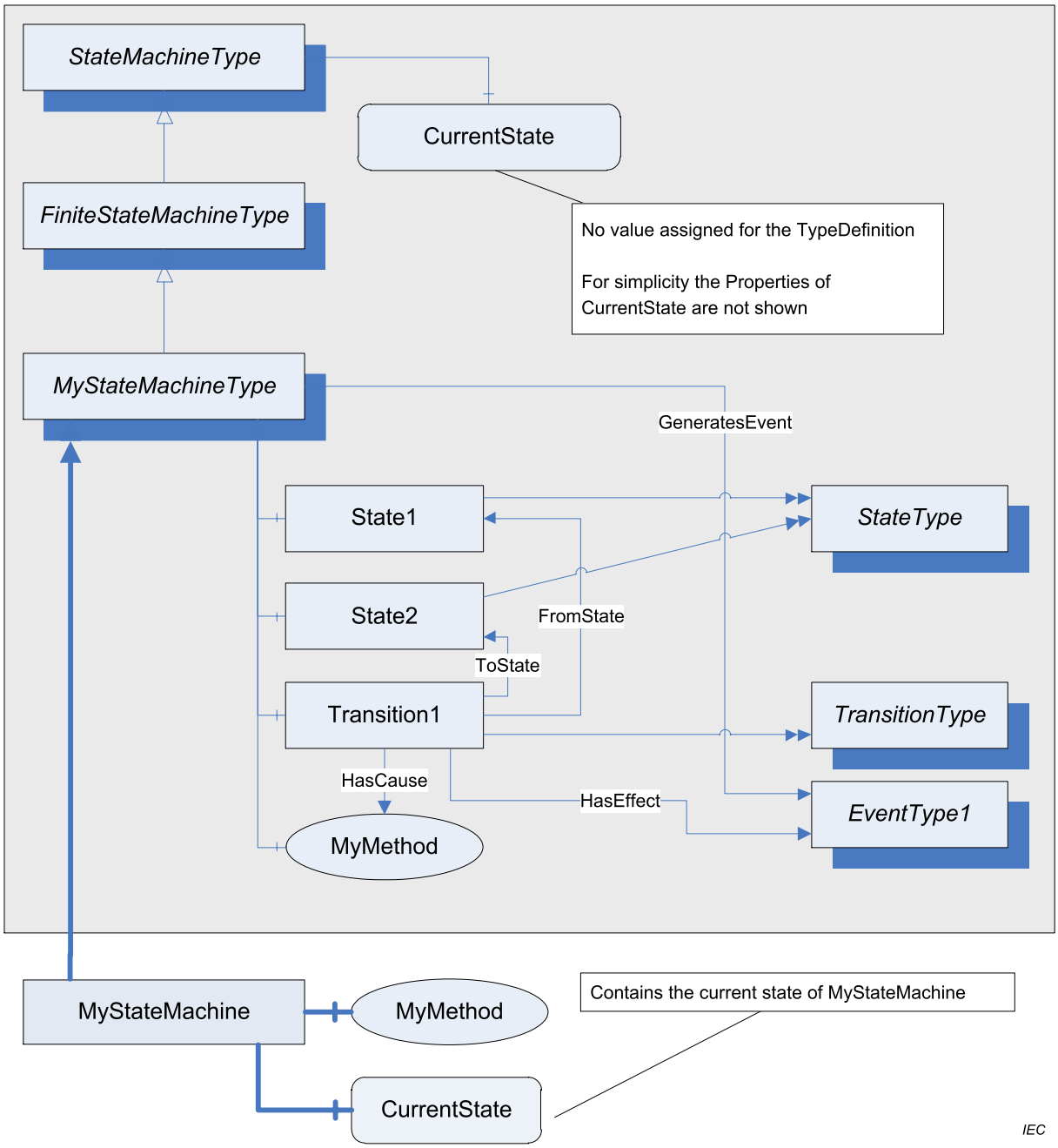
Le Tableau B.17 définit des *StatusCodes* spécifiques utilisés pour les *StateMachines*.

Tableau B.17 – StatusCodes spécifiques pour StateMachines

Id symbolique	Description
Bad_StateNotActive	L'état accédé n'est pas actif.

