



DPC: 24 / 30484408 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. 2023/02969

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

Interested committees:

Title: BS EN IEC 62541-14 OPC Unified Architecture

Part 14: Pubsub

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	Comment (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.	



COMMITTEE DRAFT FOR VOTE (CDV)

PROJECT NUMBER:

IEC 62541-14 ED2

DATE OF CIRCULATION:

2024-01-26

CLOSING DATE FOR VOTING:

2024-04-19

SUPERSEDES DOCUMENTS:

65E/974/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

☐ NOT 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.

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 14: PubSub

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.

A.1 CONTENTS

FIGURES	iii
TABLES	v
FOREWORD	xiv
1 Scope	1
2 Normative references	1
3 Terms, definitions and abbreviated terms	2
3.1 Terms and definitions	2
3.2 Abbreviated terms	2
4 Overview	3
4.1 Fields of application.....	3
4.2 Abstraction layers.....	3
4.3 Decoupling by use of middleware.....	4
4.4 Synergy of models.....	5
5 PubSub Concepts	5
5.1 General.....	5
5.2 DataSet.....	6
5.2.1 General.....	6
5.2.2 DataSetClass	7
5.2.3 DataSetMetaData	7
5.3 Messages.....	8
5.3.1 General.....	8
5.3.2 DataSetMessage field.....	9
5.3.3 DataSetMessage	9
5.3.4 NetworkMessage	10
5.3.5 Message security	10
5.3.6 Transport security	11
5.3.7 SecurityGroup	11
5.3.8 Topics.....	11
5.4 Entities.....	11
5.4.1 Publisher.....	11
5.4.2 Subscriber.....	14
5.4.3 Configuration Tool	15
5.4.4 Security Key Service	16
5.4.5 Message Oriented Middleware.....	19
6 PubSub communication parameters.....	24
6.1 Overview.....	24
6.2 Common configuration parameters.....	25
6.2.1 PubSubState state machine.....	25
6.2.2 PubSub configuration properties	27
6.2.3 PublishedDataSet parameters.....	27
6.2.4 DataSetWriter parameters	37
6.2.5 Shared PubSubGroup parameters	41
6.2.6 WriterGroup parameters	43
6.2.7 PubSubConnection parameters.....	46
6.2.8 ReaderGroup parameters	49
6.2.9 DataSetReader parameters	50

48	6.2.10	SubscribedDataSet parameters	54
49	6.2.11	Information flow and status handling	59
50	6.2.12	PubSubConfiguration.....	60
51	6.3	Message mapping configuration parameters.....	63
52	6.3.1	UADP message mapping	63
53	6.3.2	JSON message mapping	72
54	6.4	Transport Protocol mapping configuration parameters	75
55	6.4.1	Datagram Transport Protocol.....	75
56	6.4.2	Broker Transport Protocol.....	82
57	7	PubSub mappings.....	88
58	7.1	General.....	88
59	7.2	Message mappings.....	88
60	7.2.1	General.....	88
61	7.2.2	MessageTypes	88
62	7.2.3	SequenceNumber in headers.....	89
63	7.2.4	UADP message mapping	89
64	7.2.5	JSON message mapping	108
65	7.3	Transport Protocol Mappings	114
66	7.3.1	General.....	114
67	7.3.2	OPC UA UDP	114
68	7.3.3	OPC UA Ethernet	116
69	7.3.4	AMQP	117
70	7.3.5	MQTT	121
71	8	PubSub Security Key Service model.....	128
72	8.1	Overview.....	128
73	8.2	PublishSubscribe Object.....	128
74	8.3	PubSubKeyServiceType	129
75	8.3.1	PubSubKeyServiceType definition	129
76	8.3.2	GetSecurityKeys Method	129
77	8.3.3	GetSecurityGroup Method	132
78	8.4	SecurityGroupType.....	132
79	8.4.1	SecurityGroupType definition.....	132
80	8.4.2	InvalidateKeys Method	133
81	8.4.3	ForceKeyRotation Method	134
82	8.5	SecurityGroupFolderType	134
83	8.5.1	SecurityGroupFolderType definition	134
84	8.5.2	AddSecurityGroup Method	135
85	8.5.3	RemoveSecurityGroup Method	136
86	8.5.4	AddSecurityGroupFolder Method	137
87	8.5.5	RemoveSecurityGroupFolder Method.....	137
88	8.6	PubSubKeyPushTargetType	138
89	8.6.1	PubSubKeyPushTargetType definition	138
90	8.6.2	Behaviour.....	139
91	8.6.3	ConnectSecurityGroups.....	139
92	8.6.4	DisconnectSecurityGroups Method	140
93	8.6.5	TriggerKeyUpdate Method	141
94	8.6.6	HasPushedSecurityGroup.....	141
95	8.7	PubSubKeyPushTargetFolderType.....	142
96	8.7.1	PubSubKeyPushTargetFolderType definition.....	142

97	8.7.2	AddPushTarget Method	142
98	8.7.3	RemovePushTarget Method.....	143
99	8.7.4	AddPushTargetFolder Method	144
100	8.7.5	RemovePushTargetFolder Method.....	144
101	8.8	Security Key Service Roles.....	145
102	9	PubSub configuration model.....	146
103	9.1	Common configuration model.....	146
104	9.1.1	General.....	146
105	9.1.2	Configuration behaviours.....	148
106	9.1.3	Types for the PublishSubscribe Object.....	149
107	9.1.4	Published DataSet model	161
108	9.1.5	Connection model	178
109	9.1.6	Group model	183
110	9.1.7	DataSetWriter model	191
111	9.1.8	DataSetReader model	193
112	9.1.9	Subscribed DataSet model	198
113	9.1.10	PubSub Status Object	204
114	9.1.11	PubSub Diagnostics Objects.....	206
115	9.1.12	PubSub Capabilities	214
116	9.1.13	PubSub Status Events.....	216
117	9.2	Message Mapping configuration model.....	217
118	9.2.1	UADP Message mapping	217
119	9.2.2	JSON Message mapping	219
120	9.3	Transport Protocol Mapping configuration model	220
121	9.3.1	Datagram Transport Protocol mapping	220
122	9.3.2	Broker Transport Protocol mapping.....	222
123	Annex A (normative) Header Layouts.....		225
124	A.1	General.....	225
125	A.2	UADP Header Layouts.....	225
126	A.2.1	Message headers for periodic data with fixed layout.....	225
127	A.2.2	Message headers for Events and Data with dynamic layout.....	231
128	A.3	JSON Header Layouts	235
129	A.3.1	DataSets for examples	235
130	A.3.2	JSON message headers for minimal messages	238
131	A.3.3	JSON message headers for single DataSetMessage	240
132	A.3.4	JSON message headers for multiple DataSetMessages.....	242
133	Annex B (informative) Client Server vs. Publish Subscribe		245
134	B.1	Overview	245
135	B.2	Client Server Subscriptions.....	245
136	B.3	Publish-Subscribe	246
137	B.4	Synergy of models.....	247

FIGURES

141	Figure 1 – Publish Subscribe model overview	4
142	Figure 2 – Publisher and Subscriber entities	6
143	Figure 3 – DataSet in the process of publishing	7
144	Figure 4 – OPC UA PubSub message layers.....	9

145	Figure 5 – Publisher details	12
146	Figure 6 – Publisher message sending sequence	13
147	Figure 7 – Subscriber details	14
148	Figure 8 – Subscriber message reception sequence	15
149	Figure 9 – SecurityGroup management sequence	17
150	Figure 10 – Handshake used to pull keys from SKS	17
151	Figure 11 – Handshake used to push keys to Publishers and Subscribers	18
152	Figure 12 – Handshake with a Security Key Service	18
153	Figure 13 – PubSub using network infrastructure	19
154	Figure 14 – UDP Multicast overview	20
155	Figure 15 – PubSub using broker	21
156	Figure 16 – Broker overview	22
157	Figure 17 – Message Oriented Middleware providing QoS	23
158	Figure 18 – Mapping of priority-based QoS	23
159	Figure 19 – PubSub component overview	24
160	Figure 20 – PubSub mapping specific parameters overview	25
161	Figure 21 – PubSub component state dependencies	26
162	Figure 22 – PubSubState state machine	26
163	Figure 23 – PubSub information flow dependency to field representation	38
164	Figure 24 – PubSub information flow	59
165	Figure 25 – Start of the periodic publisher execution	64
166	Figure 26 – Timing offsets in a PublishingInterval	64
167	Figure 27 – DataSetOrdering and MaxNetworkMessageSize	65
168	Figure 28 – PublishingOffset options for multiple <i>NetworkMessages</i>	67
169	Figure 29 – UADP NetworkMessage	91
170	Figure 30 – UADP DataSet payload	96
171	Figure 31 – DataSetMessage header structure	97
172	Figure 32 – Data Key Frame DataSetMessage data	99
173	Figure 33 – Data Delta Frame DataSetMessage	100
174	Figure 34 – Event DataSetMessage	101
175	Figure 35 – KeepAlive message	102
176	Figure 36 – PublishSubscribe Object Types overview	128
177	Figure 37 – PubSub configuration model overview	146
178	Figure 38 – PubSub example Objects	147
179	Figure 39 – PubSub information flow	147
180	Figure 40 – PublishSubscribe Object Types overview	149
181	Figure 41 – Published DataSet overview	162
182	Figure 42 – PubSubConnectionType overview	178
183	Figure 43 – PubSubGroupType overview	184
184	Figure 44 – DataSet Writer model overview	192
185	Figure 45 – DataSet Reader model overview	193
186	Figure 46 – PubSub Diagnostics overview	206
187	Figure 47 – PubSubDiagnosticsCounterType	207

188	Figure A.1 – UADP NetworkMessage header layout	226
189	Figure A.2 – UADP NetworkMessage header layout with integrity (signing).....	227
190	Figure A.3 – UADP NetworkMessage header layout with integrity and confidentiality	228
191	Figure A.4 – UADP DataSetMessage header layout	229
192	Figure A.5 – Example for fixed message layout without security	230
193	Figure A.6 – Example for fixed message layout without signature	231
194	Figure A.7 – UADP NetworkMessage header layout	232
195	Figure A.8 – UADP NetworkMessage header layout with integrity (signing).....	232
196	Figure A.9 – UADP NetworkMessage header layout with integrity and confidentiality	233
197	Figure A.10 – UADP DataSetMessage header layout	234
198	Figure A.11 – Example for dynamic message layout without security	235
199	Figure B.1 – Subscriptions in OPC UA Client Server model	246
200	Figure B.2 – Publish Subscribe model overview	247

201

202

203

TABLES

204	Table 1 – PubSubState values.....	25
205	Table 2 – PubSubState state machine	26
206	Table 3 – PubSubState definition.....	27
207	Table 4 – General PubSub configuration properties	27
208	Table 5 – DataSetMetaData type structure	28
209	Table 6 – DataSetMetaData type definition	28
210	Table 7 – FieldMetaData structure	29
211	Table 8 – FieldMetaData definition	30
212	Table 9 – DataSetFieldFlags Values	31
213	Table 10 – DataSetFieldFlags definition.....	31
214	Table 11 – ConfigurationVersionDataType structure.....	31
215	Table 12 – ConfigurationVersionDataType definition	32
216	Table 13 – PublishedDataSetDataType structure	32
217	Table 14 – PublishedDataSetDataType definition.....	32
218	Table 15 – PublishedDataSetSourceDataType definition	33
219	Table 16 – PublishedVariableDataType structure	34
220	Table 17 – PublishedVariableDataType definition.....	34
221	Table 18 – PublishedDataItemsDataType structure	35
222	Table 19 – PublishedDataItemsDataType definition.....	35
223	Table 20 – PublishedEventsDataType structure	36
224	Table 21 – PublishedEventsDataType definition.....	36
225	Table 22 – PublishedDataSetCustomSourceDataType structure	36
226	Table 23 – PublishedDataSetCustomSourceDataType definition.....	36
227	Table 24 – DataSetFieldContentMask Values	37
228	Table 25 – DataSetFieldContentMask definition	38
229	Table 26 – DataSetMessage field representation options	39
230	Table 27 – DataSetWriterDataType structure	40

231	Table 28 – DataSetWriterDataType definition.....	40
232	Table 29 – DataSetWriterTransportDataType definition	40
233	Table 30 – DataSetWriterMessageDataType definition	41
234	Table 31 – SecurityKeyService parameter content	42
235	Table 32 – PubSubGroupDataType structure	43
236	Table 33 – PubSubGroupDataType definition.....	43
237	Table 34 – WriterGroupDataType structure	45
238	Table 35 – WriterGroupDataType definition	45
239	Table 36 – WriterGroupTransportDataType definition	45
240	Table 37 – WriterGroupMessageDataType definition	46
241	Table 38 – ConnectionProperties.....	47
242	Table 39 – PubSubConnectionDataType structure	47
243	Table 40 – PubSubConnectionDataType definition	47
244	Table 41 – ConnectionTransportDataType definition	48
245	Table 42 – NetworkAddressDataType structure.....	48
246	Table 43 – NetworkAddressDataType definition	48
247	Table 44 – NetworkAddressUrlDataType structure	48
248	Table 45 – NetworkAddressUrlDataType definition.....	48
249	Table 46 – ReaderGroupDataType structure	49
250	Table 47 – ReaderGroupDataType definition.....	49
251	Table 48 – ReaderGroupTransportDataType definition	50
252	Table 49 – ReaderGroupMessageDataType definition	50
253	Table 50 – DataSetReaderDataType structure	53
254	Table 51 – DataSetReaderDataType definition.....	54
255	Table 52 – DataSetReaderTransportDataType structure.....	54
256	Table 53 – DataSetReaderTransportDataType definition	54
257	Table 54 – DataSetReaderMessageDataType structure.....	54
258	Table 55 – DataSetReaderMessageDataType definition	54
259	Table 56 – SubscribedDataSetDataType structure	55
260	Table 57 – SubscribedDataSetDataType definition.....	55
261	Table 58 – TargetVariablesDataType structure	55
262	Table 59 – TargetVariablesDataType definition.....	55
263	Table 60 – FieldTargetDataType structure	56
264	Table 61 – FieldTargetDataType definition.....	56
265	Table 62 – OverrideValueHandling values	57
266	Table 63 – OverrideValueHandling definition	57
267	Table 64 – SubscribedDataSetMirrorDataType structure	57
268	Table 65 – SubscribedDataSetMirrorDataType definition.....	57
269	Table 66 – StandaloneSubscribedDataSetRefDataType structure	58
270	Table 67 – StandaloneSubscribedDataSetRefDataType definition	58
271	Table 68 – StandaloneSubscribedDataSetDataType structure	58
272	Table 69 – StandaloneSubscribedDataSetDataType definition.....	59
273	Table 70 – Source to message input mapping.....	59

274	Table 71 – Message output to target mapping.....	60
275	Table 72 – PubSubConfigurationDataType structure	60
276	Table 73 – PubSubConfigurationDataType definition	60
277	Table 74 – PubSubConfiguration file content	61
278	Table 75 – SecurityGroupDataType structure	61
279	Table 76 – SecurityGroupDataType definition	62
280	Table 77 – PubSubKeyPushTargetDataType structure	62
281	Table 78 – PubSubKeyPushTargetDataType definition	62
282	Table 79 – PubSubConfiguration2DataType structure	63
283	Table 80 – PubSubConfiguration2DataType definition	63
284	Table 81 – DataSetOrderingType values.....	65
285	Table 82 – DataSetOrderingType definition.....	66
286	Table 83 – UadpNetworkMessageContentMask values.....	66
287	Table 84 – UadpNetworkMessageContentMask definition.....	66
288	Table 85 – UadpWriterGroupMessageDataType structure	68
289	Table 86 – UadpWriterGroupMessageDataType definition	68
290	Table 87 – UadpDataSetMessageContentMask Values	68
291	Table 88 – UadpDataSetMessageContentMask definition	69
292	Table 89 – UadpDataSetWriterMessageDataType structure	70
293	Table 90 – UadpDataSetWriterMessageDataType definition	70
294	Table 91 – UadpDataSetReaderMessageDataType structure.....	71
295	Table 92 – UadpDataSetReaderMessageDataType definition	71
296	Table 93 – JsonNetworkMessageContentMask values.....	72
297	Table 94 – JsonNetworkMessageContentMask definition.....	72
298	Table 95 – JsonWriterGroupMessageDataType structure	72
299	Table 96 – JsonWriterGroupMessageDataType definition.....	73
300	Table 97 – JsonDataSetMessageContentMask values.....	73
301	Table 98 – JsonDataSetMessageContentMask definition.....	73
302	Table 99 – JsonDataSetWriterMessageDataType structure	74
303	Table 100 – JsonDataSetWriterMessageDataType definition	74
304	Table 101 – JsonDataSetReaderMessageDataType structure	74
305	Table 102 – JsonDataSetReaderMessageDataType definition	74
306	Table 103 – Standard QosCategory values	75
307	Table 104 – QosDataType structure	75
308	Table 105 – QosDataType definition	75
309	Table 106 – TransmitQosDataType structure	75
310	Table 107 – TransmitQosDataType definition.....	76
311	Table 108 – TransmitQosPriorityDataType structure	76
312	Table 109 – TransmitQosPriorityDataType definition	76
313	Table 110 – ReceiveQosDataType structure	76
314	Table 111 – ReceiveQosDataType definition.....	77
315	Table 112 – TransmitQosPriorityDataType structure	77
316	Table 113 – ReceiveQosPriorityDataType definition	77

