



DPC: 24 / 30485896 DC

BSI Group Headquarters

389 Chiswick High Road London W4 4AL

Tel: +44 (0)20 8996 9000

Fax: +44 (0)20 8996 7400

www.bsigroup.com

Date: 02 February 2024

Origin: International

Latest date for receipt of comments: 27 March 2024

Project No. 2024/00102

Responsible committee: GEL/65/5 Devices and integration in enterprise systems

Interested committees:

Title: BS EN IEC 62541-3 OPC unified architecture

Part 3: Address space model

Please notify the secretary if you are aware of any keywords that might assist in classifying or identifying the standard or if the content of this standard

- i) has any issues related to 3rd party IPR, patent or copyright
- ii) affects other national standard(s)
- iii) requires additional national guidance or information

**WARNING: THIS IS A DRAFT AND MUST NOT BE REGARDED OR USED AS A BRITISH STANDARD.
THIS DRAFT IS NOT CURRENT BEYOND 27 March 2024**

This draft is issued to allow comments from interested parties; all comments will be given consideration prior to publication. No acknowledgement will normally be sent. **See overleaf for information on the submission of comments.**

No copying is allowed, in any form, without prior written permission from BSI except as permitted under the Copyright, Designs and Patent Act 1988 or for circulation within a nominating organization for briefing purposes. Electronic circulation is limited to dissemination by e-mail within such an organization by committee members.

Further copies of this draft may be purchased from BSI Shop <http://shop.bsigroup.com> or from BSI Customer Services, Tel: +44(0) 20 8996 9001 or email cservices@bsigroup.com. British, International and foreign standards are also available from BSI Customer Services.

Information on the co-operating organizations represented on the committees referenced above may be obtained from <http://standardsdevelopment.bsigroup.com>

Responsible Committee Secretary: Ms Charmaine Mukherjee (BSI)

Direct tel:

E-mail: charmaine.mukherjee@bsigroup.com

Introduction

This draft standard is based on national and international discussions. Your comments on this draft are invited and will assist in the preparation of the consequent standard.

For international standards, comments will be reviewed by the relevant UK national committee before sending the consensus UK vote and comments to the international committee, which will then decide appropriate action. If the international standard is approved, it is usual for the text to be published as a British Standard.

For national standards, comments will be reviewed by the relevant UK national committee and the resulting standards published as a British Standard.

UK Vote

Please indicate whether you consider the UK should submit a negative (with supporting technical reasons) or positive vote on this draft. Please indicate if you are aware of any reason why this draft standard should not be published as a British Standard.

Submission of Comments

- **Annotated drafts are not acceptable and will be rejected.**
- All comments should be submitted online at <https://standardsdevelopment.bsigroup.com/>. You will need to register in order to comment.

Template for comments and secretariat observations

Date: xx/xx/20xx	Document: ISO/DIS xxxx
------------------	------------------------

1	2	(3)	4	5	(6)	7
MB	Clause No./ Subclause No./Annex (e.g. 3.1)	Paragraph/Figure/ Table/Note	Type of comment	Commend (justification for change) by the MB	Proposed change by the MB	Secretariat observations on each comment submitted
	3.1	Definition 1	ed	Definition is ambiguous and needs clarifying.	Amend to read '...so that the mains connector to which no connection...'	
	6.4	Paragraph 2	te	The use of the UV photometer as an alternative cannot be supported as serious problems have been encountered in its use in the UK.	Delete reference to UV photometer.	



PROJECT NUMBER:

IEC 62541-3 ED4

DATE OF CIRCULATION:

2024-02-02

CLOSING DATE FOR VOTING:

2024-04-26

SUPERSEDES DOCUMENTS:

65E/975/RR

IEC SC 65E : DEVICES AND INTEGRATION IN ENTERPRISE SYSTEMS	
SECRETARIAT: United States of America	SECRETARY: Mr Donald (Bob) Lattimer
OF INTEREST TO THE FOLLOWING COMMITTEES:	PROPOSED HORIZONTAL STANDARD: ☐ Other TC/SCs are requested to indicate their interest, if any, in this CDV to the secretary.
FUNCTIONS CONCERNED: ☐ EMC ☐ ENVIRONMENT ☐ QUALITY ASSURANCE ☐ SAFETY	
☒ SUBMITTED FOR CENELEC PARALLEL VOTING Attention IEC-CENELEC parallel voting The attention of IEC National Committees, members of CENELEC, is drawn to the fact that this Committee Draft for Vote (CDV) is submitted for parallel voting. The CENELEC members are invited to vote through the CENELEC online voting system.	☐ NOT SUBMITTED FOR CENELEC PARALLEL VOTING

This document is still under study and subject to change. It should not be used for reference purposes.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Recipients of this document are invited to submit, with their comments, notification of any relevant "In Some Countries" clauses to be included should this proposal proceed. Recipients are reminded that the CDV stage is the final stage for submitting ISC clauses. (SEE [AC/22/2007](#) OR [NEW GUIDANCE DOC](#)).

TITLE:

OPC Unified Architecture - Part 3: Address Space Model

PROPOSED STABILITY DATE: 2026

NOTE FROM TC/SC OFFICERS:

CONTENTS

3	FIGURES	vii
4	TABLES	ix
5	FOREWORD	xi
6	1 Scope	1
7	2 Normative references	1
8	3 Terms, definitions, abbreviations and conventions	2
9	3.1 Terms and definitions	2
10	3.2 Abbreviated terms	3
11	3.3 Conventions	3
12	3.3.1 Conventions for AddressSpace figures	3
13	3.3.2 Conventions for defining NodeClasses	4
14	4 AddressSpace concepts	5
15	4.1 Overview	5
16	4.2 URIs	5
17	4.3 Object Model	6
18	4.4 Node Model	7
19	4.4.1 General	7
20	4.4.2 NodeClasses	7
21	4.4.3 Attributes	7
22	4.4.4 References	7
23	4.5 Variables	8
24	4.5.1 General	8
25	4.5.2 Properties	8
26	4.5.3 DataVariables	8
27	4.6 TypeDefinitionNodes	9
28	4.6.1 General	9
29	4.6.2 Complex TypeDefinitionNodes and their InstanceDeclarations	9
30	4.6.3 Subtyping	10
31	4.6.4 Instantiation of complex TypeDefinitionNodes	10
32	4.7 Event Model	11
33	4.7.1 General	11
34	4.7.2 EventTypes	12
35	4.7.3 Event Categorization	12
36	4.8 Methods	13
37	4.9 Roles	13
38	4.9.1 Overview	13
39	4.9.2 Well Known Roles	14
40	4.9.3 Evaluating Permissions with Roles	14
41	4.10 Interfaces and AddIns for Objects	16
42	4.10.1 Overview	16
43	4.10.2 Interface Model	17
44	4.10.3 AddIn model	20
45	5 Standard NodeClasses	22
46	5.1 Overview	22
47	5.2 Base NodeClass	22

48	5.2.1	General.....	22
49	5.2.2	NodeId	22
50	5.2.3	NodeClass	22
51	5.2.4	BrowseName	22
52	5.2.5	DisplayName.....	23
53	5.2.6	Description.....	23
54	5.2.7	WriteMask.....	23
55	5.2.8	UserWriteMask	23
56	5.2.9	RolePermissions	24
57	5.2.10	UserRolePermissions.....	25
58	5.2.11	AccessRestrictions.....	25
59	5.3	ReferenceType NodeClass	25
60	5.3.1	General.....	25
61	5.3.2	Attributes	26
62	5.3.3	References	27
63	5.4	View NodeClass	28
64	5.5	Objects.....	30
65	5.5.1	Object NodeClass	30
66	5.5.2	ObjectType NodeClass	31
67	5.5.3	Standard ObjectType FolderType.....	32
68	5.5.4	Client-side creation of Objects of an ObjectType	32
69	5.6	Variables.....	33
70	5.6.1	General.....	33
71	5.6.2	Variable NodeClass	33
72	5.6.3	Properties	36
73	5.6.4	DataVariable	37
74	5.6.5	VariableType NodeClass.....	38
75	5.6.6	Client-side creation of Variables of an VariableType	40
76	5.7	Methods	40
77	5.7.1	Method NodeClass.....	40
78	5.7.2	HasArgumentDescription ReferenceType	43
79	5.7.3	HasOptionalInputArgumentDescription ReferenceType	43
80	5.8	DataTypes.....	44
81	5.8.1	DataType Model.....	44
82	5.8.2	Encoding Rules for different kinds of DataTypes	45
83	5.8.3	DataType NodeClass	45
84	5.8.4	DataTypeEncoding and Encoding Information	48
85	5.9	Summary of Attributes of the NodeClasses.....	49
86	6	Type Model for ObjectTypes and VariableTypes	49
87	6.1	Overview	49
88	6.2	Definitions	49
89	6.2.1	InstanceDeclaration	49
90	6.2.2	Instances without ModellingRules	50
91	6.2.3	InstanceDeclarationHierarchy	50
92	6.2.4	Similar Node of InstanceDeclaration	50
93	6.2.5	BrowsePath	50
94	6.2.6	BrowseName within a TypeDefinitionNode	50
95	6.2.7	Attribute Handling of InstanceDeclarations.....	50
96	6.2.8	Attribute Handling of Variable and VariableTypes.....	50