B.5 Exemples de StateMachines dans l'Espace d'Adresses

B.5.1 StateMachineType utilisant la relation d'héritage



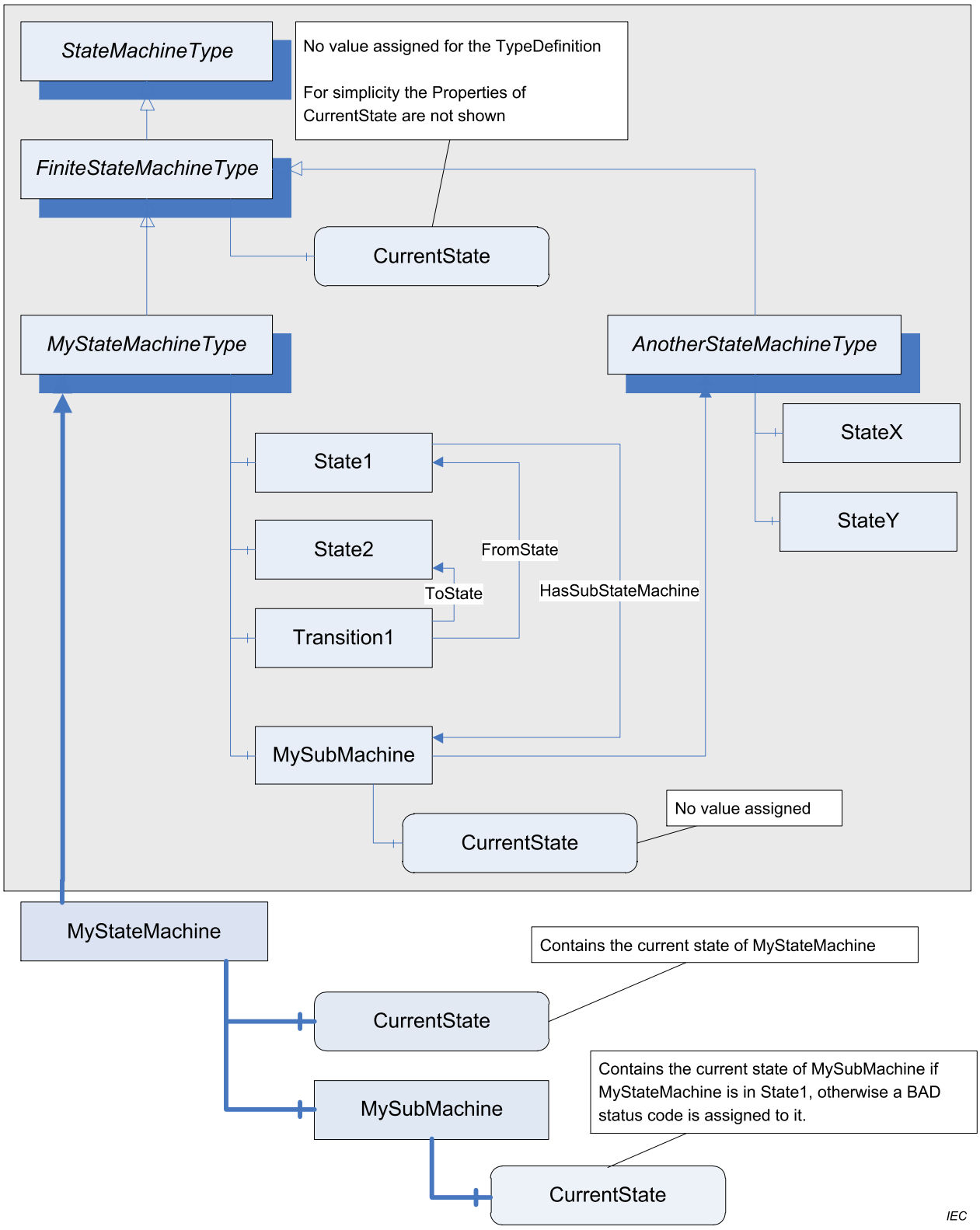
Anglais	Français
No value assigned for the TypeDefinition	Aucune valeur affectée pour le TypeDefinition
For simplicity the Properties of CurrentState are not shown	Pour des raisons de simplicité, les propriétés de CurrentState ne sont pas montrées
Contains the current state of MyStateMachine	Contient l'état courant de MyStateMachine

Figure B.5 – Exemple d'un StateMachineType utilisant la relation d'héritage

Dans la Figure B.5, un exemple de *StateMachine* est donné en utilisant la Notation définie dans l'IEC 62541-3. En premier lieu, un nouveau *StateMachineType* est défini, appelé "MyStateMachineType", héritant du *FiniteStateMachineType* de base. Il contient deux *États*, "State1" et "State2" et une *Transition* "Transition1" entre ceux-ci. La *Transition* pointe sur une *Méthode* "MyMethod" comme la *Cause* de la *Transition* et sur un *EventType* "EventType1" comme l'*Action* de la *Transition*.

Des instances de “MyStateMachineType” peuvent être créées, par exemple “MyStateMachine”. Elle a une *Variable* “CurrentState” représentant l’*État* courant. L'*Objet* “MyStateMachine” inclut seulement les *Nœuds* qui présentent des informations spécifiques à l'instance.