317	Table 114 – DatagramConnectionTransportDataType structure	77
318	Table 115 – DatagramConnectionTransportDataType definition.....	78
319	Table 116 – DatagramConnectionTransport2DataType structure	79
320	Table 117 – DatagramConnectionTransport2DataType definition.....	79
321	Table 118 – DatagramWriterGroupTransportDataType structure	79
322	Table 119 – DatagramWriterGroupTransportDataType definition	79
323	Table 120 – DatagramWriterGroupTransport2DataType structure	80
324	Table 121 – DatagramWriterGroupTransport2DataType definition	81
325	Table 122 – DatagramDataSetReaderTransportDataType structure	81
326	Table 123 – DatagramDataSetReaderTransportDataType definition	82
327	Table 124 – BrokerTransportQualityOfService values.....	82
328	Table 125 – BrokerTransportQualityOfService definition.....	82
329	Table 126 – BrokerConnectionTransportDataType structure	83
330	Table 127 – BrokerConnectionTransportDataType definition	83
331	Table 128 – BrokerWriterGroupTransportDataType structure	84
332	Table 129 – BrokerWriterGroupTransportDataType definition	84
333	Table 130 – BrokerDataSetWriterTransportDataType structure	85
334	Table 131 – BrokerDataSetWriterTransportDataType definition	86
335	Table 132 – BrokerDataSetReaderTransportDataType structure.....	87
336	Table 133 – BrokerDataSetReaderTransportDataType definition	87
337	Table 134 – PubSub MessageTypes	88
338	Table 135 – Values for different sequence number sizes	89
339	Table 136 – UADP MessageType mapping	90
340	Table 137 – UADP NetworkMessage	92
341	Table 138 – Layout of the key data for UADP message security	94
342	Table 139 – Layout of the MessageNonce for AES-CTR.....	94
343	Table 140 – Layout of the counter block for UADP message security for AES-CTR	95
344	Table 141 – Chunked NetworkMessage payload header.....	95
345	Table 142 – Chunked NetworkMessage payload fields	96
346	Table 143 – UADP DataSet payload header.....	97
347	Table 144 – UADP DataSet payload	97
348	Table 145 – DataSetMessage header structure.....	98
349	Table 146 – Data Key Frame DataSetMessage structure.....	99
350	Table 147 – Data Delta Frame DataSetMessage structure.....	100
351	Table 148 – Event DataSetMessage structure.....	101
352	Table 149 – Discovery announcement header structure	104
353	Table 150 – OPC UA <i>Application</i> information announcement message structure.....	104
354	Table 151 – DataSetMetaData announcement message structure	104
355	Table 152 – ApplicationInformationType application description fields	105
356	Table 153 – Publisher Endpoints announcement message structure	105
357	Table 154 – ApplicationInformationType status fields	105
358	Table 155 – PubSubConnection configuration announcement message structure.....	106
359	Table 156 – DataSetWriter configuration announcement message structure	106

360	Table 157 – Discovery probe header structure	107
361	Table 158 – Publisher information probe message structure.....	107
362	Table 159 – DataSetWriter settings for Publisher information probe	108
363	Table 160 – WriterGroup settings for Publisher information probe	108
364	Table 161 – PubSubConnections settings for Publisher information probe	108
365	Table 162 – JSON NetworkMessage MessageType mapping.....	109
366	Table 163 – JSON NetworkMessage definition.....	109
367	Table 164 – JSON DataSetMessage definition	111
368	Table 165 – JSON DataSetMetaData definition	112
369	Table 166 – JSON ApplicationDescription definition	112
370	Table 167 – JSON ServerEndpoints definition.....	113
371	Table 168 – JSON Status definition	113
372	Table 169 – JSON PubSubConnection definition.....	114
373	Table 170 – UADP message transported over UDP.....	114
374	Table 171 – UADP message transported over Ethernet.....	116
375	Table 172 – AMQP standard header fields	119
376	Table 173 – OPC UA AMQP standard header QualifiedName Name mappings	119
377	Table 174 – OPC UA AMQP header field conversion rules	120
378	Table 175 – MQTT ConnectionProperties	122
379	Table 176 – OPC UA MQTT standard connection property configuration	122
380	Table 177 – Examples of MQTT Wildcards.....	124
381	Table 178 – MQTT Topic level MessageType mapping.....	124
382	Table 179 – TransportProfileUri encodings	126
383	Table 180 – OPC UA MQTT message properties	126
384	Table 181 – OPC UA MQTT standard message property configuration	127
385	Table 182 – PublishSubscribe Object definition.....	129
386	Table 183 – PubSubKeyServiceType definition	129
387	Table 184 – GetSecurityKeys Method AddressSpace definition	131
388	Table 185 – GetSecurityGroup Method AddressSpace definition	132
389	Table 186 – SecurityGroupType definition.....	133
390	Table 187 – InvalidateKeys Method AddressSpace definition	134
391	Table 188 – ForceKeyRotation Method AddressSpace definition	134
392	Table 189 – SecurityGroupFolderType definition.....	135
393	Table 190 – AddSecurityGroup Method AddressSpace definition.....	136
394	Table 191 – RemoveSecurityGroup Method AddressSpace definition	137
395	Table 192 – AddSecurityGroupFolder Method AddressSpace definition	137
396	Table 193 – RemoveSecurityGroupFolder Method AddressSpace definition.....	138
397	Table 194 – PubSubKeyPushTargetType definition	138
398	Table 195 – ConnectSecurityGroups Method AddressSpace definition.....	140
399	Table 196 – DisconnectSecurityGroups Method AddressSpace definition	141
400	Table 197 – HasPushedSecurityGroup ReferenceType	142
401	Table 198 – TriggerKeyUpdate Method AddressSpace definition	142
402	Table 199 – PubSubKeyPushTargetFolderType definition	142

403	Table 200 – AddPushTarget Method AddressSpace definition	143
404	Table 201 – RemovePushTarget Method AddressSpace definition.....	144
405	Table 202 – AddPushTargetFolder Method AddressSpace definition	144
406	Table 203 – RemovePushTargetFolder Method AddressSpace definition	145
407	Table 204 – Well-Known SKS Roles	145
408	Table 205 – PublishSubscribeType definition	150
409	Table 206 – PublishSubscribeType Additional Subcomponents	151
410	Table 207 – SetSecurityKeys Method AddressSpace definition.....	152
411	Table 208 – AddConnection Method AddressSpace definition	153
412	Table 209 – RemoveConnection Method AddressSpace definition	153
413	Table 210 – HasPubSubConnection ReferenceType	154
414	Table 211 – PubSubConfigurationType definition	155
415	Table 212 – PubSubConfigurationRefMask values	156
416	Table 213 – PubSubConfigurationRefMask definition	157
417	Table 214 – PubSubConfigurationRefDataType structure	157
418	Table 215 – PubSubConfigurationRefDataType definition.....	158
419	Table 216 – PubSubConfigurationValueDataType structure.....	158
420	Table 217 – PubSubConfigurationValueDataType definition	158
421	Table 218 – Reservelds Method AddressSpace definition	159
422	Table 219 – CloseAndUpdate Method AddressSpace definition	161
423	Table 220 – PublishedDataSetType definition	163
424	Table 221 – ExtensionFieldsType definition	164
425	Table 222 – Well-Known Extension Field Names.....	164
426	Table 223 – AddExtensionField Method AddressSpace definition	165
427	Table 224 – RemoveExtensionField Method AddressSpace definition.....	165
428	Table 225 – DataSetToWriter ReferenceType	166
429	Table 226 – PublishedDataItemsType definition.....	166
430	Table 227 – AddVariables Method AddressSpace definition	168
431	Table 228 – RemoveVariables Method AddressSpace definition	169
432	Table 229 – PublishedEventsType definition	169
433	Table 230 – ModifyFieldSelection Method AddressSpace definition	171
434	Table 231 – DataSetFolderType definition	171
435	Table 232 – AddPublishedDataItems Method AddressSpace definition	173
436	Table 233 – AddPublishedEvents Method AddressSpace definition	174
437	Table 234 – AddPublishedDataItemsTemplate Method AddressSpace definition	175
438	Table 235 – AddPublishedEventsTemplate Method AddressSpace definition	176
439	Table 236 – RemovePublishedDataSet Method AddressSpace definition	177
440	Table 237 – AddDataSetFolder Method AddressSpace definition.....	177
441	Table 238 – RemoveDataSetFolder Method AddressSpace definition	178
442	Table 239 – PubSubConnectionType definition	179
443	Table 240 – AddWriterGroup Method AddressSpace definition	180
444	Table 241 – AddReaderGroup Method AddressSpace definition	181
445	Table 242 – RemoveGroup Method AddressSpace definition.....	181

446	Table 243 – NetworkAddressType definition	181
447	Table 244 – NetworkAddressUriType definition	182
448	Table 245 – ConnectionTransportType definition	182
449	Table 246 – HasWriterGroup ReferenceType	183
450	Table 247 – HasReaderGroup ReferenceType	183
451	Table 248 – PubSubGroupType definition	184
452	Table 249 – WriterGroupType definition	185
453	Table 250 – AddDataSetWriter Method AddressSpace definition	186
454	Table 251 – RemoveDataSetWriter Method AddressSpace definition	187
455	Table 252 – HasDataSetWriter ReferenceType	187
456	Table 253 – WriterGroupTransportType definition	187
457	Table 254 – WriterGroupMessageType definition	188
458	Table 255 – ReaderGroupType definition	188
459	Table 256 – AddDataSetReader Method AddressSpace definition	189
460	Table 257 – RemoveDataSetReader Method AddressSpace definition	190
461	Table 258 – HasDataSetReader ReferenceType	190
462	Table 259 – ReaderGroupTransportType definition	190
463	Table 260 – ReaderGroupMessageType definition	191
464	Table 261 – DataSetWriterType definition	192
465	Table 262 – DataSetWriterTransportType definition	193
466	Table 263 – DataSetWriterMessageType definition	193
467	Table 264 – DataSetReaderType definition	194
468	Table 265 – DataSetReaderTransportType definition	195
469	Table 266 – DataSetReaderMessageType definition	195
470	Table 267 – CreateTargetVariables Method AddressSpace definition	196
471	Table 268 – CreateDataSetMirror Method AddressSpace definition	197
472	Table 269 – SubscribedDataSetType definition	198
473	Table 270 – TargetVariablesType definition	198
474	Table 271 – AddTargetVariables Method AddressSpace definition	199
475	Table 272 – RemoveTargetVariables Method AddressSpace definition	200
476	Table 273 – SubscribedDataSetMirrorType definition	200
477	Table 274 – SubscribedDataSetFolderType definition	201
478	Table 275 – AddSubscribedDataSet Method AddressSpace definition	202
479	Table 276 – RemoveSubscribedDataSet Method AddressSpace definition	202
480	Table 277 – AddDataSetFolder Method AddressSpace definition	203
481	Table 278 – RemoveDataSetFolder Method AddressSpace definition	204
482	Table 279 – StandaloneSubscribedDataSetType definition	204
483	Table 280 – PubSubStatusType definition	205
484	Table 281 – Enable Method AddressSpace definition	205
485	Table 282 – Disable Method AddressSpace definition	206
486	Table 283 – PubSubDiagnosticsType	207
487	Table 284 – Counters for PubSubDiagnosticsType	208
488	Table 285 – Reset Method AddressSpace definition	208

489	Table 286 – DiagnosticsLevel values	209
490	Table 287 – PubSubDiagnosticsCounterType	209
491	Table 288 – PubSubDiagnosticsCounterClassification values	210
492	Table 289 – PubSubDiagnosticsRootType	210
493	Table 290 – LiveValues for PubSubDiagnosticsRootType.....	210
494	Table 291 – PubSubDiagnosticsConnectionType	210
495	Table 292 – LiveValues for PubSubDiagnosticsConnectionType.....	211
496	Table 293 – PubSubDiagnosticsWriterGroupType	211
497	Table 294 – Counters for PubSubDiagnosticsWriterGroupType	211
498	Table 295 – LiveValues for PubSubDiagnosticsWriterGroupType	211
499	Table 296 – PubSubDiagnosticsReaderGroupType	212
500	Table 297 – Counters for PubSubDiagnosticsReaderGroupType	212
501	Table 298 – LiveValues for PubSubDiagnosticsReaderGroupType	212
502	Table 299 – PubSubDiagnosticsDataSetWriterType	212
503	Table 300 – Counters for PubSubDiagnosticsDataSetWriterType	213
504	Table 301 – LiveValues for PubSubDiagnosticsDataSetWriterType	213
505	Table 302 – PubSubDiagnosticsDataSetReaderType	213
506	Table 303 – Counters for PubSubDiagnosticsDataSetReaderType	213
507	Table 304 – LiveValues for PubSubDiagnosticsDataSetReaderType.....	213
508	Table 305 – PubSubCapabilitiesType definition.....	214
509	Table 306 – PubSubStatusEventType definition	216
510	Table 307 – PubSubTransportLimitsExceedEventType definition.....	217
511	Table 308 – PubSubCommunicationFailureEventType definition.....	217
512	Table 309 – UadpWriterGroupMessageType definition	218
513	Table 310 – UadpDataSetWriterMessageType definition	218
514	Table 311 – UadpDataSetReaderMessageType definition	219
515	Table 312 – JsonWriterGroupMessageType definition	219
516	Table 313 – JsonDataSetWriterMessageType definition	220
517	Table 314 – JsonDataSetReaderMessageType definition	220
518	Table 315 – DatagramConnectionTransportType definition.....	220
519	Table 316 – DatagramWriterGroupTransportType definition	221
520	Table 317 – DatagramDataSetReaderTransportType definition.....	222
521	Table 318 – BrokerConnectionTransportType definition	222
522	Table 319 – BrokerWriterGroupTransportType definition	223
523	Table 320 – BrokerDataSetWriterTransportType definition	223
524	Table 321 – Broker Writer well-known extension field names.....	224
525	Table 322 – BrokerDataSetReaderTransportType definition	224
526	Table A.1 – UADP NetworkMessage header layout	227
527	Table A.2 – Values for configuration parameters	227
528	Table A.3 – UADP NetworkMessage header layout with integrity (signing).....	228
529	Table A.4 – UADP NetworkMessage header layout with integrity and confidentiality	228
530	Table A.5 – UADP DataSetMessage header layout	229
531	Table A.6 – Values for configuration parameters	229

532	Table A.7 – UADP NetworkMessage header layout	232
533	Table A.8 – Values for configuration parameters	232
534	Table A.9 – UADP NetworkMessage header layout with integrity (signing).....	233
535	Table A.10 – UADP NetworkMessage header layout with integrity and confidentiality	233
536	Table A.11 – UADP DataSetMessage header layout	234
537	Table A.12 – Values for configuration parameters	234
538	Table A.13 – DataSet1 fields	235
539	Table A.14 – DataSet2 fields	235
540	Table A.15 – DataSet3 fields	236
541	Table A.16 – Values for WriterGroup configuration parameters.....	238
542	Table A.17 – Values for DataSetWriter configuration parameters.....	239
543	Table A.18 – Values for WriterGroup configuration parameters.....	240
544	Table A.19 – Values for DataSetWriter configuration parameters.....	241
545	Table A.20 – Values for WriterGroup configuration parameters.....	243
546	Table A.21 – Values for DataSetWriter configuration parameters.....	243

547

548

INTERNATIONAL ELECTROTECHNICAL COMMISSION

OPC UNIFIED ARCHITECTURE –

Part 14: PubSub

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 62541-14 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

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

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

- a) Added a "Quantity Model" which can be referenced from EngineeringUnit Properties. The model defines quantities and assigned units. In addition it provides alternative units and the conversion to them.
- b) Added additional rules for ValuePrecision Property:
 - Can also be used for other subtypes like Duration and Decimal.
 - Added rules when ValuePrecision has negative values.

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

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

600
601 Full information on the voting for the approval of this International Standard can be found in the report
602 on voting indicated in the above table.

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

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

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

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

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

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

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

- 620 • reconfirmed,
- 621 • withdrawn,
- 622 • replaced by a revised edition, or
- 623 • amended.

624

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.

625

OPC Unified Architecture Specification

Part 14: PubSub

1 Scope

This part of OPC Unified Architecture (OPC UA) defines the *PubSub* communication model. It defines an OPC UA publish subscribe pattern which complements the client server pattern defined by the *Services* in IEC 62541-4. See IEC 62541-1 for an overview of the two models and their distinct uses.

PubSub allows the distribution of data and events from an OPC UA information source to interested observers inside a device network as well as in IT and analytics cloud systems.

This document consists of

- a general introduction of the *PubSub* concepts,
- a definition of the *PubSub* configuration parameters,
- mapping of *PubSub* concepts and configuration parameters to messages and transport protocols,
- and a PubSub configuration model.

