

DIN EN 62541-3

ICS 35.200; 35.240.50

Einsprüche bis 2014-12-24
Vorgesehen als Ersatz für
DIN EN 62541-3:2011-09

Entwurf

**OPC Unified Architecture –
Teil 3: Adressraummodell
(IEC 65E/374/CDV:2014);
Englische Fassung FprEN 62541-3:2014**

OPC Unified Architecture –
Part 3: Address Space Model
(IEC 65E/374/CDV:2014);
English version FprEN 62541-3:2014

Architecture unifiée OPC –
Partie 3: Modèle de l'espace d'adressage
(CEI 65E/374/CDV:2014);
Version anglaise FprEN 62541-3:2014

Anwendungswarnvermerk

Dieser Norm-Entwurf mit Erscheinungsdatum 2014-10-24 wird der Öffentlichkeit zur Prüfung und Stellungnahme vorgelegt.

Weil die beabsichtigte Norm von der vorliegenden Fassung abweichen kann, ist die Anwendung dieses Entwurfes besonders zu vereinbaren.

Stellungnahmen werden erbeten

- vorzugsweise online im Norm-Entwurfs-Portal des DIN unter www.entwuerfe.din.de bzw. für Norm-Entwürfe der DKE auch im Norm-Entwurfs-Portal der DKE unter www.entwuerfe.normenbibliothek.de, sofern dort wiedergegeben;
- oder als Datei per E-Mail an dke@vde.com möglichst in Form einer Tabelle. Die Vorlage dieser Tabelle kann im Internet unter www.din.de/stellungnahme oder für Stellungnahmen zu Norm-Entwürfen der DKE unter www.dke.de/stellungnahme abgerufen werden;
- oder in Papierform an die DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE, Stresemannallee 15, 60596 Frankfurt am Main.

Die Empfänger dieses Norm-Entwurfs werden gebeten, mit ihren Kommentaren jegliche relevanten Patentrechte, die sie kennen, mitzuteilen und unterstützende Dokumentationen zur Verfügung zu stellen.

Gesamtumfang 123 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE

Nationales Vorwort

Die Englische Fassung des europäischen Dokuments FprEN 62541-3:2014 „OPC Unified Architecture – Teil 3: Adressraummodell“ (Entwurf in der Umfrage) ist unverändert in diesen Norm-Entwurf übernommen worden.

Die Internationale Elektrotechnische Kommission (IEC) und das Europäische Komitee für Elektrotechnische Normung (CENELEC) haben vereinbart, dass ein auf IEC-Ebene erarbeiteter Entwurf für eine Internationale Norm zeitgleich (parallel) bei IEC und CENELEC zur Umfrage (CDV-Stadium) und Abstimmung als FDIS (en: Final Draft International Standard) bzw. Schluss-Entwurf für eine Europäische Norm gestellt wird, um eine Beschleunigung und Straffung der Normungsarbeit zu erreichen. Dem entsprechend ist das internationale Dokument IEC 65E/374/CDV:2014 „OPC Unified Architecture – Part 3: Address Space Model“ unverändert in den Entwurf FprEN 62541-3:2014 übernommen worden.

Das internationale Dokument wurde vom SC 65E „Devices and integration in enterprise systems“ der Internationalen Elektrotechnischen Kommission (IEC) erarbeitet.

Bei der Abstimmung zu dem Europäischen Schluss-Entwurf bei CENELEC und dem Internationalen Schluss-Entwurf bei IEC [Final Draft International Standard (FDIS)] sind jeweils nur „JA/NEIN“-Entscheidungen möglich, wobei „NEIN“-Entscheidungen fundiert begründet werden müssen. Dokumente, die bei CENELEC als Europäische Norm angenommen und ratifiziert werden, sind unverändert als Deutsche Normen zu übernehmen.

Für dieses Dokument ist das nationale Arbeitsgremium K 931 „Systemaspekte“ der DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE (www.dke.de) zuständig.

Für den Fall einer undatierten Verweisung im normativen Text (Verweisung auf eine Norm ohne Angabe des Ausgabedatums und ohne Hinweis auf eine Abschnittsnummer, eine Tabelle, ein Bild usw.) bezieht sich die Verweisung auf die jeweils neueste gültige Ausgabe der in Bezug genommenen Norm.

Für den Fall einer datierten Verweisung im normativen Text bezieht sich die Verweisung immer auf die in Bezug genommene Ausgabe der Norm.

Der Zusammenhang der zitierten Normen mit den entsprechenden Deutschen Normen ergibt sich, soweit ein Zusammenhang besteht, grundsätzlich über die Nummer der entsprechenden IEC-Publikation. Beispiel: IEC 60068 ist als EN 60068 als Europäische Norm durch CENELEC übernommen und als DIN EN 60068 ins Deutsche Normenwerk aufgenommen.

Das Präsidium des DIN hat mit Präsidialbeschluss 1/2004 festgelegt, dass DIN-Normen, deren Inhalt sich auf internationale Arbeitsergebnisse der Informationsverarbeitung gründet, unter bestimmten Bedingungen allein in englischer Sprache veröffentlicht werden dürfen. Diese Bedingungen sind für den vorliegenden Normentwurf erfüllt.

Da sich die Benutzer des vorliegenden Normentwurfs der englischen Sprache als Fachsprache bedienen, wird die Englische Fassung der EN 62541-3 veröffentlicht. Zu deren Abschnitt 3, Begriffe, Symbole, Abkürzungen und Vereinbarungen, wurde eine Übersetzung angefertigt und als informativer Nationaler Anhang NA des vorliegenden Normentwurfs hinzugefügt. Dieser Normentwurf steht nicht in unmittelbarem Zusammenhang mit Rechtsvorschriften und ist nicht als Sicherheitsnorm anzusehen.

Es ist vorgesehen, auch bei der entsprechenden zukünftigen Deutschen Norm auf die deutsche Sprachfassung zu verzichten und diese in der englischsprachigen Fassung zu veröffentlichen.

Es wird auf die Möglichkeit hingewiesen, dass einige Elemente dieses Dokuments Patentrechte berühren können. ISO [und/oder] IEC sind nicht dafür verantwortlich, einige oder alle diesbezüglichen Patentrechte zu identifizieren.

Das Original-Dokument enthält Bilder in Farbe, die in der Papierversion in einer Graustufen-Darstellung wiedergegeben werden. Elektronische Versionen dieses Dokuments enthalten die Bilder in der originalen Farbdarstellung.

Änderungen

Gegenüber DIN EN 62541-3:2011-09 wurden folgende Änderungen vorgenommen:

- a) Regeln für Untertypen für Untertypen von Aufzählungen in 8.14 hinzugefügt;
- b) Property EnumValues in 5.8.3 hinzugefügt, um die Darstellung von Aufzählungen mit ganzen Zahlen zu unterstützen, die nicht nullbasiert sind oder Lücken haben;
- c) Property ValueAsText in 5.6.2 hinzugefügt, damit Lokaltext für Aufzählungswerte dargestellt werden kann;
- d) EventType SystemStatusChangeEvent in 9.31 hinzugefügt, sodass der Verlust der Verbindung zu einem Subsystem angezeigt werden kann;
- e) Properties MaxArrayLength und MaxStringLength in 5.6.2 hinzugefügt, um die größte erlaubte Länge eines Strings und Arrays anzuzeigen für Clients, die Werte schreiben;
- f) die Vorstellung des ModelParent herausgenommen, da sie sich nicht als nützlich erwiesen hat. Der Nodeld des ReferenceType wird jedoch belassen, damit bestehende Anwendungen nicht beeinträchtigt werden;
- g) EventType ProgressEventType in 9.4 hinzugefügt, damit der Fortschritt einer Operation wie bei einem Dienstaufufr festgestellt werden kann;
- h) in 8.38 angegeben, dass es erlaubt ist, an allen Stellen, wo die UTC-Zeit verwendet wird, TAI zu benutzen, damit Probleme mit springenden Sekunden vermieden werden;
- i) Property EngineeringUnits in 5.6.2 hinzugefügt, da sie in der IEC 62541-8 verwendet werden;
- j) ModellingRules OptionalPlaceholder und MandatoryPlaceholder in 6.4.4.5.5 und 6.4.4.5.6 hinzugefügt.

Nationaler Anhang NA (informativ)

NA.3 Begriffe, Symbole, Abkürzungen und Vereinbarungen

NA.3.1 Begriffe

Für die Anwendung dieses Schriftstückes gelten die folgenden sowie die in IEC/TR 62541-1 angegebenen Begriffe.

3.1.1

DataType

(en: **DataType**)

Instanz eines *DataType Node* (Datentypknoten), die gemeinsam mit dem *ValueRank Attribute* (Werterang-attribut) zur Festlegung des Datentyps einer *Variablen* verwendet wird

3.1.2

DataTypeId

(en: **DataTypeId**)

die *NodeId* (Kennzeichnung eines Knotens) eines *DataType Node* (Datentypknotens)

3.1.3

DataVariable

(en: **DataVariable**)

Variable, die *Werte* von *Objekten*, entweder direkt oder indirekt für komplexe *Variablen*, darstellen; dabei sind die *Variablen* stets der *TargetNode* (Zielknoten) einer *HasCompetent Reference* (Besitzt-Kompetenz-Bezug)

3.1.4**EventType****(en: EventType)***ObjectType Node* (Objektypknoten), der die Typdefinition eines *Event* (Ereignis) darstellt**3.1.5****HierarchicalReference****(en: Hierarchical Reference)**Bezug, der zum Aufbau von Hierarchien im *AdressSpace* (Adressraum) verwendet wird

ANMERKUNG Alle hierarchischen *ReferenceTypes* (Bezugstypen) werden von *HierarchicalReferences* (hierarchischen Bezügen) abgeleitet.

3.1.6**InstanceDeclaration****(en: InstanceDeclaration)***Node* (Knoten), der von einem komplexen *TypeDefinitionNode* (Typdefinitions-knoten) zum Zeigen seiner komplexen Struktur verwendet wird; es ist eine Instanz, die von einer Typdefinition verwendet wird**3.1.7****ModellingRule****(en: ModellingRule)**Metadaten einer *InstanceDeclaration* (Instanzenbeschreibung), die festlegen, wie die *InstanceDeclaration* für die Instantiierung zu verwenden ist; sie legt auch Untertypenregeln für eine *InstanceDeclaration* fest**3.1.8****Eigenschaft****(en: Property)***Variable*, die der *TargetNode* für einen *HasProperty Reference* (Besitzt-Eigenschaft-Bezug) ist; *Eigenschaften* beschreiben die Merkmale eines *Node* (Knoten)**3.1.9****SourceNode****(en: SourceNode)***Node*, der ein *Reference* (Bezug) zu einem weiteren *Node* hat; z. B. in der *Reference* „A enthält B“ ist „A“ der *SourceNode* (Quellknoten)**3.1.10****TargetNode****(en: TargetNode)***Node*, auf den von einem weiteren *Node* Bezug genommen wird; z. B. in der *Reference* „A enthält B“ ist „B“ der *TargetNode* (Zielknoten)**3.1.11****TypeDefinitionNode****(en: TypeDefinitionNode)***Node*, der zur Festlegung des Typs eines weiteren *Node* verwendet wird; *ObjectType* (Objektyp-) und *VariableType Nodes* (*Variablentypknoten*) sind *TypeDefinitionNodes* (Typdefinitions-knoten)**3.1.12****VariableType****(en: VariableType)***Node*, der die Typdefinition für eine *Variable* darstellt