B.5.2 StateMachineType avec un sous-diagramme utilisant la relation d'héritage



Anglais	Français
No value assigned for the TypeDefinition	Aucune valeur affectée pour le TypeDefinition
For simplicity the Properties of CurrentState are not shown	Pour des raisons de simplicité, les propriétés de CurrentState ne sont pas montrées
Contains the current state of MyStateMachine	Contient l'état courant de MyStateMachine

Anglais	Français
No value assigned	Aucune valeur affectée
Contains the current state of MySubMachine if MyStateMachine is in State1, otherwise a BAD status code is assigned to it	Contient l'état courant de MySubMachine si MyStateMachine est dans l'état State1; autrement un code de statut BAD ("mauvais") lui est affecté.

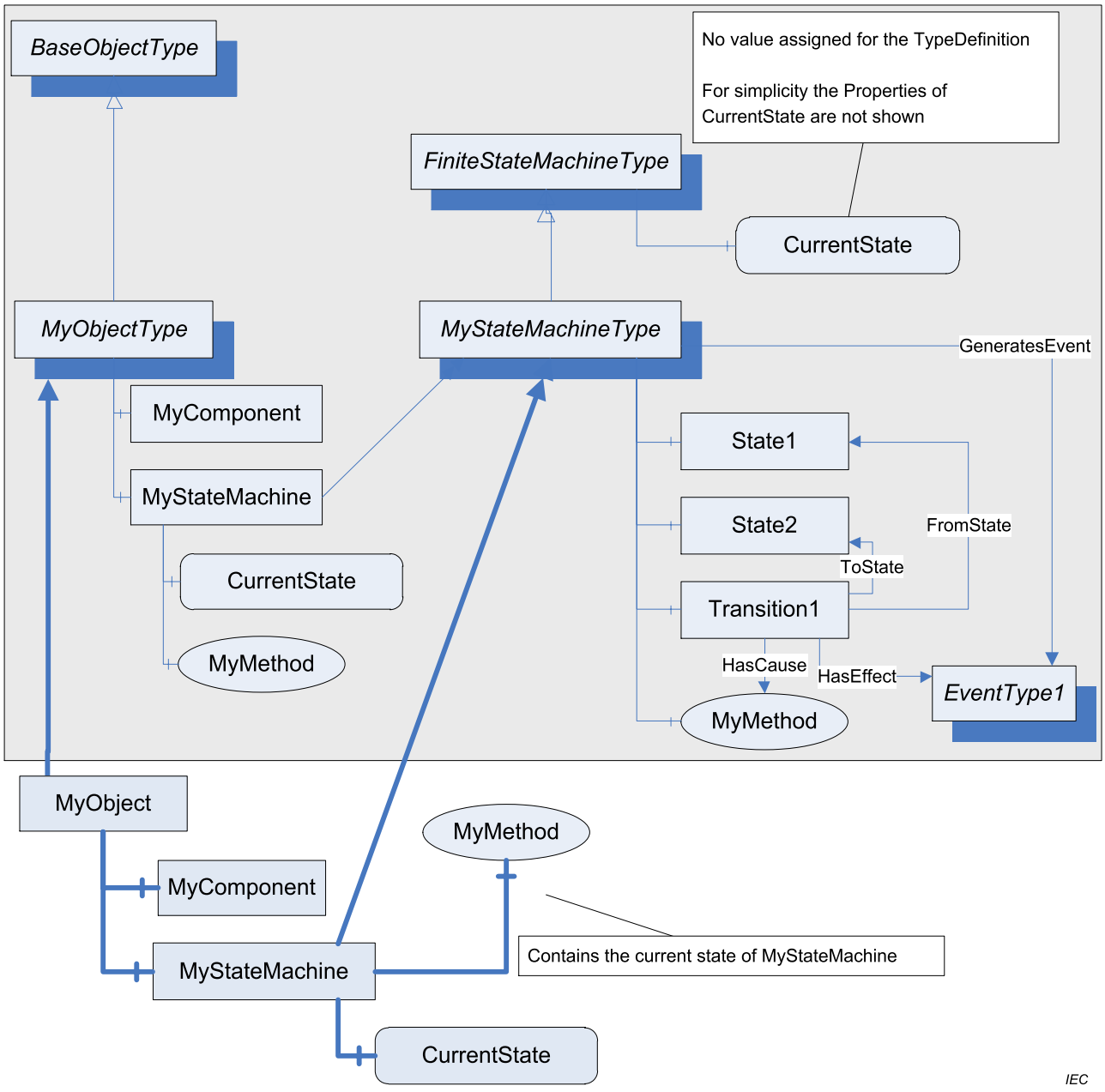
Figure B.6 – Exemple d'un StateMachineType avec un SubStateMachine utilisant la relation d'héritage

La Figure B.6 donne un exemple de *StateMachineType* ayant un *SubStateMachine* pour son "State1". Pour des raisons de simplicité, il n'est montré aucune cause et aucun effet, ni les informations relatives au type pour les *États* ou les *Règles de modélisation*.