Not all OPC UA *Applications* will need to implement all defined message and transport protocol mappings. IEC 62541-7 defines the *Profile* that dictate which mappings need to be implemented in order to be compliant with a particular *Profile*.

2 Normative references

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

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

IEC 62541-2: *OPC Unified Architecture - Part 2: Security Model*

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 62541-12: *OPC Unified Architecture - Part 12: Discovery and Global Services*

IEC 62541-20: *OPC Unified Architecture - Part 20: File Transfer*

IEC 62541-22: *OPC Unified Architecture - Part 22: Base Network Model*

ISO/IEC 19464:2014, *Advanced Message Queuing Protocol (AMQP) v1.0 specification*

ISO/IEC 20922:2016, *Message Queuing Telemetry Transport (MQTT) v3.1.1*

Message Queuing Telemetry Transport (MQTT) Version 5

<http://docs.oasis-open.org/mqtt/mqtt/v5.0>

IETF RFC 8254, *The JavaScript Object Notation (JSON) Data Interchange Format*

<http://www.ietf.org/rfc/rfc8254.txt>

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62541-1, IEC 62541-3, and IEC 62541-4, as well as the following apply.

3.1.1

DataSetClass

template declaring the content of a *DataSet*

Note 1 to entry: A *DataSetClass* is used to type *DataSets* for use in several *Publishers* and for filtering in *Subscribers*.

3.1.2

DataSetMetaData

data describing the content and semantic of a *DataSet*

3.1.3

DataSetReader

entity receiving *DataSetMessages* from a *Message Oriented Middleware*

Note 1 to entry: A *DataSetReader* is the component that extracts a *DataSetMessage* from a *NetworkMessage* received from the *Message Oriented Middleware* and decodes the *DataSetMessage* to a *DataSet* for further processing in the *Subscriber*.

3.1.4

DataSetWriter

entity creating *DataSetMessages* from *DataSets* and publishing them through a *Message Oriented Middleware*

Note 1 to entry: A *DataSetWriter* encodes a *DataSet* to a *DataSetMessage* and includes the *DataSetMessage* into a *NetworkMessage* for publishing through a *Message Oriented Middleware*.

3.1.5

PublishedDataSet

configuration of application-data to be published as *DataSet*

Note 1 to entry: A *PublishedDataSet* can be a list of monitored *Variables* or an *Event* selection.

3.1.6

SecurityGroup

grouping of security settings and security keys used to access messages from a *Publisher*

Note 1 to entry: A *SecurityGroup* is an abstraction that represents the security settings and security keys that can be used to access messages from a *Publisher*. A *SecurityGroup* is identified with a unique identifier called the *SecurityGroupId*. The *SecurityGroupId* is unique within the *Security Key Service*.

3.1.7

SubscribedDataSet

configuration for dispatching of received *DataSets*

Note 1 to entry: A *SubscribedDataSet* can be a mapping of *DataSet* fields to *Variables* in the *Subscriber AddressSpace*.

3.2 Abbreviated terms

AMQP Advanced Message Queuing Protocol

AS Authorization Service

712	CTL	Certificate Trust List
713	HMI	Human Machine Interface
714	IGMP	Internet Group Management Protocol
715	MIME	Multipurpose Internet Mail Extensions
716	MLD	Multicast Listener Discovery
717	MQTT	MQ Telemetry Transport
718	MTU	Maximum Transmission Unit
719	PCP	Priority Code Point
720	QoS	Quality of Service
721	SKS	Security Key Service
722	TSN	Time Sensitive Networking
723	UA	Unified Architecture
724	UADP	UA Datagram Protocol
725	UDP	User Datagram Protocol
726	URI	Uniform Resource Identifier
727	URL	Uniform Resource Locator
728	VID	VLAN Identifier

729 4 Overview

730 4.1 Fields of application

731 In *PubSub* the participating OPC UA *Applications* with their roles as *Publishers* and *Subscribers*
732 are decoupled. The number of *Subscribers* receiving data from a *Publisher* does not influence
733 the *Publisher*. This makes *PubSub* suitable for applications where location independence and/or
734 scalability are required.

735 The following are some example uses for *PubSub*:

- 736 • Configurable peer-to-peer communication between controllers and between controllers
737 and HMIs. The peers are not directly connected and do not even need to know about
738 the existence of each other. The data exchange often requires a fixed time-window; it
739 may be point-to-point connection or data distribution to many receivers.
- 740 • Asynchronous workflows. For example, an order processing application can place an
741 order on a message queue or an enterprise service bus. From there it can be processed
742 by one or more workers.
- 743 • Logging to multiple systems. For example, sensors or actuators can write logs to a
744 monitoring system, an HMI, an archive application for later querying, and so on.
- 745 • OPC UA *Servers* representing services or devices can stream data to applications
746 hosted in the cloud. For example, backend servers, big data analytics for system
747 optimization and predictive maintenance.

748 4.2 Abstraction layers

749 *PubSub* is designed to be flexible and is not bound to a particular messaging system. All
750 components and activities are first described abstractly in Clause 5 and do not represent a
751 specification for implementation. The concrete communication parameters are specified in
752 Clause 6. The concrete transport protocol mappings and message mappings are specified in
753 Clause 7.

754 Defined with these abstraction layers, *PubSub* can be used to transport different types of
755 information through networks with different characteristics as illustrated with two examples:

- *PubSub* with UDP transport and binary encoded messages may be well-suited in production environments for frequent transmission of small amounts of data. It also allows data exchange in one-to-one and one-to-many configurations.
- The use of established standard messaging protocols (e.g. AMQP or MQTT) with JSON data encoding supports the cloud integration path and readily allows handling of the information in modern stream and batch analytics systems.

4.3 Decoupling by use of middleware

In *PubSub* the participating OPC UA *Applications* can assume the roles *Publisher* and *Subscriber*. *Publishers* are the sources of data, while *Subscribers* consume that data. Communication in *PubSub* is message-based. *Publishers* send messages to a *Message Oriented Middleware*, without knowledge of what, if any, *Subscribers* there may be. Similarly, *Subscribers* express interest in specific types of data, and process messages that contain this data, without knowledge of what *Publishers* there are.

Message Oriented Middleware is software or hardware infrastructure that supports sending and receiving messages between distributed systems. The implementation of this distribution depends on the *Message Oriented Middleware*.

Figure 1 illustrates that *Publishers* and *Subscribers* only interact with the *Message Oriented Middleware* which provides the means to forward the data to one or more receivers.

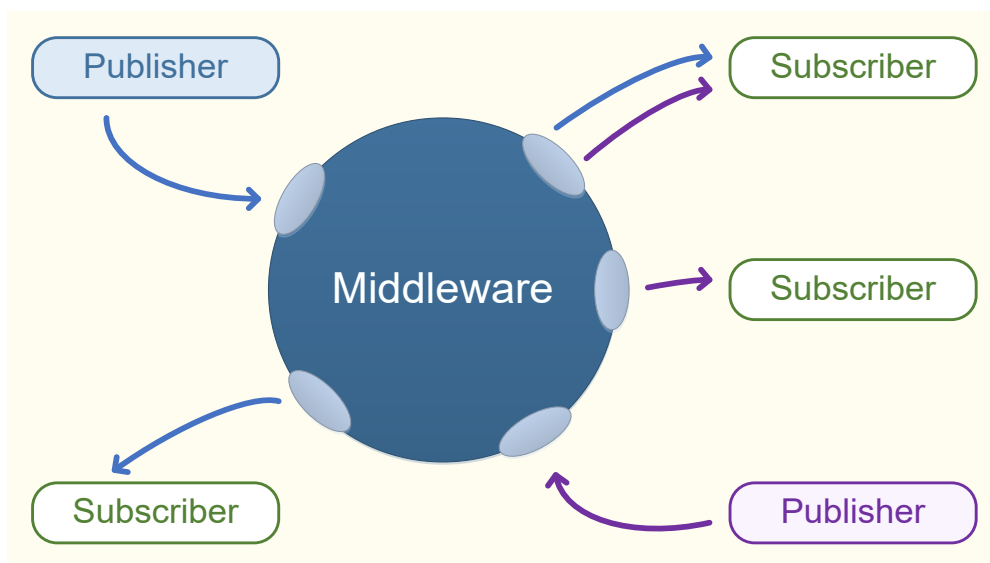


Figure 1 – Publish Subscribe model overview

To cover a large number of use cases, OPC UA *PubSub* supports two largely different *Message Oriented Middleware* variants:

- a broker-less form, where the *Message Oriented Middleware* is the network infrastructure that is able to route datagram-based messages. *Subscribers* and *Publishers* use datagram protocols like UDP;
- a broker-based form, where the core component of the *Message Oriented Middleware* is a message *Broker*. *Subscribers* and *Publishers* use standard messaging protocols like AMQP or MQTT to communicate with the *Broker*. All messages are published to specific queues (e.g. topics, nodes) that the *Broker* exposes and *Subscribers* can listen to these queues. The *Broker* may translate messages from the formal messaging protocol of the *Publisher* to the formal messaging protocol of the *Subscriber*.

4.4 Synergy of models

PubSub and *Client Server* are both based on the OPC UA *Information Model*. *PubSub* therefore can easily be integrated into OPC UA *Servers* and OPC UA *Clients*. Quite typically, a *Publisher* will be an OPC UA *Server* (the owner of information) and a *Subscriber* is often an OPC UA *Client*. Above all, the *PubSub Information Model* for configuration (see 9) promotes the configuration of *Publishers* and *Subscribers* using the OPC UA *Client Server* model.

Nevertheless, the *PubSub* communication does not require such a role dependency. I.e., OPC UA *Clients* can be *Publishers* and OPC UA *Servers* can be *Subscribers*. In fact, there is no necessity for *Publishers* or *Subscribers* to be either an OPC UA *Server* or an OPC UA *Client* to participate in *PubSub* communications.

More details on how *Subscriptions* in the *Client Server* communication model compare to *PubSub* are described in Annex B.

5 PubSub Concepts

5.1 General

Clause 5 describes the general OPC UA *PubSub* concepts.

The *DataSet* constitutes the payload of messages provided by the *Publisher* and consumed by the *Subscriber*. The *DataSet* is described in 5.2. The mapping to messages is described in 5.3. The participating entities like *Publisher* and *Subscriber* are described in 5.4.

The abstract communication parameters are described in Clause 6.

The mapping of this model to concrete message and transport protocol mappings is defined in Clause 7.

The OPC UA *Information Model* for *PubSub* configuration in Clause 9 specifies the standard *Objects* in an OPC UA *AddressSpace* used to create, modify and expose an OPC UA *PubSub* configuration.

Figure 2 provides an overview of the *Publisher* and *Subscriber* entities. It illustrates the flow of messages from a *Publisher* to one or more *Subscribers*. The *PubSub* communication model supports many other scenarios; for example, a *Publisher* may send a *DataSet* to multiple *Message Oriented Middleware* and a *Subscriber* may receive messages from multiple *Publishers*.

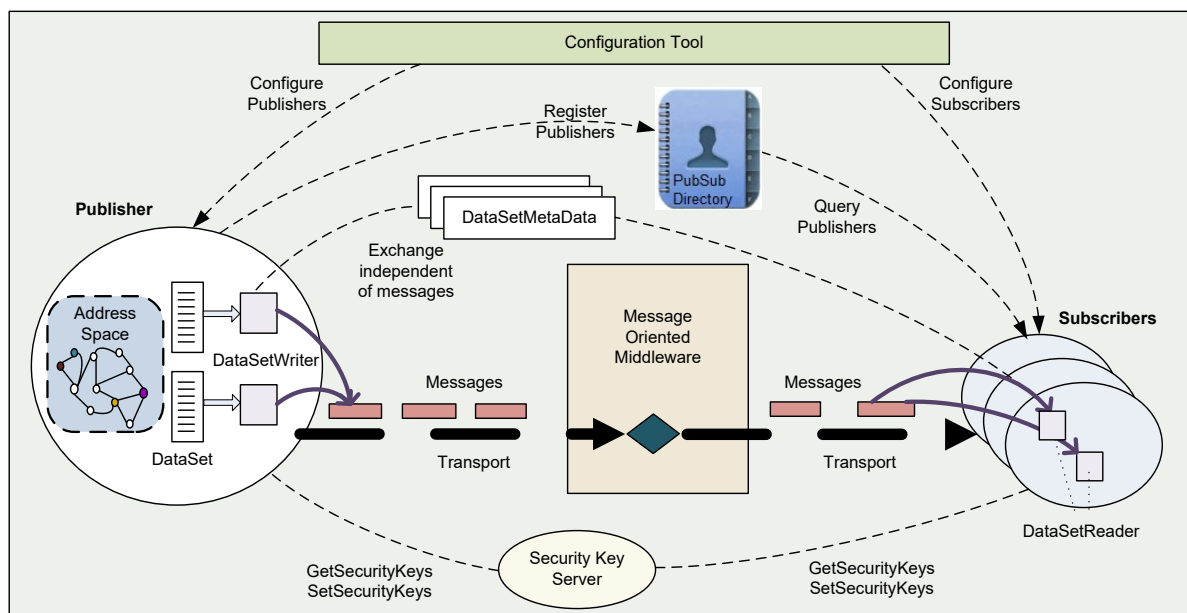


Figure 2 – Publisher and Subscriber entities

Publishers and *Subscribers* are loosely coupled. They often will not even know each other. Their primary relation is the shared understanding of specific types of data (*DataSets*), the publish characteristics of messages that include these data, and the *Message Oriented Middleware*.

The “messages” in Figure 2 represent *NetworkMessages*. Each *NetworkMessage* includes header information (e.g. identification and security data) and one or more *DataSetMessages* (the payload). The *DataSetMessages* may be signed and encrypted in accordance with the configured message security. A *Security Key Server* is responsible for the distribution of the security keys needed for message security.

Each *DataSetMessage* is created from a *DataSet*. A component of a *Publisher* called *DataSetWriter* generates a continuous sequence of *DataSetMessages*. Syntax and semantics of *DataSets* are described by *DataSetMetaData*. The selection of information for a *DataSet* in the *Publisher* and the data acquisition parameters are called *PublishedDataSet*. *DataSet*, *DataSetMetaData* and *PublishedDataSet* are detailed in 5.2.

Publishers and *Subscribers* are typically configured through a configuration tool. The configuration can be done through a generic OPC UA *PubSub* configuration tool using the *PubSub* configuration *Information Model* defined in Clause 9 or through product-specific configuration tools. To support the *PubSub* configuration *Information Model*, *Publishers* and *Subscribers* must be also OPC UA *Server*.

NOTE The *PubSub* directory is an optional entity that allows *Publishers* to advertise their *PublishedDataSets* and their communication parameters. This directory functionality is planned for a future version of this document.

5.2 DataSet

5.2.1 General

A *DataSet* can be thought of as a list of name and value pairs representing an *Event* or a list of *Variable Values*.

A *DataSet* can be created from an *Event* or from a sample of *Variable Values*. The configuration of this application-data collector is called *PublishedDataSet*. *DataSet* fields can be defined to represent any information, for example, they could be internal *Variables* in the *Publisher*, *Events* from the *Publisher* or collected by the *Publisher*, network data, or data from sub-devices.

DataSetMetaData described in 5.2.3 defines the structure and content of a *DataSet*.

For publishing, a *DataSet* will be encoded into a *DataSetMessage*. One or more *DataSetMessages* are combined to form the payload of a *NetworkMessage*.

Figure 3 illustrates the use of *DataSets* for publishing.

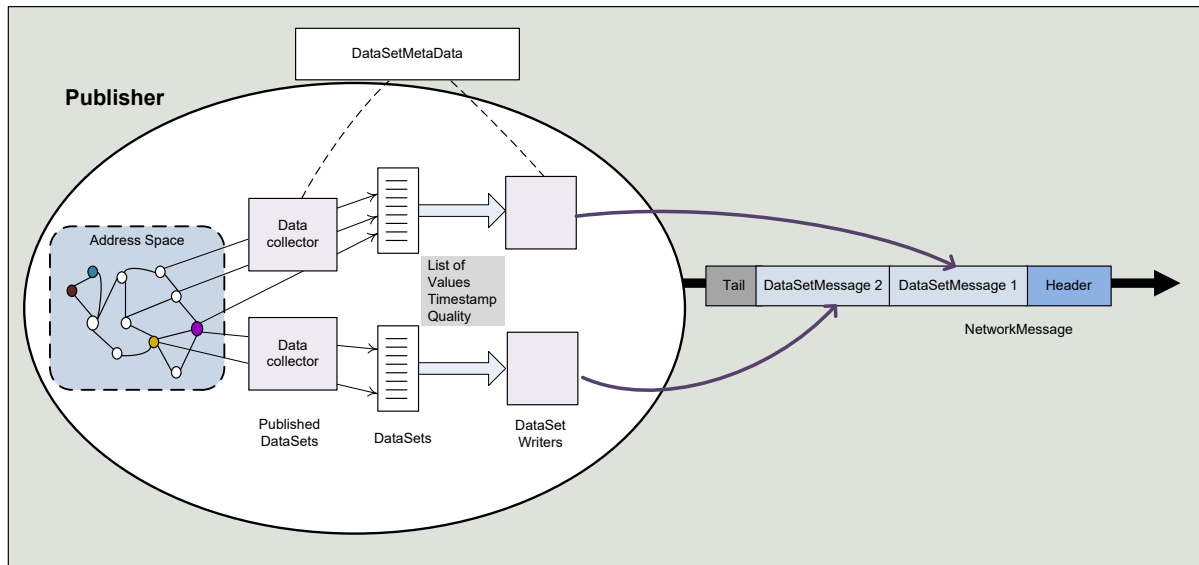


Figure 3 – DataSet in the process of publishing

A *PublishedDataSet* is similar to either an *Event MonitoredItem* or a list of data *MonitoredItems* in the *Client Server Subscription* model. Similar to an *Event MonitoredItem*, a *PublishedDataSet* can select a list of *Event* fields. Similar to data *MonitoredItems*, the *PublishedDataSet* can contain a list of *Variables*.

