



DPC: 24 / 30484400 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: 29 January 2024

Origin: International

Latest date for receipt of comments: 20 March 2024

Project No. 2023/02967

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

Interested committees:

Title: BS EN IEC 62541-100 OPC Unified Architecture

Part 100: Device Interface

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 20 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 62541-100 ED2

DATE OF CIRCULATION:

2024-01-26

CLOSING DATE FOR VOTING:

2024-04-19

SUPERSEDES DOCUMENTS:

65E/972/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 100: Device Interface

PROPOSED STABILITY DATE: 2026

NOTE FROM TC/SC OFFICERS:

Copyright © 2024 International Electrotechnical Commission, IEC. All rights reserved. It is permitted to download this electronic file, to make a copy and to print out the content for the sole purpose of preparing National Committee positions. You may not copy or "mirror" the file or printed version of the document, or any part of it, for any other purpose without permission in writing from IEC.

CONTENTS

1		
2		
3	FOREWORD	xi
4	1 Scope	1
5	2 Reference documents	1
6	3 Terms, definitions, abbreviated terms, and conventions	2
7	3.1 Terms and definitions	2
8	3.2 Abbreviated terms	3
9	3.3 Conventions used in this document	4
10	3.3.1 Conventions for Node descriptions	4
11	3.3.2 NodeIds and BrowseNames	6
12	3.3.3 Common Attributes	7
13	4 Device model	9
14	4.1 General	9
15	4.2 Usage guidelines	10
16	4.3 TopologyElementType	10
17	4.4 FunctionalGroupType	11
18	4.4.1 Model	11
19	4.4.2 Recommended FunctionalGroup BrowseNames	12
20	4.4.3 UIElement Type	13
21	4.5 Interfaces	14
22	4.5.1 Overview	14
23	4.5.2 VendorNameplate Interface	14
24	4.5.3 TagNameplate Interface	16
25	4.5.4 DeviceHealth Interface	17
26	4.5.5 OperationCounter Interface	18
27	4.5.6 SupportInfo Interface	20
28	4.6 ComponentType	22
29	4.7 DeviceType	23
30	4.8 SoftwareType	26
31	4.9 DeviceSet entry point	26
32	4.10 DeviceFeatures entry point	27
33	4.11 BlockType	28
34	4.12 DeviceHealth Alarm Types	29
35	4.12.1 General	29
36	4.12.2 DeviceHealthDiagnosticAlarmType	30
37	4.12.3 FailureAlarmType	30
38	4.12.4 CheckFunctionAlarmType	31
39	4.12.5 OffSpecAlarmType	31
40	4.12.6 MaintenanceRequiredAlarmType	31
41	5 Device communication model	32
42	5.1 General	32
43	5.2 ProtocolType	33
44	5.3 Network	35
45	5.4 ConnectionPoint	36
46	5.5 ConnectsTo and ConnectsToParent ReferenceTypes	37
47	5.6 NetworkSet Object	39
48	6 Device integration host model	40

49	6.1	General	40
50	6.2	DeviceTopology Object	41
51	6.3	Online/Offline	42
52	6.3.1	General	42
53	6.3.2	IsOnline ReferenceType	43
54	6.4	Offline-Online data transfer	44
55	6.4.1	Definition	44
56	6.4.2	TransferServices Type	44
57	6.4.3	TransferServices Object	45
58	6.4.4	TransferToDevice Method	45
59	6.4.5	TransferFromDevice Method	46
60	6.4.6	FetchTransferResultData Method	47
61	7	Locking model	51
62	7.1	Overview	51
63	7.2	LockingServices Type	51
64	7.3	LockingServices Object	53
65	7.4	MaxInactiveLockTime Property	53
66	7.5	InitLock Method	53
67	7.6	ExitLock Method	54
68	7.7	RenewLock Method	55
69	7.8	BreakLock Method	55
70	8	Software update model	57
71	8.1	Overview	57
72	8.2	Use Cases	57
73	8.2.1	Supported Use Cases	57
74	8.2.2	Unsupported Use Cases	59
75	8.3	General	59
76	8.3.1	System perspective	59
77	8.3.2	Types of software	60
78	8.3.3	Types of Devices	60
79	8.3.4	Options for the Server	60
80	8.3.5	Software Update Client	63
81	8.3.6	Safety considerations	67
82	8.3.7	Security considerations	67
83	8.3.8	Update Behavior	68
84	8.3.9	Installation of patches	68
85	8.3.10	Incompatible parameters / settings	68
86	8.3.11	AddIn model	68
87	8.4	ObjectTypes	69
88	8.4.1	SoftwareUpdateType	69
89	8.4.2	SoftwareLoadingType	72
90	8.4.3	PackageLoadingType	72
91	8.4.4	DirectLoadingType	74
92	8.4.5	CachedLoadingType	75
93	8.4.6	FileSystemLoadingType	77
94	8.4.7	SoftwareVersionType	79
95	8.4.8	PrepareForUpdateStateMachineType	81
96	8.4.9	InstallationStateMachineType	84
97	8.4.10	PowerCycleStateMachineType	89

98	8.4.11	ConfirmationStateMachineType	90
99	8.5	DataTypes.....	92
100	8.5.1	SoftwareVersionFileType.....	92
101	8.5.2	UpdateBehavior OptionSet	93
102	9	Specialized topology elements	94
103	9.1	General.....	94
104	9.2	Configurable components	94
105	9.2.1	General pattern	94
106	9.2.2	ConfigurableObjectType	94
107	9.3	Block Devices.....	95
108	9.4	Modular Devices.....	96
109	10	Lifetime model	98
110	10.1	General.....	98
111	10.2	LifetimeVariableType definition	98
112	10.2.1	Overview.....	98
113	10.2.2	VariableType definition	98
114	10.3	BaseLifetimeIndicationType definition	100
115	10.3.1	Overview.....	100
116	10.3.2	ObjectType definition.....	100
117	10.4	TimeIndicationType definition	100
118	10.4.1	Overview.....	100
119	10.4.2	ObjectType definition.....	100
120	10.5	NumberOfPartsIndicationType definition	100
121	10.5.1	Overview.....	100
122	10.5.2	ObjectType definition.....	101
123	10.6	NumberOfUsagesIndicationType definition	101
124	10.6.1	Overview.....	101
125	10.6.2	ObjectType definition.....	101
126	10.7	LengthIndicationType definition.....	101
127	10.7.1	Overview.....	101
128	10.7.2	ObjectType definition.....	101
129	10.8	DiameterIndicationType definition	101
130	10.8.1	Overview.....	101
131	10.8.2	ObjectType definition.....	102
132	10.9	SubstanceVolumeIndicationType definition.....	102
133	10.9.1	Overview.....	102
134	10.9.2	ObjectType definition.....	102
135	11	Profiles and ConformanceUnits	103
136	11.1	Conformance Units	103
137	11.2	Profiles	105
138	11.2.1	General.....	105
139	11.2.2	Profile list.....	105
140	11.2.3	Device Server Facets	106
141	11.2.4	Device Client Facets.....	108
142	12	Namespaces	111
143	12.1	Namespace Metadata	111
144	12.2	Handling of OPC UA namespaces.....	111
145	Annex A (normative)	Namespace and mappings	113

146	Annex B (informative) Examples	114
147	B.1 Functional Group Usages	114
148	B.2 Identification Functional Group	115
149	B.3 Software Update examples	115
150	B.3.1 Factory Automation Example	115
151	B.3.2 Update sequence using Direct-Loading	118
152	B.3.3 Update sequence using Cached-Loading	119
153	B.3.4 Update sequence using File System based Loading	120
154	Annex C (informative) Guidelines for the usage of OPC UA for Devices as base for	
155	Companion Specifications	122
156	C.1 Overview	122
157	C.2 Guidelines to define Companion Specifications based on OPC UA for Devices ..	124
158	C.3 Guidelines on how to combine different companion specifications based on OPC	
159	UA for Devices in one OPC UA application	125
160	C.4 Guidelines to manage the same Variables defined in different places	127
161	C.5 Guidelines on how to use functionality in companion specifications	128
162	Bibliography	130

163

FIGURES

164		
165	Figure 1 – Device model overview	9
166	Figure 2 – Components of the TopologyElementType.....	10
167	Figure 3 – FunctionalGroupType.....	12
168	Figure 4 – Overview of Interfaces for Devices and Device components.....	14
169	Figure 5 – VendorNameplate Interface.....	14
170	Figure 6 – TagNameplate Interface.....	17
171	Figure 7 – DeviceHealth Interface.....	17
172	Figure 8 – Support information Interface	20
173	Figure 9 – ComponentType	22
174	Figure 10 – DeviceType	24
175	Figure 11 – SoftwareType	26
176	Figure 12 – Standard entry point for Devices	27
177	Figure 13 – Standard entry point for DeviceFeatures.....	28
178	Figure 14 – BlockType hierarchy	28
179	Figure 15 – Device Health Alarm type hierarchy.....	30
180	Figure 16 – Device communication model overview.....	32
181	Figure 17 – Example of a communication topology.....	33
182	Figure 18 – Example of a ProtocolType hierarchy with instances that represent specific	
183	communication profiles.....	34
184	Figure 19 – NetworkType	35
185	Figure 20 – Example of ConnectionPointType hierarchy.....	36
186	Figure 21 – ConnectionPointType.....	36
187	Figure 22 – ConnectionPoint usage	37
188	Figure 23 – Type Hierarchy for ConnectsTo and ConnectsToParent References.....	38
189	Figure 24 – Example with ConnectsTo and ConnectsToParent References.....	39
190	Figure 25 – Example of an automation system	40
191	Figure 26 – Example of a Device topology	41
192	Figure 27 – Online component for access to Device data	42
193	Figure 28 – Type hierarchy for IsOnline Reference.....	43
194	Figure 29 – TransferServicesType	44
195	Figure 30 – TransferServices.....	45
196	Figure 31 – LockingServicesType	51
197	Figure 32 – LockingServices.....	53
198	Figure 33 – Example with a device and several software components	60
199	Figure 34 – Determine the type of update that the Server implements.	63
200	Figure 35 – Different flows of <i>Direct-Loading</i> , <i>Cached-Loading</i> and <i>FileSystem based</i>	
201	<i>Loading</i>	64
202	Figure 36 – Prepare and Resume activities	65
203	Figure 37 – Installation activity for <i>Direct-Loading</i>	66
204	Figure 38 – Installation activity for <i>Cached-Loading</i> and <i>File System based Loading</i>	66
205	Figure 39 – Resume activity	67
206	Figure 40 – Example how to add the SoftwareUpdate AddIn to a component	69
207	Figure 41 – SoftwareUpdateType.....	

208	Figure 42 – PackageLoadingType.....	73
209	Figure 43 – DirectLoadingType.....	74
210	Figure 44 – CachedLoadingType.....	75
211	Figure 45 – FileSystemLoadingType.....	77
212	Figure 46 – SoftwareVersionType.....	79
213	Figure 47 – PrepareForUpdate state machine.....	81
214	Figure 48 – PrepareForUpdateStateMachineType.....	82
215	Figure 49 – Installation state machine.....	85
216	Figure 50 – InstallationStateMachine.....	85
217	Figure 51 – PowerCycle state machine.....	89
218	Figure 52 – Confirmation state machine.....	91
219	Figure 53 – ConfirmationStateMachineType.....	91
220	Figure 54 – Configurable component pattern.....	94
221	Figure 55 – ConfigurableObjectType.....	95
222	Figure 56 – Block-oriented Device structure example.....	96
223	Figure 57 – Modular Device structure example.....	97
224	Figure B.1 – Analyser Device use for FunctionalGroups.....	114
225	Figure B.2 – PLCopen use for FunctionalGroups.....	114
226	Figure B.3 – Example of an Identification FunctionalGroup.....	115
227	Figure B.4 – Example.....	116
228	Figure B.5 – Example sequence of Direct-Loading.....	118
229	Figure B.6 – Example sequence of Cached-Loading.....	119
230	Figure B.7 – Example sequence of File System based Loading.....	121
231	Figure C.1 – Example of applying two companion specifications based on OPC UA for	
232	Devices.....	123
233	Figure C.2 – Using composition to compose one device representation defined by two	
234	companion specifications.....	124
235	Figure C.3 – Example of applying several companion specifications (I).....	126
236	Figure C.4 – Example of applying several companion specifications (II).....	127
237	Figure C.5 – Options how to manage the same Variable.....	128
238	Figure C.6 – Example on how to use AddIns and Interface.....	128
239	Figure C.7 – Example on how to use Interface with additional Object.....	129
240		