Le "MyStateMachineType" contient un *Objet* "MySubMachine" du type "AnotherStateMachineType" représentant un *SubStateMachine*. Le "State1" référence cet *Objet* avec une *Référence HasSubStateMachine* et, donc, il est un *SubStateMachine* de "State1". Comme "MySubMachine" est un *Objet* du type "AnotherStateMachineType", il a une *Variable* représentant l'*État* courant. Sachant qu'il est utilisé comme une *InstanceDeclaration*, il n'est affecté à cette *Variable* aucune valeur.

Un *Objet* de "MyStateMachineType", appelé "MyStateMachine" a des *Variables* pour l'*État* courant, mais il a aussi un *Objet* "MySubMachine" et une *Variable* représentant l'état courant du *SubStateMachine*. Sachant que le *SubStateMachine* est utilisé seulement lorsque "MyStateMachine" est dans le "State1", un client recevrait un *StatusCode Bad_StateNotActive* lorsqu'il lit la *Variable CurrentState* du *SubStateMachine* si "MyStateMachine" est dans un *État* différent.

B.5.3 StateMachineType utilisant la hiérarchie d'appartenance (ou emboîtement)



Anglais	Français
No value assigned for the TypeDefinition	Aucune valeur affectée pour le TypeDefinition
For simplicity the Properties of CurrentState are not shown	Pour des raisons de simplicité, les propriétés de CurrentState ne sont pas montrées
Contains the current state of MyStateMachine	Contient l'état courant de MyStateMachine

Figure B.7 – Exemple d'un StateMachineType utilisant la hiérarchie d'appartenance

La Figure B.7 donne un exemple d'un *ObjectType* non seulement représentant un *StateMachine*, mais aussi ayant une autre fonctionnalité. L'*ObjectType* "MyObjectType" a un *Objet* "MyComponent" représentant cette autre fonctionnalité. Mais il contient également un *StateMachine* "MyStateMachine" du type "MyStateMachineType". Les *Objets* de "MyObjectType" contiennent également un tel *Objet* représentant le StateMachine et une *Variable* contenant l'état courant du StateMachine, comme montré dans la Figure.

B.5.4 Exemple de StateMachine ayant une Transition vers un SubStateMachine

Les *StateMachines* montrés jusqu'ici avaient seulement des *Transitions* entre des *États* du même niveau, c'est-à-dire, sur le même *StateMachine*. Mais il est possible et parfois indispensable d'avoir des *Transitions* entre des *États* du *StateMachine* et des *États* de son *SubStateMachine*.

Étant donné qu'un *SubStateMachine* peut être défini par un autre *StateMachineType* et que ce type peut être utilisé en plusieurs endroits, il n'est pas possible d'ajouter une *Référence* bidirectionnelle issue de l'un des *États* partagés du *SubStateMachine* vers un autre *StateMachine*. Dans ce cas, il est judicieux de présenter les *Références FromState* ou *ToState* dans un seul sens, c'est-à-dire, pointant de la *Transition* vers l'*État* et de ne pas pouvoir parcourir l'autre sens. Si une *Transition* pointe d'un *État* d'un *SubStateMachine* vers un *État* d'un autre sous-diagramme, les *Références FromState* et *ToState*, sont toutes deux gérées dans un seul sens.

Un client doit pouvoir gérer les informations d'un *StateMachine* si les *Références ToState* et *FromState* sont seulement présentées comme des *Références* directes et les *Références* inverses sont omises.

La Figure B.8 donne un exemple d'un diagramme d'états ayant une transition d'un sous-état vers un état.