A *DataSet* does not define the mechanism to encode, secure and transport it. A *DataSetWriter* handles the creation of a *DataSetMessage* for a *DataSet*. The *DataSetWriter* contains settings for the encoding and transport of a *DataSetMessage*. Most of these settings depend on the selected *Message Oriented Middleware*.

The configuration of *DataSets* and the way the data is obtained for publishing can be configured using the *PubSub* configuration model defined in 8.2 or with vendor-specific configuration tools.

5.2.2 DataSetClass

DataSets can be individual for a *Publisher* or they can be derived from a *DataSetClass*. Such a *DataSetClass* acts as template declaring the content of a *DataSet*. The *DataSetClass* is identified by a globally unique id – the *DataSetClassId* (see 6.2.3.3).

The *DataSetMetaData* is identical for all *PublishedDataSets* that are configured based on this *DataSetClass*. The *DataSetClassId* shall be in the corresponding field of the *DataSetMetaData*.

When all *DataSetMessages* of a *NetworkMessage* are created from *DataSets* that are instances of the same *DataSetClass*, the *DataSetClassId* of this class can be provided in the *NetworkMessage* header.

5.2.3 DataSetMetaData

DataSetMetaData describes the content and semantics of a *DataSet*. The structure description includes overall *DataSet* attributes (e.g. name and version) and a set of fields with their name and data type. The order of the fields in the *DataSetMetaData* shall match the order of values in the published *DataSetMessages*.

The *DataSetMetaDataType* is defined in 6.2.3.2.3.

Example description (simplified, in pseudo-language):

Name:	"Temperature-Sensor Measurement"
Fields:	[1] Name=DeviceName, Type=String
	[2] Name=Temperature, Type=Float, Unit=Celsius, Range={1,100}

Subscribers use the *DataSetMetaData* for decoding the values of a *DataSetMessage* to a *DataSet*. *Subscribers* may use name and data type for further processing or display of the published data.

Each *DataSetMessage* also includes the version of the *DataSetMetaData* that it complies with. This allows *Subscribers* to verify if they have the corresponding *DataSetMetaData*. The related *ConfigurationVersionDataType* is defined in 6.2.3.2.6.

DataSetMetaData may be specific to a single *PublishedDataSet* or identical for all *PublishedDataSets* that are configured based on a *DataSetClass* (see 5.2.2).

There are multiple options for *Subscribers* to get the initial *DataSetMetaData*:

- The *Subscriber* is an OPC UA *Client* and is able to get the necessary configuration information from the *PubSub* configuration model (see 9.1.4.2.1) provided by the *Publisher* or from a configuration server.
- The *Subscriber* supports the OPC UA configuration *Methods* defined in the *PubSub* configuration model.
- The *Subscriber* receives the *DataSetMetaData* as *NetworkMessage* from the *Publisher*. This may require an option for the *Subscriber* to request this *NetworkMessage* from the *Publisher*.
- The *Subscriber* is configured with product-specific configuration means.

There are multiple options to exchange the *DataSetMetaData* between *Publisher* and *Subscriber* if the configuration changes.

- The *DataSetMetaData* is sent as a *NetworkMessage* from the *Publisher* to the *Subscriber* before *DataSetMessages* with changed content are sent. The used *Message Oriented Middleware* should ensure reliable delivery of the message. The mapping for the *Message Oriented Middleware* defines a way for the *Subscriber* to get the *DataSetMetaData*. The *Subscriber* goes to an error state if it has not received the new *DataSetMetaData* that matches the *ConfigurationVersion* of the received *DataSetMessage*.
- The *Subscriber* is automatically updated via the OPC UA configuration *Methods* defined in the *PubSub* configuration model when the *DataSet* in the *Publisher* is updated.
- The *Subscriber* is an OPC UA *Client* and is able to obtain the update from the *Publisher* or a configuration server via the information exposed by the *PubSub* configuration model.
- The *Subscriber* is updated with product-specific configuration means when the *DataSet* in the *Publisher* is changed.

5.3 Messages

5.3.1 General

The term message is used with various intentions in the messaging world. It sometimes only refers to the payload (the application data) and sometimes to the network packet that also includes protocol-, security-, or encoding-specific data. To avoid confusion, this document formally defines the term *DataSetMessage* to mean the application data (the payload) supplied by the *Publisher* and the term *NetworkMessage* to mean the message handed off and received from a specific *Message Oriented Middleware*. *DataSetMessages* are embedded in *NetworkMessages*. Figure 4 shows the relationship of these message types.

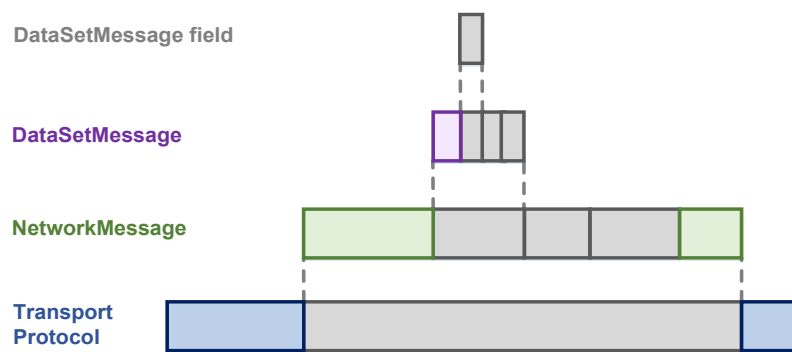


Figure 4 – OPC UA PubSub message layers

The transport protocol-specific headers and definitions are described in 7.3.

Subclauses 5.3.2 to 5.3.4 provide an abstract definition of *DataSetMessage* and *NetworkMessage*. The concrete structure depends on the message mapping and is described in 7.2.

DataSetMessages are just one of the possible *MessageTypes* transported within a *NetworkMessage*. The different *MessageTypes* are defined in 7.2.

5.3.2 DataSetMessage field

A *DataSetMessage* field is the representation of a *DataSet* field in a *DataSetMessage*.

A *DataSet* field contains the actual value as well as additional information about the value like status and timestamp.

A *DataSet* field can be represented as a *DataValue*, as a *Variant* or as a *RawData* in the *DataSetMessage* field. The representation depends on the *DataSetFieldContentMask* defined in 6.2.4.2.

The representation as a *DataValue* is used if value, status and timestamp are included in the *DataSetMessage*.

The representation as *Variant* is used if value or bad status should be included in the *DataSetMessage*.

The representation as *RawData* is the most efficient format and is used if a common status and timestamp per *DataSet* is sufficient.

5.3.3 DataSetMessage

A *DataSetMessage* is created from a *DataSet*. It consists of a header and the encoded fields of the *DataSet*.

Depending on the configured *DataSetMessageContentMask*, a *DataSetMessage* may exist in different forms and with varying detail. *DataSetMessages* do not contain any information about the data acquisition or information source in the *Publisher*.

Additional header information includes:

DataSetWriterId	Identifies the <i>DataSetWriter</i> and indirectly the <i>PublishedDataSet</i> .
Sequence number	A number that is incremented for each <i>DataSetMessage</i> . Can be used to verify the ordering and to detect missing messages.
Timestamp	A timestamp describing when the data in this <i>DataSetMessage</i> was obtained.
Version	Version information about the configuration of the <i>DataSetMetaData</i> .
Status	Status information about the data in this <i>DataSetMessage</i> .

960 Keep alive When no *DataSetMessages* are sent for a configured time period, a
 961 keep alive *DataSetMessage* is sent to signal the *Subscribers* that the
 962 *Publisher* is still alive.

963 *DataSetMessages* are either sent cyclicly or acyclicly in a publishing interval. Acyclic *DataSets*
 964 are sent as event *DataSetMessages*. Cyclic *DataSets* can create at most one *DataSetMessages*
 965 per interval. Acyclic *DataSets* can create multiple event *DataSetMessages* per interval.

966 For cyclic *DataSets*, some encodings differentiate between key frame *DataSetMessages* and
 967 delta frame *DataSetMessages*. A key frame *DataSetMessage* includes values for all fields of the
 968 *DataSet*. A delta frame *DataSetMessage* only contains the subset that changed since the
 969 previous *DataSetMessage*.

970 A key frame *DataSetMessage* is sent after a configured number of *DataSetMessages*.

971 The *DataSetMetaData* as data contract defines the fields contained in the *DataSetMessage*.
 972 The header settings for *DataSetMessage* and *NetworkMessage* define the communication
 973 contract between *Publisher* and *Subscriber*.

974 A heartbeat *DataSetMessage* is a key frame that only contains header information. It is used to
 975 indicate that the *Publisher* is still alive without sending payload. A heartbeat *DataSetMessage*
 976 is not created from a *DataSet*.

977 **5.3.4 NetworkMessage**

978 The *NetworkMessage* is a container for messages of different *MessageTypes* defined in 7.2.

979 A *NetworkMessage* can contain an array of *DataSetMessages* and includes information shared
 980 between *DataSetMessages*. This information consists of:

981 PublisherId	Identifies the <i>Publisher</i> .
982 Security data	Only available for encodings that support message security. The
983	relevant information is specified in the message mapping.
984 Promoted fields	Selected fields out of the <i>DataSet</i> also sent in the header.
985 Payload	One or more <i>DataSetMessages</i> .

986 The payload, consisting of the *DataSetMessages* will be encrypted in accordance with the
 987 configured message security. Individual fields of a *DataSetMessage* can be marked as being
 988 “promoted fields”. Such fields are intended for filtering or routing and therefore are never
 989 encrypted. How and where the values for promoted fields are inserted depends on the
 990 *NetworkMessage* format and the used protocol. The *NetworkMessage* header is not encrypted
 991 to enable efficient filtering.

992 **5.3.5 Message security**

993 Message security in *PubSub* concerns integrity and confidentiality of the published message
 994 payload. The base concepts for OPC UA security are defined in IEC 62541-2. The level of
 995 security can be:

- 996 • No security
- 997 • Signing but no encryption
- 998 • Signing and encryption

999 Message security is end-to-end security (from *Publisher* to *Subscriber*) and requires common
 1000 knowledge of the cryptographic keys necessary to sign and encrypt on the *Publisher* side as
 1001 well as validate signature and decrypt on the *Subscriber* side.

1002 The keys used for message security are managed in the context of a *SecurityGroup*. The basic
 1003 concepts of a *SecurityGroup* are described in 5.3.7.

1004 This standard defines a general distribution framework for cryptographic keys. This framework
 1005 is introduced in 5.4.4.

1006 All parameters that are relevant for message security are described in 6.2.5. These parameters
1007 are independent of any *Broker* level transport security.

1008 The message security for *PubSub* is independent of the transport protocol mapping and is
1009 completely defined by OPC UA.

1010 5.3.6 Transport security

1011 The transport security is specific to the transport protocol mapping.

1012 When using a broker-based middleware (see 5.4.5.2.2), confidentiality and integrity can be
1013 ensured with the transport security between *Publishers* and the *Broker* as well as *Subscribers*
1014 and the *Broker*. The *Broker* level security in addition requires all *Publishers* and *Subscribers* to
1015 have credentials that grant them access to a *Broker* resource.

1016 Transport security may be hop-by-hop security with some risk of man-in-the-middle attacks. It
1017 also requires trusting the *Broker* since the *Broker* can read the messages. Combining transport
1018 security with message security reduces this risk.

1019 5.3.7 SecurityGroup

1020 A *SecurityGroup* is an abstraction that represents the message security settings and security
1021 keys for a subset of *NetworkMessages* exchanged between *Publishers* and *Subscribers*. The
1022 security keys are used to encrypt and decrypt *NetworkMessages* and to generate and check
1023 signatures on a *NetworkMessage*.

1024 A *Security Key Service* (SKS) manages *SecurityGroups* and maintains a mapping between
1025 *Roles* and their access *Permissions* for a *SecurityGroup*. This mapping defines if a *Publisher*
1026 or *Subscriber* has access to the security keys of a *SecurityGroup*. The SKS is described in more
1027 detail in 5.4.4.

1028 A *SecurityGroup* is identified with a unique identifier called the *SecurityGroupId*. It is unique
1029 within the SKS. A *Publisher* for its *PublishedDataSets* needs to know the *SecurityGroupId*. For
1030 *Subscribers* the *SecurityGroupId* is distributed as metadata together with the *DataSetMetaData*.
1031 The metadata for a *SecurityGroupId* includes the *EndpointDescription* of the responsible SKS.
1032 Publishers and Subscribers use the *EndpointDescription* to access the SKS and the
1033 *SecurityGroupId* to obtain the security keys for a *SecurityGroup*.

1034 5.3.8 Topics

1035 A *Topic* is a string associated with a *NetworkMessage* that can be used to logically organize
1036 *NetworkMessages* published to a *Message Oriented Middleware*. *Topics* are most commonly
1037 used with *Broker*-based middleware where the filtering is done by the *Broker*. However, *Topics*
1038 can be used with *Broker*-less middleware where the filtering is done by the *Subscriber*.

1039 *Topics* have a hierarchical structure with different *Topic* levels separated by a delimiter like '/'.
1040 For example:

1041 `opcua/json/status/FlowController1`

1042 Mappings to different implementation technologies may add additional constraints.

1043 A *Topic* for *NetworkMessages* containing *DataSetMessages* is typically used as *QueueName*
1044 for the broker-based communication configuration.

1045 5.4 Entities

1046 5.4.1 Publisher

1047 5.4.1.1 General

1048 The *Publisher* is the *PubSub* entity that sends *NetworkMessages* to a *Message Oriented*
1049 *Middleware*. It represents a certain information source, for example, a control device, a
1050 manufacturing process, a weather station, or a stock exchange.

1051 Commonly, a *Publisher* is also an OPC UA Server. For the abstract *PubSub* concepts, however,

it is an arbitrary entity and should not be assumed to be an individual or even a specific network node (an IP or a MAC address) or a specific application. A *Publisher* may consist of one or more network nodes sending messages and management node(s) responding to discovery probe messages and providing an OPC UA *Server* for configuration and diagnostics.

Figure 5 illustrates a *Publisher* with data collection, encoding and message sending.

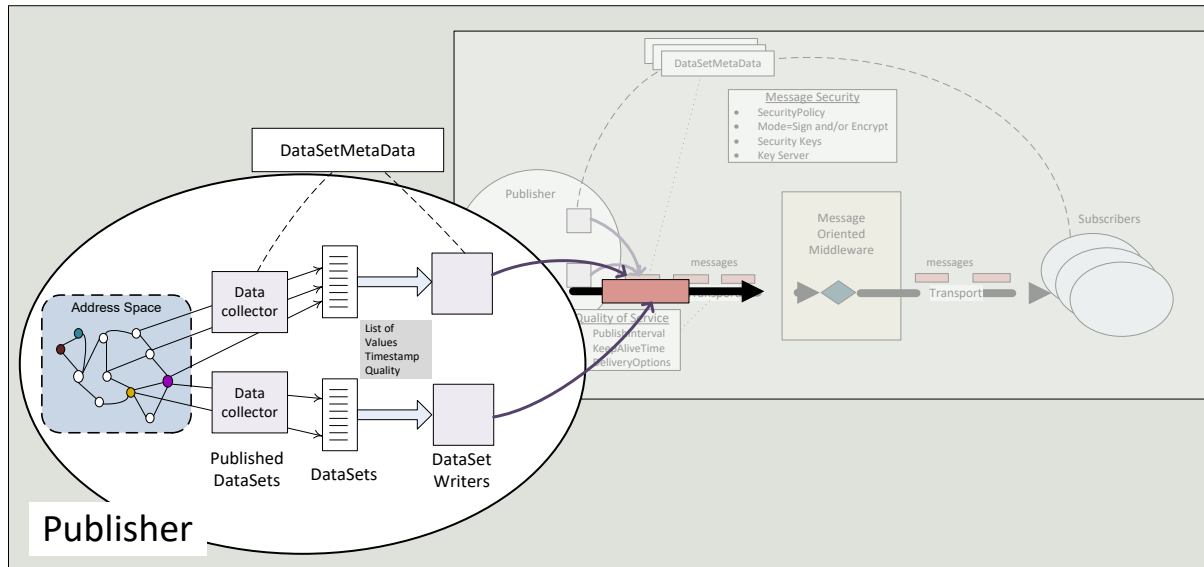


Figure 5 – Publisher details

A single *Publisher* may support multiple *PublishedDataSets* and multiple *DataSetWriters* to one or more *Message Oriented Middleware*. A *DataSetWriter* is a logical component of a *Publisher*. See 5.4.1.2 for further information about the *DataSet* writing process.

