



BSI Standards Publication

OPC unified architecture

Part 14: PubSub

National foreword

This British Standard is the UK implementation of EN IEC 62541-14:2020. It is identical to IEC 62541-14:2020.

The UK participation in its preparation was entrusted to Technical Committee GEL/65/3, Industrial communications: process measurement and control, including fieldbus.

A list of organizations represented on this committee can be obtained on request to its committee manager.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2020
Published by BSI Standards Limited 2020

ISBN 978 0 580 52157 7

ICS 35.100.05; 25.040.40; 35.240.50

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 September 2020.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

English Version

OPC unified architecture - Part 14: PubSub
(IEC 62541-14:2020)

Architecture unifiée OPC - Partie 14: PubSub
(IEC 62541-14:2020)

OPC Unified Architecture - Teil 14: Festlegungen zur
Umsetzung des Publisher-Subscriber-Modells
(IEC 62541-14:2020)

This European Standard was approved by CENELEC on 2020-08-12. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.



European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

European foreword

The text of document 65E/720/FDIS, future edition 1 of IEC 62541-14, prepared by SC 65E "Devices and integration in enterprise systems" of IEC/TC 65 "Industrial-process measurement, control and automation" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN IEC 62541-14:2020.

The following dates are fixed:

- latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement (dop) 2021-05-12
- latest date by which the national standards conflicting with the document have to be withdrawn (dow) 2023-08-12

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC shall not be held responsible for identifying any or all such patent rights.

Endorsement notice

The text of the International Standard IEC 62541-14:2020 was approved by CENELEC as a European Standard without any modification.

Annex ZA (normative)

Normative references to international publications with their corresponding European publications

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE 1 Where an International Publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

NOTE 2 Up-to-date information on the latest versions of the European Standards listed in this annex is available here: www.cenelec.eu.

<u>Publication</u>	<u>Year</u>	<u>Title</u>	<u>EN/HD</u>	<u>Year</u>
IEC/TR 62541-1	-	OPC unified architecture - Part 1: Overview and concepts	CLC/TR 62541-1	-
IEC/TR 62541-2	-	OPC unified architecture - Part 2: Security model	CLC/TR 62541-2	-
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	EN IEC 62541-7	-
IEC 62541-8	-	OPC Unified Architecture - Part 8: Data Access	EN IEC 62541-8	-
IEC 62541-12	-	OPC unified architecture - Part 12: Discovery and global services	EN IEC 62541-12	-
ISO/IEC 19464	2014	Information technology - Advanced Message Queuing Protocol (AMQP) v1.0 specification	-	-
ISO/IEC 20922	2016	Information technology - Message Queuing Telemetry Transport (MQTT) v3.1.1	-	-
IETF RFC 7159	-	The JavaScript Object Notation (JSON) Data Interchange Format	-	-

This page deliberately left blank

CONTENTS

FOREWORD.....	10
1 Scope.....	12
2 Normative references	12
3 Terms, definitions and abbreviated terms	13
3.1 Terms and definitions.....	13
3.2 Abbreviated terms.....	14
4 Overview	14
4.1 Fields of application.....	14
4.2 Abstraction layers	15
4.3 Decoupling by use of middleware.....	15
4.4 Synergy of models	16
5 PubSub Concepts.....	16
5.1 General.....	16
5.2 DataSet	17
5.2.1 General	17
5.2.2 DataSetClass	18
5.2.3 DataSetMetaData	18
5.3 Messages	19
5.3.1 General	19
5.3.2 DataSetMessage field.....	20
5.3.3 DataSetMessage	20
5.3.4 NetworkMessage	21
5.3.5 Message security.....	21
5.3.6 Transport security.....	22
5.3.7 SecurityGroup	22
5.4 Entities	22
5.4.1 Publisher	22
5.4.2 Subscriber	25
5.4.3 Security Key Service	26
5.4.4 Message Oriented Middleware.....	29
6 PubSub communication parameters.....	33
6.1 Overview.....	33
6.2 Common Configuration Parameters.....	34
6.2.1 PubSubState State Machine	34
6.2.2 PublishedDataSet parameters	36
6.2.3 DataSetWriter Parameters	44
6.2.4 Shared PubSubGroup Parameters	48
6.2.5 WriterGroup parameters	50
6.2.6 PubSubConnection Parameters	52
6.2.7 ReaderGroup parameters	55
6.2.8 DataSetReader Parameters	56
6.2.9 SubscribedDataSet Parameters	60
6.2.10 Information flow and status handling.....	63
6.2.11 PubSubConfigurationDataType.....	65
6.3 Message mapping configuration parameters	66
6.3.1 UADP message mapping	66