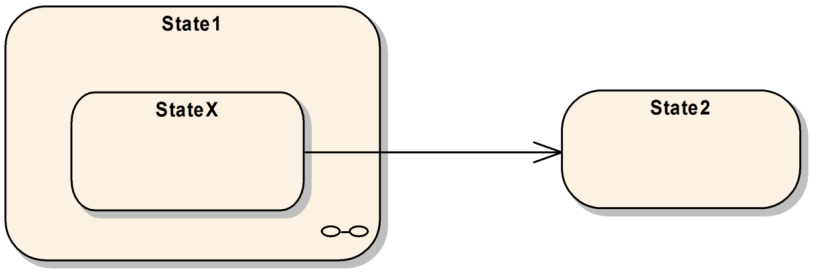
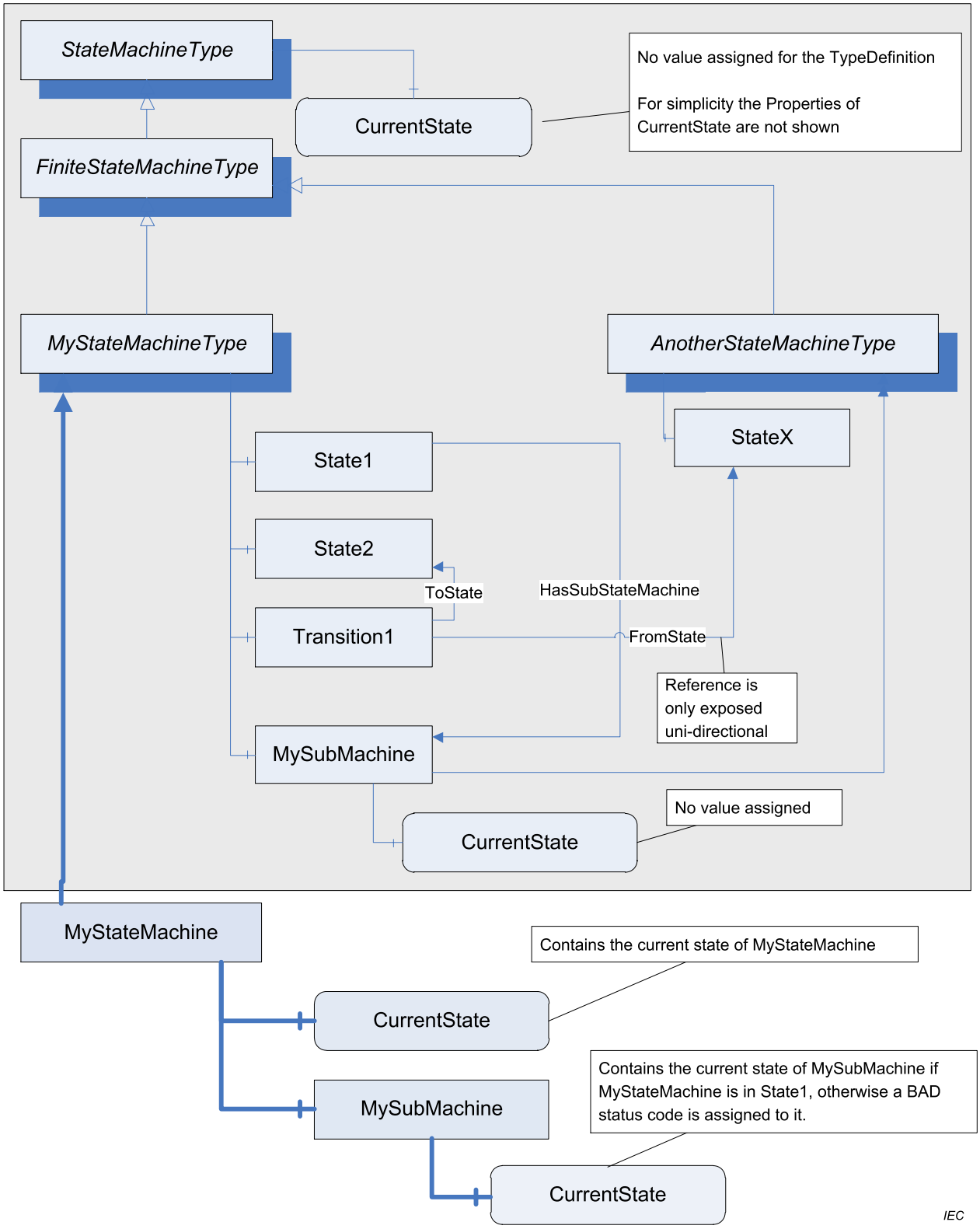


Figure B.8 – Exemple d'un diagramme d'états avec des transitions partant de sous-états

La Figure B.9 donne la représentation de cet exemple comme *StateMachineType* dans l'*Espace d'Adresses*. La "Transition1", partie intégrante de la définition du "MyStateMachineType", pointe sur le "StateX" du *StateMachineType* "AnotherStateMachineType". La *Référence* est seulement présentée comme *Référence* directe et la *Reference* inverse est omise. Ainsi, il n'y a pas de *Référence* partant du "StateX" de l'"AnotherStateMachineType" vers une partie quelconque de "MyStateMachineType", et "AnotherStateMachineType" peut être utilisé en d'autres endroits également.