TABLES

241		
242	Table 1 – Examples of DataTypes	4
243	Table 2 – Type Definition Table	5
244	Table 3 – Examples of Other Characteristics	5
245	Table 4 – <some>Type Additional References	6
246	Table 5 – <some>Type Additional Subcomponents	6
247	Table 6 – <some>Type Attribute values for child Nodes	6
248	Table 7 – Common Node Attributes	7
249	Table 8 – Common Object Attributes	8
250	Table 9 – Common Variable Attributes	8
251	Table 10 – Common VariableType Attributes	8
252	Table 11 – Common Method Attributes	8
253	Table 12 – TopologyElementType definition	10
254	Table 14 – FunctionalGroupType definition	12
255	Table 15 – Recommended FunctionalGroup BrowseNames	13
256	Table 16 – UIElementType definition	13
257	Table 17 – IVendorNameplateType definition	15
258	Table 18 – VendorNameplate Mapping to IRDIs	16
259	Table 19 – ITagNameplateType definition	17
260	Table 20 – TagNameplate Mapping to IRDIs	17
261	Table 21 – IDeviceHealthType definition	18
262	Table 22 – DeviceHealthEnumeration values	18
263	Table 23 – DeviceHealthEnumeration definition	18
264	Table 24 – IOperationCounterType definition	19
265	Table 25 – IOperationCounterType Attribute values for child Nodes	19
266	Table 26 – ISupportInfoType definition	21
267	Table 28 – ComponentType definition	23
268	Table 29 – DeviceType definition	24
269	Table 30 – SoftwareType definition	26
270	Table 31 – DeviceSet definition	27
271	Table 32 – DeviceFeatures definition	28
272	Table 33 – BlockType definition	29
273	Table 34 – DeviceHealthDiagnosticAlarmType definition	30
274	Table 35 – FailureAlarmType definition	31
275	Table 36 – CheckFunctionAlarmType definition	31
276	Table 37 – OffSpecAlarmType definition	31
277	Table 38 – MaintenanceRequiredAlarmType definition	31
278	Table 39 – ProtocolType definition	34
279	Table 40 – NetworkType definition	35
280	Table 41 – ConnectionPointType definition	37
281	Table 42 – ConnectsTo ReferenceType	38
282	Table 43 – ConnectsToParent ReferenceType	38
283	Table 44 – NetworkSet definition	39

284	Table 45 – DeviceTopology definition	42
285	Table 46 – IsOnline ReferenceType	44
286	Table 47 – TransferServicesType definition	45
287	Table 48 – TransferToDevice Method arguments	46
288	Table 49 – TransferToDevice Method AddressSpace definition	46
289	Table 50 – TransferFromDevice Method arguments	47
290	Table 51 – TransferFromDevice Method AddressSpace definition	47
291	Table 52 – FetchTransferResultData Method arguments	48
292	Table 53 – FetchTransferResultData Method AddressSpace definition	48
293	Table 54 – FetchResultDataType structure	48
294	Table 55 – TransferResultErrorDataType structure	49
295	Table 56 – TransferResultErrorDataType Definition	49
296	Table 57 – TransferResultDataDataType structure	49
297	Table 58 – TransferResultDataDataType definition	50
298	Table 59 – LockingServicesType definition	52
299	Table 60 – LockingServicesType Additional Variable Attributes	52
300	Table 61 – MaxInactiveLockTime Property definition	53
301	Table 62 – InitLock Method Arguments	54
302	Table 63 – InitLock Method AddressSpace definition.....	54
303	Table 64 – ExitLock Method Arguments	54
304	Table 65 – ExitLock Method AddressSpace definition.....	55
305	Table 66 – RenewLock Method Arguments	55
306	Table 67 – RenewLock Method AddressSpace definition	55
307	Table 68 – BreakLock Method Arguments	55
308	Table 69 – BreakLock Method AddressSpace definition	56
309	Table 70 – SoftwareUpdateType definition	70
310	Table 71 – SoftwareUpdateType Attribute values for child Nodes	72
311	Table 72 – SoftwareLoadingType definition.....	72
312	Table 73 – PackageLoadingType definition	73
313	Table 74 – TemporaryFileTransferType Result Codes	73
314	Table 75 – DirectLoadingType definition	75
315	Table 76 – CachedLoadingType definition	76
316	Table 77 – GetUpdateBehavior Method Arguments	77
317	Table 78 – GetUpdateBehavior Method AddressSpace definition.....	77
318	Table 79 – FileSystemLoadingType definition	78
319	Table 80 – GetUpdateBehavior Method Arguments	78
320	Table 81 – GetUpdateBehavior Method AddressSpace definition.....	78
321	Table 82 – ValidateFiles Method Arguments	79
322	Table 83 – ValidateFiles Method AddressSpace definition.....	79
323	Table 84 – SoftwareVersionType definition	80
324	Table 85 – PrepareForUpdateStateMachineType definition	82
325	Table 86 – PrepareForUpdateStateMachineType Attribute values for child Nodes.....	83
326	Table 87 – PrepareForUpdateStateMachineType Additional References.....	83

327	Table 88 – InstallationStateMachineType definition	86
328	Table 89 – InstallationStateMachineType Attribute values for child Nodes	86
329	Table 90 – InstallationStateMachineType Additional References	86
330	Table 91 – InstallSoftwarePackage Method Arguments	88
331	Table 92 – InstallSoftwarePackage Method AddressSpace definition	88
332	Table 93 – InstallFiles Method Arguments	88
333	Table 94 – InstallFiles Method AddressSpace definition	89
334	Table 95 – PowerCycleStateMachineType definition	90
335	Table 96 – PowerCycleStateMachineType Attribute values for child Nodes	90
336	Table 97 – PowerCycleStateMachineType Additional References	90
337	Table 98 – ConfirmationStateMachineType	91
338	Table 99 – ConfirmationStateMachineType Attribute values for child Nodes	92
339	Table 100 – ConfirmationStateMachineType TargetBrowsePath	92
340	Table 101 – SoftwareVersionFileType Items	92
341	Table 102 – SoftwareVersionFileType definition	93
342	Table 103 – UpdateBehavior OptionSet	93
343	Table 104 – UpdateBehavior OptionSet Definition	93
344	Table 105 – ConfigurableObjectType definition	95
345	Table 106 – Lifetime examples	98
346	Table 107 – LifetimeVariableType definition	98
347	Table 108 – LifetimeVariableType Attribute values for child Nodes	100
348	Table 109 – BaseLifetimeIndicationType definition	100
349	Table 110 – TimeIndicationType definition	100
350	Table 111 – NumberOfPartsIndicationType definition	101
351	Table 112 – NumberOfUsagesIndicationType definition	101
352	Table 113 – LengthIndicationType definition	101
353	Table 114 – DiameterIndicationType definition	102
354	Table 115 – SubstanceVolumeIndicationType definition	102
355	Table 116 – Conformance Units for Devices	103
356	Table 117 – Profile URIs for Devices	105
357	Table 118 – DI BaseDevice Server Facet definition	106
358	Table 119 – DI DeviceIdentification Server Facet definition	106
359	Table 120 – DI BlockDevice Server Facet definition	106
360	Table 121 – DI Locking Server Facet definition	106
361	Table 122 – DI DeviceCommunication Server Facet definition	106
362	Table 123 – DI DeviceIntegrationHost Server Facet definition	107
363	Table 124 – DI SU Software Update Base Server Facet	107
364	Table 125 – DI SU Direct Loading Server Facet	107
365	Table 126 – DI SU Cached Loading Server Facet	107
366	Table 127 – DI SU FileSystem Loading Server Facet	108
367	Table 128 – DI Lifetime Server Facet	108
368	Table 129 – DI BaseDevice Client Facet definition	108
369	Table 130 – DI DeviceIdentification Client Facet definition	108

370	Table 131 – DI BlockDevice Client Facet definition	108
371	Table 132 – DI Locking Client Facet definition	108
372	Table 133 – DI DeviceCommunication Client Facet definition	109
373	Table 134 – DI DeviceIntegrationHost Client Facet definition	109
374	Table 135 – DI SU Software Update Base Client Facet	109
375	Table 136 – DI SU Direct Loading Client Facet	109
376	Table 137 – DI SU Cached Loading Client Facet.....	109
377	Table 138 – DI SU FileSystem Loading Client Facet	110
378	Table 139 – DI Lifetime Client Facet definition	110
379	Table 140 – NamespaceMetadata Object for this Specification	111
380	Table 141 – Namespaces used in an OPC UA for Devices Server	111
381	Table 142 – Namespaces used in this specification.....	112
382		
383		

INTERNATIONAL ELECTROTECHNICAL COMMISSION

OPC UNIFIED ARCHITECTURE –

Part 100: Devices

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 62541-100 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

This third edition cancels and replaces the second edition published in 2015. This edition constitutes a technical revision.

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

- a) Defined a `ComponentType` that can be used to model any HW or SW element of a device. In addition added a `SoftwareType` as subtype of `ComponentType`.
- b) Added the new OPC UA interface concept and defined interfaces for `Nameplate`, `DeviceHealth`, and `SupportInfo`.
- c) Added a new model for Software Update (Firmware Update).
- d) Specified a new entry point for documents where each document is represented by a `FileType` instance.

- 436 e) Added a model that provides information about the lifetime, related limits and semantic of the
437 lifetime of things like tools, material or machines.

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

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

439
440 Full information on the voting for the approval of this International Standard can be found in the report
441 on voting indicated in the above table.

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

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

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

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

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

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

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

- 459 • reconfirmed,
460 • withdrawn,
461 • replaced by a revised edition, or
462 • amended.

463

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.

464

OPC UNIFIED ARCHITECTURE –

Part 100: Devices

1 Scope

This part of the IEC 62541 series defines the information model associated with *Devices*. This document describes three models which build upon each other as follows:

- The (base) Device Model is intended to provide a unified view of devices and their hardware and software parts irrespective of the underlying device protocols.
- The Device Communication Model adds Network and Connection information elements so that communication topologies can be created.
- The Device Integration Host Model finally adds additional elements and rules required for host systems to manage integration for a complete system. It allows reflecting the topology of the automation system with the devices as well as the connecting communication networks.

This document also defines AddIns that can be used for the models in this document but also for models in other information models. They are:

- Locking model – a generic AddIn to control concurrent access,
- Software update model – an AddIn to manage software in a *Device*.

2 Reference documents

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 62541-3, *OPC Unified Architecture - Part 3: Address Space 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 62541-7, *OPC Unified Architecture - Part 7: Profiles*

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

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

IEC 62541-19, *OPC Unified Architecture – Part 19: Dictionary References*

IEC 62769, *Field Device Integration (FDI)*

NAMUR Recommendation NE107: Self-monitoring and diagnosis of field devices

499 3 Terms, definitions, abbreviated terms, and conventions

500 3.1 Terms and definitions

501 For the purposes of this document, the terms and definitions given in IEC 62541-1, IEC 62541-3, and
502 IEC 62541-8 as well as the following apply.

503 3.1.1

504 **block**

505 functional *Parameter* grouping entity

506 Note 1 to entry: It could map to a function block (see IEC 62769) or to the resource parameters of the *Device* itself.

507 3.1.2

508 **blockMode**

509 mode of operation (target mode, permitted modes, actual mode, and normal mode) for a *Block*

510 Note 1 to entry: Further details about *Block* modes are defined by standard organisations.

511 3.1.3

512 **Communication Profile**

513 fixed set of mapping rules to allow unambiguous interoperability between *Devices* or Applications,
514 respectively

515 Note 1 to entry: Examples of such profiles are the "Wireless communication network and communication profiles for
516 WirelessHART" in IEC 62591 and the Protocol Mappings for OPC UA in IEC 62541-6.

517 3.1.4

518 **Connection Point**

519 logical representation of the interface between a *Device* and a *Network*

520 3.1.5

521 **device**

522 independent physical entity capable of performing one or more specified functions in a particular
523 context and delimited by its interfaces

524 Note 1 to entry: See IEC 61499-1.

525 Note 2 to entry: *Devices* provide sensing, actuating, communication, and/or control functionality. Examples include
526 transmitters, valve controllers, drives, motor controllers, PLCs, and communication gateways.

527 Note 3 to entry: A *Device* can be a system (topology) of other *Devices*, components, or parts.

528 3.1.6

529 **Device Integration Host**

530 *Server* that manages integration of multiple *Devices* in an automation system

531 3.1.7

532 **Device Topology**

533 arrangement of *Networks* and *Devices* that constitute a communication topology

534 3.1.8

535 **fieldbus**

536 communication system based on serial data transfer and used in industrial automation or process
537 control applications

538 Note 1 to entry: See IEC 61784.

539 Note 2 to entry: Designates the communication bus used by a *Device*.

540 3.1.9

541 **Parameter**

542 variable of the *Device* that can be used for configuration, monitoring or control purposes

543 Note 1 to entry: In the information model it is synonymous to an OPC UA *DataVariable*.

3.1.10**Network**

means used to communicate with one specific protocol

3.1.11**Direct-Loading**

an update method where the original software is overwritten during the transfer

3.1.12**Cached-Loading**

an update method where the new software is stored in a separate area

Note 1 to entry: Installation is performed later as an extra step.