6.3.2	JSON message mapping	74
6.4	Transport Protocol mapping configuration parameters	77
6.4.1	Datagram Transport Protocol.....	77
6.4.2	Broker Transport Protocol.....	78
7	PubSub mappings	83
7.1	General.....	83
7.2	Message mappings	83
7.2.1	General	83
7.2.2	UADP message mapping	83
7.2.3	JSON message mapping	99
7.3	Transport Protocol Mappings	102
7.3.1	General	102
7.3.2	OPC UA UDP	102
7.3.3	OPC UA Ethernet	103
7.3.4	AMQP	104
7.3.5	MQTT	109
8	PubSub security key service model	111
8.1	Overview.....	111
8.2	PublishSubscribe Object	111
8.3	PubSubKeyServiceType.....	112
8.4	GetSecurityKeys method.....	112
8.5	GetSecurityGroup method.....	114
8.6	SecurityGroupType	115
8.7	SecurityGroupFolderType	116
8.8	AddSecurityGroup Method	116
8.9	RemoveSecurityGroup Method.....	117
9	PubSub configuration model	117
9.1	Common configuration model.....	117
9.1.1	General	117
9.1.2	Configuration behaviours	120
9.1.3	Types for the PublishSubscribe Object	120
9.1.4	Published DataSet Model.....	125
9.1.5	Connection Model.....	141
9.1.6	Group Model.....	145
9.1.7	DataSetWriter Model	153
9.1.8	DataSetReader Model	155
9.1.9	Subscribed DataSet Model	160
9.1.10	PubSub Status Object.....	163
9.1.11	PubSub Diagnostics Objects.....	164
9.1.12	PubSub Status Events	173
9.2	Message Mapping Configuration Model.....	175
9.2.1	UADP Message Mapping	175
9.2.2	JSON Message Mapping	177
9.3	Transport Protocol Mapping Configuration Model.....	178
9.3.1	Datagram Transport Protocol Mapping.....	178
9.3.2	Broker Transport Protocol Mapping.....	179
Annex A (normative)	Common types	182
A.1	DataType Schema Header structures	182

A.1.1	DataTypeSchemaHeader	182
A.1.2	DataTypeDescription	183
A.1.3	StructureDescription	183
A.1.4	EnumDescription	184
A.1.5	SimpleTypeDescription	184
A.2	UABinaryFileDataType	184
A.3	NetworkAddress Model	185
A.3.1	NetworkAddressType	185
A.3.2	NetworkAddressUrlType	186
Annex B (informative)	Client Server vs. Publish Subscribe	187
B.1	Overview	187
B.2	Client Server Subscriptions	187
B.3	Publish-Subscribe	188
B.4	Synergy of models	189
Figure 1	– Publish Subscribe Model overview	15
Figure 2	– Publisher and Subscriber entities	17
Figure 3	– DataSet in the process of publishing	18
Figure 4	– OPC UA PubSub message layers	20
Figure 5	– Publisher details	23
Figure 6	– Publisher message sending sequence	24
Figure 7	– Subscriber details	25
Figure 8	– Subscriber message reception sequence	26
Figure 9	– SecurityGroup management sequence	27
Figure 10	– Handshake used to pull keys from SKS	28
Figure 11	– Handshake used to push keys to Publishers and Subscribers	28
Figure 12	– Handshake with a Security Key Service	29
Figure 13	– PubSub using network infrastructure	30
Figure 14	– UDP Multicast overview	30
Figure 15	– PubSub using broker	31
Figure 16	– Broker overview	32
Figure 17	– PubSub component overview	33
Figure 18	– PubSub mapping specific parameters overview	34
Figure 19	– PubSub component state dependencies	35
Figure 20	– PubSubState state machine	35
Figure 21	– PubSub Information Flow dependency to field representation	45
Figure 22	– PubSub information flow	64
Figure 23	– Start of the periodic publisher execution	67
Figure 24	– Timing offsets in a PublishingInterval	67
Figure 25	– DataSetOrdering and MaxNetworkMessageSize	68
Figure 26	– PublishingOffset options for multiple <i>NetworkMessages</i>	70
Figure 27	– UADP NetworkMessage	84
Figure 28	– UADP DataSet payload	90
Figure 29	– DataSetMessage header structure	91
Figure 30	– Data Key Frame DataSetMessage data	93