97	6.2.9	NodeIds of InstanceDeclarations	51
98	6.3	Subtyping of ObjectTypes and VariableTypes	51
99	6.3.1	Overview	51
100	6.3.2	Attributes	51
101	6.3.3	InstanceDeclarations	51
102	6.4	Instances of ObjectTypes and VariableTypes	54
103	6.4.1	Overview	54
104	6.4.2	Creating an Instance	54
105	6.4.3	Constraints on an Instance	55
106	6.4.4	ModellingRules	56
107	6.5	Changing Type Definitions that are already used	63
108	7	Standard ReferenceTypes	63
109	7.1	General	63
110	7.2	References ReferenceType	64
111	7.3	HierarchicalReferences ReferenceType	64
112	7.4	NonHierarchicalReferences ReferenceType	65
113	7.5	HasChild ReferenceType	65
114	7.6	Aggregates ReferenceType	65
115	7.7	HasComponent ReferenceType	65
116	7.8	HasProperty ReferenceType	65
117	7.9	HasOrderedComponent ReferenceType	66
118	7.10	HasSubtype ReferenceType	66
119	7.11	Organizes ReferenceType	66
120	7.12	HasModellingRule ReferenceType	66
121	7.13	HasTypeDefinition ReferenceType	66
122	7.14	HasEncoding ReferenceType	67
123	7.15	GeneratesEvent	67
124	7.16	AlwaysGeneratesEvent	67
125	7.17	HasEventSource	67
126	7.18	HasNotifier	68
127	7.19	HasInterface ReferenceType	69
128	7.20	HasAddIn ReferenceType	69
129	7.21	IsDeprecated ReferenceType	69
130	7.22	HasStructuredComponent ReferenceType	70
131	7.22.1	Overview	70
132	7.22.2	Differences between HasStructuredComponent and ExposesItsArray	70
133	7.23	AssociatedWith ReferenceType	71
134	8	Standard DataTypes	71
135	8.1	General	71
136	8.2	NodeId	71
137	8.2.1	General	71
138	8.2.2	NamespaceIndex	72
139	8.2.3	IdType	72
140	8.2.4	Identifier value	72
141	8.3	QualifiedName	73
142	8.4	LocaleId	73
143	8.5	LocalizedText	74
144	8.6	Argument	74
145	8.7	BaseDataType	74

146	8.8	Boolean.....	75
147	8.9	Byte	75
148	8.10	ByteString	75
149	8.11	DateTime	75
150	8.12	Double	75
151	8.13	Duration	75
152	8.14	Enumeration.....	75
153	8.15	Float.....	75
154	8.16	Guid	75
155	8.17	Sbyte	75
156	8.18	IdType.....	75
157	8.19	Image.....	75
158	8.20	ImageBMP	76
159	8.21	ImageGIF	76
160	8.22	ImageJPG	76
161	8.23	ImagePNG	76
162	8.24	Integer	76
163	8.25	Int16.....	76
164	8.26	Int32.....	76
165	8.27	Int64.....	76
166	8.28	TimeZoneDataType	76
167	8.29	NodeClass	76
168	8.30	Number	77
169	8.31	String	77
170	8.32	Structure	77
171	8.33	UInteger	77
172	8.34	UInt16	77
173	8.35	UInt32	77
174	8.36	UInt64	77
175	8.37	UtcTime	77
176	8.38	XmlElement.....	77
177	8.39	EnumValueType	77
178	8.40	OptionSet.....	78
179	8.41	Union	79
180	8.42	DateString	79
181	8.43	DecimalString.....	79
182	8.44	DurationString	79
183	8.45	NormalizedString.....	79
184	8.46	TimeString.....	80
185	8.47	DataTypeDefinition.....	80
186	8.48	StructureDefinition	80
187	8.49	StructureType	80
188	8.50	EnumDefinition.....	81
189	8.51	StructureField.....	81
190	8.52	EnumField	82
191	8.53	AudioDataType.....	82
192	8.54	Decimal.....	82
193	8.55	PermissionType.....	82
194	8.56	AccessRestrictionType	84

195	8.57	AccessLevelType	84
196	8.58	AccessLevelExType	84
197	8.59	EventNotifierType	86
198	8.60	AttributeWriteMask	86
199	8.61	CurrencyUnitType	87
200	9	Standard EventTypes	88
201	9.1	General	88
202	9.2	BaseEventType	88
203	9.3	SystemEventType	88
204	9.4	ProgressEventType	89
205	9.5	AuditEventType	89
206	9.6	AuditSecurityEventType	90
207	9.7	AuditChannelEventType	90
208	9.8	AuditOpenSecureChannelEventType	90
209	9.9	AuditSessionEventType	90
210	9.10	AuditCreateSessionEventType	91
211	9.11	AuditUrlMismatchEventType	91
212	9.12	AuditActivateSessionEventType	91
213	9.13	AuditCancelEventType	91
214	9.14	AuditCertificateEventType	91
215	9.15	AuditCertificateDataMismatchEventType	91
216	9.16	AuditCertificateExpiredEventType	91
217	9.17	AuditCertificateInvalidEventType	91
218	9.18	AuditCertificateUntrustedEventType	91
219	9.19	AuditCertificateRevokedEventType	91
220	9.20	AuditCertificateMismatchEventType	91
221	9.21	AuditNodeManagementEventType	92
222	9.22	AuditAddNodesEventType	92
223	9.23	AuditDeleteNodesEventType	92
224	9.24	AuditAddReferencesEventType	92
225	9.25	AuditDeleteReferencesEventType	92
226	9.26	AuditUpdateEventType	92
227	9.27	AuditWriteUpdateEventType	92
228	9.28	AuditHistoryUpdateEventType	92
229	9.29	AuditUpdateMethodEventType	92
230	9.30	DeviceFailureEventType	92
231	9.31	SystemStatusChangeEvent	92
232	9.32	ModelChangeEvents	92
233	9.32.1	General	92
234	9.32.2	NodeVersion Property	93
235	9.32.3	Views	93
236	9.32.4	Event compression	93
237	9.32.5	BaseModelChangeEvent	93
238	9.32.6	GeneralModelChangeEvent	93
239	9.32.7	Guidelines for ModelChangeEvents	93
240	9.33	SemanticChangeEvent	94
241	9.33.1	General	94
242	9.33.2	ViewVersion and NodeVersion Properties	94
243	9.33.3	Views	94

244	9.33.4	Event compression.....	94
245	Annex A (informative)	How to use the Address Space Model.....	95
246	A.1	Overview.....	95
247	A.2	Type definitions.....	95
248	A.3	ObjectTypes.....	95
249	A.4	VariableTypes.....	95
250	A.4.1	General.....	95
251	A.4.2	Properties or DataVariables.....	95
252	A.4.3	Many Variables and / or Structured DataTypes.....	96
253	A.5	Views.....	96
254	A.6	Methods.....	97
255	A.7	Defining ReferenceTypes.....	97
256	A.8	Defining ModellingRules.....	97
257	Annex B (informative)	OPC UA Meta Model in UML.....	98
258	B.1	Background.....	98
259	B.2	Notation.....	98
260	B.3	Meta Model.....	99
261	B.3.1	Base.....	99
262	B.3.2	ReferenceType.....	100
263	B.3.3	Predefined ReferenceTypes.....	101
264	B.3.4	Attributes.....	101
265	B.3.5	Object and ObjectType.....	102
266	B.3.6	EventNotifier.....	103
267	B.3.7	Variable and VariableType.....	103
268	B.3.8	Method.....	104
269	B.3.9	DataType.....	105
270	B.3.10	View.....	106
271	Annex C (normative)	Graphical notation.....	107
272	C.1	General.....	107
273	C.2	Notation.....	107
274	C.2.1	Overview.....	107
275	C.2.2	Simple notation.....	107
276	C.2.3	Extended notation.....	109
277	Bibliography.....		112
278			
279			

FIGURES

280		
281		
282	Figure 1 – AddressSpace Node diagrams.....	3
283	Figure 2 – OPC UA Object Model.....	6
284	Figure 3 – AddressSpace Node Model.....	7
285	Figure 4 – Reference Model	8
286	Figure 5 – Example of a Variable defined by a VariableType	9
287	Figure 6 – Example of a Complex TypeDefinition	10
288	Figure 7 – Object and its Components defined by an ObjectType	11
289	Figure 8 – Examples of Interfaces	18
290	Figure 9 – Example: Interface application to an ObjectType	19
291	Figure 10 – Example: One Interface applied to an ObjectType another one to the instance ..	19
292	Figure 11 – Example: Interface Hierarchy.....	20
293	Figure 12 – Example of AddIn with default BrowseName	21
294	Figure 13 – Example of AddIn applied to an instance	21
295	Figure 14 – Permissions in the Address Space.....	25
296	Figure 15 – Symmetric and Non-Symmetric References	27
297	Figure 16 – Method Metadata Example	43
298	Figure 17 – Variables, VariableTypes and their DataTypes.....	44
299	Figure 18 – DataType Model	44
300	Figure 19 – Example of DataType Modelling	48
301	Figure 20 – Subtyping TypeDefinitionNodes	52
302	Figure 21 – The Fully-Inherited InstanceDeclarationHierarchy for BetaType	53
303	Figure 22 – An Instance and its TypeDefinitionNode	55
304	Figure 23 – Example for several References between InstanceDeclarations.....	56
305	Figure 24 – Example on changing instances based on InstanceDeclarations.....	57
306	Figure 25 – Example on changing InstanceDeclarations based on an InstanceDeclaration ...	58
307	Figure 26 – Use of the Standard ModellingRule Mandatory	59
308	Figure 27 – Example using the Standard ModellingRules Optional and Mandatory	60
309	Figure 28 – Example on using ExposesItsArray.....	61
310	Figure 29 – Complex example on using ExposesItsArray.....	61
311	Figure 30 – Example using OptionalPlaceholder with an Object and Variable	61
312	Figure 31 – Example using OptionalPlaceholder with a Method	62
313	Figure 32 – Example on using MandatoryPlaceholder for Object and Variable.....	63
314	Figure 33 – Standard ReferenceType Hierarchy	64
315	Figure 34 – Event Reference Example	68
316	Figure 35 – Complex Event Reference Example.....	69
317	Figure 38 – Standard EventType Hierarchy	88
318	Figure 39 – Audit Behaviour of a Server	89
319	Figure 40 – Audit Behaviour of an Aggregating Server	90
320	Figure B.1 – Background of OPC UA Meta Model.....	98
321	Figure B.2 – Notation (I).....	99
322	Figure B.3 – Notation (II).....	99

323	Figure B.4 – Base	100
324	Figure B.5 – Reference and ReferenceType	100
325	Figure B.6 – Predefined ReferenceTypes	101
326	Figure B.7 – Attributes	102
327	Figure B.8 – Object and ObjectType	103
328	Figure B.9 – EventNotifier	103
329	Figure B.10 – Variable and VariableType	104
330	Figure B.11 – Method	105
331	Figure B.12 – DataType	106
332	Figure B.13 – View	106
333	Figure C.1 – Example of a Reference connecting two Nodes	108
334	Figure C.2 – Example of using a TypeDefinition inside a Node	110
335	Figure C.3 – Example of exposing Attributes	110
336	Figure C.4 – Example of exposing Properties inline	111
337		
338		