CONTENTS

	Page
1 Scope	1
2 Normative references	1
3 Terms, definitions, abbreviations and conventions	2
3.1 Terms and definitions	2
3.2 Abbreviations	3
3.3 Conventions	3
3.3.1 Conventions for AddressSpace figures	3
3.3.2 Conventions for defining NodeClasses	4
4 AddressSpace concepts	5
4.1 Overview	5
4.2 Object Model	5
4.3 Node Model	5
4.3.1 General	5
4.3.2 NodeClasses	6
4.3.3 Attributes	6
4.3.4 References	6
4.4 Variables	7
4.4.1 General	7
4.4.2 Properties	7
4.4.3 DataVariables	7
4.5 TypeDefinitionNodes	8
4.5.1 General	8
4.5.2 Complex TypeDefinitionNodes and their InstanceDeclarations	8
4.5.3 Subtyping	9
4.5.4 Instantiation of complex TypeDefinitionNodes	9
4.6 Event Model	10
4.6.1 General	10
4.6.2 EventTypes	11
4.6.3 Event Categorization	11
4.7 Methods	12
5 Standard NodeClasses	12
5.1 Overview	12
5.2 Base NodeClass	13
5.2.1 General	13
5.2.2 NodeId	13
5.2.3 NodeClass	13
5.2.4 BrowseName	13
5.2.5 DisplayName	14
5.2.6 Description	14
5.2.7 WriteMask	14
5.2.8 UserWriteMask	15
5.3 ReferenceType NodeClass	15
5.3.1 General	15
5.3.2 Attributes	15
5.3.3 References	17
5.4 View NodeClass	

50	5.5	Objects.....	21
51	5.5.1	Object NodeClass.....	21
52	5.5.2	ObjectType NodeClass.....	23
53	5.5.3	Standard ObjectType FolderType.....	24
54	5.5.4	Client-side creation of Objects of an ObjectType.....	24
55	5.6	Variables.....	24
56	5.6.1	General.....	24
57	5.6.2	Variable NodeClass.....	25
58	5.6.3	Properties.....	28
59	5.6.4	DataVariable.....	29
60	5.6.5	VariableType NodeClass.....	30
61	5.6.6	Client-side creation of Variables of an VariableType.....	31
62	5.7	Method NodeClass.....	32
63	5.8	DataTypes.....	33
64	5.8.1	DataType Model.....	33
65	5.8.2	Encoding Rules for different kinds of DataTypes.....	35
66	5.8.3	DataType NodeClass.....	36
67	5.8.4	DataTypeDictionary, DataTypeDescription, DataTypeEncoding and DataTypeSystem.....	37
69	5.9	Summary of Attributes of the NodeClasses.....	40
70	6	Type Model for ObjectTypes and VariableTypes.....	40
71	6.1	Overview.....	40
72	6.2	Definitions.....	40
73	6.2.1	InstanceDeclaration.....	40
74	6.2.2	Instances without ModellingRules.....	40
75	6.2.3	InstanceDeclarationHierarchy.....	41
76	6.2.4	Similar Node of InstanceDeclaration.....	41
77	6.2.5	BrowsePath.....	41
78	6.2.6	Attribute Handling of InstanceDeclarations.....	41
79	6.2.7	Attribute Handling of Variable and VariableTypes.....	41
80	6.2.8	NodeIds of InstanceDeclarations.....	41
81	6.3	Subtyping of ObjectTypes and VariableTypes.....	42
82	6.3.1	Overview.....	42
83	6.3.2	Attributes.....	42
84	6.3.3	InstanceDeclarations.....	42
85	6.4	Instances of ObjectTypes and VariableTypes.....	45
86	6.4.1	Overview.....	45
87	6.4.2	Creating an Instance.....	45
88	6.4.3	Constraints on an Instance.....	46
89	6.4.4	ModellingRules.....	47
90	6.5	Changing Type Definitions that are already used.....	54
91	7	Standard ReferenceTypes.....	55
92	7.1	General.....	55
93	7.2	References ReferenceType.....	55
94	7.3	HierarchicalReferences ReferenceType.....	56
95	7.4	NonHierarchicalReferences ReferenceType.....	56
96	7.5	HasChild ReferenceType.....	56
97	7.6	Aggregates ReferenceType.....	56
98	7.7	HasComponent ReferenceType.....	

99	7.8	HasProperty ReferenceType.....	57
100	7.9	HasOrderedComponent ReferenceType	57
101	7.10	HasSubtype ReferenceType	57
102	7.11	Organizes ReferenceType	57
103	7.12	HasModellingRule ReferenceType.....	58
104	7.13	HasTypeDefinition ReferenceType	58
105	7.14	HasEncoding ReferenceType	58
106	7.15	HasDescription ReferenceType	58
107	7.16	GeneratesEvent	59
108	7.17	AlwaysGeneratesEvent	59
109	7.18	HasEventSource	59
110	7.19	HasNotifier	60
111	8	Standard DataTypes.....	62
112	8.1	General	62
113	8.2	NodeId	62
114	8.2.1	General	62
115	8.2.2	NamespaceIndex.....	62
116	8.2.3	IdentifierType	62
117	8.2.4	Identifier value.....	63
118	8.3	QualifiedName	64
119	8.4	LocaleId	64
120	8.5	LocalizedText.....	64
121	8.6	Argument	65
122	8.7	BaseDataType.....	65
123	8.8	Boolean.....	65
124	8.9	Byte	65
125	8.10	ByteString	65
126	8.11	DateTime	65
127	8.12	Double	66
128	8.13	Duration	66
129	8.14	Enumeration.....	66
130	8.15	Float.....	66
131	8.16	Guid	66
132	8.17	SByte	66
133	8.18	IdType.....	66
134	8.19	Image.....	66
135	8.20	ImageBMP	66
136	8.21	ImageGIF	66
137	8.22	ImageJPG	66
138	8.23	ImagePNG	67
139	8.24	Integer	67
140	8.25	Int16.....	67
141	8.26	Int32.....	67
142	8.27	Int64.....	67
143	8.28	TimeZoneDataType	67
144	8.29	NamingRuleType	67
145	8.30	NodeClass	68
146	8.31	Number	68
147	8.32	String	68

148	8.33	Structure	68
149	8.34	UInteger	68
150	8.35	UInt16	68
151	8.36	UInt32	68
152	8.37	UInt64	68
153	8.38	UtcTime	68
154	8.39	XmlElement	69
155	8.40	EnumValueType	69
156	9	Standard EventTypes	70
157	9.1	General	70
158	9.2	BaseEventType	70
159	9.3	SystemEventType	71
160	9.4	ProgressEventType	71
161	9.5	AuditEventType	71
162	9.6	AuditSecurityEventType	72
163	9.7	AuditChannelEventType	72
164	9.8	AuditOpenSecureChannelEventType	73
165	9.9	AuditSessionEventType	73
166	9.10	AuditCreateSessionEventType	73
167	9.11	AuditUrlMismatchEventType	73
168	9.12	AuditActivateSessionEventType	73
169	9.13	AuditCancelEventType	73
170	9.14	AuditCertificateEventType	73
171	9.15	AuditCertificateDataMismatchEventType	73
172	9.16	AuditCertificateExpiredEventType	73
173	9.17	AuditCertificateInvalidEventType	73
174	9.18	AuditCertificateUntrustedEventType	74
175	9.19	AuditCertificateRevokedEventType	74
176	9.20	AuditCertificateMismatchEventType	74
177	9.21	AuditNodeManagementEventType	74
178	9.22	AuditAddNodesEventType	74
179	9.23	AuditDeleteNodesEventType	74
180	9.24	AuditAddReferencesEventType	74
181	9.25	AuditDeleteReferencesEventType	74
182	9.26	AuditUpdateEventType	74
183	9.27	AuditWriteUpdateEventType	74
184	9.28	AuditHistoryUpdateEventType	74
185	9.29	AuditUpdateMethodEventType	75
186	9.30	DeviceFailureEventType	75
187	9.31	SystemStatusChangeEvent	75
188	9.32	ModelChangeEvents	75
189	9.32.1	General	75
190	9.32.2	NodeVersion Property	75
191	9.32.3	Views	75
192	9.32.4	Event Compression	75
193	9.32.5	BaseModelChangeEvent	76
194	9.32.6	GeneralModelChangeEvent	76
195	9.32.7	Guidelines for ModelChangeEvents	76
196	9.33	SemanticChangeEvent	76

197	9.33.1 General	76
198	9.33.2 ViewVersion and NodeVersion Properties.....	77
199	9.33.3 Views	77
200	9.33.4 Event Compression	77
201	Annex A (informative) How to use the Address Space Model	78
202	A.1 Overview	78
203	A.2 Type definitions	78
204	A.3 ObjectTypes	78
205	A.4 VariableTypes	79
206	A.4.1 General	79
207	A.4.2 Properties or DataVariables.....	79
208	A.4.3 Many Variables and / or structured DataTypes.....	79
209	A.5 Views	80
210	A.6 Methods	80
211	A.7 Defining ReferenceTypes	80
212	A.8 Defining ModellingRules	80
213	Annex B (informative) OPC UA Meta Model in UML	81
214	B.1 Background	81
215	B.2 Notation	81
216	B.3 Meta Model	82
217	B.3.1 Base.....	83
218	B.3.2 ReferenceType	83
219	B.3.3 Predefined ReferenceTypes	84
220	B.3.4 Attributes.....	85
221	B.3.5 Object and ObjectType	86
222	B.3.6 EventNotifier.....	86
223	B.3.7 Variable and VariableType.....	87
224	B.3.8 Method	88
225	B.3.9 DataType.....	89
226	B.3.10 View	89
227	Annex C (normative) OPC Binary Type Description System	90
228	C.1 Concepts.....	90
229	C.2 Schema Description	91
230	C.2.1 TypeDictionary	91
231	C.2.2 TypeDescription	92
232	C.2.3 OpaqueType.....	92
233	C.2.4 EnumeratedType	93
234	C.2.5 StructuredType	93
235	C.2.6 FieldType	93
236	C.2.7 EnumeratedValue	95
237	C.2.8 ByteOrder.....	95
238	C.2.9 ImportDirective	95
239	C.3 Standard Type Descriptions	96
240	C.4 Type Description Examples	96
241	C.5 OPC Binary XML Schema.....	98
242	C.6 OPC Binary Standard TypeDictionary	99
243	Annex D (normative) Graphical Notation	102
244	D.1 General	102

245 D.2 Notation 102

246 D.2.1 Overview 102

247 D.2.2 Simple Notation 102

248 D.2.3 Extended Notation 104

249 Annex E (informative): Revision 1.02 Highlights 107

250

251

252

253

254

FIGURES

255	Figure 1 – AddressSpace Node diagrams	3
256	Figure 2 – OPC UA Object Model.....	5
257	Figure 3 – AddressSpace Node Model	6
258	Figure 4 – Reference Model.....	7
259	Figure 5 – Example of a Variable Defined by a VariableType	8
260	Figure 6 – Example of a Complex TypeDefinition	9
261	Figure 7 – Object and its Components defined by an ObjectType.....	10
262	Figure 8 – Symmetric and Non-Symmetric References.....	16
263	Figure 9 – Variables, VariableTypes and their DataTypes	33
264	Figure 10 – DataType Model.....	34
265	Figure 11 – Example of DataType Modelling	39
266	Figure 12 – Subtyping TypeDefinitionNodes.....	42
267	Figure 13 – The Fully-Inherited InstanceDeclarationHierarchy for BetaType	44
268	Figure 14 – An Instance and its TypeDefinitionNode	46
269	Figure 15 – Example for several References between InstanceDeclarations	47
270	Figure 16 – Example on changing instances based on InstanceDeclarations	49
271	Figure 17 – Example on changing InstanceDeclarations based on an	
272	InstanceDeclaration	50
273	Figure 18 – Use of the Standard ModellingRule New	51
274	Figure 19 – Example using the Standard ModellingRules Optional and Mandatory.....	52
275	Figure 20 – Example on using ExposesItsArray	53
276	Figure 21 – Complex example on using ExposesItsArray	53
277	Figure 22 – Example on using OptionalPlaceholder	53
278	Figure 23 – Example on using MandatoryPlaceholder	54
279	Figure 24 – Standard ReferenceType Hierarchy.....	55
280	Figure 25 – Event Reference Example	60
281	Figure 26 – Complex Event Reference Example	61
282	Figure 27 – Standard EventType Hierarchy.....	70
283	Figure 28 – Audit Behaviour of a Server	71
284	Figure 29 – Audit Behaviour of an Aggregating Server.....	72
285	Figure B.1 – Background of OPC UA Meta Model	81
286	Figure B.2 – Notation (I)	82
287	Figure B.3 – Notation (II)	82
288	Figure B.4 – BaseNode	83
289	Figure B.5 – Reference and ReferenceType	83
290	Figure B.6 – Predefined ReferenceTypes.....	84
291	Figure B.7 – Attributes	85
292	Figure B.8 – Object and ObjectType	86
293	Figure B.9 – EventNotifier	86
294	Figure B.10 – Variable and VariableType	87