Anglais	Français
No value assigned for the TypeDefinition	Aucune valeur affectée pour le TypeDefinition
For simplicity the Properties of CurrentState are not shown	Pour des raisons de simplicité, les propriétés de CurrentState ne sont pas montrées
Contains the current state of MyStateMachine	Contient l'état courant de MyStateMachine
No value assigned	Aucune valeur affectée

Anglais	Français
Contains the current state of MySubMachine if MyStateMachine is in State1, otherwise a BAD status code is assigned to it.	Contient l'état courant de MySubMachine si MyStateMachine est dans l'état State1, autrement un code de statut BAD ("mauvais") lui est affecté.t
Reference is only exposed uni-directional	La Référence est seulement présentée dans un seul sens.

Figure B.9 – Exemple d'un StateMachineType ayant une Transition vers un SubStateMachine

Annexe C
(normative)

Transfert de fichiers

C.1 Vue d'ensemble

Cette annexe décrit un modèle d'informations pour le transfert de fichiers. Les fichiers peuvent être modélisés au format OPC UA comme Variables simples en utilisant des ByteStrings. Cependant, la taille totale des messages au format OPC UA est limitée en raison des ressources et des questions de sécurité (attaques par déni de service). Seules les parties d'accès de la matrice peuvent générer des problèmes de concurrence lorsqu'un client lit la matrice tandis que d'autres la manipulent. Par conséquent, un *ObjectType* est défini qui représente un fichier avec des *Méthodes* permettant d'accéder à ce dernier.

Les *Services* définis dans le Jeu de Services NodeManagement peuvent être utilisés pour créer ou supprimer des fichiers dans un *Espace d'Adresses*. Le cycle de vie d'un fichier stocké sur un disque dur et une instance du *FileType* représentant le fichier dans un *Espace d'Adresses* OPC UA peuvent être indépendants. La suppression de l'*Objet* OPC UA n'implique pas que le fichier est supprimé du disque et une suppression du disque n'implique pas que l'*Objet* OPC UA est supprimé.

Cette annexe fait partie intégrante de la présente norme, c'est-à-dire que les types définis dans cette annexe sont à utiliser comme défini. Toutefois, il n'est pas exigé, mais fortement recommandé qu'un *Serveur* utilise ces types pour présenter ses fichiers. Les types définis peuvent être sous-typés afin d'affiner leur comportement.

C.2 FileType

Cet *ObjectType* définit un type pour les fichiers. Il est défini de façon formelle dans le Tableau C.1.

Tableau C.1 – FileType

Attribut	**Valeur**				
BrowseName	FileType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini en 6.2					
HasProperty	Variable	Size	UInt64	PropertyType	Obligatoire
HasProperty	Variable	Writable	Boolean	PropertyType	Obligatoire
HasProperty	Variable	UserWritable	Boolean	PropertyType	Obligatoire
HasProperty	Variable	OpenCount	UInt16	PropertyType	Obligatoire
HasComponent	Method	Open	Défini en C.3		Obligatoire
HasComponent	Method	Close	Défini en C.4		Obligatoire
HasComponent	Method	Read	Défini en C.5		Obligatoire
HasComponent	Method	Write	Défini en C.6		Obligatoire
HasComponent	Method	GetPosition	Défini en C.7		Obligatoire
HasComponent	Method	SetPosition	Défini en C.8		Obligatoire

Size (*Taille*) définit la taille du fichier en octets. Lorsqu'un fichier est ouvert pour écriture et que le fileHandle demeure valide, la taille peut ne pas être précise.

Writable (Inscriptible) indique si le fichier est inscriptible. Il ne prend pas en compte les droits d'accès de l'utilisateur quels qu'ils soient, c'est-à-dire que bien que le fichier soit inscriptible, ceci peut être limité à un certain utilisateur / groupe d'utilisateurs. La *Propriété* ne tient pas compte du fait de l'ouverture actuelle éventuelle du fichier pour son inscriptibilité par un autre

client, et ainsi du verrouillage actuel et de la non-inscriptibilité de ce même fichier par des tiers.

UserWriteable indique si le fichier est inscriptible compte tenu des droits d'accès de l'utilisateur. La Propriété ne tient pas compte du fait de l'ouverture actuelle éventuelle du fichier pour son inscriptibilité par un autre client, et ainsi du verrouillage actuel et de la non-inscriptibilité de ce même fichier par des tiers.

OpenCount indique le nombre d'indicateurs de fichier actuellement valides sur le fichier.

Noter que toutes les *Méthodes* applicables à un fichier exigent un fileHandle, qui est retourné dans la *Méthode Open*.