TABLES

339		
340		
341	Table 1 – NodeClass Table Conventions	4
342	Table 2 – Well-Known Roles	14
343	Table 3 – Example Roles	14
344	Table 4 – Example Nodes	15
345	Table 5 – Example Role Assignment	15
346	Table 6 – Examples of Evaluating Access	15
347	Table 7 – Base NodeClass	22
348	Table 9 – ReferenceType NodeClass	26
349	Table 10 – View NodeClass	29
350	Table 11 – Object NodeClass	30
351	Table 12 – ObjectType NodeClass	31
352	Table 13 – Variable NodeClass	33
353	Table 14 – VariableType NodeClass	39
354	Table 15 – Method NodeClass	41
355	Table 16 – DataType NodeClass	46
356	Table 17 – Overview of Attributes.....	49
357	Table 18 – The InstanceDeclarationHierarchy for BetaType	52
358	Table 19 – The Fully-Inherited InstanceDeclarationHierarchy for BetaType	53
359	Table 20 – Rule for ModellingRules Properties when Subtyping	57
360	Table 21 – NodeId Definition	72
361	Table 22 – IdType Values.....	72
362	Table 23 – NodeId Alternative Null Values	73
363	Table 24 – QualifiedName Definition	73
364	Table 25 – LocaleId Examples.....	74
365	Table 26 – LocalizedText Definition.....	74
366	Table 27 – Argument Definition	74
367	Table 28 – TimeZoneDataType Definition.....	76
368	Table 29 – NodeClass Values	76
369	Table 30 – EnumValueType Definition.....	78
370	Table 31 – OptionSet Definition.....	78
371	Table 32 – StructureDefinition Structure	80
372	Table 33 – StructureType Values	81
373	Table 34 – EnumDefinition Structure	81
374	Table 35 – StructureField Structure	81
375	Table 36 – EnumField Structure	82
376	Table 37 – PermissionType Definition	82
377	Table 38 – AccessRestrictionType Definition.....	84
378	Table 39 – AccessLevelType Definition	84
379	Table 40 – Use Cases of Constant and NonVolatile Fields	85
380	Table 41 – AccessLevelExType Definition	85
381	Table 42 – EventNotifierType Definition	86

382	Table 43 – Bit mask for WriteMask and UserWriteMask.....	87
383	Table 44 – CurrencyUnitType Definition	87
384	Table C.1 – Notation of Nodes depending on the NodeClass.....	108
385	Table C.2 – Simple Notation of Nodes depending on the NodeClass	109
386	Table C.3 – Extended Notation of Reference Cardinality	111
387		
388		

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.

This fourth edition cancels and replaces the third edition published in 2020. This edition constitutes a technical revision.

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

- a) Added the concept and modelling elements for Interfaces and AddIns.
- b) Added Currency
- c) Added Method Meta Data to define additional attributes for Method Arguments
- d) Added ApplyRestrictionToBrowse bit to AccessRestrictionType
- e) Added a Non-Volatile Storage bit to AccessLevelExType
- f) Added a Constant bit and ConfigurationConstant bit to AccessLevelExType
- g) The View NodeClass was changed to define the EventNotifier as an EventNotifierType in the same way the Object NodeClass defines it.
- h) Corrected HasNotifier, HasEventSource, and Organizes, to include ObjectType as valid source node

- i) Deprecated NamingRules
j) Added AssociatedWith ReferenceType

The text of this International Standard is based on the following documents:

CDV	Report on voting
65E/XX/CDV	65E/XX/RVC

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

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

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

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

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

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

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

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

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

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.

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 and errata) applies.

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

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

IEC 62541-18, *OPC Unified Architecture - Part 18: Role-Based Security*

IEC 62541-21, *OPC Unified Architecture - Part 21: Device Onboarding*

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

<https://www.iso.org/iso-639-language-codes.html>

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

<https://www.iso.org/iso-3166-country-codes.html>

ISO/IEC/IEEE 60559:2011, Information technology – Microprocessor Systems – Floating-Point arithmetic

<https://www.iso.org/standard/57469.html>

IETF RFC 5646, Tags for Identifying Languages

<http://tools.ietf.org/html/rfc5646>

ISO 8601-1:2019, Date and time – Representations for information interchange – Part 1: Basic

<https://www.iso.org/standard/70907.html>

Unicode Annex15, Unicode Standard Annex #15: Unicode Normalization Forms

<http://www.unicode.org/reports/tr15/>

W3C XML Schema Definition Language (XSD) Part 2, DataTypes

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

ISO 4217, Codes for the representation of currencies

<https://www.iso.org/iso-4217-currency-codes.html>

RFC 3986, Uniform Resource Identifier (URI): Generic Syntax:

<https://www.ietf.org/rfc/rfc3986.txt>

RFC 4151, The 'tag' URI Scheme

<https://datatracker.ietf.org/doc/html/rfc4151>

Unicode C0 Controls and Basic Latin:

<https://www.unicode.org/charts/PDF/U0000.pdf>

Unicode C1 Controls and Latin-1 Supplement:

<https://www.unicode.org/charts/PDF/U0080.pdf><https://www.unicode.org/Public/UCD/latest/ucd/PropList.txt>

3 Terms, definitions, abbreviations and conventions

3.1 Terms and definitions

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

3.1.1

DataType

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

3.1.2

DataTypeId

NodeId of a *DataType Node*

3.1.3

DataVariable

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

3.1.4

EventType

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

3.1.5

Hierarchical Reference

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

Note 1 to entry: All hierarchical *ReferenceTypes* are derived from *HierarchicalReferences*.

3.1.6

InstanceDeclaration

Node that is used by a complex *TypeDefinitionNode* to expose its complex structure

Note 1 to entry: This is an instance used by a type definition.

3.1.7

ModellingRule

metadata of an *InstanceDeclaration* that defines how the *InstanceDeclaration* will be used for instantiation and also defines subtyping rules for an *InstanceDeclaration*

3.1.8

Property

Variables that are the *TargetNode* for a *HasProperty Reference*

Note 1 to entry: *Properties* describe the characteristics of a *Node*.

3.1.9**SourceNode**

Node having a *Reference* to another *Node*

EXAMPLE: In the *Reference* "A contains B", "A" is the *SourceNode*.

3.1.10**TargetNode**

Node that is referenced by another *Node*

EXAMPLE: In the *Reference* "A contains B", "B" is the *TargetNode*.

3.1.11**TypeDefinitionNode**

Node that is used to define the type of another *Node*

Note 1 to entry: *ObjectType* and *VariableType* Nodes are *TypeDefinitionNodes*.

3.1.12**VariableType**

Node that represents the type definition for a *Variable*

3.1.13**AddIn**

a widely applicable feature or feature-set represented by an *ObjectType* that will be applied using aggregation

3.1.14**Interface**

an abstract *ObjectType* used to specify a feature or feature-set that shall be implemented by the *Object* or *ObjectType* where it is applied to

3.2 Abbreviated terms

UA Unified Architecture

UML Unified Modeling Language

URI Uniform Resource Identifier as defined by RFC 3986

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.

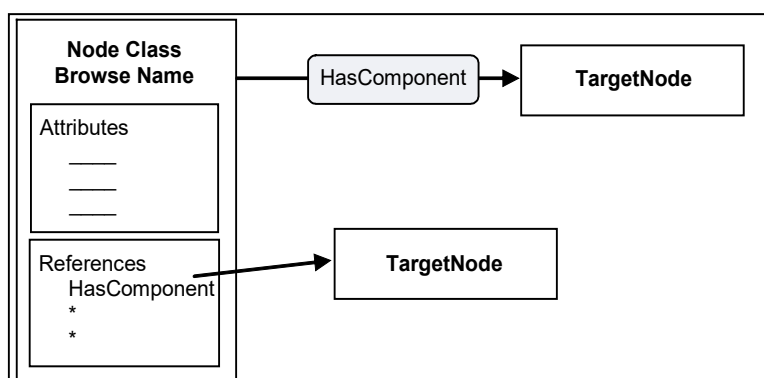


Figure 1 – AddressSpace Node diagrams

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

This document also defines *ReferenceTypes*, but *ReferenceTypes* may 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 document may 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, subtypes of the *ReferenceType* may restrict its *SourceNodes*.

This document 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 remainder of Clause 4 defines 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 8.61.

The informative Annex A describes general considerations on 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 a graphical notation for OPC UA data.

4.2 URIs

URIs provide a syntax for constructing unique identifiers for resources that are used in several different contexts with this specification. Three use cases where implementators may need to construct their own URIs are:

- *NamespaceUri* used to identify an information model. *NamespaceUri* appear in *UANodeSets* (see IEC 62541-6) and in the *NamespaceArray* in a *Server AddressSpace* (see IEC 62541-5);
- *ApplicationUri* identify an *OPC UA Application* running on a particular *Device* and are assigned by the *OwnerOperator* or automatically created by the application software. An *ApplicationInstance Certificate* has the *ApplicationUri* in the *subjectAltName* (see IEC 62541-6);
- *ProductInstanceUri* identify a *Device* and are assigned by the *Device Manufacturer* (see IEC 62541-21). A *DeviceIdentity Certificate* has the *ProductInstanceUri* in the *subjectAltName*.

These URIs conform to RFC 3986, however, this specification is very open ended. This clause provides recommendations for constructing these URIs.

The recommendations help ensure that URIs are unique by providing a scope controlled by a single authority (e.g. the domain name) and are human readable. Programs shall always treat URIs as opaque strings that can only be tested for equality with a case sensitive string comparison.

URIs should be tag URIs (RFC 4151) or HTTP URLs (RFC 3986). URLs should only be used if they link to actual webpages.

The scheme and domain name portions of the URI are in lower case. URIs may include query subcomponents and/or fragments.

Below are recommended URI formats for different use cases.