295	Figure B.11 – Method	88
296	Figure B.12 – DataType	89
297	Figure B.13 – View	89
298	Figure C.1 – OPC Binary Dictionary Structure.....	90
299	Table C.1 – TypeDictionary Components	91
300	Table C.2 – TypeDescription Components	92
301	Table C.3 – OpaqueType Components.....	92
302	Table C.4 – EnumeratedType Components	93
303	Table C.5 – StructuredType Components.....	93
304	Table C.6 – FieldType Components	94
305	Table C.7 – EnumeratedValue Components.....	95
306	Table C.8 – ImportDirective Components	95
307	Table C.9 – Standard Type Descriptions	96
308	Table D.1 – Notation of Nodes depending on the NodeClass	103
309	Figure D.1 – Example of a Reference connecting two Nodes	103
310	Table D.2 – Simple Notation of Nodes depending on the NodeClass.....	104
311	Figure D.3 – Example of using a TypeDefinition inside a Node	105
312	Figure D.4 – Example of exposing Attributes.....	105
313	Figure D.5 – Example of exposing Properties inline	106
314		
315		

316	TABLES	
317	Table 1 – NodeClass Table Conventions	4
318	Table 2 – Base NodeClass.....	13
319	Table 3 – Bit mask for WriteMask and UserWriteMask	14
320	Table 4 – ReferenceType NodeClass	15
321	Table 5 – View NodeClass	19
322	Table 6 – Object NodeClass	21
323	Table 7 – ObjectType NodeClass.....	23
324	Table 8 – Variable NodeClass.....	25
325	Table 9 – VariableType NodeClass	30
326	Table 10 – Method NodeClass	32
327	Table 11 – DataType NodeClass.....	36
328	Table 12 – Overview about Attributes	40
329	Table 13 – The InstanceDeclarationHierarchy for BetaType	43
330	Table 14 – The Fully-Inherited InstanceDeclarationHierarchy for BetaType.....	44
331	Table 15 – Rule for ModellingRules Properties when Subtyping	48
332	Table 16 – Properties of ModellingRules.....	50
333	Table 17 – NodeId Definition.....	62
334	Table 18 – IdentifierType Values.....	63
335	Table 19 – NodeId Null Values.....	63
336	Table 20 – QualifiedName Definition	64
337	Table 21 –LocaleId Examples	64
338	Table 22 – LocalizedText Definition	65
339	Table 23 – Argument Definition.....	65
340	Table 24 – TimeZoneDataType Definition	67
341	Table 25 – NamingRuleType Values	67
342	Table 26 – NodeClass Values	68
343	Table 27 – EnumValueType Definition	69
344		
345		
346		

INTERNATIONAL ELECTROTECHNICAL COMMISSION

OPC Unified Architecture –

Part 3: Address Space 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 62541-3 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

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

FDIS	Report on voting
65E/XX/FDIS	65E/XX/RVD

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

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

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.

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

- 403 • reconfirmed,
404 • withdrawn,
405 • replaced by a revised edition, or
406 • amended.

407

408 The National Committees are requested to note that for this publication the stability date
409 is 2017.

410 THIS TEXT IS INCLUDED FOR THE INFORMATION OF THE NATIONAL COMMITTEES AND WILL BE
411 DELETED AT THE PUBLICATION STAGE.

412

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

413

OPC Unified Architecture Specification

Part 3: Address Space Model

1 Scope

This specification describes the OPC Unified Architecture (OPC UA) *AddressSpace* and its *Objects*. This Part is the OPC UA meta model on which OPC UA information models are based.

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: Concepts*

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

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

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

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

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

IEC 62541-9, *OPC Unified Architecture – Part 9: Alarms & Conditions*

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

XML Schema Part 1: <http://www.w3.org/TR/xmlschema-1/>

XML Schema Part 2: <http://www.w3.org/TR/xmlschema-2/>

XPATH: <http://www.w3.org/TR/xpath/>

ISO/IEC 10918-1, *Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines*

ISO/IEC 15948, *Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification*

ISO 639 (all parts), *Codes for the representation of names of languages*

ISO 3166 (all parts), *Codes for the representation of names of countries and their subdivisions*

IETF RFC 3066, *Tags for the Identification of Languages*, <http://tools.ietf.org/html/rfc3066>

IEEE 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*,
<http://ieeexplore.ieee.org/servlet/opac?punumber=2355>

445 3 Terms, definitions, abbreviations and conventions

446 3.1 Terms and definitions

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

449 3.1.1

450 **DataType**

451 an instance of a *DataType Node* that is used together with the *ValueRank Attribute* to define the data
452 type of a *Variable*.

453 3.1.2

454 **DataTypeId**

455 the *NodeId* of a *DataType Node*.

456 3.1.3

457 **DataVariable**

458 *Variables* that represent *values* of *Objects*, either directly or indirectly for complex *Variables*, where
459 the *Variables* are always the *TargetNode* of a *HasComponent Reference*.

460 3.1.4

461 **EventType**

462 an *ObjectType Node* that represents the type definition of an *Event*

463 3.1.5

464 **Hierarchical Reference**

465 a *Reference* that is used to construct hierarchies in the *AddressSpace*

466 NOTE All hierarchical *ReferenceTypes* are derived from *HierarchicalReferences*.

467 3.1.6

468 **InstanceDeclaration**

469 a *Node* that is used by a complex *TypeDefinitionNode* to expose its complex structure; It is an
470 instance used by a type definition

471 3.1.7

472 **ModellingRule**

473 metadata of an *InstanceDeclaration* that defines how the *InstanceDeclaration* will be used for
474 instantiation; It also defines subtyping rules for an *InstanceDeclaration*

475 3.1.8

476 **Property**

477 *Variables* that are the *TargetNode* for a *HasProperty Reference*; *Properties* describe the
478 characteristics of a *Node*

479 3.1.9

480 **SourceNode**

481 a *Node* having a *Reference* to another *Node*; For example, in the *Reference* "A contains B", "A" is
482 the *SourceNode*

483 3.1.10

484 **TargetNode**

485 a *Node* that is referenced by another *Node*; For example, in the *Reference* "A Contains B", "B" is the
486 *TargetNode*

3.1.11

TypeDefinitionNode

a *Node* that is used to define the type of another *Node*; *ObjectType* and *VariableType* *Nodes* are *TypeDefinitionNodes*

3.1.12

VariableType

a *Node* that represents the type definition for a *Variable*

3.2 Abbreviations

UA	Unified Architecture
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language

3.3 Conventions

3.3.1 Conventions for AddressSpace figures

Nodes and their *References* to each other are illustrated using figures. Figure 1 illustrates the conventions used in these figures.

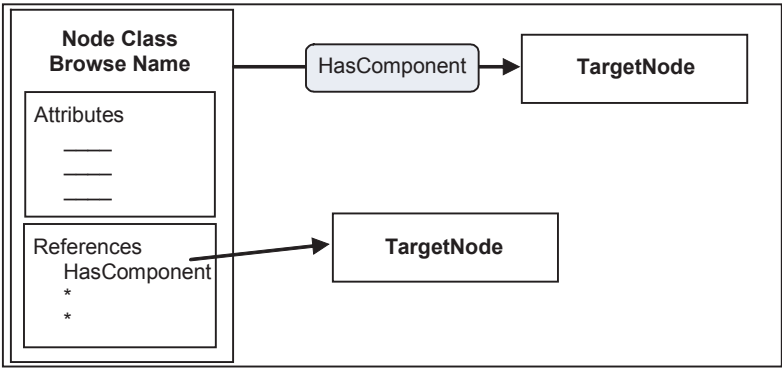


Figure 1 – AddressSpace Node diagrams

In these figures, rectangles represent *Nodes*. *Node* rectangles may be titled with one or two lines of text. When two lines are used, the first text line in the rectangle identifies the *NodeClass* and the second line contains the *BrowseName*. When one line is used, it contains the *BrowseName*.

Node rectangles may contain boxes used to define their *Attributes* and *References*. Specific names in these boxes identify specific *Attributes* and *References*.

Shaded rectangles with rounded corners and with arrows passing through them represent *References*. The arrow that passes through them begins at the *SourceNode* and points to the *TargetNode*. *References* may also be shown by drawing an arrow that starts at the *Reference* name in the “References” box and ends at the *TargetNode*.

3.3.2 Conventions for defining NodeClasses

Clause 5 defines *AddressSpace NodeClasses*. Table 1 describes the format of the tables used to define *NodeClasses*.

Table 1 – NodeClass Table Conventions

Name	Use	Data type	Description
Attributes			
"Attribute name"	"M" or "O"	Data type of the <i>Attribute</i>	Defines the <i>Attribute</i>
References			
"Reference name"	"1", "0..1" or "0..*"	Not used	Describes the use of the <i>Reference</i> by the <i>NodeClass</i> .
Standard Properties			
"Property name"	"M" or "O"	Data type of the <i>Property</i>	Defines the <i>Property</i> .

The Name column contains the name of the *Attribute*, the name of the *ReferenceType* used to create a *Reference* or the name of a *Property* referenced using the *HasProperty Reference*.

The Use column defines whether the *Attribute* or *Property* is mandatory (M) or optional (O). When mandatory the *Attribute* or *Property* shall exist for every *Node* of the *NodeClass*. For *References* it specifies the cardinality. The following values may apply:

- "0..*" identifies that there are no restrictions, that is, the *Reference* does not have to be provided but there is no limitation how often it can be provided;
- "0..1" identifies that the *Reference* is provided at most once;
- "1" identifies that the *Reference* shall be provided exactly once.

The Data Type column contains the name of the *DataType* of the *Attribute* or *Property*. It is not used for *References*.

The Description column contains the description of the *Attribute*, the *Reference* or the *Property*.

Only this standard may define *Attributes*. Thus, all *Attributes* of the *NodeClass* are specified in the table and can only be extended by other parts of this series of standards.

This standard also defines *ReferenceTypes*, but *ReferenceTypes* can also be specified by a *Server* or by a client using the *NodeManagement Services* specified in IEC 62541-4. Thus, the *NodeClass* tables contained in this standard can contain the base *ReferenceType* called *References* identifying that any *ReferenceType* may be used for the *NodeClass*, including system specific *ReferenceTypes*. The *NodeClass* tables only specify how the *NodeClasses* can be used as *SourceNodes* of *References*, not as *TargetNodes*. If a *NodeClass* table allows a *ReferenceType* for its *NodeClass* to be used as *SourceNode*, this is also true for subtypes of the *ReferenceType*. However, subclasses of the *ReferenceType* may restrict its *SourceNodes*.

This standard defines *Properties*, but *Properties* can be defined by other standard organizations or vendors and *Nodes* can have *Properties* that are not standardised. *Properties* defined in this standard are defined by their name, which is mapped to the *BrowseName* having the *NamespaceIndex* 0, which represents the *Namespace* for OPC UA.

The "use" column (optional or mandatory) does not imply a specific *ModellingRule* for *Properties*. Different *Server* implementations will choose to use *ModellingRules* appropriate for them.

4 AddressSpace concepts

4.1 Overview

The following subclauses define the concepts of the *AddressSpace*. Clause 5 defines the *NodeClasses* of the *AddressSpace* representing the *AddressSpace* concepts. Clause 6 defines details on the type model for *ObjectTypes* and *VariableTypes*. Standard *ReferenceTypes*, *DataTypes* and *EventTypes* are defined in Clauses 7 to 9.

The informative Annex A describes general considerations how to use the Address Space Model and the informative Annex B provides a UML Model of the Address Space Model. The normative Annex C defines the OPC Binary Types Description System as a format to specify data type structures and the normative Annex D defines a graphical notation for OPC UA data.

4.2 Object Model

The primary objective of the OPC UA *AddressSpace* is to provide a standard way for *Servers* to represent *Objects* to *Clients*. The OPC UA Object Model has been designed to meet this objective. It defines *Objects* in terms of *Variables* and *Methods*. It also allows relationships to other *Objects* to be expressed. Figure 2 illustrates the model.