If the *Publisher* is an OPC UA *Server*, it can expose the *Publisher* configuration in its *AddressSpace*. This information may be created through product-specific configuration tools or through the OPC UA defined *Methods*. The OPC UA *Information Model* for *PubSub* configuration is specified in Clause 9.

5.4.1.2 Message sending

Figure 6 illustrates the process inside a *Publisher* when creating and sending messages and the parameters required to accomplish it. The components, like *DataSet* collection or *DataSetWriter* should be considered abstract. They may not exist in every *Publisher* as independent entities. However, comparable processes need to exist to generate the OPC UA *PubSub* messages.

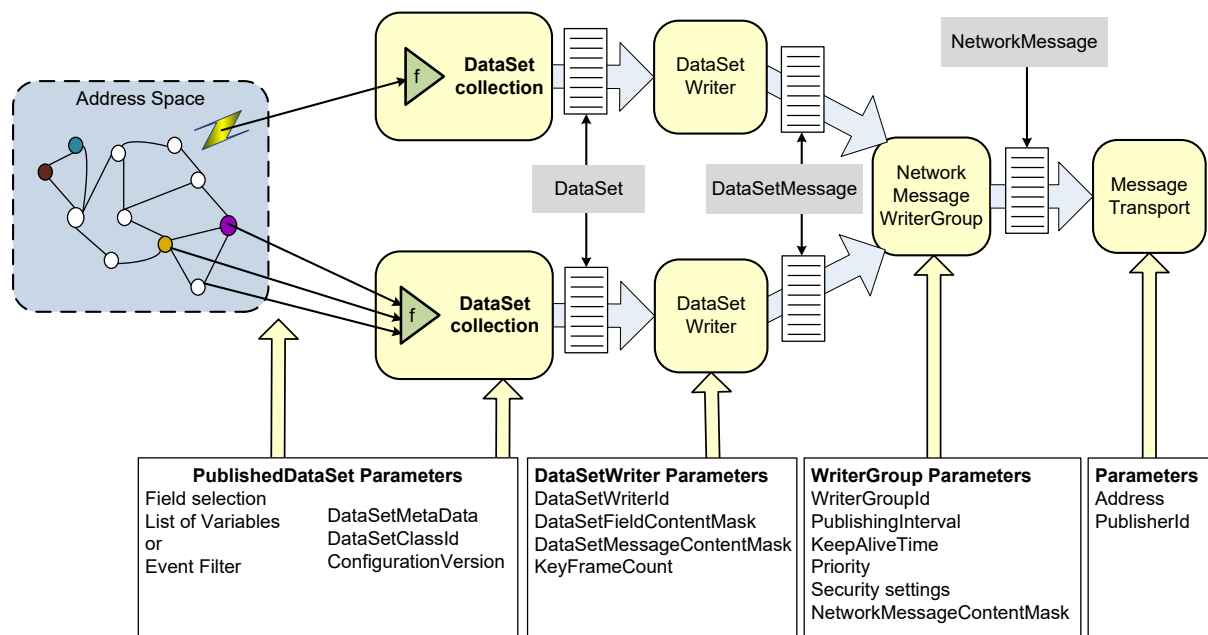


Figure 6 – Publisher message sending sequence

The sending process is guided by different parameters for different logical steps. The parameters define for example when and how often to trigger the sending sequence and the encoding and security of the messages. The PubSub communication parameters are defined in Clause 6.

The first step is the collection of data (*DataSet*) to be published. The configuration for such a collection is called *PublishedDataSet*. The *PublishedDataSet* also defines the *DataSetMetaData*. Collection is a generic expression for various different options, like monitoring of *Variables* in an OPC UA *Server AddressSpace*, processing OPC UA *Events*, or for example reading data from network packets. In the end, the collection process produces values for the individual fields of a *DataSet*. The two concrete *PublishedDataSet* options with standard OPC UA configuration are *PublishedDataItems* for *Variable* base collection and *PublishedEvents* for *Event* based collection.

In the next step, a *DataSetWriter* takes the *DataSet* and creates a *DataSetMessage*. *DataSetMessages* from *DataSetWriters* in one *WriterGroup* can be inserted into a single *NetworkMessage*. The creation of a *DataSetMessage* is guided by the following parameters:

- The *DataSetFieldContentMask* (see 6.2.4.2) controls which attributes of a value shall be encoded.
- The *DataSetMessageContentMask* (see 6.3.1.3.2) controls which header fields shall be encoded.
- The *KeyFrameCount* (see 6.2.4.3) greater than or equal to 1 controls whether a key frame or a delta frame *DataSetMessage* is to be created. A *KeyFrameCount* of 0 is used for non-cyclic *PublishedDataSets*, like *PublishedEvents*.

The resulting *DataSetMessage* is passed on to the next step together with the *DataSetWriterId* (see 6.2.4.1), the *DataSetClassId* (see 6.2.3.3), the *ConfigurationVersion* of the *DataSetMetaData* (see 6.2.3.2.6), and a list of values that match the configured propagated fields.

Next is the creation of the *NetworkMessage*. It uses the data provided from the previous step together with the *PublisherId* (see 6.2.7.1) defined on the *PubSubConnection*. The structure of this message is protocol specific. If the *SecurityMode* (see 6.2.5.2) requires message security, the *SecurityGroupId* (see 6.2.5.3) is used to fetch the *SecurityPolicy* and the security keys from

the SKS (see 5.4.4). This information is used to encrypt and/or sign the *NetworkMessage* as required by the *SecurityMode*.

The final step is delivery of the *NetworkMessage* to the *Message Oriented Middleware* through the configured *Address*.

5.4.2 Subscriber

5.4.2.1 General

Subscribers are the consumers of *NetworkMessages* from the *Message Oriented Middleware*. They may be OPC UA *Clients*, OPC UA *Servers* or applications that are neither *Client* nor *Server* but only understand the structure of OPC UA *PubSub* messages. Figure 7 illustrates a *Subscriber* with filtering, decoding and dispatching of *NetworkMessages*.

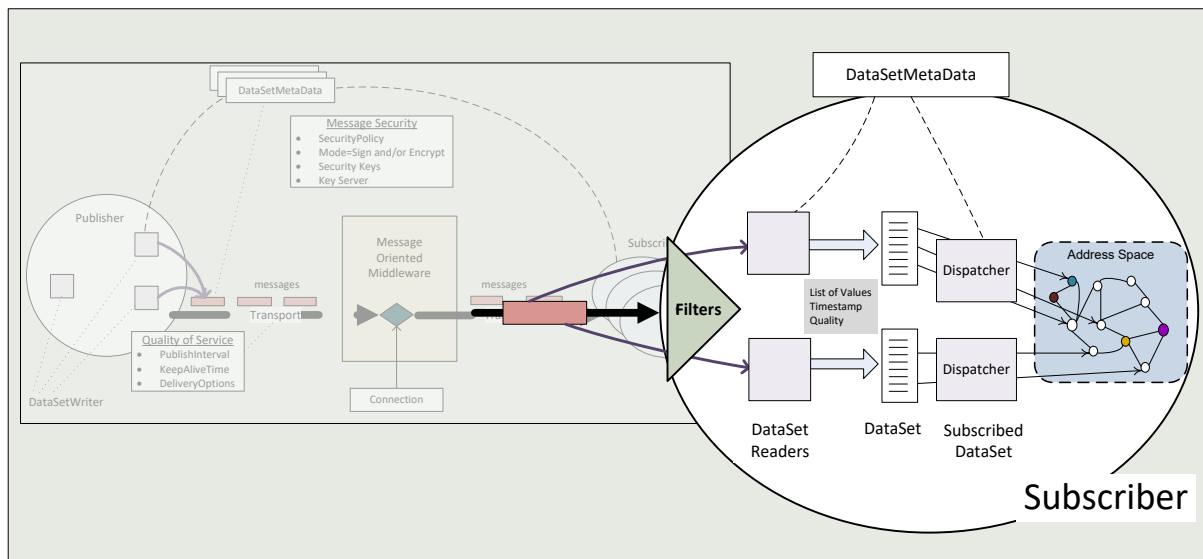


Figure 7 – Subscriber details

To determine for which *DataSetMessages* and on which *Message Oriented Middleware* to subscribe, the *Subscriber* need to be configured and/or use discovery mechanisms.

Subscribers shall be prepared to receive messages that they do not understand or are irrelevant. Each *NetworkMessage* provides unencrypted data in the *NetworkMessage* header to support identifying and filtering of relevant *Publishers*, *DataSetMessages*, *DataSetClasses* or other relevant message content (see 5.3).

If a *NetworkMessage* is signed or signed and encrypted, the *Subscriber* will need the proper security keys (see 5.3.5) to verify the signature and decrypt the relevant *DataSetMessages*.

Once a *DataSetMessage* has been selected as relevant, it will be forwarded to the corresponding *DataSetReader* for decoding into a *DataSet*. See 5.4.2.2 for further information about this *DataSet* reading process. The resulting *DataSet* is then further processed or dispatched in the *Subscriber*.

If the *Subscriber* is an OPC UA *Server*, it can expose the reader configuration in its *AddressSpace*. This information may be created through product-specific configuration tools or through the OPC UA defined configuration model. The OPC UA *Information Model* for *PubSub* configuration is specified in Clause 9.

5.4.2.2 Message reception

Figure 8 illustrates the process inside a *Subscriber* when receiving, decoding and interpreting messages and the parameter model required for accomplishing it. As for the *Publisher*, the components should be considered abstract.

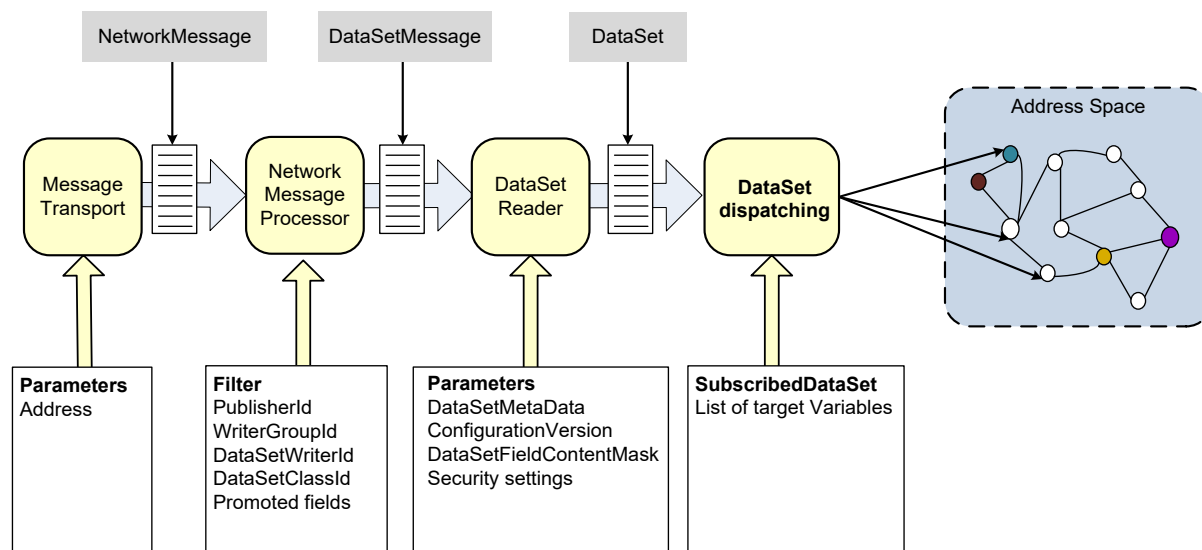


Figure 8 – Subscriber message reception sequence

The *Subscriber* need to select the required *Message Oriented Middleware* and establish a connection to it using the provided *Address*. Such a connection may simply be a multi-cast address when using OPC UA UDP or a connection to a message *Broker* when using MQTT or AMQP. Once subscribed, the *Subscriber* will start listening. The sequence starts when a *NetworkMessage* is received. The *Subscriber* may have configured filters (like a *PublisherId*, *DataSetWriterId* or a *DataSetClassId*) so that it can drop all messages that do not match the filter.

Once a *NetworkMessage* has been accepted, it is decrypted and decoded. The security parameters are the same as for the *Publisher*.

Each *DataSetMessage* of interest is passed on to a *DataSetReader*. Here, the *DataSetMetaData* is used to decode the *DataSetMessage* content to a *DataSet*. The *DataSetMetaData* in particular provides the complete field syntax including the name, data type, and other relevant *Properties* like engineering units. Version information that exists in both the *DataSetMessage* and the *DataSetMetaData* allows the *Subscriber* to detect version changes. If a major change occurs, the *Subscriber* needs to get an updated *DataSetMetaData*.

Any further processing is application-specific. For example, an additional dispatching step may map the received values to *Nodes* in the *Subscribers* OPC UA *AddressSpace*. The configuration for such a dispatching is called *SubscribedDataSet*. The two concrete *SubscribedDataSet* options with standard OPC UA configuration are *TargetVariables* and *SubscribedDataSetMirror*. The configuration of *TargetVariables* allows the dispatching of *DataSetMessage* fields to existing *Variables* in the *Subscribers* OPC UA *AddressSpace*. The configuration of *SubscribedDataSetMirror* is used if the received *DataSet* fields should be represented as *Variables* in the *Subscribers* OPC UA *AddressSpace* but the *Variables* do not exist and must be created as part of the *Subscriber* configuration.

5.4.3 Configuration Tool

An OPC UA *Application* can be pre-configured to send messages as a *Publisher* but commonly it is required to configure the information to be included into messages and also the frequency the messages are sent.

Subscribers can use discovery mechanisms to find *Publishers* and to get the *DataSetMetaData* necessary to understand the messages. One example are HMI applications where the configuration can be done inside the *Subscriber*. But if the *Subscriber* is a device, it is expected that a configuration tool is required to configure the *Subscriber* functionality in the device.

The *PubSubConfigurationDataType* and the other configuration *Structures* defined in Clause 6 can be used to prepare an offline *PubSub* configuration that can be stored in a binary file using the *UABinaryFileType*. Such a configuration can be used to configure *Publishers* and

1175 *Suscribers* if they do not have a online configuration interface or are configured through product-
1176 specific configuration tools.

1177 If *Publishers* and *Suscribers* are also OPC UA *Servers*, they can provide the *PubSub*
1178 configuration *Information Model* defined in Clause 9. This model can be used by generic *PubSub*
1179 configuration tools.

1180 A typical use case is controller to controller or machine to machine communication where both
1181 communication partners have a pre-configured list of input and output data *Variables* and a
1182 generic configuration tool establishes the communication by selecting the *Variables* to be
1183 published in the *Publisher* and then configures the *Subscriber* to receive the messages from
1184 the *Publisher* and to select the target *Variables* in the *Subscriber*.

1185 **5.4.4 Security Key Service**

1186 **5.4.4.1 General**

1187 A *Security Key Service* (SKS) provides keys for message security that can be used by the
1188 *Publisher* to sign and encrypt *NetworkMessages* and by the *Subscriber* to verify the signature
1189 of *NetworkMessages* and to decrypt them.

1190 The SKS is responsible for managing the keys used to publish or consume *PubSub*
1191 *NetworkMessages*. Separate keys are associated with each *SecurityGroupId* in the system. The
1192 *GetSecurityKeys Method* exposed by the SKS shall be called to receive necessary key material
1193 for a *SecurityGroupId*. *GetSecurityKeys* can return more than one key. In this case the next key
1194 can be used when the current key is outdated without calling *GetSecurityKeys* for every key
1195 needed. The *PubSubKeyServiceType* defined in 8.2 specifies the *GetSecurityKeys Method*.

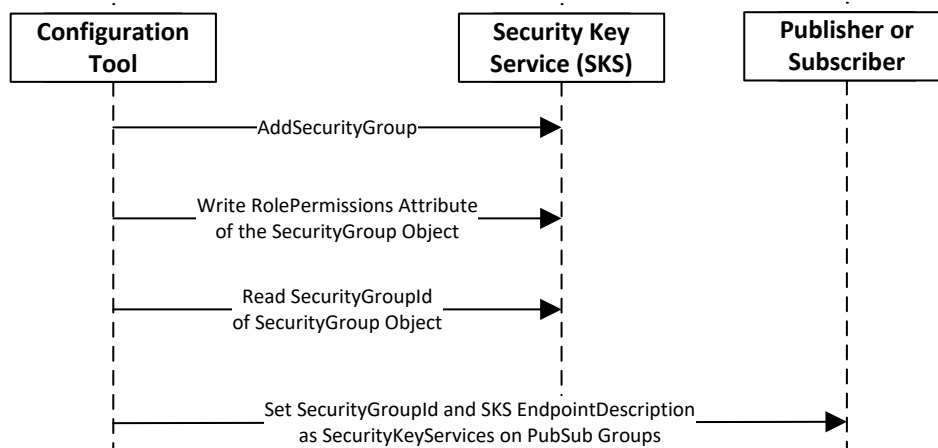
1196 The *GetSecurityKeys Method* can be implemented by a *Publisher* or by a central SKS. In both
1197 cases, the well-known *NodeIds* for the *PublishSubscribe Object* and the related
1198 *GetSecurityKeys Method* are used to call the *GetSecurityKeys Method*. The *PublishSubscribe*
1199 *Object* is defined in 8.3.2.