1) *NamespaceUri* assigned by the creator of a *InformationModel*:

tag:<authority-domain-name>,<yyyy-MM>;UA:<model-short-name>

http://<authority-domain-name>/UA/<model-short-name>

Where the <authority-domain-name> is a domain name owned by the authority creating the information model. In many cases this will be 'opcfoundation.org'. <yyyy-MM> is the year and month that the URI was created. The <model-short-name> is a short identifier for the *InformationModel*.

2) *ApplicationUri* assigned by the *OwnerOperator*

tag:<device-domain-name>,<yyyy-MM>:<product>

http://<device-domain-name>/<product>

Where the <device-domain-name> is a domain name or IP address for the *Device* that is unique within the *OwnerOperator* system. <yyyy-MM> is the year and month that the URI was created. The <product> is an identifier for the product which could contain many path segments.

3) *ApplicationUri* automatically generated by product software:

tag:<vendor-domain-name>,<yyyy-MM>:<product>:<guid>

http://<vendor-domain-name>/<product>/<guid>

Where <vendor-domain-name> is a domain name owned by the product vendor. <yyyy-MM> is the year and month that the URI was created. The <product> is an identifier for the product which could contain many path segments and <guid> is unique identifier generated when the product is first initialized on a particular *Device*.

4) *ProductInstanceUri* assigned by a *Device Manufacturer*:

tag:<manufacturer-domain-name>,<yyyy-MM>:<product>:<serial-number>

http://<manufacturer-domain-name>/<product>/<serial-number>

Where <manufacturer-domain-name> is a domain name owned by the *Manufacturer*. <yyyy-MM> is the year and month that the URI was created. The <product> is an identifier for the product which could contain many path segments and <serial-number> is unique identifier for the *Device* assigned by the *Manufacturer*.

4.3 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.

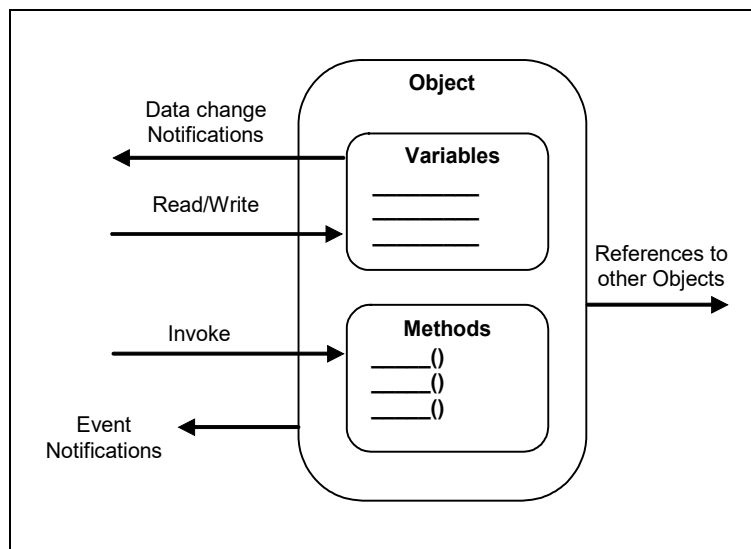


Figure 2 – OPC UA Object Model

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 4.10 defines the *NodeClasses* used to represent this model.

4.4 Node Model

4.4.1 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 4.3 discusses the details of the Node Model.

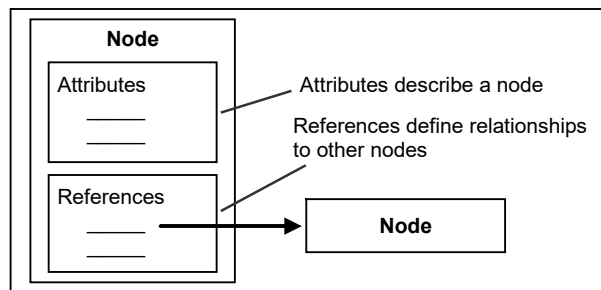


Figure 3 – AddressSpace Node Model

4.4.2 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 4.4.3 and *References* in 4.4.4.

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*.

4.4.3 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 4.10 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.

4.4.4 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*.

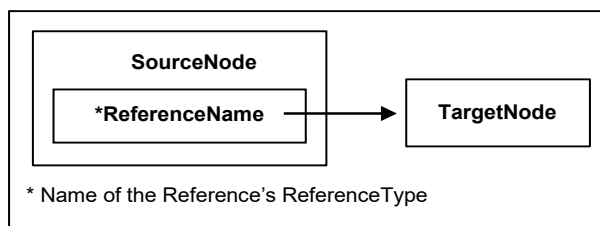


Figure 4 – Reference Model

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.

4.5 Variables

4.5.1 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*.

4.5.2 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*.

4.5.3 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 4.5.2, 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.

4.6 TypeDefinitionNodes

4.6.1 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*.

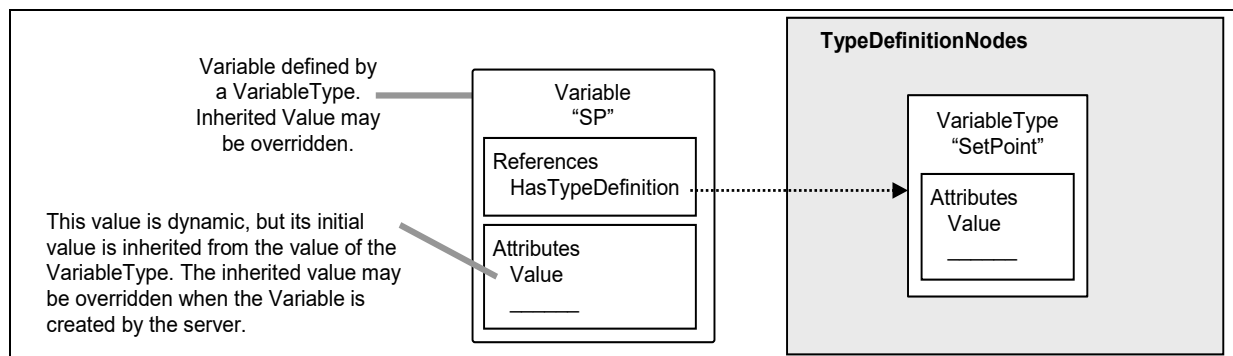


Figure 5 – Example of a Variable defined by a VariableType

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

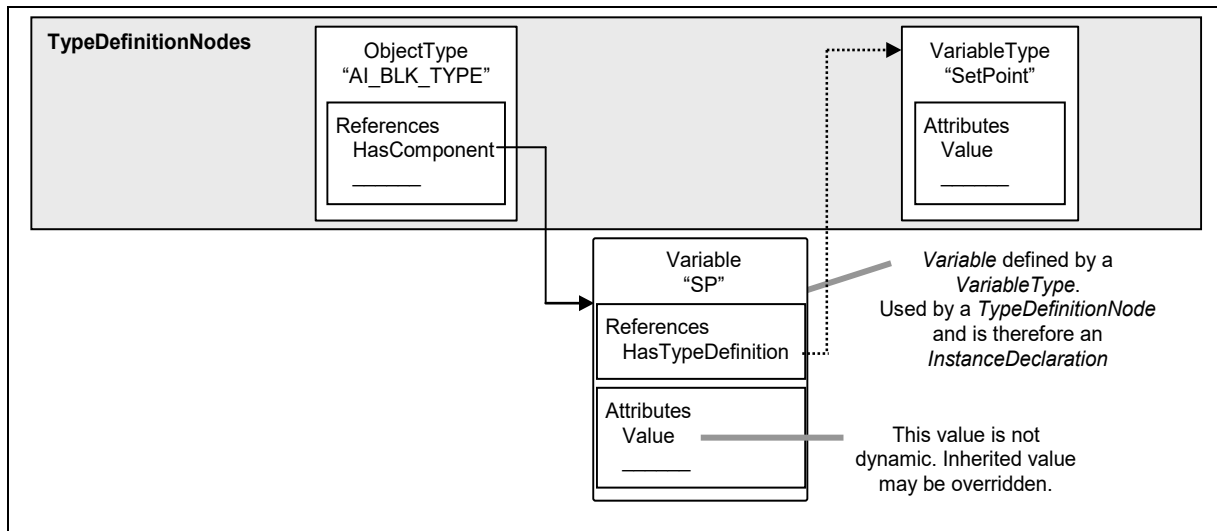


Figure 6 – Example of a Complex TypeDefinition

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

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

4.6.3 Subtyping

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

- *Clients* that only know the supertype to handle an instance of the subtype as if it were an instance of the supertype;
- instances of the supertype to be replaced by instances of the subtype;
- specialised types that inherit common characteristics of the base type.

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

4.6.4 Instantiation of complex TypeDefinitionNodes

The instantiation of complex *TypeDefinitionNodes* depends on the *ModellingRules* defined in 6.4.4. However, the intention is that instances of a type definition will reflect the structure defined by the *TypeDefinitionNode*. Figure 7 shows an instance of the *TypeDefinitionNode* “AI_BLK_TYPE”, where the *ModellingRule Mandatory*, defined in 6.4.4.4.1, 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*.