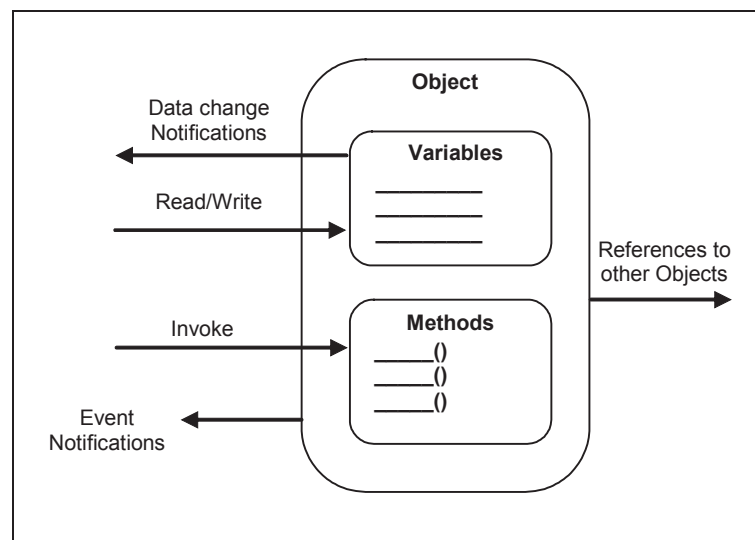


Figure 2 – OPC UA Object Model

The elements of this model are represented in the *AddressSpace* as *Nodes*. Each *Node* is assigned to a *NodeClass* and each *NodeClass* represents a different element of the Object Model. Clause 5 defines the *NodeClasses* used to represent this model.

4.3 Node Model

4.3.1 General

The set of *Objects* and related information that the OPC UA *Server* makes available to *Clients* is referred to as its *AddressSpace*. The model for *Objects* is defined by the OPC UA Object Model (see 4.2).

Objects and their components are represented in the *AddressSpace* as a set of *Nodes* described by *Attributes* and interconnected by *References*. Figure 3 illustrates the model of a *Node* and the remainder of this subclause discusses the details of the Node Model.

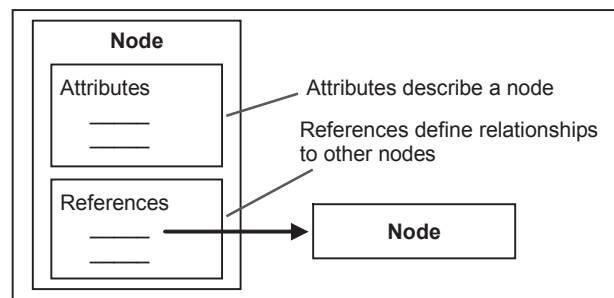


Figure 3 – AddressSpace Node Model

4.3.2 NodeClasses

NodeClasses are defined in terms of the *Attributes* and *References* that shall be instantiated (given values) when a *Node* is defined in the *AddressSpace*. *Attributes* are discussed in 4.3.3 and *References* in 4.3.4.

Clause 5 defines the *NodeClasses* for the *OPC UA AddressSpace*. These *NodeClasses* are referred to collectively as the metadata for the *AddressSpace*. Each *Node* in the *AddressSpace* is an instance of one of these *NodeClasses*. No other *NodeClasses* shall be used to define *Nodes*, and as a result, *Clients* and *Servers* are not allowed to define *NodeClasses* or extend the definitions of these *NodeClasses*.

4.3.3 Attributes

Attributes are data elements that describe *Nodes*. *Clients* can access *Attribute* values using Read, Write, Query, and Subscription/MonitoredItem *Services*. These *Services* are defined in IEC 62541-4.

Attributes are elementary components of *NodeClasses*. *Attribute* definitions are included as part of the *NodeClass* definitions in Clause 5 and, therefore, are not included in the *AddressSpace*.

Each *Attribute* definition consists of an attribute id (for attribute ids of *Attributes*, see IEC 62541-6), a name, a description, a data type and a mandatory/optional indicator. The set of *Attributes* defined for each *NodeClass* shall not be extended by *Clients* or *Servers*.

When a *Node* is instantiated in the *AddressSpace*, the values of the *NodeClass Attributes* are provided. The mandatory/optional indicator for the *Attribute* indicates whether the *Attribute* has to be instantiated.

4.3.4 References

References are used to relate *Nodes* to each other. They can be accessed using the browsing and querying *Services* defined in IEC 62541-4.

Like *Attributes*, they are defined as fundamental components of *Nodes*. Unlike *Attributes*, *References* are defined as instances of *ReferenceType Nodes*. *ReferenceType Nodes* are visible in the *AddressSpace* and are defined using the *ReferenceType NodeClass* (see 5.3).

The *Node* that contains the *Reference* is referred to as the *SourceNode* and the *Node* that is referenced is referred to as the *TargetNode*. The combination of the *SourceNode*, the *ReferenceType* and the *TargetNode* are used in *OPC UA Services* to uniquely identify *References*. Thus, each *Node* can reference another *Node* with the same *ReferenceType* only once. Any subtypes of concrete *ReferenceTypes* are considered to be equal to the base concrete *ReferenceTypes* when identifying *References* (see 5.3 for subtypes of *ReferenceTypes*). Figure 4 illustrates this model of a *Reference*.

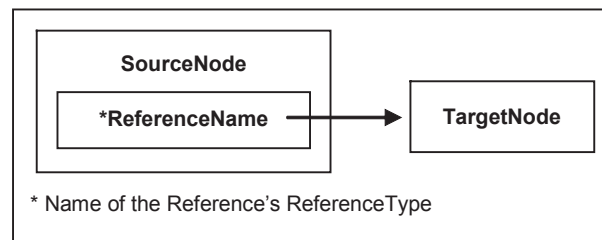


Figure 4 – Reference Model

The *TargetNode* of a *Reference* may be in the same *AddressSpace* or in the *AddressSpace* of another OPC UA *Server*. *TargetNodes* located in other *Servers* are identified in OPC UA *Services* using a combination of the remote *Server* name and the identifier assigned to the *Node* by the remote *Server*.

OPC UA does not require that the *TargetNode* exists, thus *References* may point to a *Node* that does not exist.

4.4 Variables

4.4.1 General

Variables are used to represent *values*. Two types of *Variables* are defined, *Properties* and *DataVariables*. They differ in the kind of data that they represent and whether they can contain other *Variables*.

4.4.2 Properties

Properties are *Server*-defined characteristics of *Objects*, *DataVariables* and other *Nodes*. *Properties* differ from *Attributes* in that they characterise *what* the *Node* represents, such as a device or a purchase order. *Attributes* define additional metadata that is instantiated for all *Nodes* from a *NodeClass*. *Attributes* are common to all *Nodes* of a *NodeClass* and only defined by this specification whereas *Properties* can be *Server*-defined.

For example, an *Attribute* defines the *DataType* of *Variables* whereas a *Property* can be used to specify the engineering unit of some *Variables*.

To prevent recursion, *Properties* are not allowed to have *Properties* defined for them. To easily identify *Properties*, the *BrowseName* of a *Property* shall be unique in the context of the *Node* containing the *Properties* (see 5.6.3 for details).

A *Node* and its *Properties* shall always reside in the same *Server*.

4.4.3 DataVariables

DataVariables represent the content of an *Object*. For example, a file *Object* may be defined that contains a stream of bytes. The stream of bytes may be defined as a *DataVariable* that is an array of bytes. *Properties* may be used to expose the creation time and owner of the file *Object*.

For example, if a *DataVariable* is defined by a data structure that contains two fields, “*startTime*” and “*endTime*” then it might have a *Property* specific to that data structure, such as “*earliestStartTime*”.

As another example, function blocks in control systems might be represented as *Objects*. The parameters of the function block, such as its setpoints, may be represented as *DataVariables*. The function block *Object* might also have *Properties* that describe its execution time and its type.

DataVariables may have additional *DataVariables*, but only if they are complex. In this case, their *DataVariables* shall always be elements of their complex definitions. Following the example

introduced by the description of *Properties* in 4.4.2, the *Server* could expose “startTime” and “endTime” as separate components of the data structure.

As another example, a complex *DataVariable* may define an aggregate of temperature values generated by three separate temperature transmitters that are also visible in the *AddressSpace*. In this case, this complex *DataVariable* could define *HasComponent References* from it to the individual temperature values that it is composed of.

4.5 TypeDefinitionNodes

4.5.1 General

OPC UA Servers shall provide type definitions for *Objects* and *Variables*. The *HasTypeDefinition Reference* shall be used to link an instance with its type definition represented by a *TypeDefinitionNode*. Type definitions are required; however, IEC 62541-5 defines a *BaseObjectType*, a *PropertyType*, and a *BaseDataVariableType* so a *Server* can use such a base type if no more specialised type information is available. *Objects* and *Variables* inherit the *Attributes* specified by their *TypeDefinitionNode* (see 6.4 for details).

In some cases, the *NodeId* used by the *HasTypeDefinition Reference* will be well-known to *Clients* and *Servers*. Organizations may define *TypeDefinitionNodes* that are well-known in the industry. Well-known *NodeIds* of *TypeDefinitionNodes* provide for commonality across OPC UA Servers and allow *Clients* to interpret the *TypeDefinitionNode* without having to read it from the *Server*. Therefore, *Servers* may use well-known *NodeIds* without representing the corresponding *TypeDefinitionNodes* in their *AddressSpace*. However, the *TypeDefinitionNodes* shall be provided for generic *Clients*. These *TypeDefinitionNodes* may exist in another *Server*.

The following example, illustrated in Figure 5, describes the use of the *HasTypeDefinition Reference*. In this example, a setpoint parameter “SP” is represented as a *DataVariable* in the *AddressSpace*. This *DataVariable* is part of an *Object* not shown in the figure.

To provide for a common setpoint definition that can be used by other *Objects*, a specialised *VariableType* is used. Each setpoint *DataVariable* that uses this common definition will have a *HasTypeDefinition Reference* that identifies the common “SetPoint” *VariableType*.

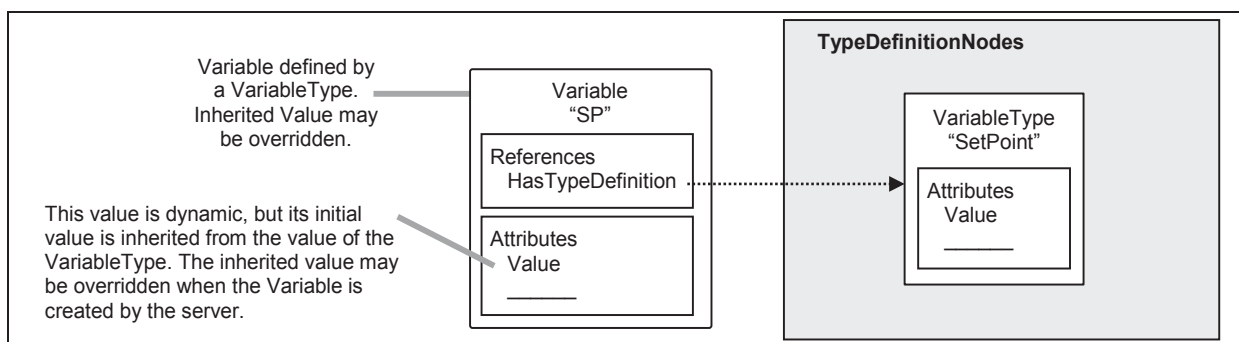


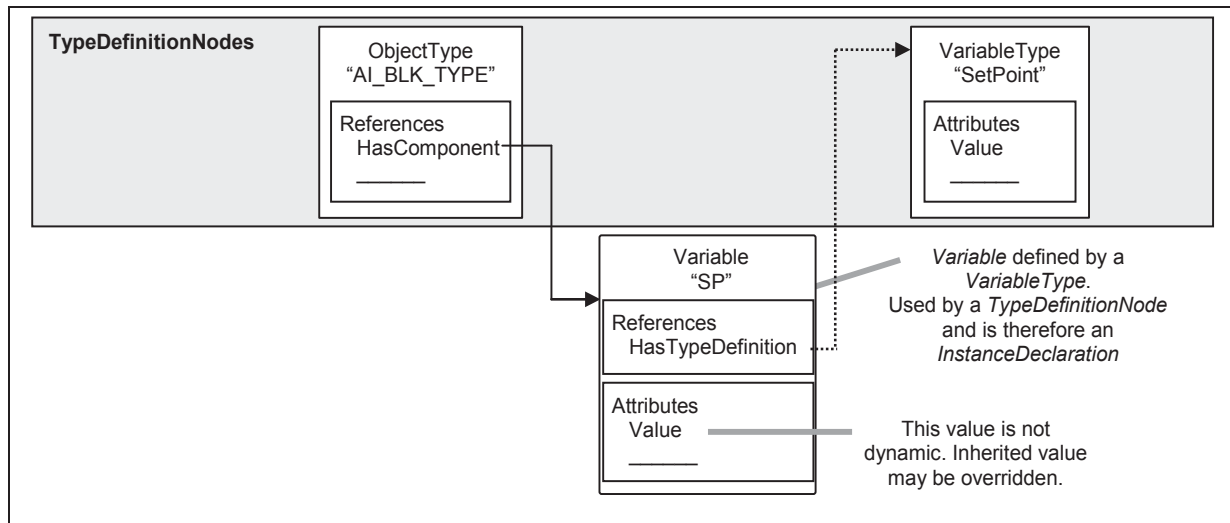
Figure 5 – Example of a Variable Defined by a VariableType

4.5.2 Complex TypeDefinitionNodes and their InstanceDeclarations

TypeDefinitionNodes can be complex. A complex *TypeDefinitionNode* also defines *References* to other *Nodes* as part of the type definition. The *ModellingRules* defined in 6.4.4 specify how those *Nodes* are handled when creating an instance of the type definition.