3.1.13**File System based Loading**

an update method based on an accessible directory structure and a separate install method

3.1.14**Software Package**

a single file that contains the data for the software update in a device specific format

3.1.15**Software Update Client**

an update client that can be used for devices of several vendors

Note 1 to entry: There can be different Software Update Clients for different domains (e.g. process industry or manufacturing).

3.1.16**Current Version**

version information of the software that is currently installed

3.1.17**Pending Version**

version information for a *Software Package* that was transferred before and is ready to be installed

3.1.18**Fallback Version**

version information about an alternatively installable software that is located on the *Server*

Note 1 to entry: Examples: factory default version or the version before the latest update

3.2 Abbreviated terms

ADI	Analyser Device Integration
CP	Communication Processor (hardware module)
CPU	Central Processing Unit (of a <i>Device</i>)
DA	Data Access
DI	Device Integration (the short name for this part)
ERP	Enterprise Resource Planning
IRDI	International Registration Data Identifiers
UA	Unified Architecture
UML	Unified Modelling Language
XML	Extensible Mark-up Language

3.3 Conventions used in this document

3.3.1 Conventions for Node descriptions

3.3.1.1 Node definitions

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

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

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

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

Table 1 – Examples of DataTypes

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

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

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

Note that if a symbolic name of a different namespace is used, it is prefixed by the *NamespaceIndex* (see 3.3.2.2).

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

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

625 **Table 2 – Type Definition Table**

Attribute	Value				
Attribute name	Attribute value. If it is an optional Attribute that is not set "--" is used.				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
ReferenceType name	NodeClass of the target Node.	BrowseName of the target Node.	DataType of the referenced Node, only applicable for Variables.	TypeDefinition of the referenced Node, only applicable for Variables and Objects.	Additional characteristics of the TargetNode such as the ModellingRule or AccessLevel.
NOTE Notes referencing footnotes of the table content.					

626
 627 Components of *Nodes* can be complex that is containing components by themselves. The
 628 *TypeDefinition*, *NodeClass* and *DataType* can be derived from the type definitions, and the symbolic
 629 name can be created as defined in 3.3.3.1. Therefore, those containing components are not explicitly
 630 specified; they are implicitly specified by the type definitions.

631 The Other column defines additional characteristics of the Node. Examples of characteristics that can
 632 appear in this column are show in Table 3.

633 **Table 3 – Examples of Other Characteristics**

Name	Short Name	Description
0:Mandatory	M	The Node has the <i>Mandatory ModellingRule</i> .
0:Optional	O	The Node has the <i>Optional ModellingRule</i> .
0:MandatoryPlaceholder	MP	The Node has the <i>MandatoryPlaceholder ModellingRule</i> .
0:OptionalPlaceholder	OP	The Node has the <i>OptionalPlaceholder ModellingRule</i> .
ReadOnly	RO	The <i>AccessLevel</i> has the <i>CurrentRead</i> bit set but not the <i>CurrentWrite</i> bit.
ReadWrite	RW	The <i>AccessLevel</i> has the <i>CurrentRead</i> and <i>CurrentWrite</i> bits set.
WriteOnly	WO	The <i>AccessLevel</i> has the <i>CurrentWrite</i> bit set but not the <i>CurrentRead</i> bit.

634 If multiple characteristics are defined, they are separated by commas. The name or the short name
 635 may be used.

636 Each *Node* defined in this document has *ConformanceUnits* defined in 11.1 that require the *Node* to
 637 be in the *AddressSpace*. If a *Server* supports a *ConformanceUnit*, it shall expose the *Nodes* related
 638 to the *ConformanceUnit* in its *AddressSpace*. If two *Nodes* are exposed, all *References* between the
 639 *Nodes* defined in this document shall be exposed as well.

640 The relations between *Nodes* and *ConformanceUnits* are defined at the end of the tables defining
 641 *Nodes*, one row per *ConformanceUnit*. The *ConformanceUnit* is reflected with a *Category* element in
 642 the *UANodeSet* file (see IEC 62541-6).

643 The *Nodes* defined in a table are not only the *Node* defined on top level, for example an *ObjectType*,
 644 but also the *Nodes* that are referenced, as long as they are not defined in other tables. For example,
 645 the *ObjectType TopologyElementType* defines its *InstanceDeclarations* in the same table, so the
 646 *InstanceDeclarations* are also bound to the *ConformanceUnits* defined for the table. The table even
 647 indirectly defines additional *InstanceDeclarations* as components of the top-level
 648 *InstanceDeclarations*, that are not directly visible in the table. The *TypeDefinitions* and *DataTypes*
 649 used in the *InstanceDeclarations*, and the *ReferenceTypes* are defined in their individual tables and
 650 not in the table itself, therefore they are not bound to the *ConformanceUnits* of the table.

3.3.1.2 Additional References

To provide information about additional *References*, the format as shown in Table 4 is used.

Table 4 – <some>Type Additional References

SourceBrowsePath	Reference Type	Is Forward	TargetBrowsePath
SourceBrowsePath is always relative to the <i>TypeDefinition</i> . Multiple elements are defined as separate rows of a nested table.	<i>ReferenceType</i> name	True = forward <i>Reference</i> .	TargetBrowsePath points to another <i>Node</i> , which can be a well-known instance or a <i>TypeDefinition</i> . You can use <i>BrowsePaths</i> here as well, which is either relative to the <i>TypeDefinition</i> or absolute. If absolute, the first entry needs to refer to a type or well-known instance, uniquely identified within a namespace by the <i>BrowseName</i> .

References can be to any other *Node*.

3.3.1.3 Additional sub-components

To provide information about sub-components, the format as shown in Table 5 is used.

Table 5 – <some>Type Additional Subcomponents

BrowsePath	References	NodeClass	BrowseName	DataType	TypeDefinition	Other
BrowsePath is always relative to the <i>TypeDefinition</i> . Multiple elements are defined as separate rows of a nested table	NOTE Same as for Table 2					

3.3.1.4 Additional Attribute values

The type definition table provides columns to specify the values for required *Node Attributes* for *InstanceDeclarations*. To provide information about additional *Attributes*, the format as shown in Table 6 is used.

Table 6 – <some>Type Attribute values for child Nodes

BrowsePath	<Attribute name> Attribute
BrowsePath is always relative to the <i>TypeDefinition</i> . Multiple elements are defined as separate rows of a nested table	The values of <i>Attributes</i> are converted to text by adapting the reversible JSON encoding rules defined in IEC 62541-6. If the JSON encoding of a value is a JSON string or a JSON number then that value is entered in the value field. Double quotes are not included. If the <i>DataType</i> includes a <i>NamespaceIndex</i> (<i>QualifiedNames</i>, <i>NodeIds</i> or <i>ExpandedNodeIds</i>) then the notation used for <i>BrowseNames</i> is used. If the value is an <i>Enumeration</i> the name of the enumeration value is entered. If the value is a <i>Structure</i> then a sequence of name and value pairs is entered. Each pair is followed by a newline. The name is followed by a colon. The names are the names of the fields in the <i>DataTypeDefinition</i>. If the value is an array of non-<i>Structures</i> then a sequence of values is entered where each value is followed by a newline. If the value is an array of <i>Structures</i> or a <i>Structure</i> with fields that are arrays or with nested <i>Structures</i> then the complete JSON array or JSON object is entered.

There can be multiple columns to define more than one *Attribute*.

3.3.2 NodeIds and BrowseNames

3.3.2.1 NodeIds

The *NodeIds* of all *Nodes* described in this standard are only symbolic names. Annex A defines the actual *NodeIds*.

The symbolic name of each *Node* defined in this document is its *BrowseName*, or, when it is part of another *Node*, the *BrowseName* of the other *Node*, a “.”, and the *BrowseName* of itself. In this case “part of” means that the whole has a *HasProperty* or *HasComponent Reference* to its part. Since all *Nodes* not being part of another *Node* have a unique name in this document, the symbolic name is unique.

The *NamespaceUri* for all *NodeIds* defined in this document is defined in Annex A. The *NamespaceIndex* for this *NamespaceUri* is vendor-specific and depends on the position of the *NamespaceUri* in the server namespace table.

Note that this document not only defines concrete *Nodes*, but also requires that some *Nodes* shall be generated, for example one for each *Session* running on the *Server*. The *NodeIds* of those *Nodes* are *Server*-specific, including the namespace. But the *NamespaceIndex* of those *Nodes* cannot be the *NamespaceIndex* used for the *Nodes* defined in this document, because they are not defined by this document but generated by the *Server*.

3.3.2.2 BrowseNames

The text part of the *BrowseNames* for all *Nodes* defined in this document is specified in the tables defining the *Nodes*. The *NamespaceUri* for all *BrowseNames* defined in this document is defined in Annex A.

For *InstanceDeclarations* of *NodeClass Object* and *Variable* that are placeholders (*OptionalPlaceholder* and *MandatoryPlaceholder ModellingRule*), the *BrowseName* and the *DisplayName* are enclosed in angle brackets (<>) as recommended in IEC 62541-3. If the *BrowseName* is not defined by this document, a namespace index prefix is added to the *BrowseName* (e.g., prefix '0' leading to '0:EngineeringUnits' or prefix '2' leading to '2:DeviceRevision'). This is typically necessary if a *Property* of another specification is overwritten or used in the OPC UA types defined in this document. Clause 12.2 provides a list of namespaces and their indexes as used in this document.

3.3.3 Common Attributes

3.3.3.1 General

The *Attributes* of *Nodes*, their *DataTypes* and descriptions are defined in IEC 62541-3. Attributes not marked as optional are mandatory and shall be provided by a *Server*. The following tables define if the *Attribute* value is defined by this document or if it is server-specific.

For all *Nodes* specified in this document, the *Attributes* named in Table 7 shall be set as specified in the table.

Table 7 – Common Node Attributes

Attribute	Value
DisplayName	The <i>DisplayName</i> is a <i>LocalizedText</i> . Each <i>Server</i> shall provide the <i>DisplayName</i> identical to the <i>BrowseName</i> of the <i>Node</i> for the <i>LocaleId</i> “en”. Whether the server provides translated names for other <i>LocaleIds</i> are server-specific.
Description	Optionally a server-specific description is provided.
NodeClass	Shall reflect the <i>NodeClass</i> of the <i>Node</i> .
NodeId	The <i>NodeId</i> is described by <i>BrowseNames</i> as defined in 3.3.2.1.
WriteMask	Optionally the <i>WriteMask Attribute</i> can be provided. If the <i>WriteMask Attribute</i> is provided, it shall set all non-server-specific <i>Attributes</i> to not writable. For example, the <i>Description Attribute</i> may be set to writable since a <i>Server</i> may provide a server-specific description for the <i>Node</i> . The <i>NodeId</i> shall not be writable, because it is defined for each <i>Node</i> in this document.
UserWriteMask	Optionally the <i>UserWriteMask Attribute</i> can be provided. The same rules as for the <i>WriteMask Attribute</i> apply.
RolePermissions	Optionally server-specific role permissions can be provided.
UserRolePermissions	Optionally the role permissions of the current <i>Session</i> can be provided. The value is server-specific and depends on the <i>RolePermissions Attribute</i> (if provided) and the current <i>Session</i> .
AccessRestrictions	Optionally server-specific access restrictions can be provided.

3.3.3.2 Objects

For all *Objects* specified in this document, the *Attributes* named in Table 8 shall be set as specified in the Table 8. The definitions for the *Attributes* can be found in IEC 62541-3.

Table 8 – Common Object Attributes

Attribute	Value
EventNotifier	Whether the <i>Node</i> can be used to subscribe to <i>Events</i> or not is server-specific.

3.3.3.3 Variables

For all *Variables* specified in this document, the *Attributes* named in Table 9 shall be set as specified in the table. The definitions for the *Attributes* can be found in IEC 62541-3.