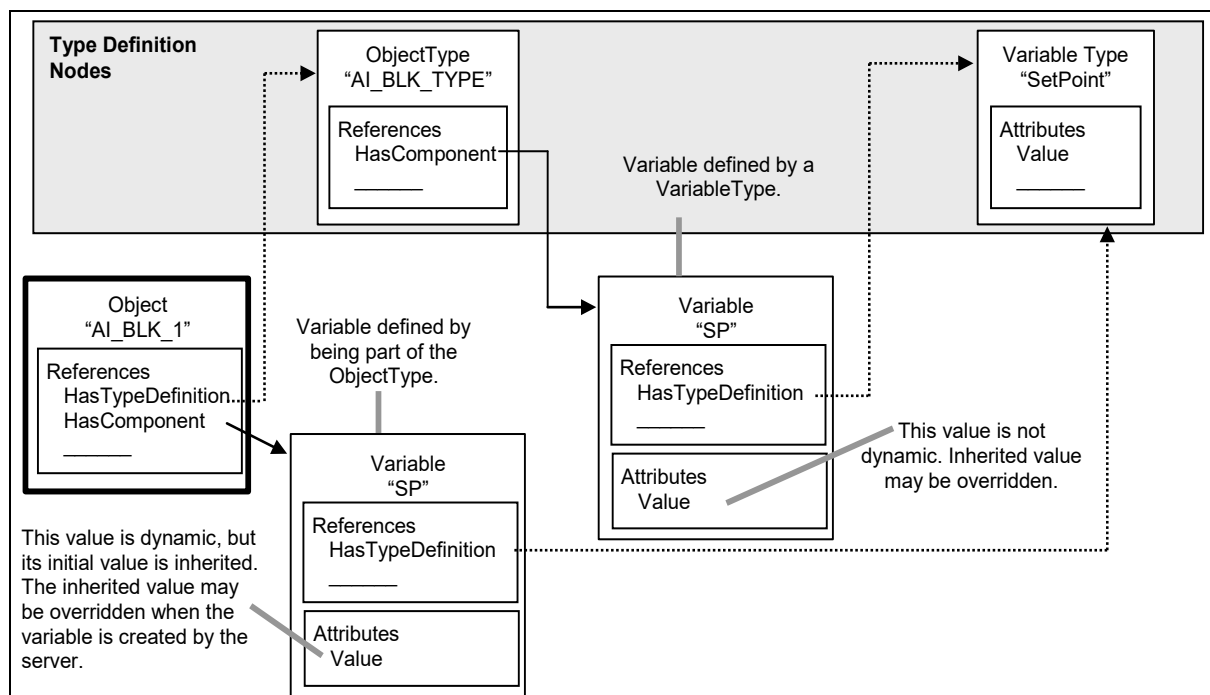


Figure 7 – Object and its Components defined by an ObjectType

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 forward *hierarchical References*. 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*.

4.7 Event Model

4.7.1 General

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

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

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

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

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

4.7.2 EventTypes

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

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

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

Standard *EventTypes* are described in Clause 8.61. Their representation in the *AddressSpace* is specified in IEC 62541-5.

4.7.3 Event Categorization

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

Event sources can also be organised into groups by using the *Event ReferenceTypes* described in 7.16 and 7.18. For example, a *Server* may define *Objects* in the *AddressSpace* representing *Events* related to physical devices, or *Event* areas of a plant or functionality contained in the *Server*. *Event References* would be used to indicate which *Event* sources represent physical devices and which ones represent some *Server*-based functionality. In addition, *References* can be used to group the physical devices or *Server*-based functionality into hierarchical *Event* areas. In some cases, an *Event* source may be categorised as being both a device and a *Server* function. In this case, two relationships would be established. Refer to the description of the *Event ReferenceTypes* for additional examples.

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

4.8 Methods

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

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

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

Methods are invoked by using the Call *Service* defined in IEC 62541-4.

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

4.9 Roles

4.9.1 Overview

A *Role* is a function assumed by a *Client* when it accesses a *Server*. *Roles* are used to separate authentication (determining who a *Client* is) from authorization (determining what the *Client* is allowed to do). By separating these tasks *Servers* can allow centralized services to manage user identities and credentials while the *Server* only manages the *Permissions* on its *Nodes* assigned to *Roles*.

The set of *Roles* supported by a *Server* are published as components of the *RoleSet Object* defined in IEC 62541-18. *Servers* should define a base set of *Roles* and allow configuration *Clients* to add system specific *Roles*. Adding, deleting, and modifying *Roles* is restricted to callers with appropriate permissions.

When a *Session* is created, or a *Session-less Service* is called, the *Server* must determine what *Roles* are granted to that *Session* or *Session-less Service* invocation. This specification defines standard mapping rules which *Servers* may support. *Servers* may also use vendor specific mapping rules in addition to or instead of the standard rules.

The *Anonymous Role* is the default *Role* which is always assigned to all *Sessions*.

The *AuthenticatedUser Role* is always assigned when a *Session* has been authenticated with a *UserIdentityToken* other than the *AnonymousIdentityToken* (see IEC 62541-4).

The standard mapping rules allow *Roles* to be granted based on:

- User identity;
- Application identity;
- Endpoint;

User identity mappings can be based on user names, user certificates or user groups.

Application identity mappings are based on the *ApplicationUri* specified in the *Client Certificate*. Application identity can only be enforced if the *Client* proves possession of a trusted *Certificate* by using it to create a *Secure Channel* or by providing a signature in *ActivateSession* (see IEC 62541-4).

Endpoint identity mappings are based on the URL used to connect to the *Server*. Endpoint identity can be used to restrict access to *Clients* running on particular networks. Endpoint identity mappings should not be used as the only criteria unless access to the endpoint is restricted by the network infrastructure. For example, an endpoint on a loopback address is only accessible from the same machine.

IEC 62541-5 defines the *Objects*, *Methods* and *DataTypes* used to represent and manage these mapping rules in the *Address Space*.

4.9.2 Well Known Roles

The *NodeIds* for the well-known *Roles* are defined in IEC 62541-6. All *Servers* should support the well-known *Roles* which are defined in Table 2.

Table 2 – Well-Known Roles

BrowseName	Suggested Permissions
Anonymous	The <i>Role</i> is allowed to browse and read non-security related <i>Nodes</i> only in the <i>Server Object</i> and all type <i>Nodes</i> .
AuthenticatedUser	The <i>Role</i> is allowed to browse and read non-security related <i>Nodes</i> .
Observer	The <i>Role</i> is allowed to browse, read live data, read historical data/events or subscribe to data/events.
Operator	The <i>Role</i> is allowed to browse, read live data, read historical data/events or subscribe to data/events. In addition, the <i>Session</i> is allowed to write some live data and call some <i>Methods</i> .
Engineer	The <i>Role</i> is allowed to browse, read/write configuration data, read historical data/events, call <i>Methods</i> or subscribe to data/events.
Supervisor	The <i>Role</i> is allowed to browse, read live data, read historical data/events, call <i>Methods</i> or subscribe to data/events.
ConfigureAdmin	The <i>Role</i> is allowed to change the non-security related configuration settings.
SecurityAdmin	The <i>Role</i> is allowed to change security related settings.

4.9.3 Evaluating Permissions with Roles

When a *Client* attempts to access a *Node*, the *Server* goes through the list of *Roles* granted to the *Session* and logically ORs the *Permissions* for the *Role* on the *Node*. If there are no *Node specific Permissions* then the default *Permissions* for the *Role* in the *DefaultRolePermissions Property* of the *NamespaceMetadata* for the namespace the *Node* belongs to are used (see IEC 62541-5). The resulting mask is the effective *Permissions*. If the bits corresponding to current operation are set, then the operation can proceed. If they are not set the *Server* returns *Bad_UserAccessDenied*.

Roles appear under the *Roles Object* in the *Server Address Space*. Each *Role* has mapping rules defined which appear as *Properties* of the *Role Object* (see IEC 62541-5). The examples shown in Table 3 illustrate how the standard mapping rules can be used to determine which *Roles* a *Session* has access to and, consequently, the *Permissions* that are granted to the *Session*.

Table 3 – Example Roles

Role	Mapping Rules	Description
Anonymous	Identities = Anonymous Applications = Endpoints =	An identity mapping rule that specifies the <i>Role</i> applies to anonymous users.
AuthenticatedUser	Identities = AuthenticatedUser Applications = Endpoints =	An identity mapping rule that specifies the <i>Role</i> applies to authenticated users.
Operator1	Identities = User with name 'Joe' Applications = urn:OperatorStation1 Endpoints =	An identity mapping rule that specifies specific users that have access to the <i>Role</i> with a application rule that restricts access to a single <i>Client</i> application.
Operator2	Identities = Users with name 'Joe' or 'Ann' Applications = urn:OperatorStation2 Endpoints =	An identity mapping rule that specifies specific users that have access to the <i>Role</i> with a application rule that restricts

		access to a single Client application.
Supervisor	Identities = User with name 'Root' Applications = Endpoints =	An identity mapping rule that specifies specific users that have access to the Role
Administrator	Identities = User with name 'Root' Applications = Endpoints = opc.tcp://127.0.0.1:48000	An identity mapping rule that specifies specific users that have access to the Role when they connect via a specific Endpoint.

The examples also make use of the *Nodes* defined in Table 4. The table specifies the value of the *RolePermissions Attribute* for each *Node*.

Table 4 – Example Nodes

Node	Role Permissions
Unit1.Measurement	AuthenticatedUser = Browse Operator1 = Browse, Read
Unit2.Measurement	AuthenticatedUser = Browse Operator2 = Browse, Read
SetPoint	AuthenticatedUser = Browse Operator1 and Operator2 = Browse, Read, Write Supervisor = Browse, Read
DisableDevice	AuthenticatedUser = Browse Operator1 and Operator2 = Browse, Read Administrator = Browse, Read, Write

When a *Client* creates a *Session* the *Roles* assigned to the *Session* depend on the rules defined for each *Role*. Table 5 lists the assigned *Roles* for different *Sessions* created with different *Users*, *Client* applications and *Endpoints*.

Table 5 – Example Role Assignment

User Provided by Client	Roles Assigned to Session
Anonymous	Anonymous
Sam	AuthenticatedUser
Joe using OperatorStation1 application.	AuthenticatedUser, Operator1
Joe using OperatorStation2 application.	AuthenticatedUser, Operator2
Joe using generic application.	AuthenticatedUser
Root using OperatorStation1 application.	AuthenticatedUser, Supervisor
Root using generic application and 127.0.0.1 endpoint.	AuthenticatedUser, Supervisor, Administrator
Root using generic application and another endpoint.	AuthenticatedUser, Supervisor

When a *Client* application accesses a *Node* the *RolePermissions* for the *Node* are compared to the *Roles* assigned to the *Session*. Any *Permissions* available to at least one *Role* is granted to the *Client*. Table 6 provides a number of scenarios and examples and the resulting decision on access.

Table 6 – Examples of Evaluating Access

Use Case	Role Permissions
Anonymous user on localhost browses Unit1.Measurement Node.	Access denied because no rule defined for Anonymous users.