A *TypeDefinitionNode* references instances instead of other *TypeDefinitionNodes* to allow unique names for several instances of the same type, to define default values and to add *References* for those instances that are specific to this complex *TypeDefinitionNode* and not to the *TypeDefinitionNode* of the instance. For example, in Figure 6 the *ObjectType* “AL_BLK_TYPE” representing a function block, has a *HasComponent Reference* to a *Variable* “SP” of the

686 *VariableType* “SetPoint”. “AI_BLK_TYPE” could have an additional setpoint *Variable* of the same
 687 type using a different name. It could add a *Property* to the *Variable* that was not defined by its
 688 *TypeDefinitionNode* “SetPoint”. And it could define a default value for “SP”, that is, each instance of
 689 “AI_BLK_TYPE” would have a *Variable* “SP” initially set to this value.



690
691 **Figure 6 – Example of a Complex TypeDefinition**

692 This approach is commonly used in object-oriented programming languages in which the variables of
 693 a class are defined as instances of other classes. When the class is instantiated, each variable is
 694 also instantiated, but with the default values (constructor values) defined for the containing class.
 695 That is, typically, the constructor for the component class runs first, followed by the constructor for
 696 the containing class. The constructor for the containing class may override component values set by
 697 the component class.

698 To distinguish instances used for the type definitions from instances that represent real data, those
 699 instances are called *InstanceDeclarations*. However, this term is used to simplify this specification, if
 700 an instance is an *InstanceDeclaration* or not is only visible in the *AddressSpace* by following its
 701 *References*. Some instances may be shared and therefore referenced by *TypeDefinitionNodes*,
 702 *InstanceDeclarations* and instances. This is similar to class variables in object-oriented programming
 703 languages.

704 4.5.3 Subtyping

705 This standard allows subtyping of type definitions. The subtyping rules are defined in Clause 6.
 706 Subtyping of *ObjectTypes* and *VariableTypes* allows:

- 707 • *Clients* that only know the supertype are able to handle an instance of the subtype as if it were
708 an instance of the supertype;
- 709 • instances of the supertype can be replaced by instances of the subtype;
- 710 • specialised types that inherit common characteristics of the base type.

711 In other words, subtypes reflect the structure defined by their supertype but may add additional
 712 characteristics. For example, a vendor may wish to extend a general “TemperatureSensor”
 713 *VariableType* by adding a *Property* providing the next maintenance interval. The vendor would do
 714 this by creating a new *VariableType* which is a *TargetNode* for a *HasSubtype* reference from the
 715 original *VariableType* and adding the new *Property* to it.

716 4.5.4 Instantiation of complex TypeDefinitionNodes

717 The instantiation of complex *TypeDefinitionNodes* depends on the *ModellingRules* defined in 6.4.4.
 718 However, the intention is that instances of a type definition will reflect the structure defined by the
 719 *TypeDefinitionNode*. Figure 7 shows an instance of the *TypeDefinitionNode* “AI_BLK_TYPE”, where

the *ModellingRule Mandatory*, defined in 6.4.4.5.2, was applied for its containing *Variable*. Thus, an instance of “AI_BLK_TYPE”, called AI_BLK_1”, has a *HasTypeDefinition Reference* to “AI_BLK_TYPE”. It also contains a *Variable* “SP” having the same *BrowseName* as the *Variable* “SP” used by the *TypeDefinitionNode* and thereby reflects the structure defined by the *TypeDefinitionNode*.

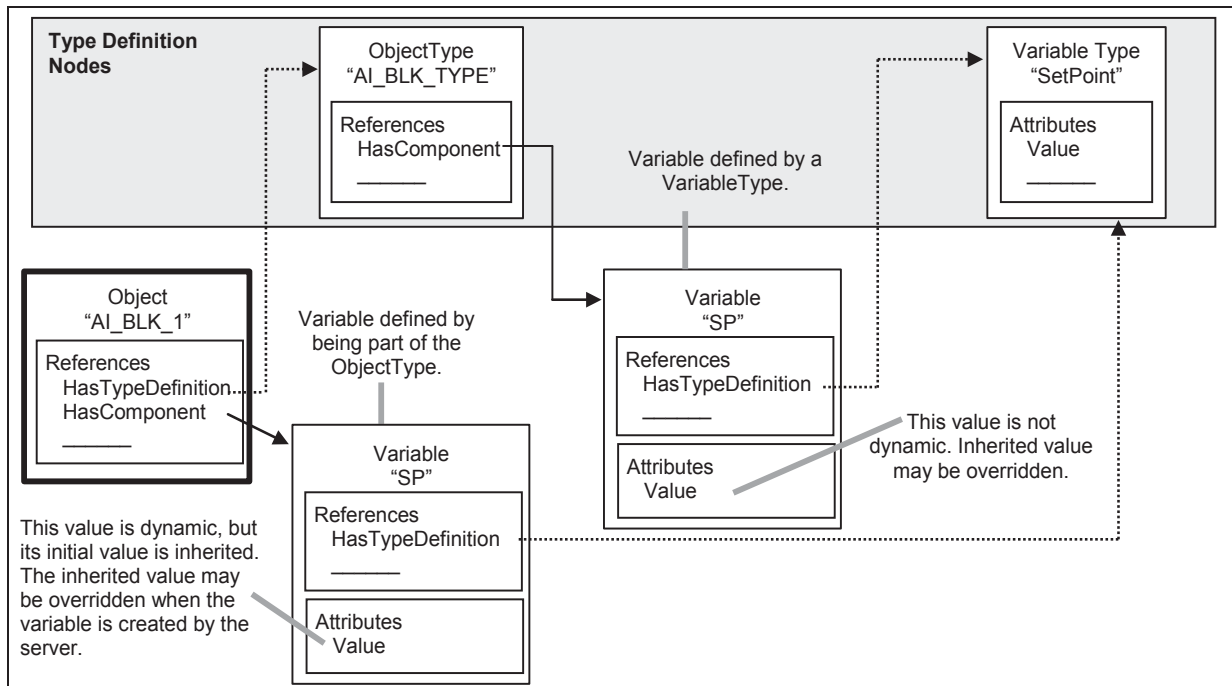


Figure 7 – Object and its Components defined by an ObjectType

A client knowing the *ObjectType* “AI_BLK_TYPE” can use this knowledge to directly browse to the containing *Nodes* for each instance of this type. This allows programming against the *TypeDefinitionNode*. For example, a graphical element may be programmed in the client that handles all instances of “AI_BLK_TYPE” in the same way by showing the value of “SP”.

There are several constraints related to programming against the *TypeDefinitionNode*. A *TypeDefinitionNode* or an *InstanceDeclaration* shall never reference two *Nodes* having the same *BrowseName* using *hierarchical References* in forward direction. Instances based on *InstanceDeclarations* shall always keep the same *BrowseName* as the *InstanceDeclaration* they are derived from. A special *Service* defined in IEC 62541-4 called *TranslateBrowsePathsToNodeIds* may be used to identify the instances based on the *InstanceDeclarations*. Using the simple *Browse Service* might not be sufficient since the uniqueness of the *BrowseName* is only required for *TypeDefinitionNodes* and *InstanceDeclarations*, not for other instances. Thus, “AI_BLK_1” may have another *Variable* with the *BrowseName* “SP”, although this one would not be derived from an *InstanceDeclaration* of the *TypeDefinitionNode*.

Instances derived from an *InstanceDeclaration* shall be of the same *TypeDefinitionNode* or a subtype of this *TypeDefinitionNode*.

A *TypeDefinitionNode* and its *InstanceDeclarations* shall always reside in the same *Server*. However, instances may point with their *HasTypeDefinition Reference* to a *TypeDefinitionNode* in a different *Server*.

4.6 Event Model

4.6.1 General

The Event Model defines a general purpose eventing system that can be used in many diverse vertical markets.

750 *Events* represent specific transient occurrences. System configuration changes and system errors
751 are examples of *Events*. *Event Notifications* report the occurrence of an *Event*. *Events* defined in this
752 document are not directly visible in the OPC UA *AddressSpace*. *Objects* and *Views* can be used to
753 subscribe to *Events*. The *EventNotifier Attribute* of those *Nodes* identifies if the *Node* allows
754 subscribing to *Events*. *Clients* subscribe to such *Nodes* to receive *Notifications* of *Event*
755 occurrences.

756 *Event Subscriptions* use the Monitoring and Subscription *Services* defined in IEC 62541-4 to
757 subscribe to *Event Notifications* of a *Node*.

758 Any OPC UA *Server* that supports eventing shall expose at least one *Node* as *EventNotifier*. The
759 *Server Object* defined in IEC 62541-5 is used for this purpose. *Events* generated by the *Server* are
760 available via this *Server Object*. A *Server* is not expected to produce *Events* if the connection to the
761 event source is down for some reason (i.e. the system is offline).

762 *Events* may also be exposed through other *Nodes* anywhere in the *AddressSpace*. These *Nodes*
763 (identified via the *EventNotifier Attribute*) provide some subset of the *Events* generated by the
764 *Server*. The position in the *AddressSpace* dictates what this subset will be. For example, a process
765 area *Object* representing a functional area of the process would provide *Events* originating from that
766 area of the process only. It should be noted that this is only an example and it is fully up to the
767 *Server* to determine what *Events* should be provided by which *Node*.

768 4.6.2 EventTypes

769 Each *Event* is of a specific *EventType*. A *Server* may support many types. This part defines the
770 *BaseEventType* that all other *EventTypes* derive from. It is expected that other companion
771 specifications will define additional *EventTypes* deriving from the base types defined in this part.

772 The *EventTypes* supported by a *Server* are exposed in the *AddressSpace* of a *Server*. *EventTypes*
773 are represented as *ObjectTypes* in the *AddressSpace* and do not have a special *NodeClass*
774 associated to them. IEC 62541-5 defines how a *Server* exposes the *EventTypes* in detail.

775 *EventTypes* defined in this document are specified as abstract and therefore never instantiated in
776 the *AddressSpace*. Event occurrences of those *EventTypes* are only exposed via a *Subscription*.
777 *EventTypes* exist in the *AddressSpace* to allow *Clients* to discover the *EventType*. This information
778 is used by a client when establishing and working with *Event Subscriptions*. *EventTypes* defined by
779 other parts of this standard or companion specifications as well as *Server* specific *EventTypes* may
780 be defined as not abstract and therefore instances of those *EventTypes* may be visible in the
781 *AddressSpace* although *Events* of those *EventTypes* are also accessible via the *Event Notification*
782 mechanisms.

783 Standard *EventTypes* are described in Clause 9. Their representation in the *AddressSpace* is
784 specified in IEC 62541-5.