C.3 Open (Ouvrir)

Open sert à ouvrir un fichier représenté par un *Objet* de FileType. Lorsqu'un client ouvre un fichier, il obtient un indicateur de fichier valide alors que la session est ouverte. Les clients doivent utiliser la *Méthode* Close pour diffuser l'indicateur lorsque l'accès au fichier ne leur est plus nécessaire. Les clients peuvent ouvrir le même fichier plusieurs fois pour lecture. Une demande d'ouverture pour écriture doit retourner Bad_NotWritable lorsque le fichier est déjà ouvert. Une demande d'ouverture pour lecture doit retourner Bad_NotReadable lorsque le fichier est déjà ouvert pour écriture.

Signature

```
Open(
    [in] Byte mode
    [out] UInt32 fileHandle
  );
```

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>mode</td><td>Indique s'il convient d'ouvrir le fichier uniquement pour les opérations de lecture, ou pour les opérations de lecture et d'écriture, et lorsque la position initiale est définie.
Le mode est un entier non signé à 8 bits utilisé comme masque de bits dont la structure est définie dans le tableau suivant:
<table>
<tr><th>Field (champ)</th><th>Bit</th><th>Description</th></tr>
<tr><td>Read</td><td>0</td><td>Le fichier est ouvert pour lecture. Si ce bit n'est pas défini, la Méthode Lecture (Read) ne peut pas être exécutée.</td></tr>
<tr><td>Write</td><td>1</td><td>Le fichier est ouvert pour écriture. Si ce bit n'est pas défini, la Méthode Ecriture (Write) ne peut pas être exécutée.</td></tr>
<tr><td>EraseExisiting</td><td>2</td><td>Ce bit peut être défini uniquement si le fichier est ouvert pour écriture (Bit Ecriture défini). Le contenu actuel du fichier est effacé et un fichier vide est fourni.</td></tr>
<tr><td>Append</td><td>3</td><td>Lorsque le bit Ajouter (Append) est défini, le fichier est ouvert à la fin ou au début dans le cas contraire. La Méthode SetPosition peut être utilisée pour changer la position.</td></tr>
<tr><td>Reserved</td><td>4:7</td><td>Réservé pour une utilisation ultérieure. Doit toujours être zéro.</td></tr>
</table>
</td></tr>
<tr><td>fileHandle</td><td>Indicateur du fichier utilisé dans d'autres appels de méthode, désignant non pas le fichier (ceci est effectué par l'Objet de l'appel de la Méthode), mais la demande d'accès, et donc la position dans le fichier. Le fileHandle est généré par le serveur et est unique pour la Session. Les clients ne peuvent pas transférer le fileHandle vers une autre Session, mais il est nécessaire qu'ils obtiennent un nouveau fileHandle par un appel de la Méthode Open.</td></tr>
</table>

Codes de résultat de la méthode (définis dans le Service Call)

Code de résultat	Description
Bad_NotReadable	Voir l'IEC 62541-4 pour une description générale. Le fichier peut être verrouillé et donc non lisible.
Bad_NotWritable	Voir l'IEC 62541-4 pour une description générale.
Bad_InvalidState	Voir l'IEC 62541-4 pour une description générale. Le fichier est verrouillé et donc non inscriptible.
Bad_InvalidArguments	Voir l'IEC 62541-4 pour une description générale. Le réglage de mode n'est pas valide.
Bad_NotFound	Voir l'IEC 62541-4 pour une description générale.
Bad_UnexpectedError	Voir l'IEC 62541-4 pour une description générale.

Le Tableau C.2 spécifie la représentation de l'*Espace d'Adresses* pour la *Méthode Open*.

Tableau C.2 – Définition de l'Espace d'Adresses de la Méthode Open

Attribut	Valeur				
BrowseName	Open				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

C.4 Close (Fermer)

Close sert à fermer un fichier représenté par un FileType. Lorsqu'un client ferme un fichier, l'indicateur devient non valide.

Signature

```
Close(
    [in] UInt32 fileHandle
  );
```

Argument	Description
fileHandle	Indicateur spécifiant la demande d'accès et donc indirectement la position interne au fichier.

Codes de résultat de la méthode (définis dans le Service Call)

Result Code	Description
Bad_InvalidArgument	Voir l'IEC 62541-4 pour une description générale. Appel d'un indicateur de fichier non valide.

Le Tableau C.3 spécifie la représentation de l'*Espace d'Adresses* pour la *Méthode Close*.

Tableau C.3 – Définition de l'Espace d'Adresses de la Méthode Close

Attribut	Valeur				
BrowseName	Close				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire

C.5 Read (Lecture)

Read sert à lire une partie du fichier en commençant par sa position actuelle. La position du fichier est avancée par le nombre d'octets lus.