User 'Sam' using OperatorStation1 application browses Unit1.Measurement Node.	Allowed because AuthenticatedUser is granted Browse Permission.
User 'Sam' using OperatorStation2 application reads Value of Unit1.Measurement Node.	Access denied because AuthenticatedUser is not granted Read Permission.
User 'Joe' using OperatorStation1 application reads Value of Unit1.Measurement Node.	Allowed because Operator1 is granted Read Permission.
User 'Joe' using OperatorStation2 application reads Value of Unit1.Measurement Node.	Access denied because AuthenticatedUser and Operator2 are not granted Read Permission.
User 'Joe' using generic OPC UA application reads Value of Measurement Node.	Access denied because AuthenticatedUser is not granted Read Permission.
User 'Joe' using OperatorStation1 application write Value of SetPoint Node.	Allowed because Operator1 is granted Write Permission.
User 'Root' using OperatorStation1 application write the Value of SetPoint Node.	Denied because AuthenticatedUser and Supervisor are not granted Write Permission.
User 'Joe' using OperatorStation1 application write Value of DisableDevice Node.	Access denied because AuthenticatedUser and Operator1 are not granted Write Permission.
User 'Root' using OperatorStation1 application write the Value of DisableDevice Node.	Access denied because AuthenticatedUser and Supervisor are not granted Write Permission.
User 'Root' using endpoint 127.0.0.1 to write Value of DisableDevice Node.	Allowed because Administrator is granted Write Permission.

4.10 Interfaces and AddIns for Objects

4.10.1 Overview

OPC UA defines a type model supporting one object-oriented type hierarchy for *ObjectTypes*. Although the specification does not restrict those hierarchies to be single inheritance (i.e. a type can only have one super-type) it only specifies the semantic (inheritance rules) for single inheritance.

In general, good object-oriented design is accomplished by using composition to aggregate an object which provides several functions instead of over-using inheritance [GH95], [FF04] .

Interfaces and *AddIns* complement the type model and can be used when subtyping is not suitable for the required extension. They:

- allow enhancing multiple types at arbitrary positions in the type hierarchy.
- also allow enhancing just instances.

4.10.2 Interface Model

Interfaces are *ObjectTypes* that represent a generic feature (functionality), assumed to be usable by different *ObjectTypes* or *Objects*. The *Interface* model specifies the rules and mechanisms to achieve this.

The “*InterfaceTypes*” *Object* (see IEC 62541-5) has been defined so that all *Interfaces* of the *Server* are either directly or indirectly accessible browsing *HierarchicalReferences* starting from this *Node*.

Rules for the definition of *Interfaces*:

- *Interface ObjectTypes* shall be abstract subtypes of the *BaseInterfaceType ObjectType*.
- *InstanceDeclarations* on an *Interface* shall only have *ModellingRules Optional* or *Mandatory*.
- *Interfaces* can be subtyped as specified in clause 6.3.
- *Interfaces* shall not be the source of *HasInterface References*.
- Recommended convention: The first letter of an *Interface* should be ‘I’. See examples below.

Rules for applying *Interfaces*:

- When an *ObjectType* references an *Interface* with a *HasInterface Reference* or a subtype, the following rules apply:
 - Each mandatory *InstanceDeclaration* of the fully-inherited *InstanceDeclaration-Hierarchy* of the *Interface* shall have for each *BrowsePath* a similar *Node* (see 6.2.4) with the *ModellingRule Mandatory* using the same *BrowsePath* in the fully-inherited *InstanceDeclarationHierarchy* of the *ObjectType*. The rules for instantiating *InstanceDeclarations* defined in 6.2.6 shall be applied.
 - Each optional *InstanceDeclaration* of the fully-inherited *InstanceDeclaration-Hierarchy* of the *Interface* should have for each *BrowsePath* a similar *Node* (see 6.2.4) with the *ModellingRule Mandatory* or *Optional* using the same *BrowsePath* in the fully-inherited *InstanceDeclarationHierarchy* of the *ObjectType*. The rules for instantiating *InstanceDeclarations* defined in 6.2.6 shall be applied. If no similar *Node* with the same *BrowsePath* exists, the *ObjectType* and its sub-types shall not use the same *BrowsePath* for a different *Node* (e.g. with a different *NodeClass*).

If the rules cannot be fulfilled (e.g. name collisions) the *ObjectType* cannot apply the *Interface*, i.e. it shall not reference the *Interface* with a *HasInterface Reference* of a subtype.

The rules apply for each referenced *Interface*. As a consequence, an *ObjectType* cannot reference two *Interfaces* using the same *BrowsePath* for *Nodes* that are not similar *Nodes* or have *TypeDefinitionNodes* that are not compatible (compatible

means they have either the same *TypeDefinitionNode* or one *TypeDefinitionNode* is the subtype of the other *TypeDefinitionNode*).

- Subtypes should not have a *HasInterface* Reference to an *Interface* if it was already applied to a super-type.
- When an *Object* references an *Interface* with a *HasInterface* Reference or a subtype, the following rules apply:
 - The *Interface* shall not be applied on the *Object* when the *Interface* cannot be applied on the *TypeDefinitionNode* of the *Object*.
 - The same rules on the *Object* apply as if the *Interface* would have been applied on the *TypeDefinitionNode* of the *Object* (e.g. all *Mandatory InstanceDeclarations* need to be applied).
 - The *Nodes* defined based on the *Interface* shall be handled as if they were defined by the *TypeDefinitionNode*. For example, the *TranslateBrowsePathsToNodeIds* Service shall return them first.
 - If several *Interfaces* should be applied to the *Object*, they should be treated as if they were all applied to the *ObjectType* of the *Object* at the same time. If this is not possible, the *Interfaces* cannot be applied to the *Object* together.
 - Instances should not have a *HasInterface* Reference to an *Interface* if it was already applied to the *TypeDefinitionNode*.
- A *BaseInterfaceType* or any subtype of *BaseInterfaceType* shall not be the *TargetNode* of a *HasTypeDefinition* Reference.

Figure 8 illustrates example *Interfaces*:

- *ISerializeServiceType*, an *Interface* to convert the *Object* and its components into a stream of bytes.
- *ITransactionServiceType*, an *Interface* to perform a sequence of changes to the *Object* as a single operation.
- *ILocationType*, an *Interface* to specify the installation location of the *Object*.

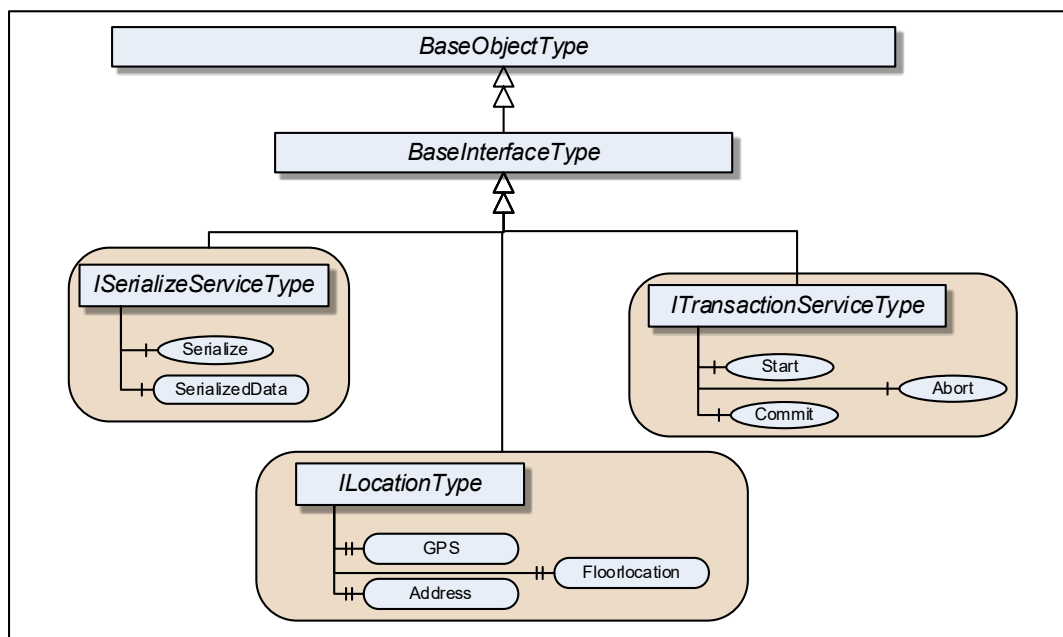


Figure 8 – Examples of Interfaces

The following examples illustrate the application of these *Interfaces*. In Figure 9 the example *Interface ILocationType* is applied to the *XYZ-DeviceType* *ObjectType*. It also illustrates the overriding of *Property* “Address” by changing the *ModellingRule* from *Optional* to *Mandatory*.

Figure 10 in addition shows how to use the *ISerializeService Interface* on the instance only. Figure 11 shows an *Interface* hierarchy where *InstanceDeclarations* of the referenced *Interface* and its parent type(s) are applied (the fully-inherited *InstanceDeclarationHierarchy*).