Table 9 – Common Variable Attributes

Attribute	Value
MinimumSamplingInterval	Optionally, a server-specific minimum sampling interval is provided.
AccessLevel	The access level for <i>Variables</i> used for type definitions is server-specific, for all other <i>Variables</i> defined in this document, the access level shall allow reading; other settings are server-specific.
UserAccessLevel	The value for the <i>UserAccessLevel</i> Attribute is server-specific. It is assumed that all <i>Variables</i> can be accessed by at least one user.
Value	For <i>Variables</i> used as <i>InstanceDeclarations</i> , the value is server-specific; otherwise it shall represent the value described in the text.
ArrayDimensions	If the <i>ValueRank</i> does not identify an array of a specific dimension (i.e. <i>ValueRank</i> ≤ 0) the <i>ArrayDimensions</i> can either be set to null or the <i>Attribute</i> is missing. This behavior is server-specific. If the <i>ValueRank</i> specifies an array of a specific dimension (i.e. <i>ValueRank</i> > 0) then the <i>ArrayDimensions</i> Attribute shall be specified in the table defining the <i>Variable</i> .
Historizing	The value for the <i>Historizing</i> Attribute is server-specific.
AccessLevelEx	If the <i>AccessLevelEx</i> Attribute is provided, it shall have the bits 8, 9, and 10 set to 0, meaning that read and write operations on an individual <i>Variable</i> are atomic, and arrays can be partly written.

3.3.3.4 VariableTypes

For all *VariableTypes* specified in this document, the *Attributes* named in Table 10 shall be set as specified in the table. The definitions for the *Attributes* can be found in IEC 62541-3.

Table 10 – Common VariableType Attributes

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

3.3.3.5 Methods

For all *Methods* specified in this document, the *Attributes* named in Table 11 shall be set as specified in the table. The definitions for the *Attributes* can be found in IEC 62541-3.

Table 11 – Common Method Attributes

Attributes	Value
Executable	All <i>Methods</i> defined in this document shall be executable (<i>Executable</i> Attribute set to "True"), unless it is defined differently in the <i>Method</i> definition.
UserExecutable	The value of the <i>UserExecutable</i> Attribute is server-specific. It is assumed that all <i>Methods</i> can be executed by at least one user.

4 Device model

4.1 General

Figure 1 depicts the main *ObjectTypes* of the base device model and their relationship. The drawing is not intended to be complete. For the sake of simplicity only a few components and relations were captured to give a rough idea of the overall structure.

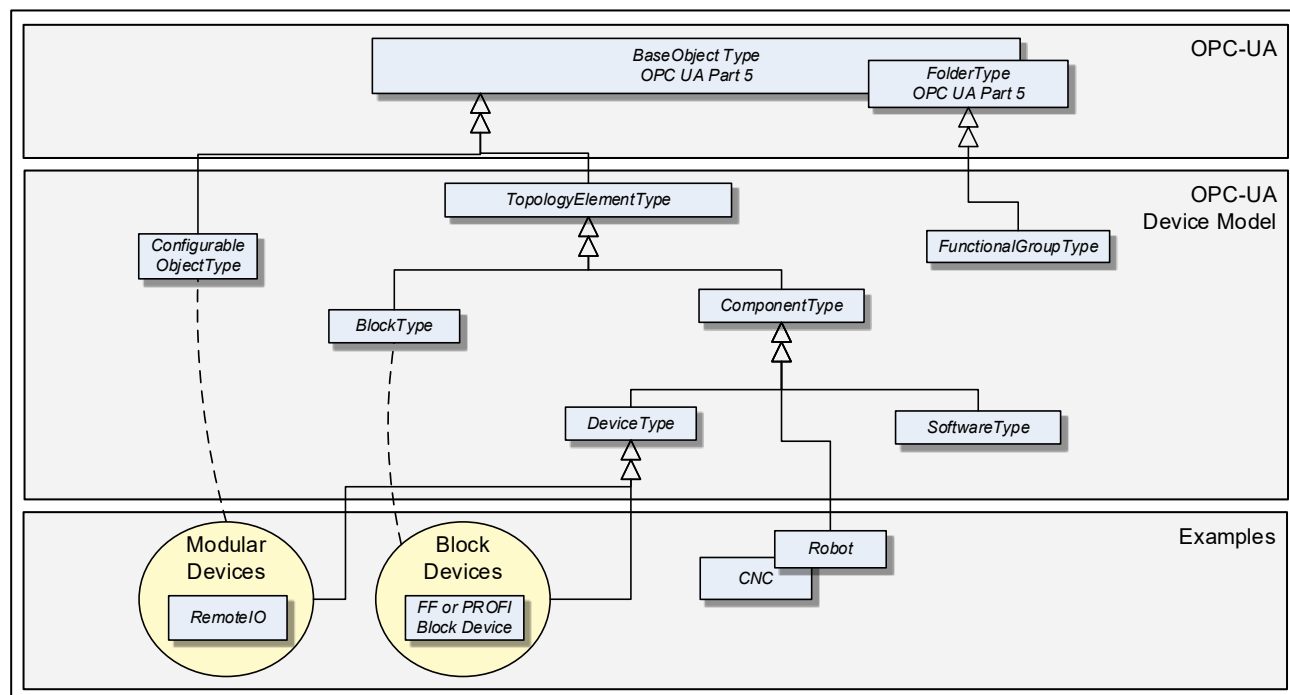


Figure 1 – Device model overview

The boxes in this drawing show the *ObjectTypes* used in this document as well as some elements from other parts that help understand some modelling decisions. The upper grey box shows the OPC UA core *ObjectTypes* from which the *TopologyElementType* is derived. The grey box in the second level shows the main *ObjectTypes* that the device model introduces. The components of those *ObjectTypes* are illustrated only in an abstract way in this overall picture.

The grey box in the third level shows real-world examples as they will be used in products and plants. In general, such subtypes are defined by other organizations.

The *TopologyElementType* is the base *ObjectType* for elements in a device topology. Its most essential aspect is the functional grouping concept.

The *ComponentType* *ObjectType* provides a generic definition for a *Device* or parts of a *Device* where parts include mechanics and/or software. *DeviceType* is commonly used to represent field *Devices*.

Modular Devices are introduced to support subdevices and *Block Devices* to support *Blocks*. *Blocks* are typically used by field communication foundations as means to organize the functionality within a *Device*. Specific types of *Blocks* will therefore be specified by these foundations.

The *ConfigurableObjectType* is used as a general means to create modular topology units. If needed an instance of this type will be added to the head object of the modular unit. *Modular Devices*, for example, will use this *ObjectType* to organize their modules. *Block-oriented Devices* use it to expose and organize their *Blocks*.

4.2 Usage guidelines

Annex C describes guidelines for the usage of the device model as base for creating companion specifications as well as guidelines on how to combine different aspects of the same device – defined in different companion specifications – in one OPC UA application.

4.3 TopologyElementType

This *ObjectType* defines a generic model for elements in a device or component topology. Among others, it introduces *FunctionalGroups*, *ParameterSet*, and *MethodSet*. Figure 2 shows the *TopologyElementType*. It is formally defined in Table 12.

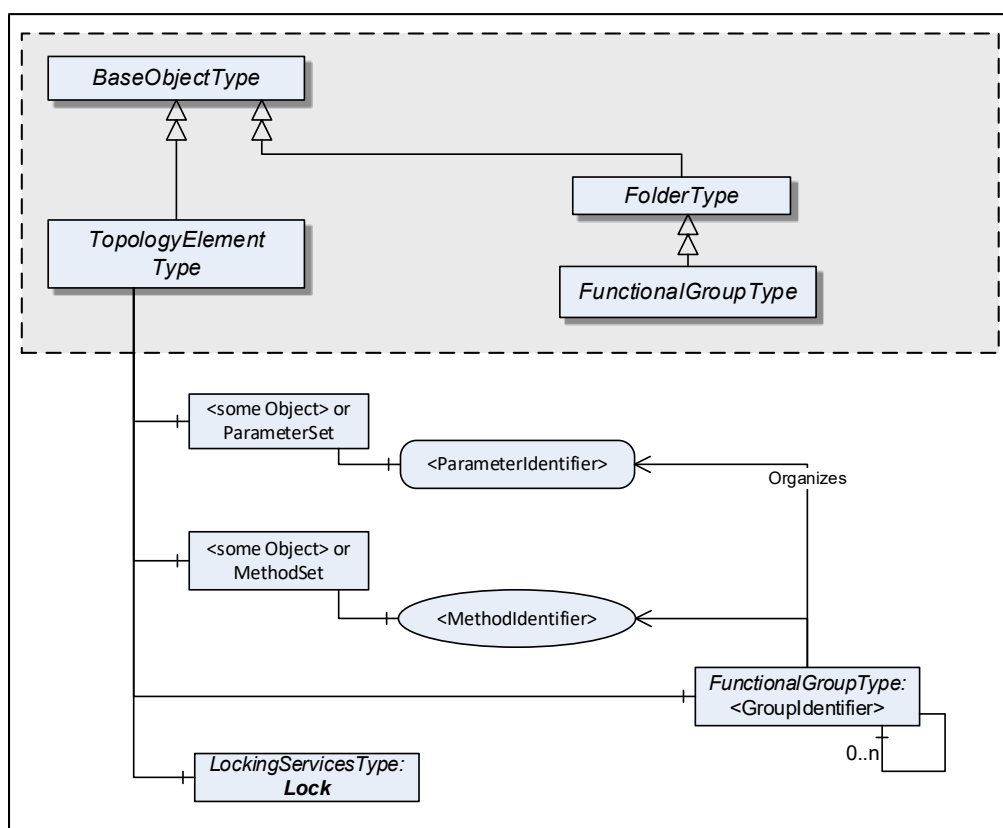


Figure 2 – Components of the TopologyElementType

Table 12 – TopologyElementType definition

Attribute	Value				
BrowseName	TopologyElementType				
IsAbstract	True				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
Subtype of the 0:BaseObjectType defined in IEC 62541-5					
0:HasSubtype	ObjectType	ComponentType	Defined in 4.6		
0:HasSubtype	ObjectType	BlockType	Defined in 4.11		
0:HasSubtype	ObjectType	ConnectionPointType	Defined in 5.4		
0:HasComponent	Object	<GroupIdentifier>		FunctionalGroupType	OP
0:HasComponent	Object	Identification		FunctionalGroupType	O
0:HasComponent	Object	Lock		LockingServicesType	O
0:HasComponent	Object	ParameterSet		0:BaseObjectType	O
0:HasComponent	Object	MethodSet		0:BaseObjectType	O
Conformance Units					
DI Information Model					

The *TopologyElementType* is abstract. There will be no instances of a *TopologyElementType* itself, but there will be instances of subtypes of this type. In this document, the term *TopologyElement* generically refers to an instance of any *ObjectType* derived from the *TopologyElementType*.

FunctionalGroups are an essential aspect introduced by the *TopologyElementType*. *FunctionalGroups* are used to structure *Nodes* like *Properties*, *Parameters* and *Methods* according to their application such as configuration, diagnostics, asset management, condition monitoring and others.

FunctionalGroups are specified in 4.4.

A *FunctionalGroup* called **Identification** can be used to organize identification information of this *TopologyElement* (see 4.4.2). Identification information typically includes the *Properties* defined by the *VendorNameplate* or *TagNameplate Interfaces* and additional application specific information.

TopologyElements may also support *LockingServices* (defined in 7).

Clients shall use the *LockingServices* if they need to make a set of changes (for example, several *Write* operations and *Method* invocations) and where a consistent state is available only after all of these changes have been performed. The main purpose of locking a *TopologyElement* is avoiding concurrent modifications.

The lock applies to the complete *TopologyElement* (including all components such as blocks or modules). *Servers* may expose a Lock Object on a component *TopologyElement* to allow independent locking of components, if no lock is applied to the top-level *TopologyElement*.

If the Online/Offline model is supported (see 6.3), the lock always applies to both the online and the offline version.

ParameterSet and *MethodSet* are defined as standard containers for systems that have a flat list of Parameters or Methods with unique names. In such cases, the *Parameters* are components of the “ParameterSet” as a flat list of *Parameters*. The *Methods* are kept the same way in the “MethodSet”.

The *MethodSet* is only available if it includes at least one *Method*.

The components of the *TopologyElementType* have additional references as defined in Table 13.

Table 13 – TopologyElementType Additional Subcomponents

Source Path	References	NodeClass	BrowseName	DataType	TypeDefinition	Other
ParameterSet	0:HasComponent	Variable	<ParameterIdentifier>	0:BaseDataType	0:BaseDataVariableType	MP

4.4 FunctionalGroupType

4.4.1 Model

This subtype of the OPC UA *FolderType* is used to structure *Nodes* like *Properties*, *Parameters* and *Methods* according to their application (e.g. maintenance, diagnostics, condition monitoring). *Organizes References* should be used when the elements are components in other parts of the *TopologyElement* that the *FunctionalGroup* belongs to. This includes *Properties*, *Variables*, and *Methods* of the *TopologyElement* or in *Objects* that are components of the *TopologyElement* either directly or via a subcomponent. The same *Property*, *Parameter* or *Method* might be useful in different application scenarios and therefore referenced from more than one *FunctionalGroup*.