1200 The *SetSecurityKeys Method* is typically used by a central SKS to push the security keys for a
1201 *SecurityGroup* into a *Publisher* or *Subscriber*. The *Method* is exposed by *Publishers* or
1202 *Subscribers* that have no OPC UA *Client* functionality. The *Method* is part of the
1203 *PublishSubscribeType* defined in 9.1.3.2.

1204 **5.4.4.2 SecurityGroup Management**

1205 The SKS is the entity with knowledge of *SecurityGroups* and it maintains a mapping between
1206 *Roles* and *SecurityGroups*. The related *User Authorization* model is defined in IEC 62541-3.
1207 The *User Authorization* model defines the mapping of identities to *Roles* and the mechanism to
1208 set *Permissions* for *Roles* on a *Node*. The *Permissions* on a *SecurityGroup Object* is used to
1209 determine if a *Role* has access to the keys for the *SecurityGroup*.

1210 An example for setting up a *SecurityGroup* and the configuration of affected *Publishers* and
1211 *Subscribers* is shown in Figure 9.



1212

1213

Figure 9 – SecurityGroup management sequence

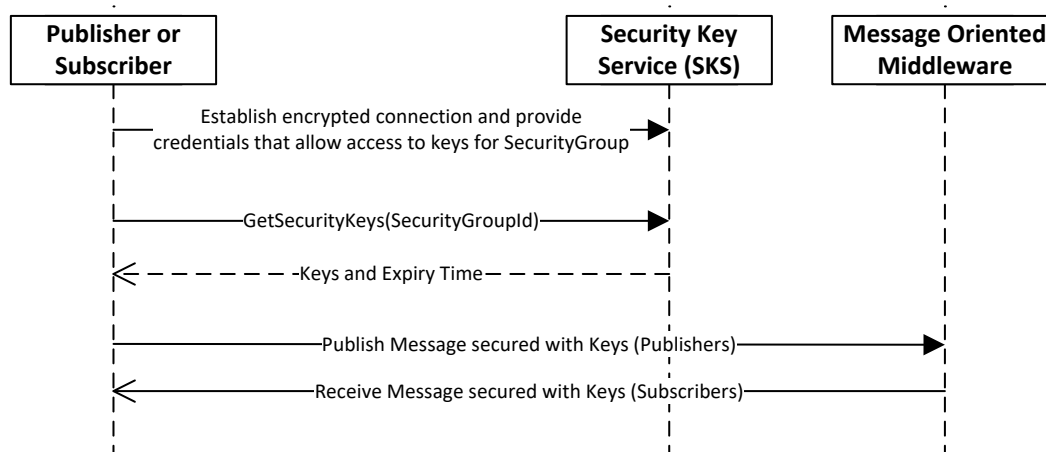
1214 To secure *NetworkMessages*, the *NetworkMessages* shall be secured with keys provided in the
 1215 context of a *SecurityGroup*. A *SecurityGroup* is created on a SKS using the *Method*
 1216 *AddSecurityGroup*.

1217 To limit access to the *SecurityGroup* and therefore to the security keys, *Permissions* shall be
 1218 set on the *SecurityGroup Object*. This requires the management of *Roles* and *Permissions* in
 1219 the SKS.

1220 To set the *SecurityGroup* relation on the *Publishers* and *Subscribers*, the *SecurityGroupId* and
 1221 the SKS *EndpointDescriptions* are configured in a *PubSub* group.

1222 5.4.4.3 Key acquisition handshakes

1223 The *Publisher* or *Subscriber* use keys provided by an SKS to secure messages exchanged via
 1224 the *Message Oriented Middleware*. The handshake to pull the keys from a SKS is shown in
 1225 Figure 10. The handshake to push the keys from an SKS to *Publishers* and *Subscribers* is
 1226 shown in Figure 11.



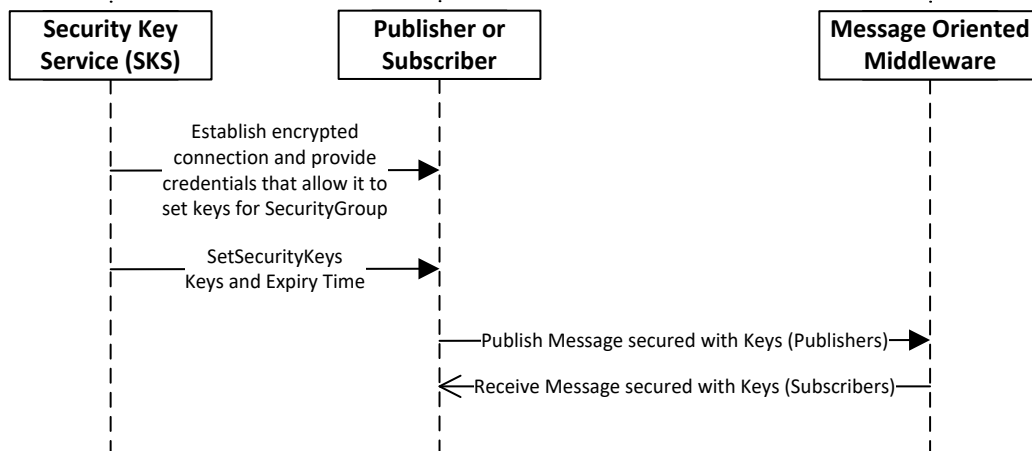
1227

1228

Figure 10 – Handshake used to pull keys from SKS

1229 To pull keys, the *Publisher* or *Subscriber* creates an encrypted connection and provides
 1230 credentials that allow it access to the *SecurityGroup*. Then it passes the identifier of the
 1231 *SecurityGroup* to the *GetSecurityKeys Method* that verifies the *identity* and returns the keys
 1232 used to secure messages for the *PubSubGroup*. The *GetSecurityKeys Method* is defined in
 1233 8.3.2.

1234 The access to the *GetSecurityKeys Method* may use *SessionlessInvoke Service* calls. These
 1235 calls typically use an *Access Token* that is retrieved from an *Authorization Service*. Both
 1236 concepts are defined in IEC 62541-4.



1237

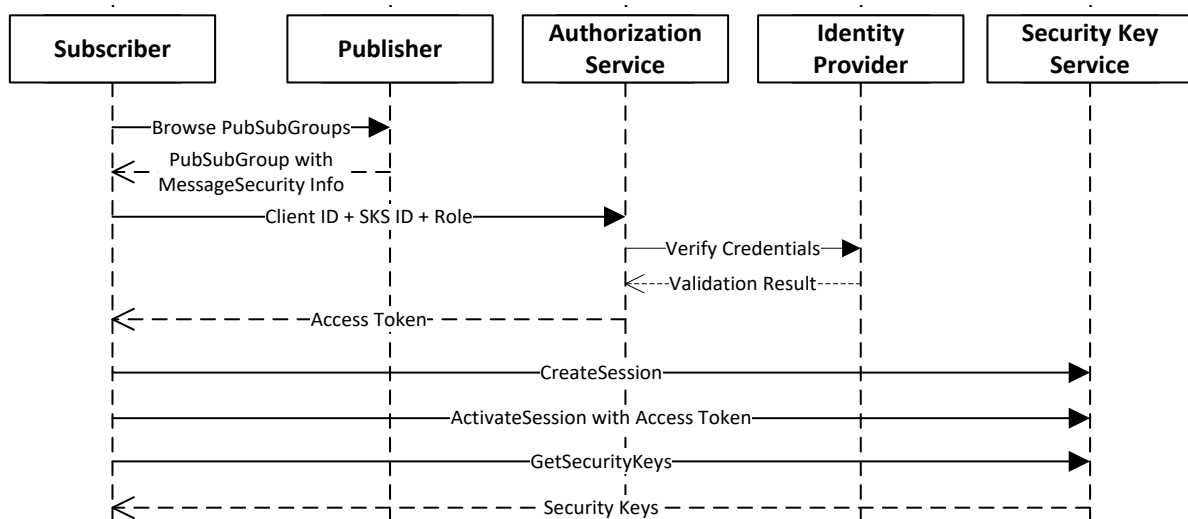
1238 **Figure 11 – Handshake used to push keys to Publishers and Subscribers**

1239 To push keys, the SKS creates an encrypted connection to a *Publisher* or *Subscriber* and
 1240 provides credentials that allow it to provide keys for a *SecurityGroup*. Then it passes the
 1241 identifier of the *SecurityGroup* and the keys used to secure messages for the *SecurityGroup* to
 1242 the *SetSecurityKeys Method*. The *SetSecurityKeys Method* is defined in 9.1.3.3.

1243 If the initial pull or push fails, the affected *PubSub* components like *WriterGroup* or
 1244 *DataSetReader* stay in the *PreOperational* state. If the updates fail and the *PubSub* components
 1245 do not have up to date key material, the state of the affected components change to *Error*. For
 1246 both pull and push, the *Client* executing the key exchange needs to retry the key exchange at
 1247 a faster rate than the key lifetime.

1248 5.4.4.4 Authorization Services and Security Key Service

1249 Access to the SKS can be managed by an *Authorization Service* as shown in Figure 12.



1250

1251 **Figure 12 – Handshake with a Security Key Service**

1252 The SKS is a *Server* that exposes a *Method* called *GetSecurityKeys*. The *Access Token* is used
 1253 to determine if the calling application is allowed to access the keys. One way to do this would

be to check the *Permissions* assigned to the *SecurityGroup Object* identified by the *GetSecurityKeys Method* arguments. *Publishers* and *Subscribers* can request keys if the *Access Token* they provide is mapped to *Roles* that have been granted *Permission* to *Browse* the *SecurityGroup Object*.

5.4.5 Message Oriented Middleware

5.4.5.1 General

Message Oriented Middleware as used in this document is any infrastructure supporting sending and receiving *NetworkMessages* between distributed applications. OPC UA does not define a *Message Oriented Middleware*, rather it uses protocols that allow connecting, sending and receiving data. The transport protocol mappings for *PubSub* are described in 7.3.

This document describes two general types of *Message Oriented Middleware* to cover a large number of use cases. The two types, broker-less and broker-based middleware, are described in 5.4.5.2 and 5.4.5.3.

5.4.5.2 Broker-less Middleware

5.4.5.2.1 General

With this option, OPC UA *PubSub* relies on the network infrastructure to deliver *NetworkMessages* to one or more receivers. Network devices – like network routers, switches, or bridges – are typically used for this purpose.

One example is a switched network and the use of UDP with unicast or multicast messages shown in Figure 13.

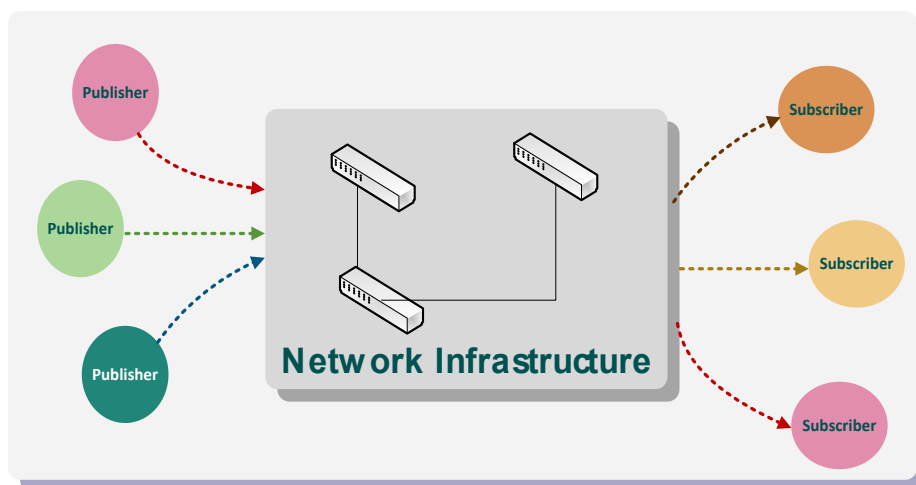


Figure 13 – PubSub using network infrastructure

Advantages of this model include:

- Only requires standard network equipment and no additional software components like a *Broker*.
- Message delivery is assumed to be direct without software intermediaries and therefore provides reduced latency and overhead.
- UDP protocol supports multiple subscribers using multicast addressing.

5.4.5.2.2 Broker-less model with OPC UA UDP

Figure 14 depicts the applications, entities and messages involved in peer-to-peer communication using UDP as a protocol that does not require a *Broker*.

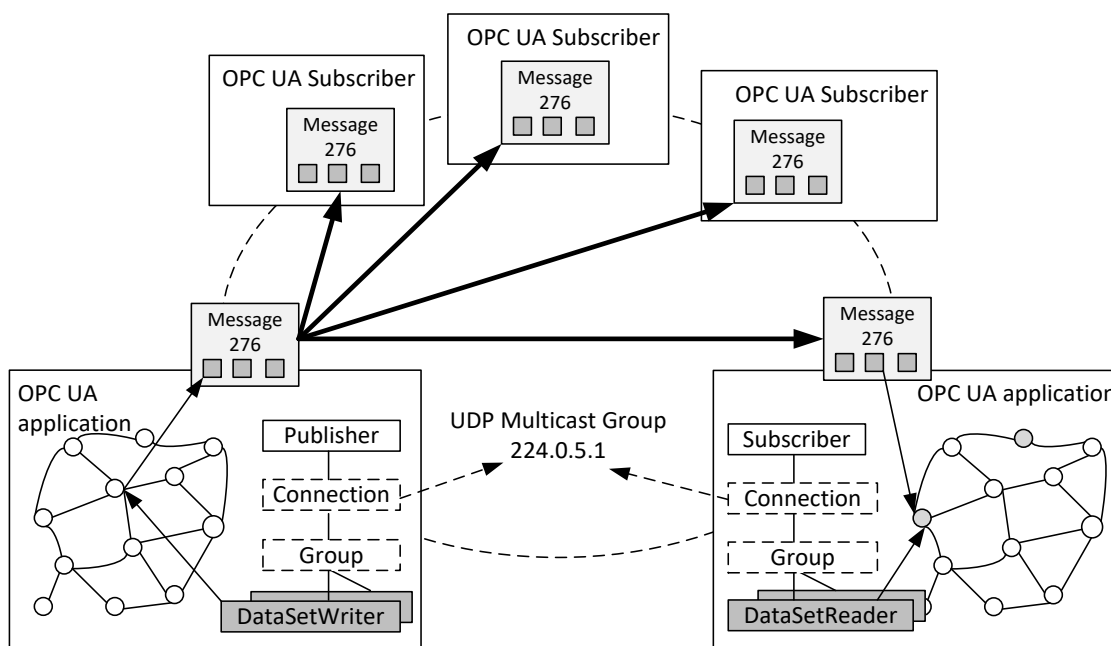


Figure 14 – UDP Multicast overview

The *PublishSubscribe Object* contains a connection *Object* for each address like an IP multicast address. The connection can have one or more groups with *DataSetWriters*. A group can publish *DataSets* at the defined publishing interval.

In each publishing interval, a *DataSet* is collected for a *PublishedDataSet* which can be a list of sampled data items in the *Publisher OPC UA Address Space*. For each *DataSet* a *DataSetMessage* is created. The *DataSetMessages* are sent in a *NetworkMessage* to the IP multicast address.

OPC UA *Applications* like HMI applications would use the values of the *DataSetMessage* that they are interested in.

An OPC UA *Application* that maps data fields from UADP *DataSetMessages* to internal *Variables* can be configured through the *DataSetReader Object* and dispatcher in the *Subscriber*. The configuration of a *DataSetReader* defines how to decode the *DataSetMessage* to a *DataSet*. The *SubscribedDataSet* defines which field in the *DataSet* is mapped to which *Variable* in the OPC UA *Application*.

With OPC UA UDP there is no guarantee of timeliness, delivery, ordering, or duplicate protection. The sequence numbers in *DataSetMessages* provide a solution for ordering and duplicate detection. The reliability is improved by the option to send the complete *DataSet* in every *DataSetMessage* or with the option to repeat *NetworkMessages*.

Other transport protocol mappings used with the broker-less model could provide guarantee of timeliness, delivery, ordering, or duplicate protection.

5.4.5.3 Broker-based Middleware

5.4.5.3.1 General

This option assumes a messaging *Broker* in the middle as shown in Figure 15. No application is speaking directly to other applications. All the communication is passed through the *Broker*. The *Broker* routes the *NetworkMessages* to the right applications based on business criteria ("queue name", "routing key", "topic" etc.) rather than on physical topology (IP addresses, host names).