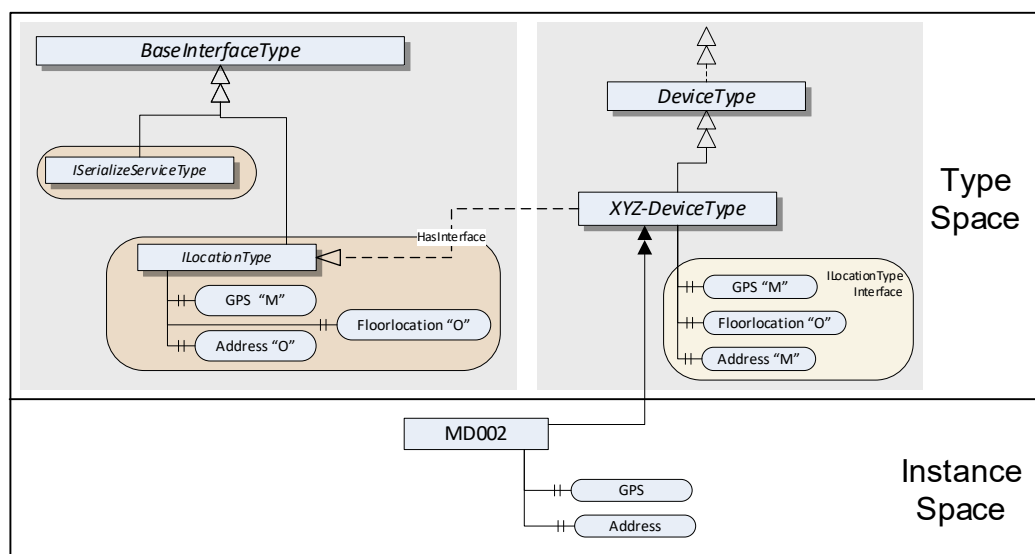


Figure 9 – Example: Interface application to an ObjectType

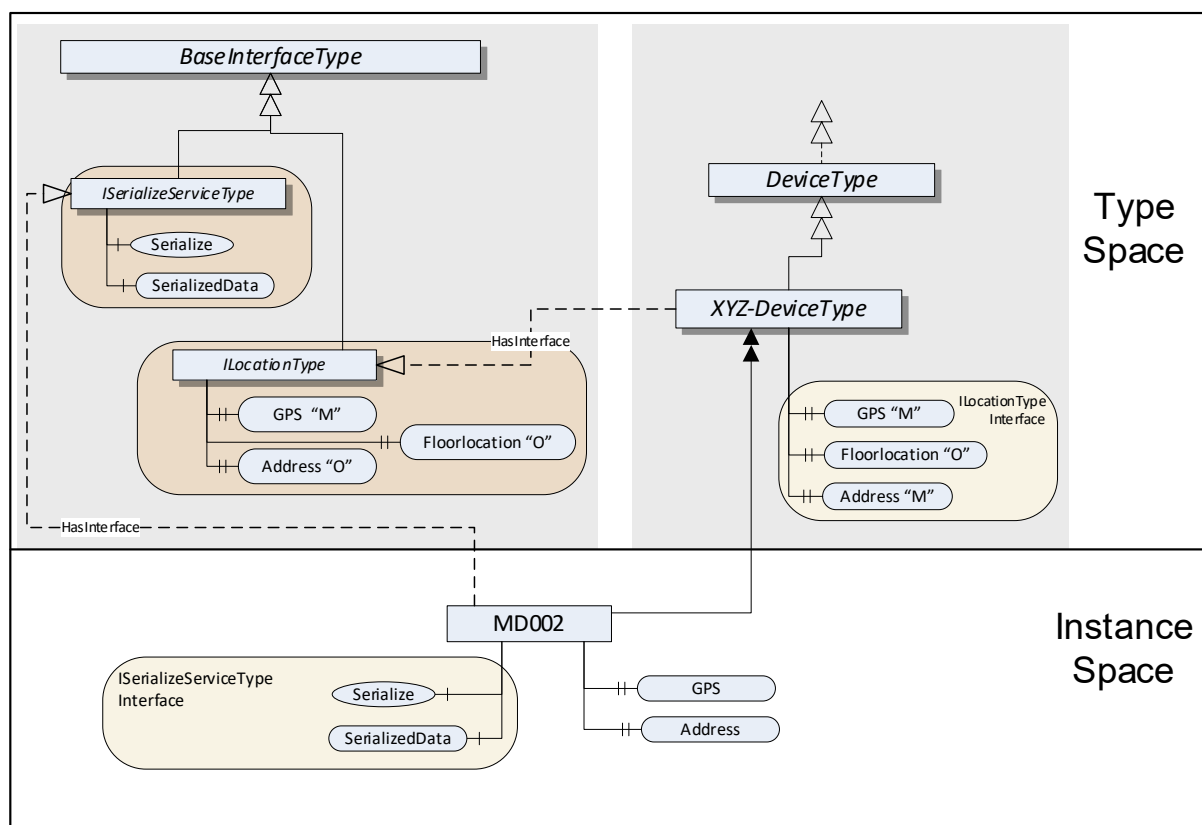


Figure 10 – Example: One Interface applied to an ObjectType another one to the instance

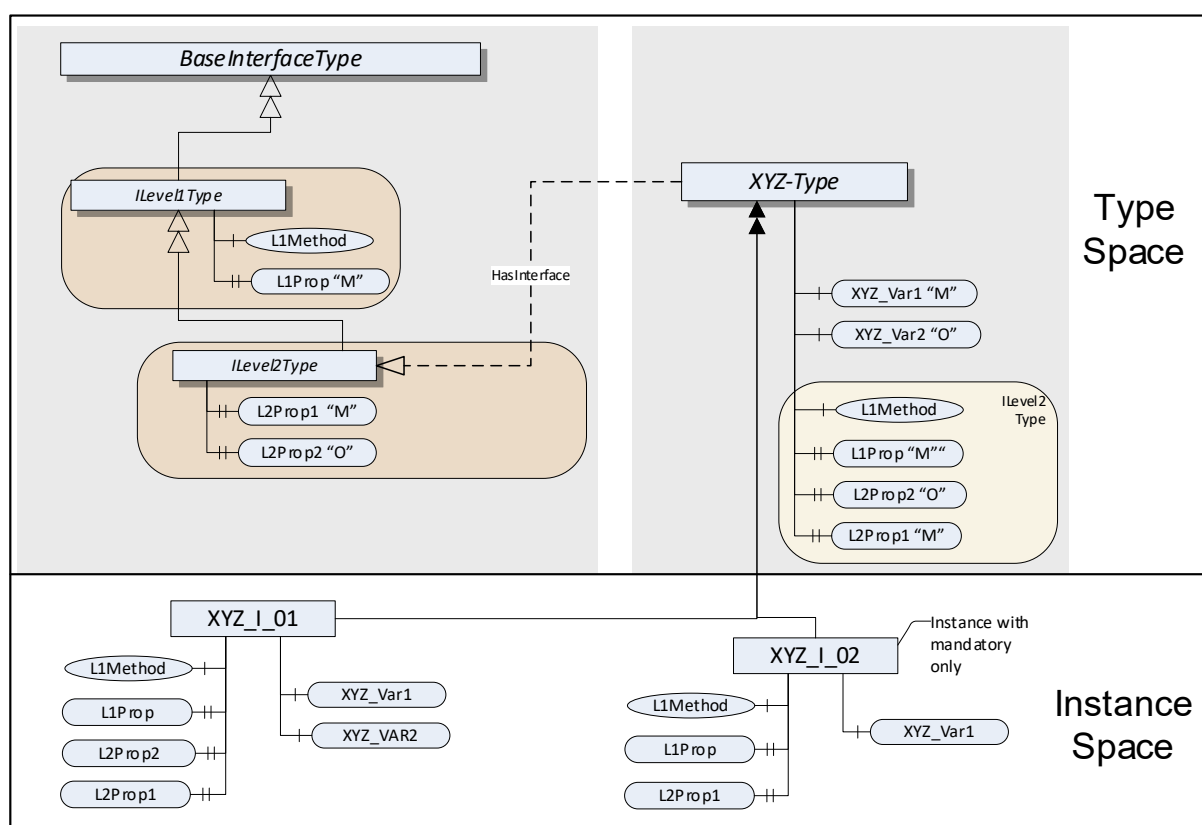


Figure 11 – Example: Interface Hierarchy

Clients can detect the implementation of Interfaces by filtering for the *HasInterface* Reference into the *Browse Service* request.

On instances, the *Browse Service* will return elements derived from an *Interface* together with elements of the *Node's* base type. Clients can also use the *TranslateBrowsePathsToNodeId* Service with *BrowseNames* of *Interface* members to get the *NodeId* of these members directly.

In the example in Figure 10 “Address” with the starting node MD002 can be used to request the *NodeId* of this *Property*.

On *Object* instances, some *Nodes* of an *Interface* may not be available if defined with *ModellingRule Optional*.

4.10.3 AddIn model

AddIns associate a feature or feature-set, represented by an *ObjectType* to the *Node* (an *Object* or *ObjectType*) they are applied to. The *Interface* model is different than the *AddIn* model in that it is based on composition. An *AddIn* is applied to a *Node* by adding a *Reference* to the *AddIn* instance.

There are no restrictions for *AddIn* *ObjectTypes* and there is no special supertype for *AddIns*. To identify instances as an *AddIn*, the *HasAddIn* Reference or a subtype shall be used.

The *AddIn* *ObjectType* shall include the definition of a default *BrowseName* using the *DefaultInstanceBrowseName* Property. Instances of such an *AddIn* should use this default *BrowseName*. If an *AddIn* is instantiated multiple times in the same parent, only one instance can have the default *BrowseName*.

The definition of an *AddIn* and its use with a default *BrowseName* is illustrated in Figure 12.

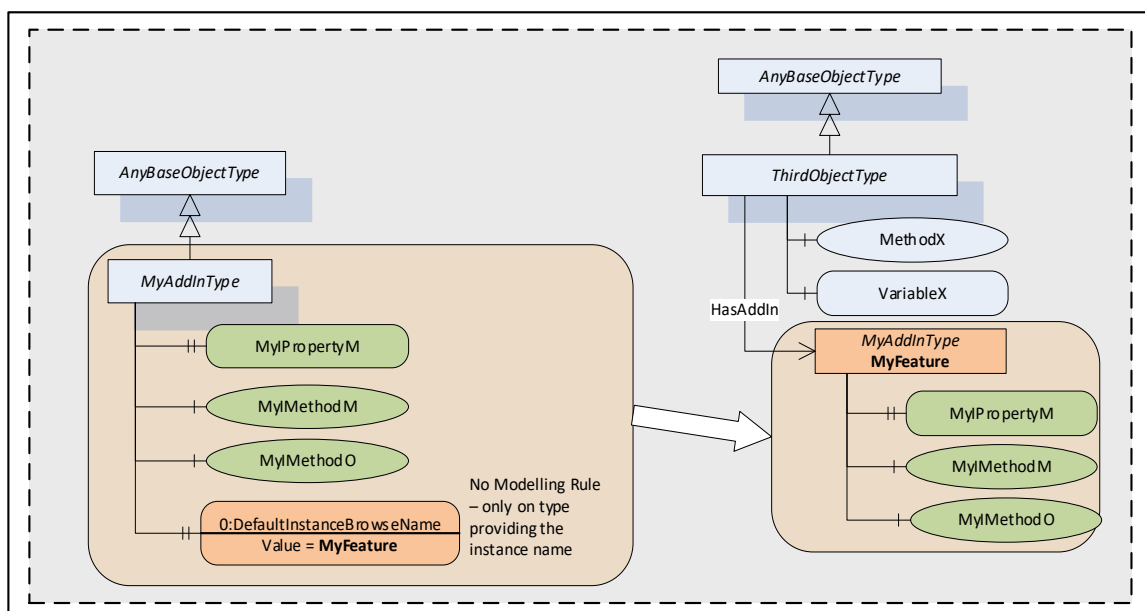


Figure 12 – Example of AddIn with default BrowseName

As already described, an *AddIn* can be applied on types and instances. The use on an instance is shown in Figure 13.