FunctionalGroups can be nested.

FunctionalGroups can directly be instantiated. In this case, the *BrowseName* of a *FunctionalGroup* should indicate its purpose. A list of recommended *BrowseNames* is in 4.4.2.

Figure 3 shows the *FunctionalGroupType* components. It is formally defined in Table 14.

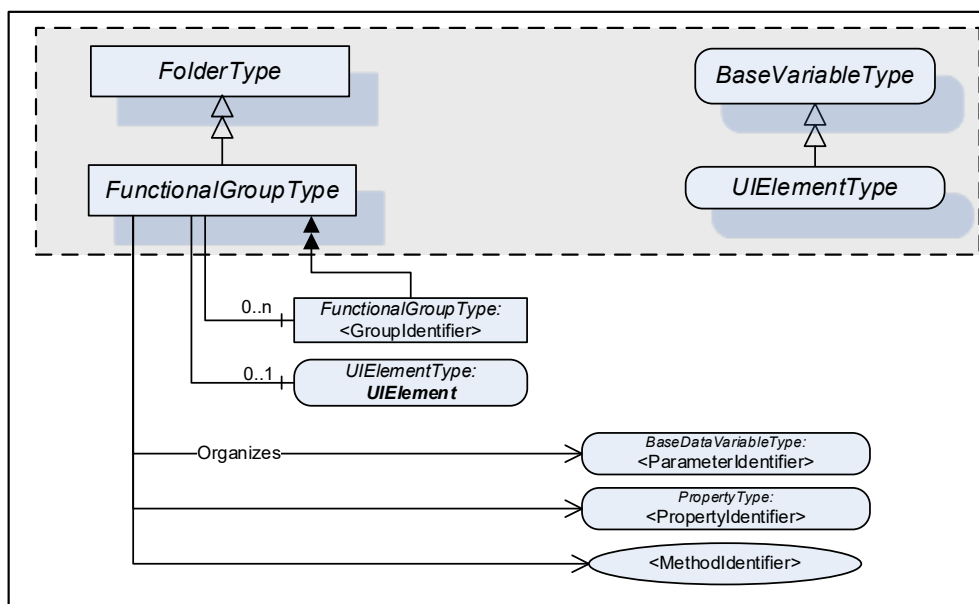


Figure 3 – FunctionalGroupType

Table 14 – FunctionalGroupType definition

Attribute	Value				
BrowseName	FunctionalGroupType				
IsAbstract	False				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
Subtype of the 0:FolderType defined in IEC 62541-5					
0:HasComponent	Object	<GroupIdentifier>		FunctionalGroupType	OP
0:HasComponent	Variable	UIElement	0:BaseDataType	UIElementType	O
Conformance Units					
DI Information Model					

810

811 All *BrowseNames* for *Nodes* referenced by a *FunctionalGroup* with an *Organizes Reference* shall be
 812 unique.

813 The *Organizes References* may be present only at the instance, not the type. Depending on the
 814 current state of the *TopologyElement* the *Server* may decide to hide or unhide certain
 815 *FunctionalGroups* or (part of) their *References*. If a *FunctionalGroup* may be hidden on an instance
 816 the *TypeDefinition* shall use an appropriate *ModellingRule* like "Optional".

817 If desirable, *Nodes* can be also children of *FunctionalGroups*. If such *Nodes* are defined, it is
 818 recommended to define a subtype of the *FunctionalGroupType*.

819 *UIElement* is the user interface element for this *FunctionalGroup*. See 4.4.3 for the definition of
 820 *UIElements*.

821 Examples in Annex B.1 illustrate the use of *FunctionalGroups*.

822 4.4.2 Recommended FunctionalGroup BrowseNames

823 Table 15 includes a list of *FunctionalGroups* with name and purpose. If *Servers* expose a
 824 *FunctionalGroup* that corresponds to the described purpose, they should use the recommended
 825 *BrowseName* with the Namespace of this document.

Table 15 – Recommended FunctionalGroup BrowseNames

BrowseName	Purpose
Configuration	<i>Parameters</i> representing the configuration items of the <i>TopologyElement</i> . If the <i>CurrentWrite</i> bit is set in the <i>AccessLevel Attribute</i> they can be modified by <i>Clients</i> .
Tuning	<i>Parameters</i> and <i>Methods</i> to optimize the behavior of the <i>TopologyElement</i> .
Maintenance	<i>Parameters</i> and <i>Methods</i> useful for maintenance operations.
Diagnostics	<i>Parameters</i> and <i>Methods</i> for diagnostics.
Statistics	<i>Parameters</i> and <i>Methods</i> for statistics.
Status	<i>Parameters</i> which describe the general health of the <i>TopologyElement</i> . This can include diagnostic <i>Parameters</i> .
Operational	<i>Parameters</i> and <i>Methods</i> useful for during normal operation, like process data.
OperationCounters	<i>Parameters</i> representing numbers of interest when managing a <i>TopologyElement</i> while it is operated. Examples are the hours of operation, hours in standby, etc. Those are often the base to calculate KPIs (key performance indicators) like the OEE (overall equipment efficiency). <i>Parameters</i> are often domain specific. Some common ones are defined in the OperationCounter Interface (see 4.5.5). This <i>FunctionalGroup</i> might be organized into other <i>FunctionalGroups</i>, so <i>Clients</i> shall expect that they need to browse several hops to get to all <i>OperationCounters</i>.
Identification	The <i>Properties</i> of the <i>VendorNameplate Interface</i> , like Manufacturer, SerialNumber or <i>Properties</i> of the TagNameplate will usually be sufficient as identification. If other <i>Parameters</i> or even <i>Methods</i> are required, all elements needed shall be organized in a <i>FunctionalGroup</i> called Identification . See Annex B.1 for an example.

4.4.3 UIElement Type

Servers can expose *UIElements* providing user interfaces in the context of their *FunctionalGroup* container. *Clients* can load such a user interface and display it on the *Client* side. The hierarchy of *FunctionalGroups* represents the tree of user interface elements.

The *UIElementType* is abstract and is mainly used as filter when browsing a *FunctionalGroup*. Only subtypes can be used for instances. No concrete *UIElements* are defined in this document. FDI (Field Device Integration, see IEC 62769) specifies two concrete subtypes

- UIDs (UI Descriptions), descriptive user interface elements, and
- UIPs (UI Plug-Ins), programmed user interface elements.

The *UIElementType* is specified in Table 16.

Table 16 – UIElementType definition

Attribute	Value				
BrowseName	UIElementType				
IsAbstract	True				
DataType	0:BaseDataType				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
Subtype of the <i>0:BaseDataVariableType</i> defined in IEC 62541-5.					
Conformance Units					
DI Information Model					

The *Value* attribute of the *UIElement* contains the user interface element. Subtypes have to define the *DataType* (e.g., *XmlElement* or *ByteString*).

4.5 Interfaces

4.5.1 Overview

This clause describes *Interfaces* with specific functionality that may be applied to multiple types at arbitrary positions in the type hierarchy.

Interfaces are defined in IEC 62541-3.

Figure 4 shows the *Interfaces* described in this document.

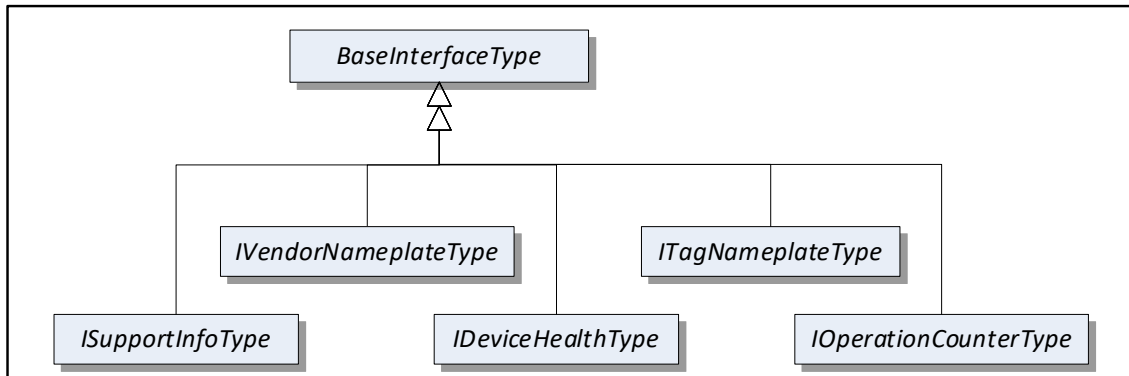


Figure 4 – Overview of Interfaces for Devices and Device components

4.5.2 VendorNameplate Interface

IVendorNameplateType includes *Properties* that are commonly used to describe a *TopologyElement* from a manufacturer point of view. They can be used as part of the identification. The *Values* of these *Properties* are typically provided by the component vendor.

The *VendorNameplate Interface* is illustrated in Figure 5 and formally defined in Table 17.

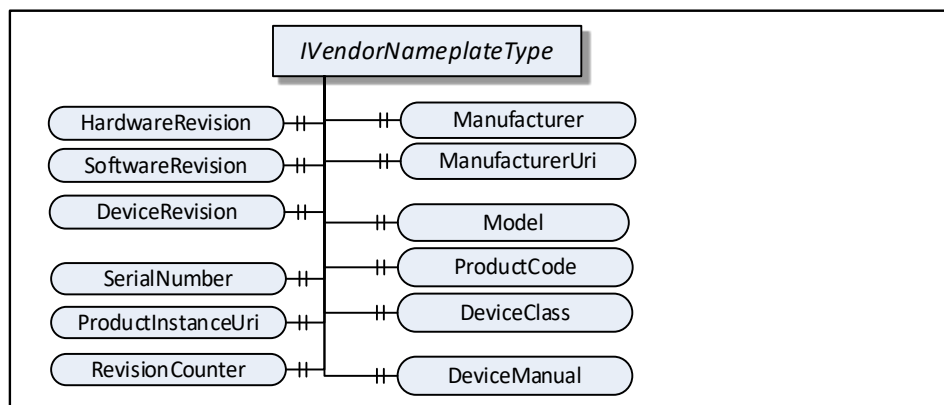


Figure 5 – VendorNameplate Interface

Table 17 – IVendorNameplateType definition

Attribute	Value				
BrowseName	IVendorNameplateType				
IsAbstract	True				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
Subtype of the <i>BaseInterfaceType</i> defined in IEC 62541-5					
Product-specific Properties					
0:HasProperty	Variable	Manufacturer	0:LocalizedText	0:PropertyType	O
0:HasProperty	Variable	ManufacturerUri	0:String	0:PropertyType	O
0:HasProperty	Variable	Model	0:LocalizedText	0:PropertyType	O
0:HasProperty	Variable	ProductCode	0:String	0:PropertyType	O
0:HasProperty	Variable	HardwareRevision	0:String	0:PropertyType	O
0:HasProperty	Variable	SoftwareRevision	0:String	0:PropertyType	O
0:HasProperty	Variable	DeviceRevision	0:String	0:PropertyType	O
0:HasProperty	Variable	DeviceManual	0:String	0:PropertyType	O
0:HasProperty	Variable	DeviceClass	0:String	0:PropertyType	O
Product instance-specific Properties					
0:HasProperty	Variable	SerialNumber	0:String	0:PropertyType	O
0:HasProperty	Variable	ProductInstanceUri	0:String	0:PropertyType	O
0:HasProperty	Variable	RevisionCounter	0:Int32	0:PropertyType	O
0:HasProperty	Variable	SoftwareReleaseDate	0:DateTime	0:PropertyType	O
0:HasProperty	Variable	PatchIdentifiers	0:String[]	0:PropertyType	O
Conformance Units					
DI Nameplate					

Product type specific *Properties*:

Manufacturer provides the name of the company that manufactured the item this *Interface* is applied to. *ManufacturerUri* provides a unique identifier for this company. This identifier should be a fully qualified domain name; however, it may be a GUID or similar construct that ensures global uniqueness.

Model provides the name of the product.

ProductCode provides a unique combination of numbers and letters used to identify the product. It may be the order information displayed on type shields or in ERP systems.

HardwareRevision provides the revision level of the hardware. *SemanticVersionString* (a sub-type of *String* defined in IEC 62541-5) may be used when using the Semantic Versioning format.

SoftwareRevision provides the version or revision level of the software component, the software/firmware of a hardware component, or the software/firmware of the *Device*. *SemanticVersionString* (a sub-type of *String* defined in IEC 62541-5) may be used when using the Semantic Versioning format.

DeviceRevision provides the overall revision level of a hardware component or the *Device*. As an example, this *Property* can be used in ERP systems together with the *ProductCode Property*. *SemanticVersionString* (a sub-type of *String* defined in IEC 62541-5) may be used when using the Semantic Versioning format.