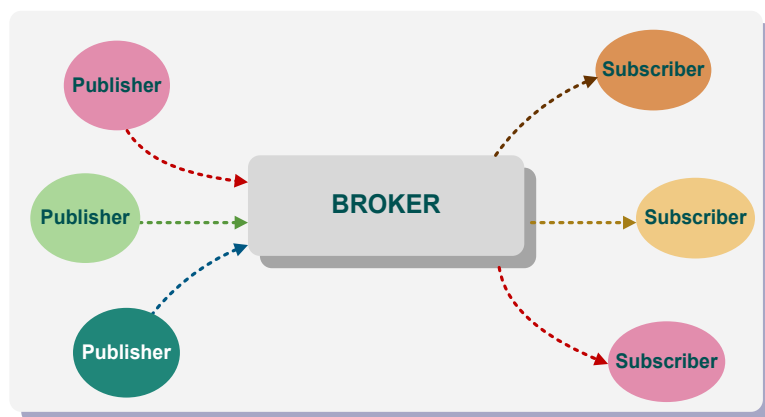


Figure 15 – PubSub using broker

Advantages of this model (partly depending on used *Broker* and its configuration) include:

- *Publisher* and *Subscriber* do not have to be directly addressable. They can be anywhere as long as they have access to the *Broker*.
- Fan out can be handled to a very large list of *Subscribers*, multiple networks or even chained *Brokers* or scalable *Brokers*.
- *Publisher* and *Subscriber* lifetimes do not have to overlap. The *Publisher* application can push *NetworkMessages* to the *Broker* and terminate. The *NetworkMessages* will be available for the *Subscriber* application later.
- *Publisher* and *Subscriber* can use different messaging protocols to communicate with the *Broker*.

In addition, the *Broker* model is to some extent resistant to the application failure. So, if the application is buggy and prone to failure, the *NetworkMessages* that are already in the *Broker* will be retained even if the application fails.

5.4.5.3.2 Broker-based model

Figure 16 depicts the applications, entities and messages involved in typical communication scenarios with a *Broker*. It requires use of messaging protocols that a *Broker* understands, like AMQP defined in ISO/IEC 19464:2014 or MQTT defined in ISO/IEC 20922:2016. In this model the *Message Oriented Middleware* will be a *Broker* that relays *NetworkMessages* from *Publishers* to *Subscribers*. The *Broker* may also be able to queue messages and send the same message to multiple *Subscribers*.

Note that the *Broker* functionality is outside the scope of this document. In terms of the messaging protocols, the *Broker* is a messaging server (the OPC UA *Publisher* and the OPC UA *Subscriber* are messaging clients). The messaging protocols define how to connect to a messaging server and what fields in a message influence the *Broker* functionality.

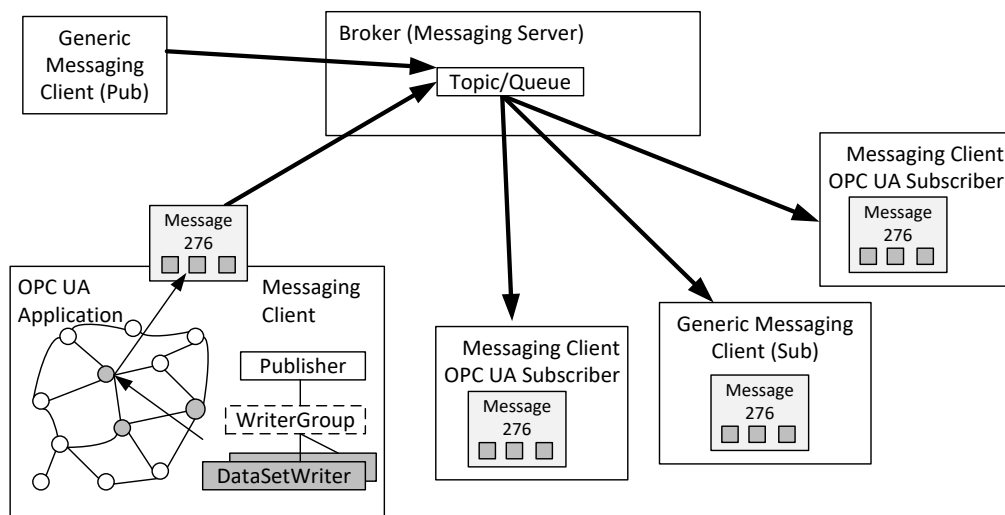


Figure 16 – Broker overview

An OPC UA *Publisher* that publishes data may be configured through the *PubSub* configuration model. It contains one connection *Object* per *Broker*. The *Broker* is configured through an URL in the connection. The connection can have one or more groups which identify specific queues or topics. Each group may have one or more *DataSetWriters* that format a *DataSet* as required for the messaging protocol. A *DataSet* can be collected from a list of *Event* fields and/or selected *Variables*. Such a configuration is called *PublishedDataSet*.

Each *DataSet* is sent as a separate *DataSetMessage* serialized with a format that depends on the *DataSetWriter*. One *DataSetMessage* format is the JSON message mapping which represents the *DataSet* in a format which *Subscribers* can understand without knowledge of OPC UA. Another *DataSetMessage* format is the UADP message mapping.

Message confidentiality and integrity with the *Broker* based communication model can be ensured at two levels:

- transport security between *Publishers* or *Subscribers* and the *Broker*, or
- message security as end-to-end security between *Publisher* and *Subscriber*.

The *Broker* level security requires all *Publishers* and *Subscribers* to have credentials that grant them access to the necessary queue or topic. In addition, all communication with the *Broker* uses transport level security to ensure confidentiality. The security parameters are specified on the connection and group.

The message security provided by the *Publisher* is only defined for the UADP message mapping.

5.4.5.4 QoS configuration

OPC UA *Applications* may demand Quality of Service (QoS) for the transport of *NetworkMessages*. These QoS requirements have to be fulfilled by the broker-less *Message Oriented Middleware* and therefore need to be mapped to concrete network technologies like TSN, Deterministic Networking (DetNet) or differentiated services (DiffServ). This mapping should be hidden to the application engineer from a PubSub perspective but may be monitored or configurable via the information model of IEC 62541-22.

Priority based QoS

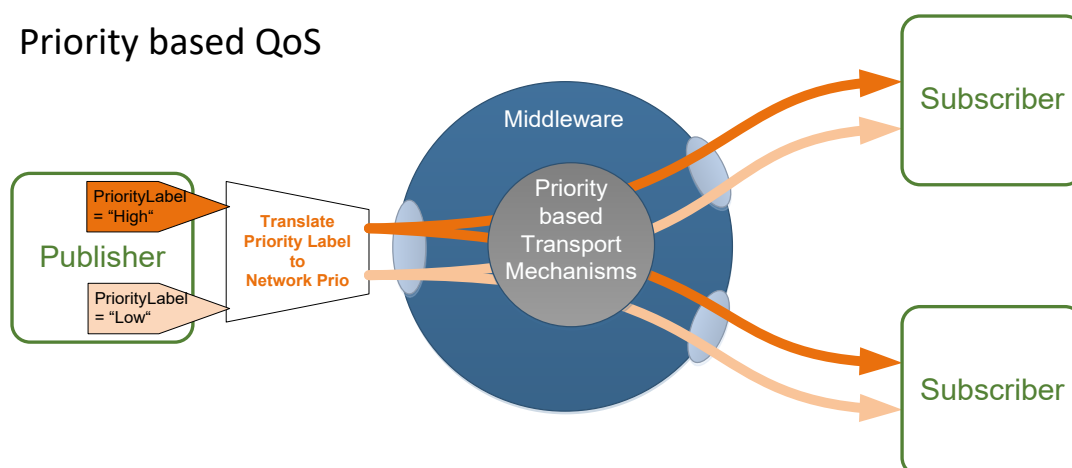


Figure 17 – Message Oriented Middleware providing QoS

QoS requirements of an OPC UA *Applications* shall be configurable with OPC UA means and without dependencies to the underlying network technology. Hiding network details from the application simplifies to migrate OPC UA *Applications* from one network technology to another or to interconnect OPC UA *Applications* over different network technologies.

QoS requirements can be fulfilled by different network mechanisms and may require different QoS control mechanisms in the network depending on the requested level of QoS.

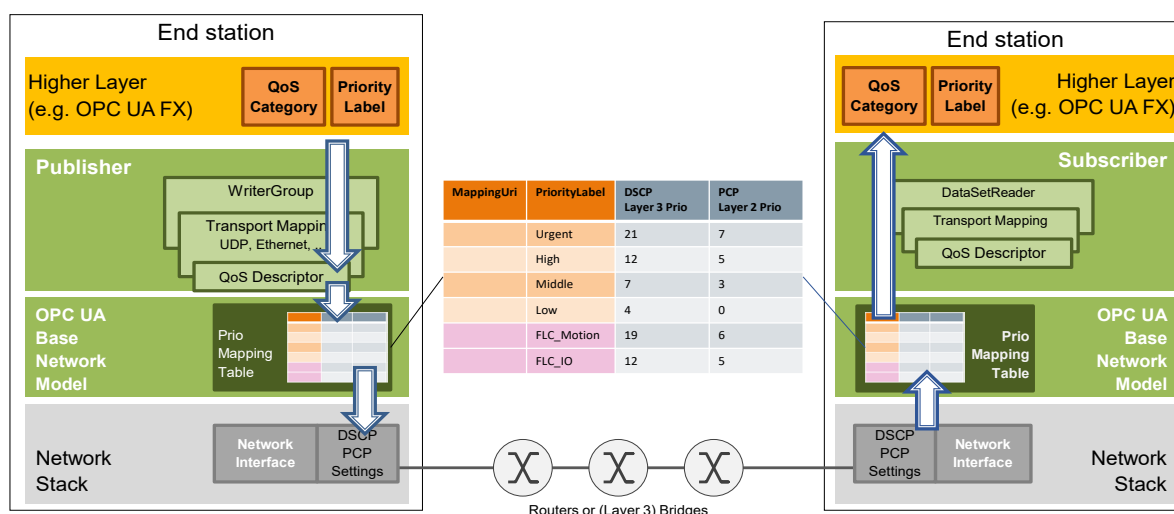


Figure 18 – Mapping of priority-based QoS

As shown in Figure 17 and Figure 18 the *PriorityLabel* from a *WriterGroup*, *DataSetReader* or *PubSubConnection TransportSettings* will be translated into actual values to be used on the wire. Together with the optional *QosCategory* the *PriorityLabel* will be present in the mapping table for the network interface used to transmit the data. If the combination of *QosCategory* and *PriorityLabel* is not present in the mapping table, the communication cannot be established. The reference from the network interface to the mapping table is defined in IEC 62541-22.

Standard values for *QosCategory* with the according required structures in the *DatagramQos* array are defined in Table 103. This list can be extended by specifications built on top of OPC UA *PubSub*. Each *QosCategory* will be described in detail by a list of measurable QoS KPIs like assured bandwidth or maximum latency in the parameter *DatagramQos*.

6 PubSub communication parameters

6.1 Overview

PubSub defines different configuration parameters for the various *PubSub* components. They define the behaviour of *Publisher* and *Subscriber*. The parameters are grouped by component and are partitioned into 'common', 'message mapping', and 'transport protocol mapping'.

The common parameters are defined in 6.2. The parameters for the different message mappings are defined in 6.3. The parameters for the different transport protocol mappings are defined in 6.4.

The application of communication parameters for concrete message and transport protocol mappings is defined in Clause 7.

Configuration of these parameters can be performed through the OPC UA *Information Model* for *PubSub* configuration defined in Clause 9 or through vendor-specific mechanisms. The parameter groupings in this clause define the parameters and also define *Structures* used to represent the parameters of the groupings. These *Structures* are used in the *PubSub* configuration model described in Clause 9 but they can also be used for offline configuration or vendor-specific configuration mechanisms.

Figure 19 depicts the different components and their relation to each other. The *WriterGroup*, *DataSetWriter* and *PublishedDataSet* components define the data acquisition for the *DataSets*, the message generation and the sending on the *Publisher* side. These parameters need to be known on the *Subscriber* side to configure *DataSetReaders* and to filter and process *DataSetMessages*.

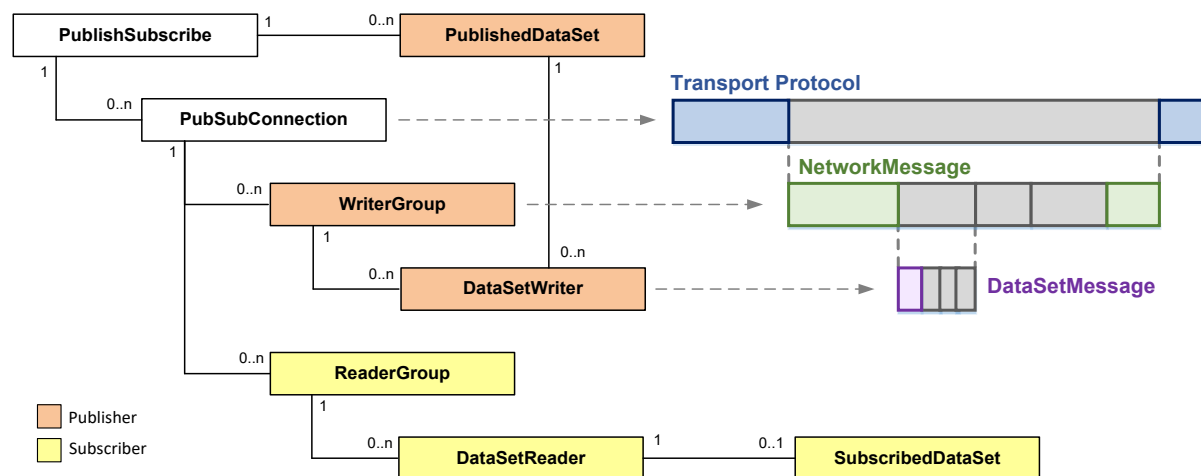


Figure 19 – PubSub component overview

Figure 19 shows the following components:

- *PublishedDataSet* contains the *DataSetMetaData* describing the content of the *DataSets* produced by the *PublishedDataSet* and the corresponding data acquisition parameters.
- *DataSetWriter* parameters are necessary for creating *DataSetMessages*. Each *DataSetWriter* is bound to a single *PublishedDataSet*. A *PublishedDataSet* can have multiple *DataSetWriters*.
- *WriterGroup* parameters are necessary for creating a *NetworkMessage*. Each writer group can have one or more *DataSetWriters*. Some of these parameters are used for creating the *DataSetMessages*. They are grouped here since they are the same for all *DataSetMessages* in a single *NetworkMessage*.

- *PubSubConnection* parameters represent settings needed for the transport protocol. One connection can have a number of writer groups and reader groups.
- *ReaderGroup* is used to group a list of *DataSetReaders* and contains a few shared settings for them. It is not symmetric to a *WriterGroup* and it is not related to a particular *NetworkMessage*. The *NetworkMessage* related filter settings are on the *DataSetReaders*.
- *DataSetReader* parameters represent settings for filtering of received *NetworkMessages* and *DataSetMessages* as well as settings for decoding of the *DataSetMessages* of interest.
- *SubscribedDataSet* parameters define the processing of the decoded *DataSet* in the *Subscriber* for one *DataSetReader*.
- *PublishSubscribe* is the overall management of the *PubSub* groupings. It contains a list of *PublishedDataSets* and a list of *PubSubConnections*.

The different PubSub mapping specific parameter groupings are shown in Figure 20.

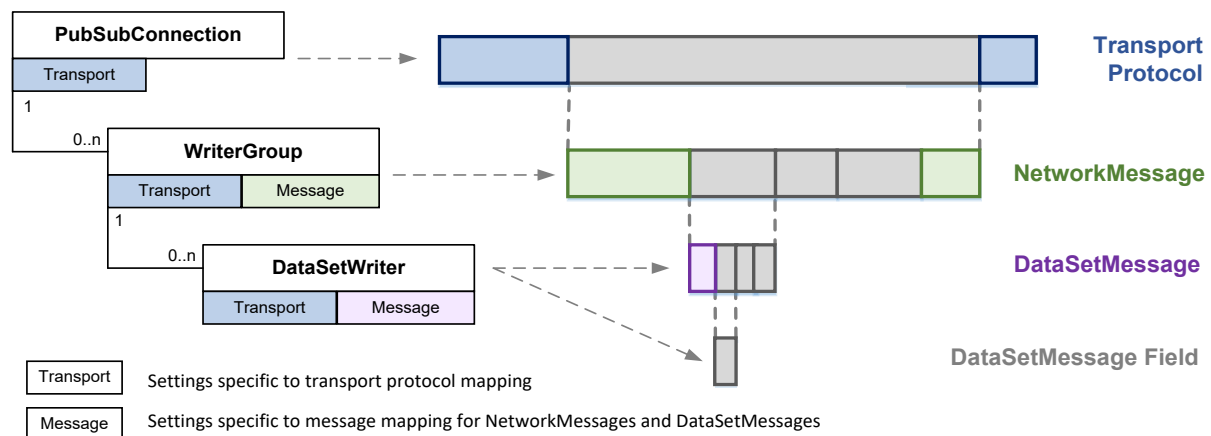


Figure 20 – PubSub mapping specific parameters overview

Transport protocol mapping specific parameters may be defined for the *PubSubConnection*, the *WriterGroup* or the *DataSetWriter*.