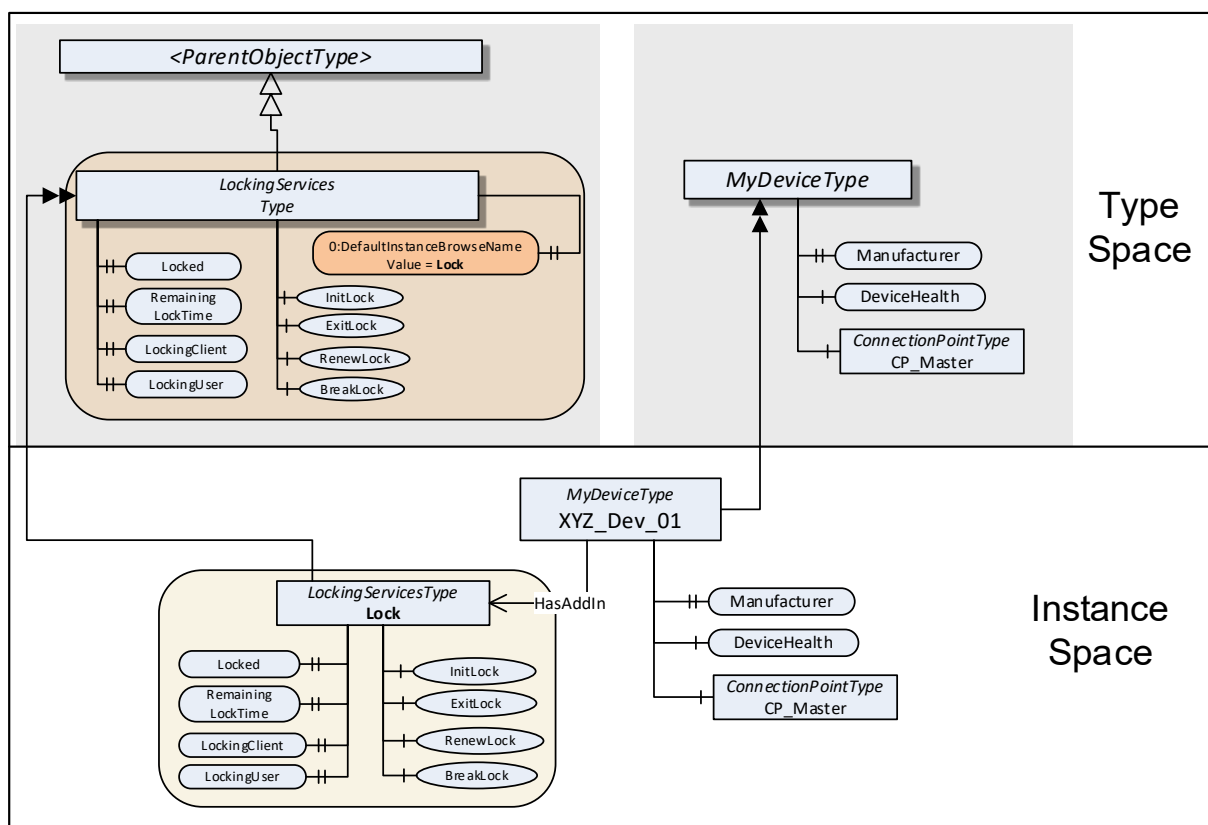


Figure 13 – Example of AddIn applied to an instance

Clients can detect the implementation of *AddIns* by passing the *HasAddIn* Reference as filter to the *Browse* Service request. If an *AddIn* has a default *BrowseName*, *Clients* can use the *TranslateBrowsePathsToNodeId* Service with the default *BrowseName* to get the *NodeId* of an *AddIn*.

In the example in Figure 12 the relative path “**MyFeature/MyPropertyM**” with the starting node **MD002** can be used to request the *NodeId* of this *Property* and the relative path “**MyFeature/MyMethodO**” can be used for the respective *Method*.

5 Standard NodeClasses

5.1 Overview

Clause 5 defines the *NodeClasses* used to define *Nodes* in the OPC UA *AddressSpace*. *NodeClasses* are derived from a common *Base NodeClass*. This *NodeClass* is defined first, followed by those used to organise the *AddressSpace* and then by the *NodeClasses* used to represent *Objects*.

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

5.2 Base NodeClass

5.2.1 General

The OPC UA Address Space Model defines a *Base NodeClass* from which all other *NodeClasses* are derived. The derived *NodeClasses* represent the various components of the OPC UA Object Model (see 4.2). The *Attributes* of the *Base NodeClass* are specified in Table 7. There are no *References* specified for the *Base NodeClass*.

Table 7 – 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 5.2.5
Description	O	LocalizedText	See 5.2.6
WriteMask	O	AttributeWriteMask	See 5.2.7
UserWriteMask	O	AttributeWriteMask	See 5.2.8
RolePermissions	O	RolePermissionType[]	See 5.2.9
UserRolePermissions	O	RolePermissionType[]	See 5.2.10
AccessRestrictions	O	AccessRestrictionType	See 5.2.11
References			
			No <i>References</i> specified for this <i>NodeClass</i>

5.2.2 NodeId

Nodes are unambiguously identified using a constructed identifier called the *NodeId*. Some *Servers* may accept alternative *NodeIds* in addition to the canonical *NodeId* represented in this *Attribute*. A *Server* shall persist the *identifierType* and *identifier NodeId* elements of a *Node* as well as the *Namespace Uri* which the *namespaceIndex NodeId* element references. A *Server* may change the *namespaceIndex NodeId* element of a *Node* with future *Sessions* and therefore a *Client* shall not assume the *namespaceIndex* will not change. The structure of the *NodeId* is defined in 8.2.

5.2.3 NodeClass

The *NodeClass Attribute* identifies the *NodeClass* of a *Node*. Its data type is defined in 8.29.

5.2.4 BrowseName

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

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

1201 Unlike *NodeIds*, the *BrowseName* cannot be used to unambiguously identify a *Node*. Different
1202 *Nodes* may have the same *BrowseName*.

1203 Subclause 8.3 defines the structure of the *BrowseName*. It contains a namespace and a string.
1204 The namespace is provided to make the *BrowseName* unique in some cases in the context of
1205 a *Node* (e.g. *Properties* of a *Node*) although not unique in the context of the *Server*. If different
1206 organizations define *BrowseNames* for *Properties*, the namespace of the *BrowseName* provided
1207 by the organization makes the *BrowseName* unique, although different organizations may use
1208 the same string having a slightly different meaning.

1209 *Servers* may often choose to use the same namespace for the *NodeId* and the *BrowseName*.
1210 However, if they want to provide a standard *Property*, its *BrowseName* shall have the
1211 namespace of the standards body although the namespace of the *NodeId* reflects something
1212 else, for example the local *Server*.

1213 Standards bodies defining standard type definitions shall use their namespace(s) for the *NodeId*
1214 of the *TypeDefinitionNode* as well as for the *BrowseName* of the *TypeDefinitionNode*.
1215 *BrowseNames* of *TypeDefinitionNodes*, *ReferenceTypes*, and *DataTypes* shall be unique. Any
1216 well-known instances used as entry points shall also be unique. For example, the *Root Node*
1217 defined in IEC 62541-5.

1218 The string-part of the *BrowseName* is case sensitive. That is, *Clients* shall consider them case
1219 sensitive. *Servers* are allowed to handle *BrowseNames* passed in *Service* requests as case
1220 insensitive. Examples are the *TranslateBrowsePathsToNodeIds Service* or *Event* filter. If a
1221 *Server* accepts a case insensitive *BrowseName* it needs to ensure that the uniqueness of the
1222 *BrowseName* does not depend on case.

1223 5.2.5 DisplayName

1224 The *DisplayName Attribute* contains the localised name of the *Node*. *Clients* should use this
1225 *Attribute* if they want to display the name of the *Node* to the user. They should not use the
1226 *BrowseName* for this purpose. The *Server* may maintain one or more localised representations
1227 for each *DisplayName*. *Clients* negotiate the locale to be returned when they open a session
1228 with the *Server*. Refer to IEC 62541-4 for a description of session establishment and locales.
1229 Subclause 8.5 defines the structure of the *DisplayName*. The string part of the *DisplayName* is
1230 restricted to 512 characters.

1231 5.2.6 Description

1232 The optional *Description Attribute* shall explain the meaning of the *Node* in a localised text using
1233 the same mechanisms for localisation as described for the *DisplayName* in 5.2.5.

1234 5.2.7 WriteMask

1235 The optional *WriteMask Attribute* exposes the possibilities of a client to write the *Attributes* of
1236 the *Node*. The *WriteMask Attribute* does not take any user access rights into account, that is,
1237 although an *Attribute* is writeable this may be restricted to a certain user/user group.

1238 If the OPC UA *Server* does not have the ability to get the *WriteMask* information for a specific
1239 *Attribute* from the underlying system, it should state that it is writeable. If a write operation is
1240 called on the *Attribute*, the *Server* should transfer this request and return the corresponding
1241 *StatusCode* if such a request is rejected. *StatusCodes* are defined in IEC 62541-4.

1242 The *AttributeWriteMask DataType* is defined in 8.60.

1243 5.2.8 UserWriteMask

1244 The optional *UserWriteMask Attribute* exposes the possibilities of a client to write the *Attributes*
1245 of the *Node* taking user access rights into account. It uses the *AttributeWriteMask DataType*
1246 which is defined in 8.60.

1247 The *UserWriteMask Attribute* can only further restrict the *WriteMask Attribute*, when it is set to
1248 not writeable in the general case that applies for every user.