DeviceManual allows specifying an address of the user manual. It may be a pathname in the file system or a URL (Web address).

DeviceClass indicates in which domain or for what purpose a certain item for which the *Interface* is applied is used. Examples are “ProgrammableController”, “RemoteIO”, and “TemperatureSensor”. This standard does not predefine any *DeviceClass* names. More specific standards that utilize this *Interface* will likely introduce such classifications (e.g., IEC 62769, and others).

Product instance specific *Properties*:

884 *SerialNumber* is a unique production number provided by the manufacturer. This is often stamped on
885 the outside of a physical component and may be used for traceability and warranty purposes.

886 *ProductInstanceUri* is a globally unique resource identifier provided by the manufacturer. This is often
887 stamped on the outside of a physical component and may be used for traceability and warranty
888 purposes. The maximum length is 255 characters. The recommended syntax of the
889 *ProductInstanceUri* is: <ManufacturerUri>/<any string> where <any string> is unique among all
890 instances using the same *ManufacturerUri*.

891 Examples: "some-company.com/5ff40f78-9210-494f-8206-c2c082f0609c", "some-company.com/snr-
892 16273849" or "some-company.com/model-xyz/snr-16273849".

893 *RevisionCounter* is an incremental counter indicating the number of times the configuration data has
894 been modified. An example would be a temperature sensor where the change of the unit would
895 increment the *RevisionCounter* but a change of the measurement value would not affect the
896 *RevisionCounter*.

897 *SoftwareReleaseDate* defines the date when the software is released. If the version information is
898 about patches, this should be the date of the latest patch. It is additional information for the user.

899 *PatchIdentifiers* identify the list of patches that are applied to a software version. The format and
900 semantics of the strings are vendor-specific. The order of the strings shall not be relevant.

901 Companion specifications may specify additional semantics for the contents of these *Properties*.

902 Table 18 specifies the mapping of these *Properties* to the International Registration Data Identifiers
903 (IRDI) defined in ISO/IEC 11179-6. They should be used if a *Server* wants to expose a dictionary
904 reference as defined in IEC 62541-19.

905 **Table 18 – VendorNameplate Mapping to IRDIs**

Property	IRDI
Manufacturer	0112/2///61987#ABA565#007
ManufacturerUri	0112/2///61987#ABN591#001
Model	0112/2///61987#ABA567#007
SerialNumber	0112/2///61987#ABA951#007
HardwareRevision	0112/2///61987#ABA926#006
SoftwareRevision	0112/2///61987#ABA601#006
DeviceRevision	-
RevisionCounter	0112/2///61987#ABN603#001
ProductCode	0112/2///61987#ABA300#006
ProductInstanceUri	0112/2///61987#ABN590#001
DeviceManual	-
DeviceClass	0112/2///61987#ABA566 - type of product

906
907
908

909 4.5.3 TagNameplate Interface

910 *ITagNameplateType* includes *Properties* that are commonly used to describe a *TopologyElement* from
911 a user point of view.

The *TagNameplate Interface* is illustrated in Figure 6 and formally defined in Table 19.

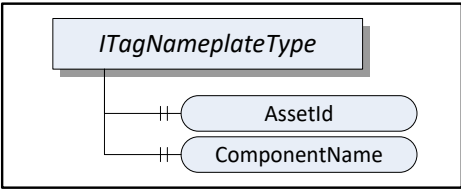


Figure 6 – TagNameplate Interface

Table 19 – ITagNameplateType definition

Attribute	Value				
BrowseName	ITagNameplateType				
IsAbstract	True				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
Subtype of the <i>BaseInterfaceType</i> defined in IEC 62541-5					
0:HasProperty	Variable	AssetId	0:String	0:PropertyType	O
0:HasProperty	Variable	ComponentName	0:LocalizedText	0:PropertyType	O
Conformance Units					
DI TagNameplate					

AssetId is a user writable alphanumeric character sequence uniquely identifying a component. The ID is provided by the integrator or user of the device. It contains typically an identifier in a branch, use case or user specific naming scheme. This could be for example a reference to an electric scheme.

ComponentName is a user writable name provided by the integrator or user of the component.

Table 20 specifies the mapping of these *Properties* to the International Registration Data Identifiers (IRDI) defined in ISO/IEC 11179-6. They should be used if a *Server* wants to expose a dictionary reference as defined in IEC 62541-19.

Table 20 – TagNameplate Mapping to IRDIs

Property	IRDI
AssetId	0112/2//61987#ABA038 - identification code of device
ComponentName	0112/2//61987#ABA251 - designation of device

4.5.4 DeviceHealth Interface

The *DeviceHealth Interface* includes *Properties* and *Alarms* that are commonly used to expose the health status of a *Device*. It is illustrated in Figure 7 and formally defined in Table 21.

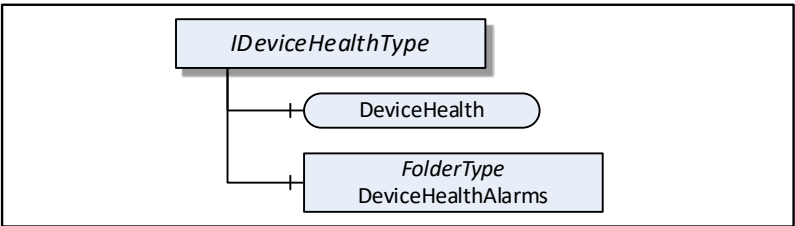


Figure 7 – DeviceHealth Interface

Table 21 – IDeviceHealthType definition

Attribute	Value				
BrowseName	IDeviceHealthType				
IsAbstract	True				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
Subtype of the 0:BaseInterfaceType defined in IEC 62541-5					
0:HasComponent	Variable	DeviceHealth	DeviceHealthEnumeration	0:BaseDataVariableType	O
0:HasComponent	Object	DeviceHealthAlarms		0:FolderType	O
Conformance Units					
DI DeviceHealth					

DeviceHealth indicates the status as defined by NAMUR Recommendation NE107. *Clients* can read or monitor this *Variable* to determine the device condition.

The *DeviceHealthEnumeration* *DataType* is an enumeration that defines the device condition. Its values are defined in Table 22. Its representation in the *AddressSpace* is defined in Table 23.

Table 22 – DeviceHealthEnumeration values

Name	Value	Description
NORMAL	0	The <i>Device</i> functions normally.
FAILURE	1	Malfunction of the <i>Device</i> or any of its peripherals. Typically caused device-internal or is process related.
CHECK_FUNCTION	2	Functional checks are currently performed. Examples: Change of configuration, local operation, and substitute value entered.
OFF_SPEC	3	"Off-spec" means that the <i>Device</i> is operating outside its specified range (e.g. measuring or temperature range) or that internal diagnoses indicate deviations from measured or set values due to internal problems in the <i>Device</i> or process characteristics.
MAINTENANCE_REQ UIRED	4	Although the output signal is valid, the wear reserve is nearly exhausted or a function will soon be restricted due to operational conditions e.g. build-up of deposits.

Table 23 – DeviceHealthEnumeration definition

Attribute	Value				
BrowseName	DeviceHealthEnumeration				
IsAbstract	False				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
Subtype of the 0:Enumeration type defined in IEC 62541-5					
0:HasProperty	Variable	0:EnumStrings	0:LocalizedText []	0:PropertyType	
Conformance Units					
DI DeviceHealth					

DeviceHealthAlarms shall be used for instances of the DeviceHealth Alarm Types specified in 4.12.

DeviceHealthAlarms may also be used for other *Alarm* instances that relate to the health condition of the *Device*.

4.5.5 OperationCounter Interface

The *IOperationCounterType* defines counters for the duration of operation. It is formally defined in Table 24.

Table 24 – IOperationCounterType definition

Attribute	Value				
BrowseName	IOperationCounterType				
IsAbstract	True				
Description	Interface defining counters for the duration of operation				
References	Node Class	BrowseName	DataType	TypeDefinition	Other
Subtype of the <i>0:BaseInterfaceType</i>					
0:HasProperty	Variable	PowerOnDuration	0:Duration	0:PropertyType	O
0:HasProperty	Variable	OperationDuration	0:Duration	0:PropertyType	O
0:HasProperty	Variable	OperationCycleCounter	0:UInteger	0:PropertyType	O
Conformance Units					
DI OperationCounter Interface					

PowerOnDuration is the duration the *Device* has been powered. The main purpose is to determine the time in which degradation of the *Device* occurred. The details, when the time is counted, is implementation-specific. Companion specifications might define specific rules. Typically, when the *Device* has supply voltage and the main CPU is running, the time is counted. This may include any kind of sleep mode, but may not include pure Wake on LAN. This value shall only increase during the lifetime of the *Device* and shall not be reset when the *Device* is restarted. The *PowerOnDuration* is provided as *Duration*, i.e., in milliseconds or even fractions of a millisecond. However, the *Server* is not expected to update the value in such a high frequency, but maybe once a minute or once an hour, depending on the application.

OperationDuration is the duration the *Device* has been powered and performing an activity. This counter is intended for *Devices* where a distinction is made between switched on and in operation. For example, a drive might be powered on but not operating. It is not intended for *Devices* always performing an activity like sensors always measuring data. This value shall only increase during the lifetime of the *Device* and shall not be reset when the *Device* is restarted. The *OperationDuration* is provided as *Duration*, i.e., in milliseconds or even fractions of a millisecond. However, the *Server* is not expected to update the value in such a high frequency, but maybe once a minute or once an hour, depending on the application.

OperationCycleCounter is counting the times the *Device* switches from not performing an activity to performing an activity. For example, each time a valve starts moving, is counted. This value shall only increase during the lifetime of the *Device* and shall not be reset when the *Device* is restarted.

The child *Nodes* of the *IOperationCounterType* have additional *Attribute* values defined in Table 25.

Table 25 – IOperationCounterType Attribute values for child Nodes

BrowsePath	Description Attribute
PowerOnDuration	PowerOnDuration is the duration the Device has been powered. The main purpose is to determine the time in which degradation of the Device occurred. The details, when the time is counted, is implementation-specific. Companion specifications might define specific rules. Typically, when the Device has supply voltage and the main CPU is running, the time is counted. This may include any kind of sleep mode, but may not include pure Wake on LAN. This value shall only increase during the lifetime of the Device and shall not be reset when the Device is restarted. The PowerOnDuration is provided as Duration, i.e., in milliseconds or even fractions of a millisecond. However, the Server is not expected to update the value in such a high frequency, but maybe once a minute or once an hour, depending on the application.
OperationDuration	OperationDuration is the duration the Device has been powered and performing an activity. This counter is intended for Devices where a distinction is made between switched on and in operation. For example, a drive might be powered on but not operating. It is not intended for Devices always performing an activity like sensors always measuring data. This value shall only increase during the lifetime of the Device and shall not be reset when the Device is restarted. The OperationDuration is provided as Duration, i.e., in milliseconds or even fractions of a millisecond. However, the Server is not expected to update the value in such a high frequency, but maybe once a minute or once an hour, depending on the application.
OperationCycleCounter	OperationCycleCounter is counting the times the Device switches from not performing an activity to performing an activity. For example, each time a valve starts moving, is counted. This value shall only increase during the lifetime of the Device and shall not be reset when the Device is restarted.

4.5.6 SupportInfo Interface

The *SupportInfo Interface* defines a number of additional data that a commonly exposed for *Devices* and their components. These include mainly images, documents, or protocol-specific data. The various types of information is organized into different folders. Each information element is represented by a readonly *Variable*. The information can be retrieved by reading the *Variable* value.

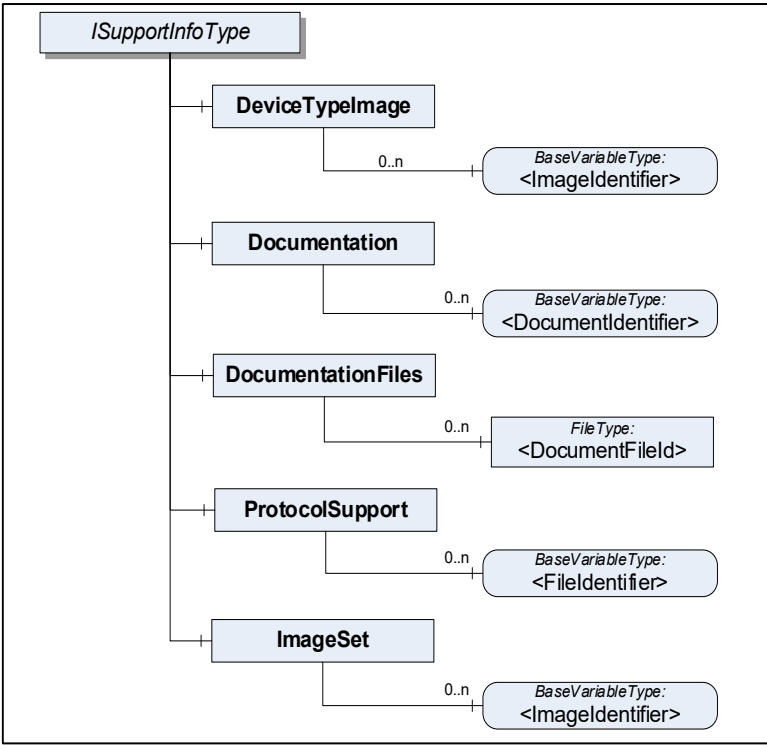


Figure 8 Illustrates the *SupportInfo Interface*. It is formally defined in Table 26.