Figure 31 – Data Delta Frame DataSetMessage	94
Figure 32 – Event DataSetMessage	95
Figure 33 – KeepAlive message.....	95
Figure 34 – PublishSubscribe Object Types overview	111
Figure 35 – PubSub configuration model overview	118
Figure 36 – PubSub example Objects	119
Figure 37 – PubSub information flow	119
Figure 38 – PublishSubscribe Object Types overview	121
Figure 39 – Published DataSet overview.....	125
Figure 40 – PubSubConnectionType overview	142
Figure 41 – PubSubGroupType overview	145
Figure 42 – DataSet Writer Model Overview.....	153
Figure 43 – DataSet Reader Model overview	155
Figure 44 – PubSub Diagnostics overview	165
Figure 45 – PubSubDiagnosticsCounterType	165
Figure B.1 – Subscriptions in OPC UA Client Server Model	188
Figure B.2 – Publish Subscribe Model Overview	189
Table 1 – PubSubState values	35
Table 2 – PubSubState state machine	36
Table 3 – DataSetMetaData type structure.....	36
Table 4 – DataSetMetaData type definition	37
Table 5 – FieldMetaData structure	37
Table 6 – DataSetFieldFlags values.....	39
Table 7 – DataSetFieldFlags definition.....	39
Table 8 – ConfigurationVersionDataType structure	40
Table 9 – PublishedDataSetDataType structure	41
Table 10 – PublishedDataSetSourceDataType definition.....	41
Table 11 – PublishedVariableDataType structure	42
Table 12 – PublishedDataItemsDataType structure	43
Table 13 – PublishedEventsDataType structure	43
Table 14 – DataSetFieldContentMask values	44
Table 15 – DataSetFieldContentMask definition	45
Table 16 – DataSetMessage field representation options	46
Table 17 – DataSetWriterDataType structure	47
Table 18 – DataSetWriterTransportDataType definition.....	47
Table 19 – DataSetWriterMessageDataType structure	48
Table 20 – PubSubGroupDataType structure	49
Table 21 – PubSubGroupDataType definition.....	49
Table 22 – WriterGroupDataType structure	51
Table 23 – WriterGroupDataType definition	51
Table 24 – WriterGroupTransportDataType definition.....	52
Table 25 – WriterGroupMessageDataType structure	52

Table 26 – PubSubConnectionDataType structure	53
Table 27 – ConnectionTransportDataType definition	54
Table 28 – NetworkAddressDataType structure	54
Table 29 – NetworkAddressDataType definition	54
Table 30 – NetworkAddressUrlDataType structure	54
Table 31 – NetworkAddressUrlDataType definition	55
Table 32 – ReaderGroupDataType structure	55
Table 33 – ReaderGroupDataType definition	55
Table 34 – ReaderGroupTransportDataType definition	56
Table 35 – ReaderGroupMessageDataType structure	56
Table 36 – DataSetReaderDataType structure	59
Table 37 – DataSetReaderTransportDataType structure	59
Table 38 – DataSetReaderTransportDataType definition	60
Table 39 – DataSetReaderMessageDataType structure	60
Table 40 – DataSetReaderMessageDataType definition	60
Table 41 – SubscribedDataSetDataType structure	60
Table 42 – SubscribedDataSetDataType Definition	61
Table 43 – TargetVariablesDataType structure	61
Table 44 – FieldTargetDataType structure	62
Table 45 – OverrideValueHandling values	63
Table 46 – SubscribedDataSetMirrorDataType structure	63
Table 47 – Source to message input mapping	64
Table 48 – Message output to target mapping	65
Table 49 – PubSubConfigurationDataType structure	65
Table 50 – PubSubConfiguration file content	66
Table 51 – DataSetOrderingType values	68
Table 52 – UadpNetworkMessageContentMask values	69
Table 53 – UadpNetworkMessageContentMask definition	69
Table 54 – UadpWriterGroupMessageDataType structure	71
Table 55 – UadpDataSetMessageContentMask values	71
Table 56 – UadpDataSetMessageContentMask definition	72
Table 57 – UadpDataSetWriterMessageDataType structure	73
Table 58 – UadpDataSetReaderMessageDataType structure	74
Table 59 – JsonNetworkMessageContentMask values	75
Table 60 – JsonNetworkMessageContentMask definition	75
Table 61 – JsonWriterGroupMessageDataType structure	75
Table 62 – JsonDataSetMessageContentMask values	76
Table 63 – JsonDataSetMessageContentMask definition	76
Table 64 – JsonDataSetWriterMessageDataType structure	76
Table 65 – JsonDataSetReaderMessageDataType structure	77
Table 66 – DatagramConnectionTransportDataType structure	77
Table 67 – DatagramWriterGroupTransportDataType structure	78
Table 68 – BrokerConnectionTransportDataType structure	79