785 4.6.3 Event Categorization

786 *Events* can be categorised by creating new *EventTypes* which are subtypes of existing *EventTypes*
787 but do not extend an existing type. They are used only to identify an event as being of the new
788 *EventType*. For example, the *EventType* *DeviceFailureEventType* could be subtyped into
789 *TransmitterFailureEventType* and *ComputerFailureEventType*. These new subtypes would not add
790 new *Properties* or change the semantic inherited from the *DeviceFailureEventType* other than purely
791 for categorization of the *Events*.

792 *Event* sources can also be organised into groups by using the *Event ReferenceTypes* described in
793 7.17 and 7.19. For example, a *Server* may define *Objects* in the *AddressSpace* representing *Events*
794 related to physical devices, or *Event* areas of a plant or functionality contained in the *Server*. *Event*
795 *References* would be used to indicate which *Event* sources represent physical devices and which
796 ones represent some *Server*-based functionality. In addition, *References* can be used to group the
797 physical devices or *Server*-based functionality into hierarchical *Event* areas. In some cases, an

798 *Event* source may be categorised as being both a device and a *Server* function. In this case, two
799 relationships would be established. Refer to the description of the *Event ReferenceTypes* for
800 additional examples.

801 *Clients* can select a category or categories of *Events* by defining content filters that include terms
802 specifying the *EventType* of the *Event* or a grouping of *Event* sources. The two mechanisms allow
803 for a single *Event* to be categorised in multiple manners. A client could obtain all *Events* related to a
804 physical device or all failures of a particular device.

805 4.7 Methods

806 *Methods* are “lightweight” functions, whose scope is bounded by an owning (see Note) *Object*,
807 similar to the methods of a class in object-oriented programming or an owning *ObjectType*, similar to
808 static methods of a class. *Methods* are invoked by a client, proceed to completion on the *Server* and
809 return the result to the client. The lifetime of the *Method*’s invocation instance begins when the client
810 calls the *Method* and ends when the result is returned.

811 NOTE The owning *Object* or *ObjectType* is specified in the service call when invoking the *Method*.

812 While *Methods* may affect the state of the owning *Object*, they have no explicit state of their own. In
813 this sense, they are stateless. *Methods* can have a varying number of input arguments and return
814 resultant arguments. Each *Method* is described by a *Node* of the *Method NodeClass*. This *Node*
815 contains the metadata that identifies the *Method*’s arguments and describes its behaviour.

816 *Methods* are invoked by using the *Call Service* defined in IEC 62541-4. Each *Method* is invoked
817 within the context of an existing session. If the session is terminated during *Method* execution, the
818 results of the *Method*’s execution cannot be returned to the client and are discarded. In that case,
819 the *Method* execution is undefined, that is, the *Method* may be executed until it is finished or it may
820 be aborted.

821 *Clients* discover the *Methods* supported by a *Server* by browsing for the owning *Objects References*
822 that identify their supported *Methods*.

823 5 Standard NodeClasses

824 5.1 Overview

825 This clause defines the *NodeClasses* used to define *Nodes* in the *OPC UA AddressSpace*.
826 *NodeClasses* are derived from a common, *Base NodeClass*. This *NodeClass* is defined first, followed
827 by those used to organise the *AddressSpace* and then by the *NodeClasses* used to represent
828 *Objects*.

829 The *NodeClasses* defined to represent *Objects* fall into three categories: those used to define
830 instances, those used to define types for those instances and those used to define data types. 6.3
831 describes the rules for subtyping and 6.4 the rules for instantiation of the type definitions.

832 **5.2 Base NodeClass**833 **5.2.1 General**

834 The OPC UA Address Space Model defines a *Base NodeClass* from which all other *NodeClasses* are
 835 derived. The derived *NodeClasses* represent the various components of the OPC UA Object Model
 836 (see 4.2). The *Attributes* of the *Base NodeClass* are specified in Table 2. There are no *References*
 837 specified for the *Base NodeClass*.

838 **Table 2 – Base NodeClass**

Name	Use	Data Type	Description
Attributes			
NodeId	M	NodeId	See 5.2.2
NodeClass	M	NodeClass	See 5.2.3
BrowseName	M	QualifiedName	See 5.2.4
DisplayName	M	LocalizedText	See 0
Description	O	LocalizedText	See 5.2.6
WriteMask	O	UInt32	See 5.2.7
UserWriteMask	O	UInt32	See 5.2.8
References			
			No <i>References</i> specified for this <i>NodeClass</i>

839

840 **5.2.2 NodeId**

841 *Nodes* are unambiguously identified using a constructed identifier called the *NodeId*. Some *Servers*
 842 may accept alternative *NodeIds* in addition to the canonical *NodeId* represented in this *Attribute*. A
 843 *Server* shall persist the *NodeId* of a *Node*, that is, it shall not generate new *NodeIds* when rebooting.
 844 The structure of the *NodeId* is defined in 8.2.

845 **5.2.3 NodeClass**

846 The *NodeClass Attribute* identifies the *NodeClass* of a *Node*. Its data type is defined in 8.30.

847 **5.2.4 BrowseName**

848 *Nodes* have a *BrowseName Attribute* that is used as a non-localised human-readable name when
 849 browsing the *AddressSpace* to create paths out of *BrowseNames*. The
 850 *TranslateBrowsePathsToNodeIds Service* defined in IEC 62541-4 can be used to follow a path
 851 constructed of *BrowseNames*.

852 A *BrowseName* should never be used to display the name of a *Node*. The *DisplayName* should be
 853 used instead for this purpose.

854 Unlike *NodeIds*, the *BrowseName* cannot be used to unambiguously identify a *Node*. Different *Nodes*
 855 may have the same *BrowseName*.

856 Clause 8.3 defines the structure of the *BrowseName*. It contains a namespace and a string. The
 857 namespace is provided to make the *BrowseName* unique in some cases in the context of a *Node*
 858 (e.g. *Properties* of a *Node*) although not unique in the context of the *Server*. If different organizations
 859 define *BrowseNames* for *Properties*, the namespace of the *BrowseName* provided by the
 860 organization makes the *BrowseName* unique, although different organizations may use the same
 861 string having a slightly different meaning.

862 *Servers* may often choose to use the same namespace for the *NodeId* and the *BrowseName*.
 863 However, if they want to provide a standard *Property*, its *BrowseName* shall have the namespace of
 864 the standards body although the namespace of the *NodeId* reflects something else, for example the
 865 local *Server*.

866 It is recommended that standard bodies defining standard type definitions use their namespace for
 867 the *NodeId* of the *TypeDefinitionNode* as well as for the *BrowseName* of the *TypeDefinitionNode*.

868 The string-part of the *BrowseName* is case sensitive.

869 5.2.5 DisplayName

870 The *DisplayName Attribute* contains the localised name of the *Node*. *Clients* should use this *Attribute*
871 if they want to display the name of the *Node* to the user. They should not use the *BrowseName* for
872 this purpose. The *Server* may maintain one or more localised representations for each *DisplayName*.
873 *Clients* negotiate the locale to be returned when they open a session with the *Server*. Refer to IEC
874 IEC 62541-4 for a description of session establishment and locales. 8.5 defines the structure of the
875 *DisplayName*. The string part of the *DisplayName* is restricted to 512 characters.

876 5.2.6 Description

877 The optional *Description Attribute* shall explain the meaning of the *Node* in a localised text using the
878 same mechanisms for localisation as described for the *DisplayName* in 0.

879 5.2.7 WriteMask

880 The optional *WriteMask Attribute* exposes the possibilities of a client to write the *Attributes* of the
881 *Node*. The *WriteMask Attribute* does not take any user access rights into account, that is, although
882 an *Attribute* is writeable this may be restricted to a certain user / user group.

883 If the OPC UA *Server* does not have the ability to get the *WriteMask* information for a specific
884 *Attribute* from the underlying system, it should state that it is writable. If a write operation is called on
885 the *Attribute*, the *Server* should transfer this request and return the corresponding *StatusCode* if
886 such a request is rejected. *StatusCodes* are defined in IEC 62541-4.

887 The *WriteMask Attribute* is a 32-bit unsigned integer with the structure defined in Table 3. If the bit is
888 set to 0, it means the *Attribute* is not writeable, if it is set to 1, it means it is writable. If a *Node* does
889 not support a specific *Attribute*, the corresponding bit has to be set to 0.

890 **Table 3 – Bit mask for WriteMask and UserWriteMask**

Field	Bit	Description
AccessLevel	0	Indicates if the AccessLevel Attribute is writable.
ArrayDimensions	1	Indicates if the ArrayDimensions Attribute is writable.
BrowseName	2	Indicates if the BrowseName Attribute is writable.
ContainsNoLoops	3	Indicates if the ContainsNoLoops Attribute is writable.
DataType	4	Indicates if the DataType Attribute is writable.
Description	5	Indicates if the Description Attribute is writable.
DisplayName	6	Indicates if the DisplayName Attribute is writable.
EventNotifier	7	Indicates if the EventNotifier Attribute is writable.
Executable	8	Indicates if the Executable Attribute is writable.
Historizing	9	Indicates if the Historizing Attribute is writable.
InverseName	10	Indicates if the InverseName Attribute is writable.
IsAbstract	11	Indicates if the IsAbstract Attribute is writable.
MinimumSamplingInterval	12	Indicates if the MinimumSamplingInterval Attribute is writable.
NodeClass	13	Indicates if the NodeClass Attribute is writable.
NodeId	14	Indicates if the NodeId Attribute is writable.
Symmetric	15	Indicates if the Symmetric Attribute is writable.
UserAccessLevel	16	Indicates if the UserAccessLevel Attribute is writable.
UserExecutable	17	Indicates if the UserExecutable Attribute is writable.
UserWriteMask	18	Indicates if the UserWriteMask Attribute is writable.
ValueRank	19	Indicates if the ValueRank Attribute is writable.
WriteMask	20	Indicates if the WriteMask Attribute is writable.
ValueForVariableType	21	Indicates if the <i>Value Attribute</i> is writable for a <i>VariableType</i> . It does not apply for <i>Variables</i> since this is handled by the <i>AccessLevel</i> and <i>UserAccessLevel Attributes</i> for the <i>Variable</i> . For <i>Variables</i> this bit shall be set to 0.
Reserved	22:31	Reserved for future use. Shall always be zero.

891 **5.2.8 UserWriteMask**

892 The optional *UserWriteMask Attribute* exposes the possibilities of a client to write the *Attributes* of
 893 the *Node* taking user access rights into account. It uses the same bit mask as used in the *WriteMask*
 894 *Attribute*, defined in Table 3.

895 The *UserWriteMask Attribute* can only further restrict the *WriteMask Attribute*, when it is set to not
 896 writeable in the general case that applies for every user.

897 **5.3 ReferenceType NodeClass**898 **5.3.1 General**

899 *References* are defined as instances of *ReferenceType Nodes*. *ReferenceType Nodes* are visible in
 900 the *AddressSpace* and are defined using the *ReferenceType NodeClass* as specified in Table 4. In
 901 contrast, a *Reference* is an inherent part of a *Node* and no *NodeClass* is used to represent
 902 *References*.

903 This standard defines a set of *ReferenceTypes* provided as an inherent part of the OPC UA Address
 904 Space Model. These *ReferenceTypes* are defined in Clause 7 and their representation in the
 905 *AddressSpace* is defined in IEC 62541-5. *Servers* may also define *ReferenceTypes*. In addition,
 906 IEC 62541-4 defines *NodeManagement Services* that allow *Clients* to add *ReferenceTypes* to the
 907 *AddressSpace*.