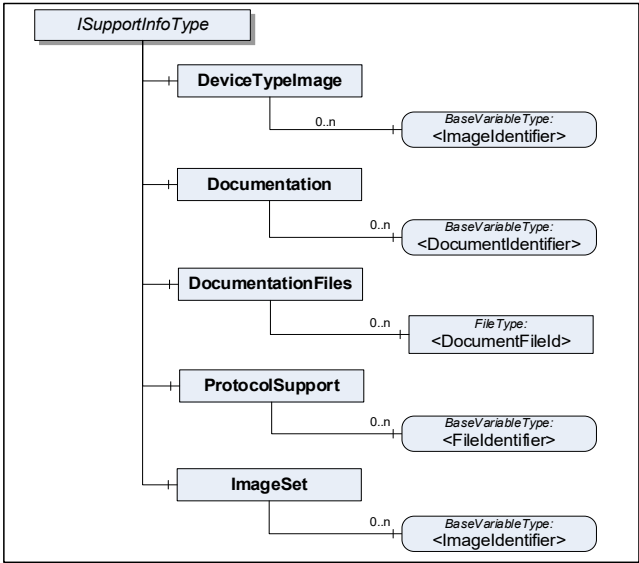


Figure 8 – Support information Interface

Table 26 – ISupportInfoType definition

Attribute	Value				
BrowseName	ISupportInfoType				
IsAbstract	True				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
Subtype of the <i>BaseInterfaceType</i> defined in IEC 62541-5					
0:HasComponent	Object	DeviceTypeImage		0:FolderType	O
0:HasComponent	Object	Documentation		0:FolderType	O
0:HasComponent	Object	DocumentationFiles		0:FolderType	O
0:HasComponent	Object	ProtocolSupport		0:FolderType	O
0:HasComponent	Object	ImageSet		0:FolderType	O
Conformance Units					
DI DeviceSupportInfo					

Clients need to be aware that the contents that these *Variables* represent may be large. Reading large values with a single Read operation may not be possible due to configured limits in either the *Client* or the *Server* stack. The default maximum size for an array of bytes is 1 megabyte. It is recommended that *Clients* use the *IndexRange* in the OPC UA Read Service (see IEC 62541-4) to read these *Variables* in chunks, for example, one-megabyte chunks. It is up to the *Client* whether it starts without an index and repeats with an *IndexRange* only after an error or whether it always uses an *IndexRange*.

The components of the *ISupportInfoType* have additional references as defined in Table 27.

Table 27 – ISupportInfoType Additional Subcomponents

Source Path	References	Node Class	BrowseName	Data Type	TypeDefinition	Other
DeviceTypeImage	0:HasComponent	Variable	<ImageIdentifier>	0:Image	0:BaseDataVariableType	MP
Documentation	0:HasComponent	Variable	<DocumentIdentifier>	0:ByteString	0:BaseDataVariableType	MP
DocumentationFiles	0:HasComponent	Object	<DocumentFileId>		0:FileType	MP
ProtocolSupport	0:HasComponent	Variable	<ProtocolSupportIdentifier>	0:ByteString	0:BaseDataVariableType	MP
ImageSet	0:HasComponent	Variable	<ImageIdentifier>	0:Image	0:BaseDataVariableType	MP

Pictures can be exposed as *Variables* organized in the *DeviceTypeImage* folder. There may be multiple images of different resolutions. Each image is a separate *Variable*.

All images are transferred as a *ByteString*. The *DataType* of the *Variable* specifies the image format. OPC UA defines BMP, GIF, JPG and PNG (see IEC 62541-3).

Documents in many cases will represent a product manual. They may be exposed as *Variables* or as *FileType* instances. Files are useful in particular for large documents.

- Documents as *Variables* are represented as a *ByteString* and organized in the *Documentation* folder. The *BrowseName* of each *Node* will consist of the filename including the extension that can be used to identify the document type. Typical extensions are “.pdf” or “.txt”.
- Documents as *FileType* instances are organized in the *DocumentationFiles* folder. They are retrieved from the *Server* by using the *FileType Methods*. It is recommended to use the *MimeType Property* to specify the media type of the file based on RFC 2046.

Protocol support files are exposed as *Variables* organized in the *ProtocolSupport* folder. They may represent various types of information as defined by a protocol. Examples are a GSD or a CFF file.

All protocol support files are transferred as a *ByteString*. The *BrowseName* of each *Variable* shall consist of the complete filename including the extension that can be used to identify the type of information.

1012 Images that are used within *UIElements* are exposed as separate *Variables* rather than embedding
 1013 them in the element. All image *Variables* will be aggregated by the *ImageSet* folder. The *UIElement*
 1014 shall specify an image by its name that is also the *BrowseName* of the image *Variable*. *Clients* can
 1015 cache images so they don't have to be transferred more than once.

1016 The *DataType* of the *Variable* specifies the image format. OPC UA defines BMP, GIF, JPG and PNG
 1017 (see IEC 62541-3).

1018 4.6 ComponentType

1019 Compared to *DeviceType* the *ComponentType* is more universal. It includes the same components
 1020 but does not mandate any *Properties*. This makes it usable for representation of a *Device* or parts of
 1021 a *Device*. Parts include both mechanical and software parts.

1022 The *ComponentType* applies the *VendorNameplate* and the *TagNameplate Interface*. Figure 9
 1023 illustrates the *ComponentType*. It is formally defined in Table 28.

1024

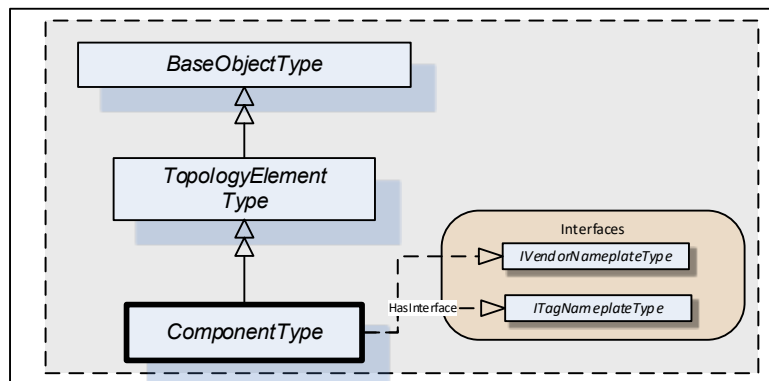


Figure 9 – ComponentType

1025

1026

1027

Table 28 – ComponentType definition

Attribute	Value				
BrowseName	ComponentType				
IsAbstract	True				
References	NodeClass	BrowseName	Data Type	Type Definition	Other
Subtype of the <i>TopologyElementType</i> defined in 4.3.					
HasSubtype	ObjectType	DeviceType	Defined in 4.7.		
HasSubtype	ObjectType	SoftwareType	Defined in 4.8.		
HasInterface	ObjectType	IVendorNameplateType	Defined in 4.5.2.		
HasInterface	ObjectType	ITagNameplateType	Defined in 4.5.3.		
Applied from <i>IVendorNameplateType</i>					
0:HasProperty	Variable	Manufacturer	0:LocalizedText	0:PropertyType	O
0:HasProperty	Variable	ManufacturerUri	0:String	0:PropertyType	O
0:HasProperty	Variable	Model	0:LocalizedText	0:PropertyType	O
0:HasProperty	Variable	ProductCode	0:String	0:PropertyType	O
0:HasProperty	Variable	HardwareRevision	0:String	0:PropertyType	O
0:HasProperty	Variable	SoftwareRevision	0:String	0:PropertyType	O
0:HasProperty	Variable	DeviceRevision	0:String	0:PropertyType	O
0:HasProperty	Variable	DeviceManual	0:String	0:PropertyType	O
0:HasProperty	Variable	DeviceClass	0:String	0:PropertyType	O
0:HasProperty	Variable	SerialNumber	0:String	0:PropertyType	O
0:HasProperty	Variable	ProductInstanceUri	0:String	0:PropertyType	O
0:HasProperty	Variable	RevisionCounter	0:Int32	0:PropertyType	O
Applied from <i>ITagNameplateType</i>					
0:HasProperty	Variable	AssetId	0:String	0:PropertyType	O
0:HasProperty	Variable	ComponentName	0:LocalizedText	0:PropertyType	O
Conformance Units					
DI Information Model					

The *ComponentType* is abstract. *DeviceType* and *SoftwareType* are subtypes of *ComponentType*. There will be no instances of a *ComponentType* itself, only of concrete subtypes.

IVendorNameplateType and its members are described in 4.5.2.

ITagNameplateType and its members are described in 4.5.3.

4.7 DeviceType

This *ObjectType* can be used to define the structure of a *Device*. Figure 10 shows the *DeviceType*. It is formally defined in Table 29.

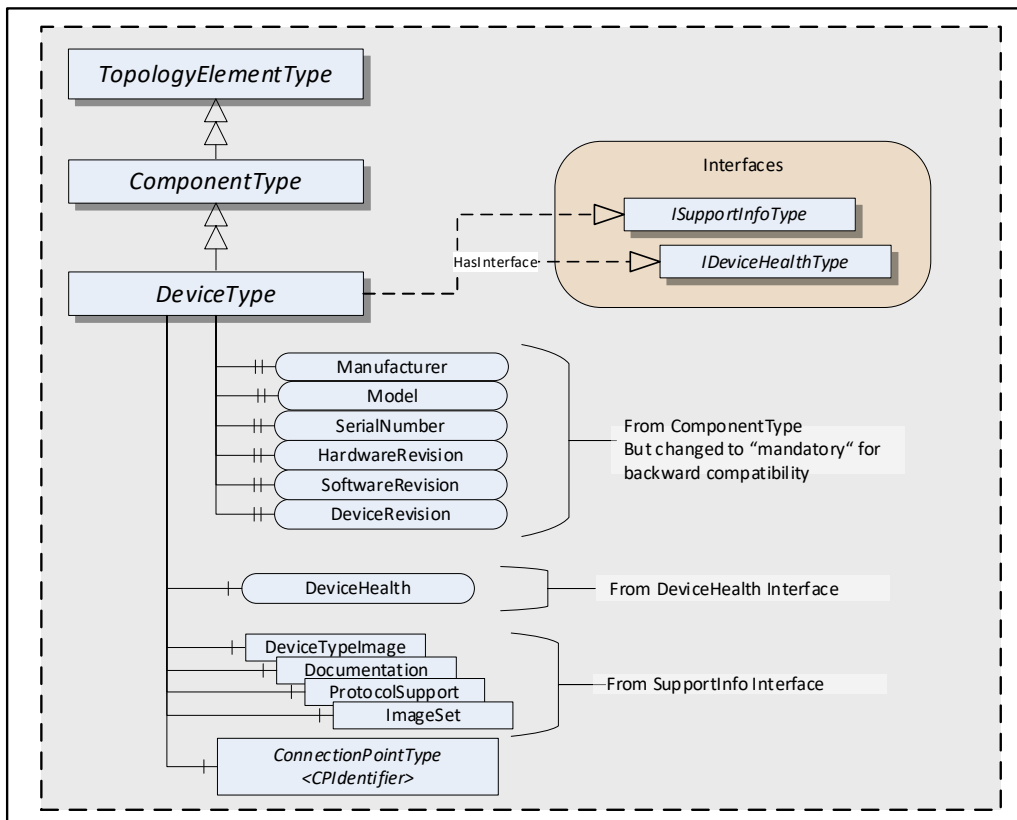


Figure 10 – DeviceType

Table 29 – DeviceType definition

Attribute	Value				
BrowseName	DeviceType				
IsAbstract	True				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
Subtype of the <i>ComponentType</i> defined in 4.6					
HasInterface	ObjectType	ISupportInfoType		Defined in 4.5.3.	
HasInterface	ObjectType	IDeviceHealthType		Defined in 4.5.3.	
0:HasComponent	Object	<CPIdentifier>		ConnectionPointType	OP
0:HasProperty	Variable	SerialNumber	0:String	0:PropertyType	M
0:HasProperty	Variable	RevisionCounter	0:Int32	0:PropertyType	M
0:HasProperty	Variable	Manufacturer	0:LocalizedText	0:PropertyType	M
0:HasProperty	Variable	Model	0:LocalizedText	0:PropertyType	M
0:HasProperty	Variable	DeviceManual	0:String	0:PropertyType	M
0:HasProperty	Variable	DeviceRevision	0:String	0:PropertyType	M
0:HasProperty	Variable	SoftwareRevision	0:String	0:PropertyType	M
0:HasProperty	Variable	HardwareRevision	0:String	0:PropertyType	M
0:HasProperty	Variable	DeviceClass	0:String	0:PropertyType	O
0:HasProperty	Variable	ManufacturerUri	0:String	0:PropertyType	O
0:HasProperty	Variable	ProductCode	0:String	0:PropertyType	O
0:HasProperty	Variable	ProductInstanceUri	0:String	0:PropertyType	O

Applied from IDeviceHealthType					
0:HasComponent	Variable	DeviceHealth	DeviceHealthEnumeration	0:BaseDataVariableType	O
0:HasComponent	Object	DeviceHealthAlarms		0:FolderType	O
Applied from ISupportInfoType					
0:HasComponent	Object	DeviceTypeImage		0:FolderType	O
0:HasComponent	Object	Documentation		0:FolderType	O
0:HasComponent	Object	ProtocolSupport		0:FolderType	O
0:HasComponent	Object	ImageSet		0:FolderType	O
Conformance Units					
DI DeviceType					

1042

1043 *DeviceType* is a subtype of *ComponentType* which means it inherits all *InstanceDeclarations*.