Signature

```
Read(
    [in] Uint32 fileHandle
```

```
    [in] Int32 length
    [out] ByteString data
  );
```

Argument	Description
fileHandle	Indicateur spécifiant la demande d'accès et donc indirectement la position interne au fichier.
Length	Définit la longueur en octets, qu'il convient de retourner sous forme de données, en commençant par la position actuelle de l'indicateur de fichier. Lorsque la fin du fichier est atteinte, seules les données jusqu'à la fin du fichier sont toutes retournées. Lorsque les longueurs spécifiées sont supérieures à la taille de message maximale admise de la communication, seules les données adaptées à la taille du message sont retournées. Seules les valeurs positives sont admises.
Data	Contient les données retournées du fichier.

Codes de résultat de la méthode (définis dans le Service Call)

Code de résultat	Description
Bad_InvalidArgument	Voir l'IEC 62541-4, Appel d'un indicateur de fichier non valide ou longueur non positive.
Bad_UnexpectedError	Voir l'IEC 62541-4 pour une description générale.
Bad_InvalidState	See IEC 62541-4 pour une description générale. Le fichier n'a pas été ouvert pour un accès en lecture.

Le Tableau C.4 spécifie la représentation de l'*Espace d'Adresses* pour la *Méthode Read*.

Tableau C.4 – Définition de l'Espace d'Adresses de la Méthode Read

Attribut	Valeur				
BrowseName	Read				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

C.6 Write (Ecriture)

Write sert à écrire une partie du fichier en commençant par sa position actuelle. La position du fichier est avancée par le nombre d'octets inscrits.

Signature

```
Write(
    [in] UInt32 fileHandle
    [in] ByteString data
  );
```

Argument	Description
fileHandle	Indicateur spécifiant la demande d'accès et donc indirectement la position interne au fichier.
data	Contient les données à inscrire à la position du fichier. Le fait de déterminer si les données inscrites sont stockées de façon permanente si la session est achevée sans appeler la Méthode Close avec le fileHandle, dépend du serveur.

Codes de résultat de la méthode (définis dans le Service Call)

Code de résultat	Description
Bad_InvalidArgument	Voir l'IEC 62541-4 pour une description générale. Appel d'un indicateur de fichier non valide.
Bad_NotWritable	Voir l'IEC 62541-4 pour une description générale. Le fichier peut être verrouillé et donc non inscriptible.
Bad_InvalidState	Voir l'IEC 62541-4 pour une description générale. Le fichier n'était pas ouvert pour un accès en écriture.

Le Tableau C.5 spécifie la représentation de l'*Espace d'Adresses* pour la *Méthode Write*

Tableau C.5 – Définition de l'Espace d'Adresses de la Méthode Write

Attribut	Valeur				
BrowseName	Write				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire

C.7 GetPosition

GetPosition sert à fournir la position actuelle de l'indicateur de fichier.

Signature

```
GetPosition(
    [in] UInt32 fileHandle
    [out] UInt64 position
   );
```

Argument	Description
fileHandle	Indicateur spécifiant la demande d'accès et donc indirectement la position interne au fichier.
Position	La position du fileHandle dans le fichier. Une opération de lecture ou d'écriture appelée commence à cette position.

Codes de résultat de la méthode (définis dans le Service Call)

Code de résultat	Description
Bad_InvalidArgument	Voir l'IEC 62541-4 pour une description générale. Appel d'un indicateur de fichier non valide.

Le Tableau C.6 spécifie la représentation de l'*Espace d'Adresses* pour la *Méthode GetPosition*.

Tableau C.6 – Définition de l'Espace d'Adresses de la Méthode GetPosition

Attribut	Valeur				
BrowseName	GetPosition				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

C.8 SetPosition

SetPosition sert à définir la position actuelle de l'indicateur de fichier.

Signature

```
SetPosition(
    [in] UInt32 fileHandle
    [in] UInt64 position
   );
```

Argument	Description
fileHandle	Indicateur spécifiant la demande d'accès et donc indirectement la position interne au fichier.
Position	Position à définir pour le fileHandle dans le fichier. Une opération de lecture ou d'écriture appelée commence à cette position. Lorsque la position est supérieure à la taille du fichier, la position est définie à la fin du fichier.

Codes de résultat de la méthode (définis dans le Service Call)

Code de résultat	Description
Bad_InvalidArgument	Voir l'IEC 62541-4 pour une description générale. Appel d'un indicateur de fichier non valide.

Le Tableau C.7 spécifie la représentation de l'*Espace d'Adresses* pour la *Méthode SetPosition*.

Tableau C.7 – Définition de l'Espace d'Adresses de la Méthode SetPosition

Attribut	**Valeur**				
BrowseName	SetPosition				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire

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