908 **Table 4 – ReferenceType NodeClass**

Name	Use	Data Type	Description
Attributes			
Base NodeClass Attributes	M	--	Inherited from the <i>Base NodeClass</i> . See 5.2
IsAbstract	M	Boolean	A boolean <i>Attribute</i> with the following values: TRUE it is an abstract <i>ReferenceType</i> , i.e. no <i>References</i> of this type shall exist, only of its subtypes. FALSE it is not an abstract <i>ReferenceType</i> , i.e. <i>References</i> of this type can exist.
Symmetric	M	Boolean	A boolean <i>Attribute</i> with the following values: TRUE the meaning of the <i>ReferenceType</i> is the same as seen from both the <i>SourceNode</i> and the <i>TargetNode</i> . FALSE the meaning of the <i>ReferenceType</i> as seen from the <i>TargetNode</i> is the inverse of that as seen from the <i>SourceNode</i> .
InverseName	O	LocalizedText	The inverse name of the <i>Reference</i> , that is the meaning of the <i>ReferenceType</i> as seen from the <i>TargetNode</i> .
References			
HasProperty	0..*		Used to identify the <i>Properties</i> (see 5.3.3.2)
HasSubtype	0..*		Used to identify subtypes (see 5.3.3.3)
Standard Properties			
NodeVersion	O	String	The <i>NodeVersion Property</i> is used to indicate the version of a <i>Node</i> . The <i>NodeVersion Property</i> is updated each time a <i>Reference</i> is added or deleted to the <i>Node</i> the <i>Property</i> belongs to. <i>Attribute</i> value changes do not cause the <i>NodeVersion</i> to change. <i>Clients</i> may read the <i>NodeVersion Property</i> or subscribe to it to determine when the structure of a <i>Node</i> has changed.

909

910 **5.3.2 Attributes**

911 The *ReferenceType NodeClass* inherits the base *Attributes* from the *Base NodeClass* defined in 5.2.
 912 The inherited *BrowseName Attribute* is used to specify the meaning of the *ReferenceType* as seen
 913 from the *SourceNode*. For example, the *ReferenceType* with the *BrowseName* “Contains” is used in
 914 *References* that specify that the *SourceNode* contains the *TargetNode*. The inherited *DisplayName*
 915 *Attribute* contains a translation of the *BrowseName*.

916 The *BrowseName* of a *ReferenceType* shall be unique in a *Server*. It is not allowed that two different
917 *ReferenceTypes* have the same *BrowseName*.

918 The *IsAbstract Attribute* indicates if the *ReferenceType* is abstract. Abstract *ReferenceTypes* cannot
919 be instantiated and are used only for organizational reasons, for example to specify some general
920 semantics or constraints that are inherited to its subtypes.

921 The *Symmetric Attribute* is used to indicate whether or not the meaning of the *ReferenceType* is the
922 same for both the *SourceNode* and *TargetNode*.

923 If a *ReferenceType* is symmetric, the *InverseName Attribute* shall be omitted. Examples of
924 symmetric *ReferenceTypes* are “Connects To” and “Communicates With”. Both imply the same
925 semantic coming from the *SourceNode* or the *TargetNode*. Therefore both directions are considered
926 to be forward References.

927 If the *ReferenceType* is non-symmetric and not abstract, the *InverseName Attribute* shall be set. The
928 *InverseName Attribute* specifies the meaning of the *ReferenceType* as seen from the *TargetNode*.
929 Examples of non-symmetric *ReferenceTypes* include “Contains” and “Contained In”, and “Receives
930 From” and “Sends To”.

931 *References* that use the *InverseName*, such as “Contained In” *References*, are referred to as inverse
932 *References*.

933 Figure 8 provides examples of symmetric and non-symmetric *References* and the use of the
934 *BrowseName* and the *InverseName*.

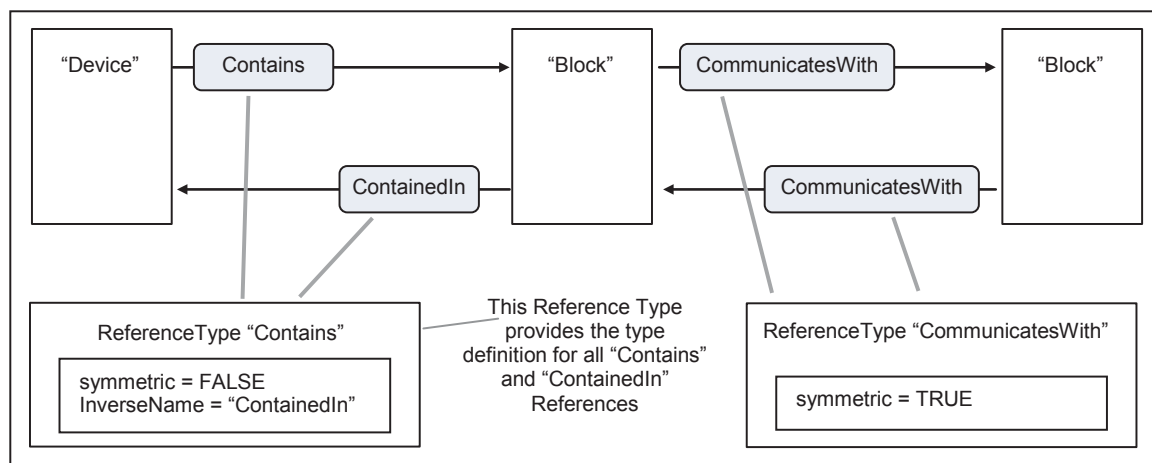


Figure 8 – Symmetric and Non-Symmetric References

937 It might not always be possible for *Servers* to instantiate both forward and inverse *References* for
938 non-symmetric *ReferenceTypes* as shown in Figure 8. When they do, the *References* are referred to
939 as *bidirectional*. Although not required, it is recommended that all *hierarchical References* be
940 instantiated as bidirectional to ensure browse connectivity. A bidirectional *Reference* is modelled as
941 two separate *References*.

942 As an example of a *unidirectional Reference*, it is often the case that a subscriber knows its
943 publisher, but its publisher does not know its subscribers. The subscriber would have a “Subscribes
944 To” *Reference* to the publisher, without the publisher having the corresponding “Publishes To”
945 inverse *References* to its subscribers.

946 The *DisplayName* and the *InverseName* are the only standardised places to indicate the semantic of
947 a *ReferenceType*. There may be more complex semantics associated with a *ReferenceType* than
948 can be expressed in those *Attributes* (e.g. the semantic of *HasSubtype*). This standard does not

949 specify how this semantic should be exposed. However, the *Description Attribute* can be used for
950 this purpose. This standard provides a semantic for the *ReferenceTypes* specified in Clause 7.

951 A *ReferenceType* can have constraints restricting its use. For example, it can specify that starting
952 from *Node A* and only following *References* of this *ReferenceType* or one of its subtypes shall never
953 be able to return to *A*, that is, a “No Loop” constraint.

954 This standard does not specify how those constraints could or should be made available in the
955 *AddressSpace*. Nevertheless, for the standard *ReferenceTypes*, some constraints are specified in
956 Clause 7. This standard does not restrict the kind of constraints valid for a *ReferenceType*. It can, for
957 example, also affect an *ObjectType*. The restriction that a *ReferenceType* can only be used relating
958 *Nodes* of some *NodeClasses* with a defined cardinality is a special constraint of a *ReferenceType*.

959 5.3.3 References

960 5.3.3.1 General

961 *HasSubtype References* and *HasProperty References* are the only *ReferenceTypes* that may be
962 used with *ReferenceType Nodes* as *SourceNode*. *ReferenceType Nodes* shall not be the
963 *SourceNode* of other types of *References*.

964 5.3.3.2 HasProperty References

965 *HasProperty References* are used to identify the *Properties* of a *ReferenceType* and shall only refer
966 to *Nodes* of the *Variable NodeClass*.

967 The *Property NodeVersion* is used to indicate the version of the *ReferenceType*.

968 There are no additional *Properties* defined for *ReferenceTypes* in this standard. Additional parts of
969 this series of standards may define additional *Properties* for *ReferenceTypes*.

970 5.3.3.3 HasSubtype References

971 *HasSubtype References* are used to define subtypes of *ReferenceTypes*. It is not required to provide
972 the *HasSubtype Reference* for the supertype, but it is required that the subtype provides the inverse
973 *Reference* to its supertype. The following rules for subtyping apply.

974 a) The semantic of a *ReferenceType* (e.g. “spans a hierarchy”) is inherited to its subtypes and can
975 be refined there (e.g. “spans a special hierarchy”). The *DisplayName*, and also the *InverseName*
976 for non-symmetric *ReferenceTypes*, reflect the specialization.

977 b) If a *ReferenceType* specifies some constraints (e.g. “allow no loops”) this is inherited and can
978 only be refined (e.g. inheriting “no loops” could be refined as “shall be a tree – only one parent”)
979 but not lowered (e.g. “allow loops”).

980 c) The constraints concerning which *NodeClasses* can be referenced are also inherited and can
981 only be further restricted. That is, if a *ReferenceType* “A” is not allowed to relate an *Object* with
982 an *ObjectType*, this is also true for its subtypes.

983 d) A *ReferenceType* shall have exactly one supertype, except for the *References ReferenceType*
984 defined in 7.2 as the root type of the *ReferenceType* hierarchy. The *ReferenceType* hierarchy
985 does not support multiple inheritances.

986 5.4 View NodeClass

987 Underlying systems are often large and *Clients* often have an interest in only a specific subset of the
988 data. They do not need, or want, to be burdened with viewing *Nodes* in the *AddressSpace* for which
989 they have no interest.

990 To address this problem, this specification defines the concept of a *View*. Each *View* defines a
991 subset of the *Nodes* in the *AddressSpace*. The entire *AddressSpace* is the default *View*. Each *Node*
992 in a *View* may contain only a subset of its *References*, as defined by the creator of the *View*. The
993 *View Node* acts as the root for the *Nodes* in the *View*. *Views* are defined using the *View NodeClass*,
994 which is specified in Table 5.

995 All *Nodes* contained in a *View* shall be accessible starting from the *View Node* when browsing in the
 996 context of the *View*. It is not expected that all containing *Nodes* can be browsed directly from the
 997 *View Node* but rather browsed from other *Nodes* contained in the *View*.

998 A *View Node* may not only be used as additional entry point into the *AddressSpace* but as a
 999 construct to organize the *AddressSpace* and thus as the only entry point into a subset of the
 1000 *AddressSpace*. Therefore *Clients* shall not ignore *View Nodes* when exposing the *AddressSpace*.
 1001 Simple *Clients* that do not deal with *Views* for filtering purposes can, for example, handle a *View*
 1002 *Node* like an *Object* of type *FolderType* (see 5.5.3).