Message mapping specific parameters are defined for the *NetworkMessages* on the *WriterGroup* and for the *DataSetMessages* on the *DataSetWriter*.

6.2 Common configuration parameters

6.2.1 PubSubState state machine

The *PubSubState* is used to expose and control the operation of a *PubSub* component. It is an enumeration of the possible states. The enumeration values are described in Table 1.

Table 1 – PubSubState values

Name	Value	Description
Disabled	0	The <i>PubSub</i> component is configured but currently disabled.
Paused	1	The <i>PubSub</i> component is enabled but currently paused by a parent component. The parent component is either <i>Disabled</i> or <i>Paused</i> .
Operational	2	The <i>PubSub</i> component is operational.
Error	3	The <i>PubSub</i> component is in an error state.
PreOperational	4	The <i>PubSub</i> component is enabled but currently in the process to execute the steps necessary to enter the <i>Operational</i> state.

Figure 21 depicts the *PubSub* components that have a *PubSub* state and their parent-child relationship. State changes of children are based on changes of the parent state. The root of the hierarchy is the *PublishSubscribe* component.

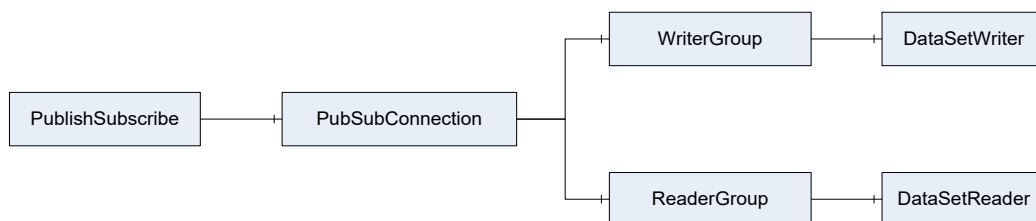


Figure 21 – PubSub component state dependencies

Figure 22 describes the formal state machine with the possible transitions.

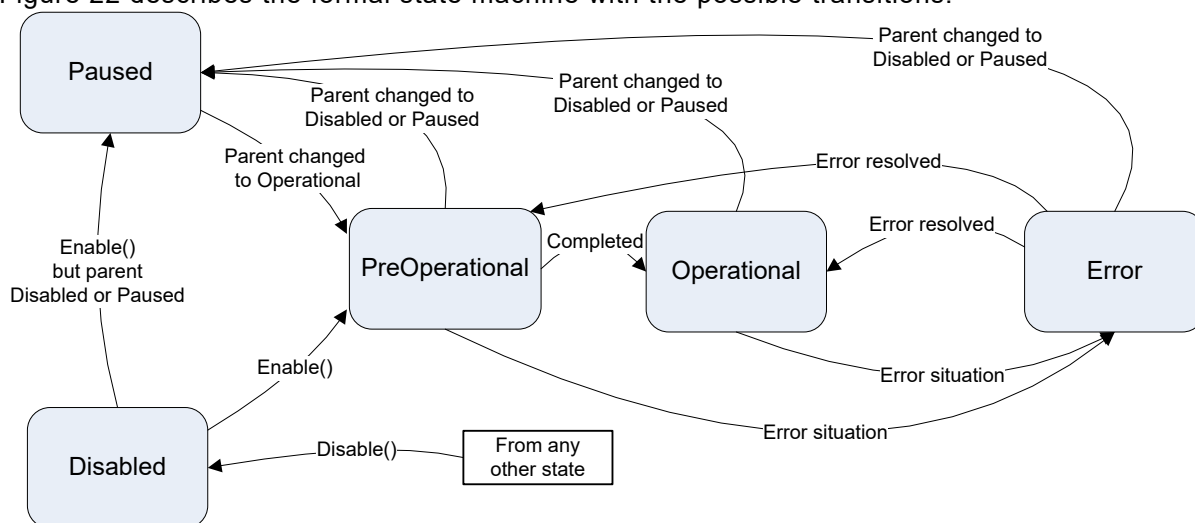


Figure 22 – PubSubState state machine

Table 2 formally defines the transitions of the state machine.

Table 2 – PubSubState state machine

Source State	Target State	Trigger Description
Disabled	Paused	The component was successfully enabled but the parent component is in the state <i>Disabled</i> or <i>Paused</i> .
Disabled	PreOperational	The component was successfully enabled.
Paused	Disabled	The component was successfully disabled.
Paused	PreOperational	The state of the parent component changed to <i>Operational</i> .
PreOperational	Operational	The component completed the steps necessary to enter the <i>Operational</i> state. These steps include setup of network communication and security keys. If the <i>PubSub</i> component is a <i>DataSetReader</i> , the state shall change to <i>Operational</i> after the first key frame or event <i>DataSetMessage</i> was received.
PreOperational	Disabled	The component was successfully disabled.
PreOperational	Paused	The state of the parent component changed to <i>Disabled</i> or <i>Paused</i> .
PreOperational	Error	There is a pending error situation for the related <i>PubSub</i> component.
Operational	Disabled	The component was successfully disabled.
Operational	Paused	The state of the parent component changed to <i>Disabled</i> or <i>Paused</i> .
Operational	Error	There is a pending error situation for the related <i>PubSub</i> component.
Error	Disabled	The component was successfully disabled.
Error	Paused	The state of the parent component changed to <i>Disabled</i> or <i>Paused</i> .
Error	Operational	The error situation was resolved for the related <i>PubSub</i> component.
Error	PreOperational	The error situation was resolved for the related <i>PubSub</i> component.

The *PubSubState* representation in the *AddressSpace* is defined in Table 3.

Table 3 – PubSubState definition

Attribute	Value				
BrowseName	PubSubState				
IsAbstract	False				
References	Node Class	BrowseName	DataType	TypeDefinition	Others
Subtype of Enumeration defined in IEC 62541-5					
HasProperty	Variable	EnumStrings	LocalizedText []	PropertyType	
Conformance Units					
PubSub Model Base					

6.2.2 PubSub configuration properties

The *PubSub* components have configuration property lists as *KeyValuePair* arrays. These optional configuration properties extend the configuration parameters defined for the different *PubSub* components.

The configuration properties are mainly used as protocol or product specific parameters influencing the behaviour of the PubSub application. These properties may contain information that should not be visible outside the configuration tools and PubSub applications. Therefore the properties shall not be included in the *PubSubConnectionDataType*, *WriterGroupDataType* and *DataSetWriterDataType* when sent in discovery messages.

The configuration are defined in different sections in the scope where they are used. The properties that can be applied to all PubSub components are defined in Table 4.

The *NamespaceIndex* of the *QualifiedName* in the *KeyValuePair* for properties defined in this document shall be 0. The *Name* of the *QualifiedName* is the property name from Table 4. The *DataType* of the *Value* in the *KeyValuePair* shall be the *DataType* defined in Table 4.

Table 4 formally defines the general *PubSub* configuration properties.

Table 4 – General PubSub configuration properties

Key	DataType	Description
0:NotPersisted	Boolean	Indicates if the component configuration is persisted. If this property is not present the default value is false and the component configuration is persisted.
0>ErrorDelay	Duration	A time delay in milliseconds from the time the first error is detected until transitioning to the <i>PubSubState Error</i> .

6.2.3 PublishedDataSet parameters

6.2.3.1 Overview

A *PublishedDataSet* defines the content of a *DataSetMessage* and the configuration of the information source for a *DataSet*. See 5.2 for the introduction to *DataSets*, 5.3 for the introduction to *DataSetMessages* and 5.4.1.2 for an introduction to the different source options and the parameters for sending of *DataSetMessages*.

The content of a *DataSetMessage* is defined by the *DataSetMetaData*. This information is required for interoperability between *Publisher* and *Subscriber*. See 6.2.3.2.

The information source is only necessary for the configuration of the *Publisher*. The standard configuration options are published data items for cyclic *DataSets* as defined in 6.2.3.7 and published events for acyclic *DataSets* as defined in 6.2.3.8. OPC UA Applications can provide *PublishedDataSets* where the information source is application specific. The custom *PublishedDataSet* source *DataType* defined in 6.2.3.9 indicates if the *DataSet* is cyclic or acyclic. Cyclic *DataSets* are sent as key frame or delta frame *DataSetMessages*. Acyclic *DataSets* are sent as event *DataSetMessages*.

6.2.3.2 DataSetMetaData

6.2.3.2.1 General

DataSetMetaData describe the content and semantic of a *DataSet*. The order of the fields in the *DataSetMetaData* shall match the order of *DataSet* fields when they are included in the published *DataSetMessages*. The *DataSetMetaDataType* is defined in 6.2.3.2.3.

6.2.3.2.2 DataTypeSchemaHeader

The *DataSetMetaData* is a subtype of *DataTypeSchemaHeader*. The *DataTypeSchemaHeader* provides OPC UA *DataType* definitions used in the *DataSetMetaData*. The *DataTypeSchemaHeader* is defined in IEC 62541-5.

The *DataTypeSchemaHeader* provides information that is required when the *DataSetMetaData* is used outside the scope of an OPC UA Server *e.g* when sent in PubSub messages or when the *PubSubConfiguration* is exchanged with the *UABinaryFileType*. This information includes a namespace array and *DataType* descriptions. OPC UA namespace defined *DataTypes* are not included.

The *DataTypeSchemaHeader* of the *DataSetMetaData* shall be populated with the necessary information. This includes all namespaces and *DataTypes* that are potentially contained in the associated *DataSetMessages*.

6.2.3.2.3 DataSetMetaDataType

This *Structure DataType* is a subtype of *DataTypeSchemaHeader* and is used to provide the metadata for a *DataSet*. The *DataSetMetaDataType* is formally defined in Table 5.

Table 5 – DataSetMetaDataType structure

Name	Type	Description
DataSetMetaDataType	Structure	Subtype of <i>DataTypeSchemaHeader</i> defined in IEC 62541-5.
Name	String	Name of the <i>DataSet</i> .
Description	LocalizedText	Description of the <i>DataSet</i> . The default value is a null or empty <i>LocalizedText</i> .
Fields	FieldMetaData[]	The metadata for the fields in the <i>DataSet</i> . The <i>FieldMetaData DataType</i> is defined in 6.2.3.2.4.
DataSetClassId	Guid	This field provides the globally unique identifier of the class of <i>DataSet</i> if the <i>DataSet</i> is based on a <i>DataSetClass</i> . In this case, this field shall match the <i>DataSetClassId</i> of the concrete <i>DataSet</i> configuration. If the <i>DataSets</i> are not created from a class, this field is null.
ConfigurationVersion	ConfigurationVersionDataType	The configuration version for the current configuration of the <i>DataSet</i> .

Its representation in the AddressSpace is defined in Table 6.

Table 6 – DataSetMetaDataType definition

Attributes	Value
BrowseName	DataSetMetaDataType
IsAbstract	False
Subtype of <i>DataTypeSchemaHeader</i> defined in IEC 62541-5.	
Conformance Units	
PubSub Parameters Discovery	

6.2.3.2.4 FieldMetaData

This *Structure DataType* is used to provide the metadata for a field in a *DataSet*. The *FieldMetaData* is formally defined in Table 7.

Table 7 – FieldMetaData structure

Name	Type	Description
FieldMetaData	Structure	
Name	String	Name of the field. The name shall be unique in the <i>DataSet</i> .
Description	LocalizedText	Description of the field. The default value shall be a null or empty <i>LocalizedText</i> .
FieldFlags	DataSetFieldFlags	Flags for the field.
BuiltInType	Byte	The built-in data type of the field. The possible built-in type values are defined in IEC 62541-6. All data types are transferred in <i>DataSetMessages</i> as one of the built-in data types. In most cases the identifier of the <i>DataType NodeId</i> matches the built-in type. The following special cases need to be handled in addition: (1) Abstract types always have the built-in type <i>Variant</i> since they can result in different concrete types in a <i>DataSetMessage</i> . The <i>dataType</i> field may provide additional restrictions e.g. if the abstract type is <i>Number</i> . Abstract types shall not be used if the field is represented as <i>RawData</i> set by the <i>DataSetFieldContentMask</i> defined in 6.2.4.2. (2) <i>Enumeration DataTypes</i> are encoded as <i>Int32</i> . The <i>Enumeration</i> strings are defined through a <i>DataType</i> referenced through the <i>dataType</i> field. (3) <i>Structure</i> and <i>Union DataTypes</i> are encoded as <i>ExtensionObject</i> . The encoding rules are defined through a <i>DataType</i> referenced through the <i>dataType</i> field. (4) <i>DataTypes</i> derived from built-in types have the <i>BuiltInType</i> of the corresponding base <i>DataType</i> . The concrete subtype is defined through the <i>dataType</i> field. (5) <i>OptionSet DataTypes</i> are either encoded as one of the concrete <i>UInteger DataTypes</i> or as an instance of an <i>OptionSetType</i> in an <i>ExtensionObject</i> .
DataType	NodeId	The <i>NodeId</i> of the <i>DataType</i> of this field. If the <i>DataType</i> is an <i>Enumeration</i> or an <i>OptionSet</i> , the semantic of the <i>Enumeration DataType</i> is provided through the <i>enumDataTypes</i> field of the <i>DataSetMetaData</i> base type <i>DataTypeSchemaHeader</i> . If the <i>DataType</i> is a <i>Structure</i> or <i>Union</i> , the encoding and decoding description of the <i>Structure DataType</i> is provided through the <i>structureDataTypes</i> field of the <i>DataSetMetaData</i> base type <i>DataTypeSchemaHeader</i> .

Name	Type	Description
ValueRank	Int32	Indicates whether the <i>dataType</i> is an array and how many dimensions the array has. It may have the following values: <i>n</i> > 1: the <i>dataType</i> is an array with the specified number of dimensions. OneDimension (1): The <i>dataType</i> is an array with one dimension. OneOrMoreDimensions (0): The <i>dataType</i> is an array with one or more dimensions. Scalar (−1): The <i>dataType</i> is not an array. Any (−2): The <i>dataType</i> can be a scalar or an array with any number of dimensions. ScalarOrOneDimension (−3): The <i>dataType</i> can be a scalar or a one dimensional array. NOTE All <i>DataTypes</i> are considered to be scalar, even if they have array-like semantics like <i>ByteString</i> and <i>String</i> . Only a concrete valueRank with values <i>n</i> =−1 or <i>n</i> >0 shall be used if the field is represented as <i>RawData</i> set by the <i>DataSetFieldContentMask</i> defined in 6.2.4.2.
ArrayDimensions	UInt32[]	This field specifies the maximum supported length of each dimension. If the maximum is unknown the value shall be 0. The number of elements shall be equal to the value of the <i>valueRank</i> field. This field shall be null or empty if <i>valueRank</i> ≤ 0. The maximum number of elements of an array transferred on the wire is 2.147.483.647 (max Int32). It is the total number of elements in all dimensions based on the UA Binary encoding rules for arrays.
MaxStringLength	UInt32	If the <i>dataType</i> field is a <i>String</i> , <i>LocalizedText</i> (the text field) or <i>ByteString</i> then this field specifies the maximum supported length of the data in number of bytes. If the maximum is unknown the value shall be 0. If the <i>dataType</i> field is not a <i>String</i> , <i>LocalizedText</i> or <i>ByteString</i> the value shall be 0. If the <i>valueRank</i> is greater than 0 this field applies to each element of the array.
DataSetFieldId	Guid	The unique ID for the field in the <i>DataSet</i> . The ID is generated when the field is added to the list. A change of the position of the field in the list shall not change the ID.
Properties	KeyValuePair[]	List of <i>Property</i> values providing additional semantic for the field. If at least one <i>Property</i> value changes, the <i>MajorVersion</i> of the <i>ConfigurationVersion</i> shall be updated. The <i>Property</i> in the <i>FieldMetaData</i> shall correctly describe the <i>Field Value</i> in the <i>DataSetMessages</i> . For example if the <i>Property</i> is <i>EngineeringUnits</i> , the unit of the <i>Field Value</i> shall match the unit of the <i>FieldMetaData</i> . The <i>KeyValuePair</i> <i>DataType</i> is defined in IEC 62541-5. For this field the key in the <i>KeyValuePair</i> structure is the <i>BrowseName</i> of the <i>Property</i> and the value in the <i>KeyValuePair</i> structure is the <i>Value</i> of the <i>Property</i> .

Its representation in the AddressSpace is defined in Table 8.

Table 8 – FieldMetaData definition

Attributes	Value
BrowseName	FieldMetaData
IsAbstract	False
Subtype of Structure defined in IEC 62541-5.	
Conformance Units	
PubSub Parameters Discovery	

6.2.3.2.5 DataSetFieldFlags

This *DataType* defines flags for *DataSet* fields.

The *DataSetFieldFlags* is formally defined in Table 9.