Table 69 – BrokerTransportQualityOfService values	80
Table 70 – BrokerWriterGroupTransportDataType structure	80
Table 71 – BrokerDataSetWriterTransportDataType structure	82
Table 72 – BrokerDataSetReaderTransportDataType structure	83
Table 73 – UADP NetworkMessage	84
Table 74 – Layout of the key data for UADP message security	87
Table 75 – Layout of the MessageNonce for AES-CTR	88
Table 76 – Layout of the counter block for UADP message security	88
Table 77 – Chunked NetworkMessage payload header	89
Table 78 – Chunked NetworkMessage payload fields	89
Table 79 – UADP DataSet payload header	90
Table 80 – UADP DataSet payload	91
Table 81 – DataSetMessage header structure	92
Table 82 – Data Key Frame DataSetMessage structure	93
Table 83 – Data Delta Frame DataSetMessage structure	94
Table 84 – Event DataSetMessage structure	95
Table 85 – Discovery request header structure	97
Table 86 – Publisher information request message structure	97
Table 87 – Discovery response header structure	98
Table 88 – Publisher Endpoints message structure	98
Table 89 – DataSetMetaData message structure	98
Table 90 – DataSetWriter configuration message structure	99
Table 91 – JSON NetworkMessage definition	99
Table 92 – JSON DataSetMessage definition	101
Table 93 – JSON DataSetMetaData definition	102
Table 94 – UADP message transported over UDP	103
Table 95 – UADP message transported over Ethernet	104
Table 96 – AMQP standard header fields	106
Table 97 – OPC UA AMQP standard header QualifiedName Name mappings	107
Table 98 – OPC UA AMQP header field conversion rules	108
Table 99 – PublishSubscribe Object definition	112
Table 100 – PubSubKeyType definition	112
Table 101 – SecurityGroupType definition	115
Table 102 – SecurityGroupFolderType definition	116
Table 103 – PublishSubscribeType definition	122
Table 104 – HasPubSubConnection ReferenceType	125
Table 105 – PublishedDataSetType definition	126
Table 106 – ExtensionFieldsType definition	127
Table 107 – Well-Known Extension Field Names	128
Table 108 – DataSetToWriter ReferenceType	129
Table 109 – PublishedDataItemsType definition	130
Table 110 – PublishedEventsType definition	133
Table 111 – DataSetFolderType definition	134

Table 112 – PubSubConnectionType definition	142
Table 113 – ConnectionTransportType definition	145
Table 114 – PubSubGroupType definition	146
Table 115 – WriterGroupType definition	147
Table 116 – HasDataSetWriter ReferenceType	149
Table 117 – WriterGroupTransportType definition	149
Table 118 – WriterGroupMessageType definition	150
Table 119 – ReaderGroupType definition	150
Table 120 – HasDataSetReader ReferenceType	152
Table 121 – ReaderGroupTransportType definition	152
Table 122 – ReaderGroupMessageType Definition	152
Table 123 – DataSetWriterType definition	153
Table 124 – DataSetWriterTransportType definition	154
Table 125 – DataSetWriterMessageType definition	154
Table 126 – DataSetReaderType definition	156
Table 127 – DataSetReaderTransportType definition	157
Table 128 – DataSetReaderMessageType definition	158
Table 129 – SubscribedDataSetType definition	160
Table 130 – TargetVariablesType definition	160
Table 131 – SubscribedDataSetMirrorType definition	162
Table 132 – PubSubStatusType definition	163
Table 133 – Status Object definition	164
Table 134 – PubSubDiagnosticsType	166
Table 135 – Counters for PubSubDiagnosticsType	166
Table 136 – DiagnosticsLevel Values	167
Table 137 – PubSubDiagnosticsCounterType	168
Table 138 – PubSubDiagnosticsCounterClassification Values	168
Table 139 – PubSubDiagnosticsRootType	169
Table 140 – LiveValues for PubSubDiagnosticsRootType	169
Table 141 – PubSubDiagnosticsConnectionType	169
Table 142 – LiveValues for PubSubDiagnosticsConnectionType	170
Table 143 – PubSubDiagnosticsWriterGroupType	170
Table 144 – Counters for PubSubDiagnosticsWriterGroupType	170
Table 145 – LiveValues for PubSubDiagnosticsWriterGroupType	170
Table 146 – PubSubDiagnosticsReaderGroupType	171
Table 147 – Counters for PubSubDiagnosticsReaderGroupType	171
Table 148 – LiveValues for PubSubDiagnosticsReaderGroupType	171
Table 149 – PubSubDiagnosticsDataSetWriterType	172
Table 150 – Counters for PubSubDiagnosticsDataSetWriterType	172
Table 151 – LiveValues for PubSubDiagnosticsDataSetWriterType	172
Table 152 – PubSubDiagnosticsDataSetReaderType	172
Table 153 – Counters for PubSubDiagnosticsDataSetReaderType	173
Table 154 – LiveValues for PubSubDiagnosticsDataSetReaderType	173