1003

Table 5 – View NodeClass

Name	Use	Data Type	Description																		
Attributes																					
Base NodeClass Attributes	M	--	Inherited from the <i>Base NodeClass</i> . See 5.2																		
ContainsNoLoops	M	Boolean	If set to “true” this <i>Attribute</i> indicates that following <i>References</i> in the context of the <i>View</i> contains no loops, i.e. starting from a <i>Node</i> “A” contained in the <i>View</i> and following the forward <i>References</i> in the context of the <i>View Node</i> “A” will not be reached again. It does not specify that there is only one path starting from the <i>View Node</i> to reach a <i>Node</i> contained in the <i>View</i> . If set to “false” this <i>Attribute</i> indicates that following <i>References</i> in the context of the <i>View</i> may lead to loops.																		
EventNotifier	M	Byte	The <i>EventNotifier Attribute</i> is used to indicate if the <i>Node</i> can be used to subscribe to <i>Events</i> or to read / write historic <i>Events</i> . The <i>EventNotifier</i> is an 8-bit unsigned integer with the structure defined in the following table.																		
			<table><tr><th>Field</th><th>Bit</th><th>Description</th></tr><tr><td>SubscribeTo Events</td><td>0</td><td>Indicates if it can be used to subscribe to <i>Events</i> (0 means cannot be used to subscribe to <i>Events</i>, 1 means can be used to subscribe to <i>Events</i>).</td></tr><tr><td>Reserved</td><td>1</td><td>Reserved for future use. Shall always be zero.</td></tr><tr><td>HistoryRead</td><td>2</td><td>Indicates if the history of the <i>Events</i> is readable (0 means not readable, 1 means readable).</td></tr><tr><td>HistoryWrite</td><td>3</td><td>Indicates if the history of the <i>Events</i> is writable (0 means not writable, 1 means writable).</td></tr><tr><td>Reserved</td><td>4:7</td><td>Reserved for future use. Shall always be zero.</td></tr></table>	Field	Bit	Description	SubscribeTo Events	0	Indicates if it can be used to subscribe to <i>Events</i> (0 means cannot be used to subscribe to <i>Events</i> , 1 means can be used to subscribe to <i>Events</i>).	Reserved	1	Reserved for future use. Shall always be zero.	HistoryRead	2	Indicates if the history of the <i>Events</i> is readable (0 means not readable, 1 means readable).	HistoryWrite	3	Indicates if the history of the <i>Events</i> is writable (0 means not writable, 1 means writable).	Reserved	4:7	Reserved for future use. Shall always be zero.
			Field	Bit	Description																
			SubscribeTo Events	0	Indicates if it can be used to subscribe to <i>Events</i> (0 means cannot be used to subscribe to <i>Events</i> , 1 means can be used to subscribe to <i>Events</i>).																
			Reserved	1	Reserved for future use. Shall always be zero.																
			HistoryRead	2	Indicates if the history of the <i>Events</i> is readable (0 means not readable, 1 means readable).																
			HistoryWrite	3	Indicates if the history of the <i>Events</i> is writable (0 means not writable, 1 means writable).																
			Reserved	4:7	Reserved for future use. Shall always be zero.																
The second two bits also indicate if the history of the <i>Events</i> is available via the OPC UA <i>Server</i> .																					
References																					
HierarchicalReferences	0..*		Top level <i>Nodes</i> in a <i>View</i> are referenced by <i>hierarchical References</i> (see 7.3).																		
HasProperty	0..*		<i>HasProperty References</i> identify the <i>Properties</i> of the <i>View</i> .																		
Standard Properties																					
NodeVersion	O	String	The <i>NodeVersion Property</i> is used to indicate the version of a <i>Node</i> . The <i>NodeVersion Property</i> is updated each time a <i>Reference</i> is added or deleted to the <i>Node</i> the <i>Property</i> belongs to. <i>Attribute</i> value changes do not cause the <i>NodeVersion</i> to change. <i>Clients</i> may read the <i>NodeVersion Property</i> or subscribe to it to determine when the structure of a <i>Node</i> has changed.																		
ViewVersion	O	UInt32	The version number for the <i>View</i> . When <i>Nodes</i> are added to or removed from a <i>View</i> , the value of the <i>ViewVersion Property</i> is updated. <i>Clients</i> may detect changes to the composition of a <i>View</i> using this <i>Property</i> . The value of the <i>ViewVersion</i> shall always be greater than 0.																		

1004

1005 The *View NodeClass* inherits the base *Attributes* from the *Base NodeClass* defined in 5.2. It also
 1006 defines two additional *Attributes*.

- 1007 The mandatory *ContainsNoLoops Attribute* is set to false if the *Server* is not able to identify if the
1008 *View* contains loops or not.
- 1009 The mandatory *EventNotifier Attribute* identifies if the *View* can be used to subscribe to *Events* that
1010 either occur in the content of the *View* or as *ModelChangeEvents* (see 9.32) of the content of the
1011 *View* or to read / write the history of the *Events*. A *View* that supports *Events* shall provide all *Events*
1012 that occur in any *Object* used as *EventNotifier* that is part of the content of the *View*. In addition, it
1013 shall provide all *ModelChangeEvents* that occur in the context of the *View*.
- 1014 To avoid recursion, i.e. getting all *Events* of the *Server*, the *Server Object* defined in IEC 62541-5
1015 shall never be part of any *View* since it provides all *Events* of the *Server*.
- 1016 *Views* are defined by the *Server*. The browsing and querying *Services* defined in IEC 62541-4 expect
1017 the *NodeId* of a *View Node* to provide these *Services* in the context of the *View*.
- 1018 *HasProperty References* are used to identify the *Properties* of a *View*. The *Property NodeVersion* is
1019 used to indicate the version of the *View Node*. The *ViewVersion Property* indicates the version of the
1020 content of the *View*. In contrast to the *NodeVersion*, the *ViewVersion Property* is updated even if
1021 *Nodes* not directly referenced by the *View Node* are added to or deleted from the *View*. This
1022 *Property* is optional because it might not be possible for *Servers* to detect changes in the *View*
1023 contents. *Servers* may also generate a *ModelChangeEvent*, described in 9.32, if *Nodes* are added to
1024 or deleted from the *View*. There are no additional *Properties* defined for *Views* in this document.
1025 Additional parts of this series of standards may define additional *Properties* for *Views*.
- 1026 *Views* can be the *SourceNode* of any *hierarchical Reference*. They shall not be the *SourceNode* of
1027 any *non-hierarchical Reference*.

1028 **5.5 Objects**1029 **5.5.1 Object NodeClass**

1030 *Objects* are used to represent systems, system components, real-world objects and software
 1031 objects. *Objects* are defined using the *Object NodeClass*, specified in Table 6.

1032

Table 6 – Object NodeClass

Name	Use	Data Type	Description																		
Attributes																					
Base NodeClass Attributes	M	--	Inherited from the <i>Base NodeClass</i> . See 5.2																		
EventNotifier	M	Byte	The <i>EventNotifier Attribute</i> is used to indicate if the <i>Node</i> can be used to subscribe to <i>Events</i> or the read / write historic <i>Events</i>. The <i>EventNotifier</i> is an 8-bit unsigned integer with the structure defined in the following table: <table><tr><th>Field</th><th>Bit</th><th>Description</th></tr><tr><td>SubscribeTo Events</td><td>0</td><td>Indicates if it can be used to subscribe to <i>Events</i> (0 means cannot be used to subscribe to <i>Events</i>, 1 means can be used to subscribe to <i>Events</i>).</td></tr><tr><td>Reserved</td><td>1</td><td>Reserved for future use. shall always be zero.</td></tr><tr><td>HistoryRead</td><td>2</td><td>Indicates if the history of the <i>Events</i> is readable (0 means not readable, 1 means readable).</td></tr><tr><td>HistoryWrite</td><td>3</td><td>Indicates if the history of the <i>Events</i> is writable (0 means not writable, 1 means writable).</td></tr><tr><td>Reserved</td><td>4:7</td><td>Reserved for future use. Shall always be zero.</td></tr></table> The second two bits also indicate if the history of the <i>Events</i> is available via the OPC UA Server.	Field	Bit	Description	SubscribeTo Events	0	Indicates if it can be used to subscribe to <i>Events</i> (0 means cannot be used to subscribe to <i>Events</i> , 1 means can be used to subscribe to <i>Events</i>).	Reserved	1	Reserved for future use. shall always be zero.	HistoryRead	2	Indicates if the history of the <i>Events</i> is readable (0 means not readable, 1 means readable).	HistoryWrite	3	Indicates if the history of the <i>Events</i> is writable (0 means not writable, 1 means writable).	Reserved	4:7	Reserved for future use. Shall always be zero.
Field	Bit	Description																			
SubscribeTo Events	0	Indicates if it can be used to subscribe to <i>Events</i> (0 means cannot be used to subscribe to <i>Events</i> , 1 means can be used to subscribe to <i>Events</i>).																			
Reserved	1	Reserved for future use. shall always be zero.																			
HistoryRead	2	Indicates if the history of the <i>Events</i> is readable (0 means not readable, 1 means readable).																			
HistoryWrite	3	Indicates if the history of the <i>Events</i> is writable (0 means not writable, 1 means writable).																			
Reserved	4:7	Reserved for future use. Shall always be zero.																			
References																					
HasComponent	0..*		<i>HasComponent References</i> identify the <i>DataVariables</i> , the <i>Methods</i> and <i>Objects</i> contained in the <i>Object</i> .																		
HasProperty	0..*		<i>HasProperty References</i> identify the <i>Properties</i> of the <i>Object</i> .																		
HasModellingRule	0..1		<i>Objects</i> can point to at most one <i>ModellingRule Object</i> using a <i>HasModellingRule Reference</i> (see 6.4.4 for details on <i>ModellingRules</i>).																		
HasTypeDefinition	1		The <i>HasTypeDefinition Reference</i> points to the type definition of the <i>Object</i> . Each <i>Object</i> shall have exactly one type definition and therefore be the <i>SourceNode</i> of exactly one <i>HasTypeDefinition Reference</i> pointing to an <i>ObjectType</i> . See 4.5 for a description of type definitions.																		
HasEventSource	0..*		The <i>HasEventSource Reference</i> points to event sources of the <i>Object</i> . <i>References</i> of this type can only be used for <i>Objects</i> having their "SubscribeToEvents" bit set in the <i>EventNotifier Attribute</i> . See 7.17 for details.																		
HasNotifier	0..*		The <i>HasNotifier Reference</i> points to notifiers of the <i>Object</i> . <i>References</i> of this type can only be used for <i>Objects</i> having their "SubscribeToEvents" bit set in the <i>EventNotifier Attribute</i> . See 7.19 for details.																		
Organizes	0..*		This <i>Reference</i> should be used only for <i>Objects</i> of the <i>ObjectType FolderType</i> (see 5.5.3).																		
HasDescription	0..1		This <i>Reference</i> shall be used only for <i>Objects</i> of the <i>ObjectType DataTypeEncodingType</i> (see 5.8.4).																		
<other References>	0..*		<i>Objects</i> may contain other <i>References</i> .																		
Standard Properties																					
NodeVersion	O	String	The <i>NodeVersion Property</i> is used to indicate the version of a <i>Node</i> . The <i>NodeVersion Property</i> is updated each time a <i>Reference</i> is added or deleted to the <i>Node</i> the <i>Property</i> belongs to. <i>Attribute</i> value changes do not cause the <i>NodeVersion</i> to change. <i>Clients</i> may read the <i>NodeVersion Property</i> or subscribe to it to determine when the structure of a <i>Node</i> has changed.																		
Icon	O	Image	The <i>Icon Property</i> provides an image that can be used by <i>Clients</i> when displaying the <i>Node</i> . It is expected that the <i>Icon Property</i> contains a relatively small image.																		
NamingRule	O	NamingRuleType	The <i>NamingRule Property</i> defines the <i>NamingRule</i> of a <i>ModellingRule</i> (see 6.4.4.2.1 for details). This <i>Property</i> shall only be used for <i>Objects</i> of the type <i>ModellingRuleType</i> defined in 6.4.4.1.																		

1033

- 1034 The *Object NodeClass* inherits the base *Attributes* from the *Base NodeClass* defined in 5.2.
- 1035 The mandatory *EventNotifier Attribute* identifies whether the *Object* can be used to subscribe to
1036 *Events* or to read and write the history of the *Events*.
- 1037 The *Object NodeClass* uses the *HasComponent Reference* to define the *DataVariables*, *Objects* and
1038 *Methods* of an *Object*.
- 1039 It uses the *HasProperty Reference* to define the *Properties* of an *Object*. The *Property NodeVersion*
1040 is used to indicate the version of the *Object*. The *Property Icon* provides an icon of the *Object*. The
1041 *Property NamingRule* defines the *NamingRule* of a *ModellingRule* and shall only be applied to
1042 *Objects* of type *ModellingRuleType*. There are no additional *Properties* defined for *Objects* in this
1043 document. Additional parts of this series of standards may define additional *Properties* for *Objects*.
- 1044 To specify its *ModellingRule*, an *Object* can use at most one *HasModellingRule Reference* pointing
1045 to a *ModellingRule Object*. *ModellingRules* are defined in 6.4.4.
- 1046 *HasNotifier* and *HasEventSource References* are used to provide information about eventing and
1047 can only be applied to *Objects* used as event notifiers. Details are defined in 7.17 and 7.19.
- 1048 The *HasTypeDefinition Reference* points to the *ObjectType* used as type definition of the *Object*.
- 1049 *Objects* may use any additional *References* to define relationships to other *Nodes*. No restrictions
1050 are placed on the types of *References* used or on the *NodeClasses* of the *Nodes* that may be
1051 referenced. However, restrictions may be defined by the *ReferenceType* excluding its use for
1052 *Objects*. Standard *ReferenceTypes* are described in Clause 7.
- 1053 If the *Object* is used as an *InstanceDeclaration* (see 4.5) then all *Nodes* referenced with *hierarchical*
1054 *References* in a forward direction shall have unique *BrowseNames* in the context of this *Object*.
- 1055 If the *Object* is created based on an *InstanceDeclaration* then it shall have the same *BrowseName*
1056 as its *InstanceDeclaration*.