1044 The *DeviceType ObjectType* is abstract. There will be no instances of a *DeviceType* itself, only of
1045 concrete subtypes.

1046 *ConnectionPoints* (see 5.4) represent the interface (interface card) of a *DeviceType* instance to a
1047 *Network*. Multiple *ConnectionPoints* may exist if multiple protocols and/or multiple *Communication*
1048 *Profiles* are supported.

1049 The *Interfaces* and their members are described in 4.5. Some of the *Properties* inherited from the
1050 *ComponentType* are declared mandatory for backward compatibility.

1051 Although mandatory, some of the *Properties* may not be supported for certain types of *Devices*. In
1052 this case vendors shall provide the following defaults:

- 1053 • *Properties* with *DataType String*: **empty string**
- 1054 • *Properties* with *DataType LocalizedText*: **empty text field**
- 1055 • *RevisionCounter Property*: **- 1**

1056 *Clients* can ignore the *Properties* when they have these defaults.

1057 When *Properties* are not supported, *Servers* should initialize the corresponding *Property* declaration
1058 on the *DeviceType* with the default value. Relevant *Browse Service* requests can then return a
1059 *Reference* to this *Property* on the type definition. That way, no extra *Nodes* are needed.

1060

4.8 SoftwareType

This *ObjectType* can be used for software modules of a *Device* or a part of a *Device*. *SoftwareType* is a concrete subtype of *ComponentType* and can be used directly.

Figure 11 Illustrates the *SoftwareType*. It is formally defined in Table 30.

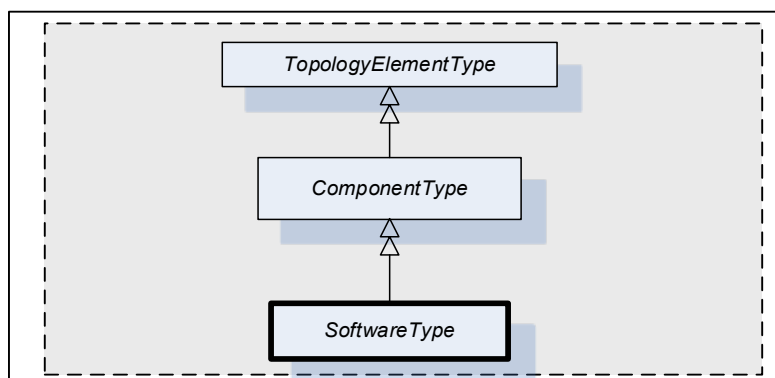


Figure 11 – SoftwareType

Table 30 – SoftwareType definition

Attribute	Value				
BrowseName	SoftwareType				
IsAbstract	False				
References	NodeClass	BrowseName	DataType	TypeDefinition	Other
Subtype of the <i>ComponentType</i> defined in 4.6.					
0:HasProperty	Variable	Manufacturer	0:LocalizedText	0:PropertyType	M
0:HasProperty	Variable	Model	0:LocalizedText	0:PropertyType	M
0:HasProperty	Variable	SoftwareRevision	0:String	0:PropertyType	M
Conformance Units					
DI Software Component					

SoftwareType is a subtype of *ComponentType* which means it inherits all *InstanceDeclarations*.

The *Properties Manufacturer*, *Model*, and *SoftwareRevision* inherited from *ComponentType* are declared mandatory for *SoftwareType* instances.

4.9 DeviceSet entry point

The *DeviceSet Object* is the starting point to locate *Devices*. It shall either directly or indirectly reference all instances of a subtype of *ComponentType* with a *Hierarchical Reference*. For complex *Devices* that are composed of various components that are also *Devices*, only the root instance shall be referenced from the *DeviceSet Object*. The components of such complex *Devices* shall be locatable by following *Hierarchical References* from the root instance. An example is the *Modular Device* defined in 9.4 and also illustrated in Figure 12.

Examples:

- UA Server represents a monolithic or modular *Device*: *DeviceSet* only contains one instance
- UA Server represents a host system that has access to a number of *Devices* that it manages: *DeviceSet* contains several instances that the host provides access to.
- UA Server represents a gateway *Device* that acts as representative for *Devices* that it has access to: *DeviceSet* contains the gateway *Device* instance and instances for the *Devices* that it represents.

- *UA Server* represents a robotic system consisting of mechanics and controls. *DeviceSet* only contains the instance for the root of the robotic system. The mechanics and controls are represented by *ComponentType* instances which are organized as sub-components of the root instance.

Figure 12 shows the *AddressSpace* organisation with this standard entry point and examples.

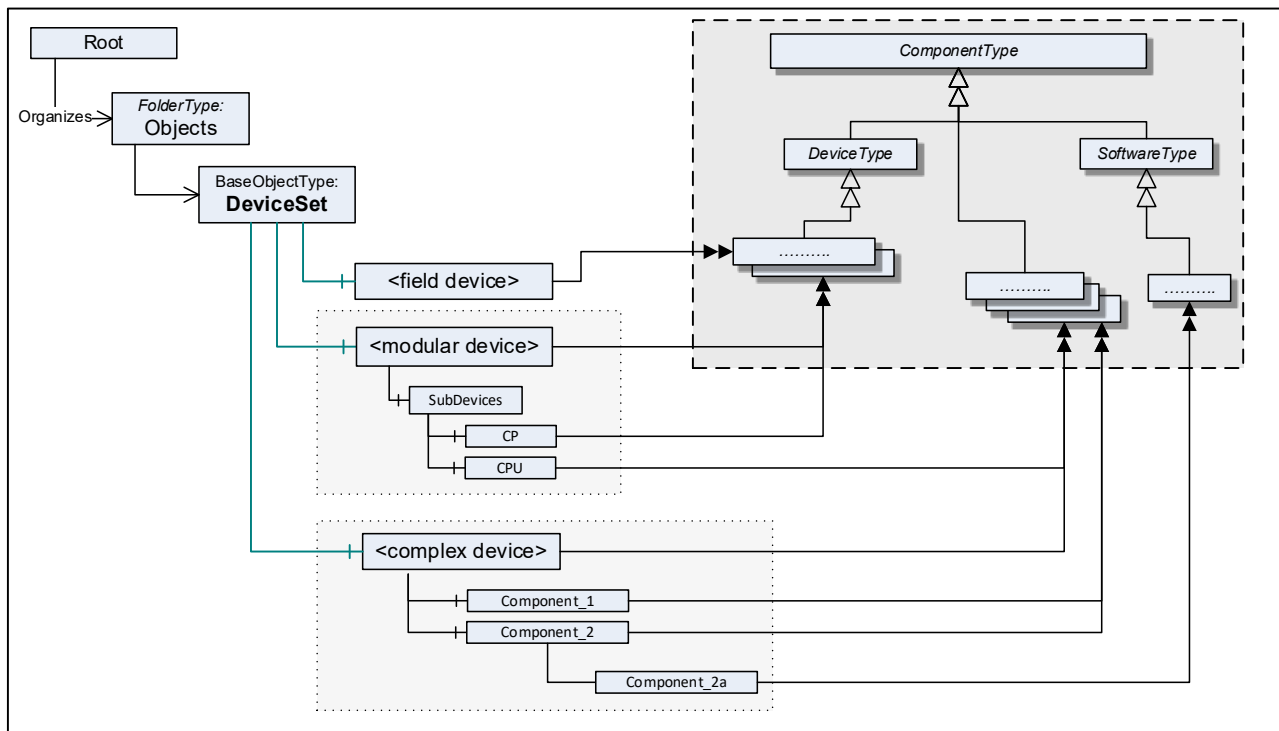


Figure 12 – Standard entry point for Devices

The *DeviceSet Node* is formally defined in Table 31.

Table 31 – DeviceSet definition

Attribute		Value	
BrowseName		DeviceSet	
References	NodeClass	BrowseName	TypeDefinition
OrganizedBy by the 0:Objects Folder defined in IEC 62541-5			
0:HasTypeDefinition	ObjectType	0:BaseObjectType	
Conformance Units			
DI DeviceSet			

4.10 DeviceFeatures entry point

The *DeviceFeatures Object* can be used to organize other functional entities that are related to the *Devices* referenced by the *DeviceSet*. Companion specifications may standardize such instances and their *BrowseNames*. Figure 13 shows the *AddressSpace* organisation with this standard entry point.

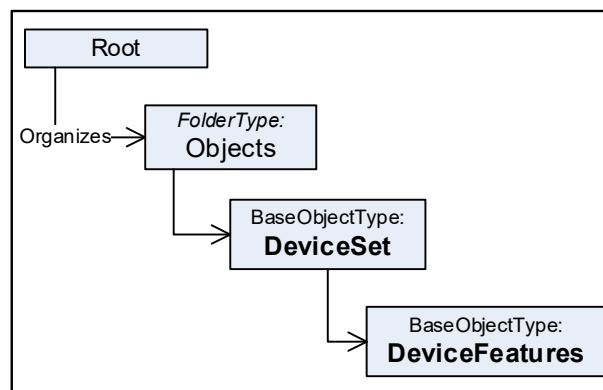


Figure 13 – Standard entry point for DeviceFeatures

The *DeviceFeatures Node* is formally defined in Table 32.

Table 32 – DeviceFeatures definition

Attribute	Value		
BrowseName	DeviceFeatures		
References	NodeClass	BrowseName	TypeDefinition
OrganizedBy by the DeviceSet Object defined in 4.9			
0:HasTypeDefinition	ObjectType	0:BaseObjectType	
Conformance Units			
DI DeviceSet			

4.11 BlockType

This *ObjectType* defines the structure of a *Block Object*. Figure 14 depicts the *BlockType* hierarchy. It is formally defined in Table 33.

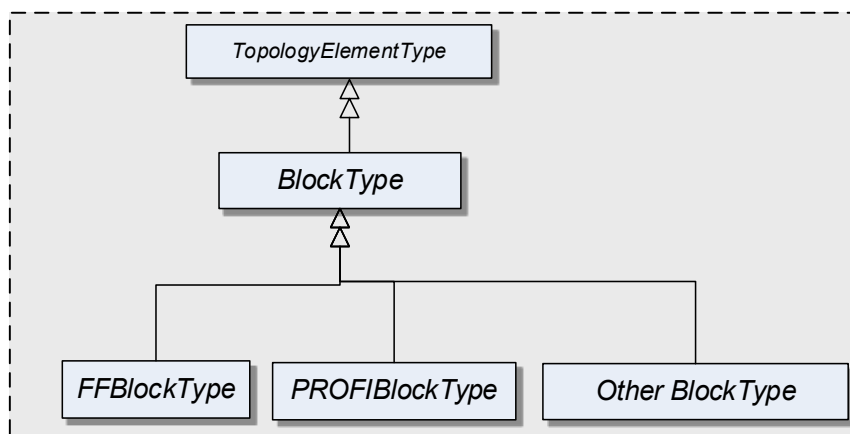


Figure 14 – BlockType hierarchy

FFBlockType and PROFIBlockType are examples. They are not further defined in this document. It is expected that industry groups will standardize general purpose *BlockTypes*.