Table 155 – PubSubStatusEventType definition	173
Table 156 – PubSubTransportLimitsExceedEventType definition	174
Table 157 – PubSubCommunicationFailureEventType definition	174
Table 158 – UadpWriterGroupMessageType definition	175
Table 159 – UadpDataSetWriterMessageType definition	176
Table 160 – UadpDataSetReaderMessageType definition	176
Table 161 – JsonWriterGroupMessageType Definition	177
Table 162 – JsonDataSetWriterMessageType definition	177
Table 163 – JsonDataSetReaderMessageType definition	178
Table 164 – DatagramConnectionTransportType definition	178
Table 165 – DatagramWriterGroupTransportType definition	178
Table 166 – BrokerConnectionTransportType definition	179
Table 167 – BrokerWriterGroupTransportType definition	179
Table 168 – BrokerDataSetWriterTransportType definition	180
Table 169 – Broker Writer well-known extension field names	180
Table 170 – BrokerDataSetReaderTransportType definition	181
Table A.1 – DataTypeSchemaHeader structure	182
Table A.2 – DataTypeSchemaHeader definition	182
Table A.3 – DataTypeDescription structure	183
Table A.4 – DataTypeDescription definition	183
Table A.5 – StructureDescription structure	183
Table A.6 – StructureDescription definition	183
Table A.7 – EnumDescription Structure	184
Table A.8 – EnumDescription definition	184
Table A.9 – SimpleTypeDescription structure	184
Table A.10 – UABinaryFileDataType structure	185
Table A.11 – UABinaryFileDataType definition	185
Table A.12 – NetworkAddressType definition	185
Table A.13 – NetworkAddressUriType definition	186

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.

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

FDIS	Report on voting
65E/720/FDIS	65E/736/RVD

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

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

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

Italics are used to denote a defined term or definition that appears in Clause 3 in one of the parts of the series.

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

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

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

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

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

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

OPC UNIFIED ARCHITECTURE –

Part 14: PubSub

1 Scope

This part of IEC 62541 defines the OPC Unified Architecture (OPC UA) *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. IEC TR 62541-1 gives 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 dictates which mappings need to be implemented in order to be compliant with a particular *Profile*.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

IEC TR 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*

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

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

IETF RFC 7159, *The JavaScript Object Notation (JSON) Data Interchange Format*
<http://www.ietf.org/rfc/rfc7159.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 TR 62541-1, IEC 62541-3, and IEC 62541-4 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at <http://www.electropedia.org/>
- ISO Online browsing platform: available at <http://www.iso.org/obp>

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 a *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
CTL	certificate trust list
HMI	human-machine interface
IGMP	Internet Group Management Protocol
MIME	Multipurpose Internet Mail Extensions
MQTT	MQ Telemetry Transport
MTU	maximum transmission unit
PCP	priority code point
QoS	quality of service
SKS	Security Key Service
UA	Unified Architecture
UADP	UA Datagram Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VID	VLAN identifier

4 Overview

4.1 Fields of application

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

The following are some example uses for *PubSub*:

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

4.2 Abstraction layers

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

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

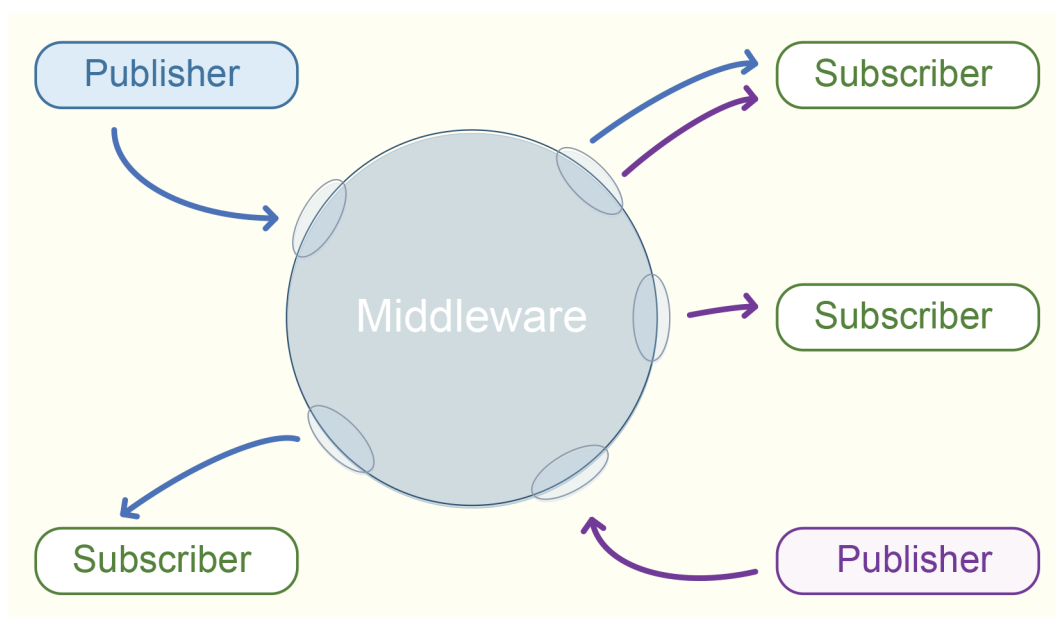
- *PubSub* with UDP transport and binary encoded messages may be well-suited in production environments for frequent transmission of small amounts of data. It also allows data exchange in one-to-one and one-to-many configurations.
- The use of established standard messaging protocols (e.g. AMQP or MQTT) 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.



IEC

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 6.2.2) 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*.

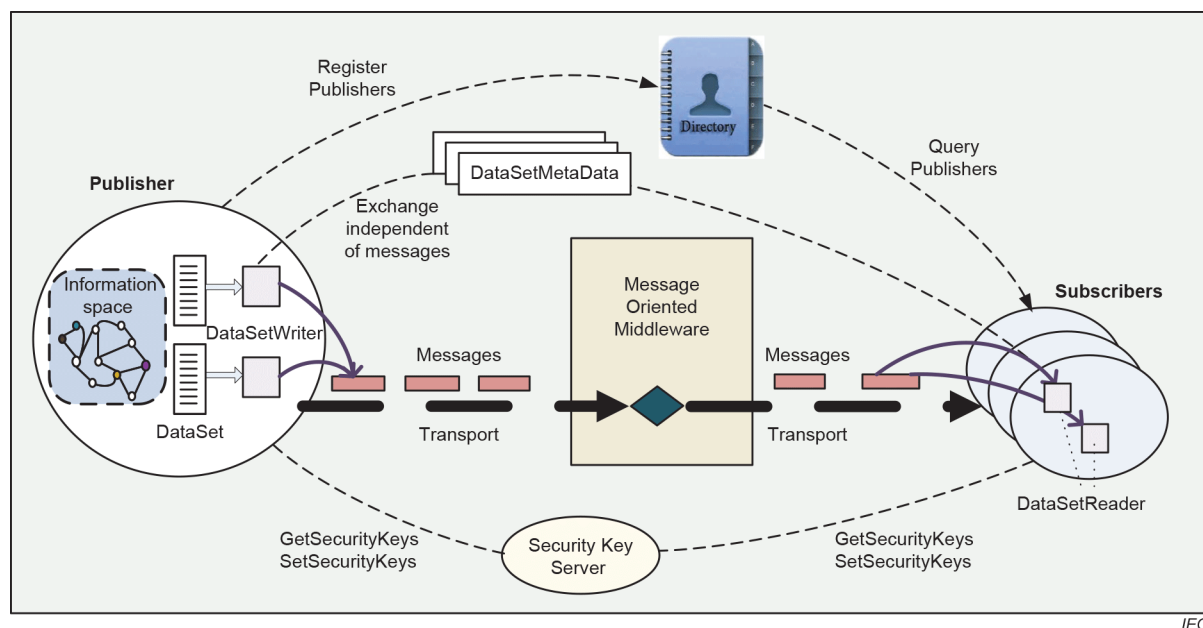


Figure 2 – Publisher and Subscriber entities

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.

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 edition of IEC 62541-14.

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.

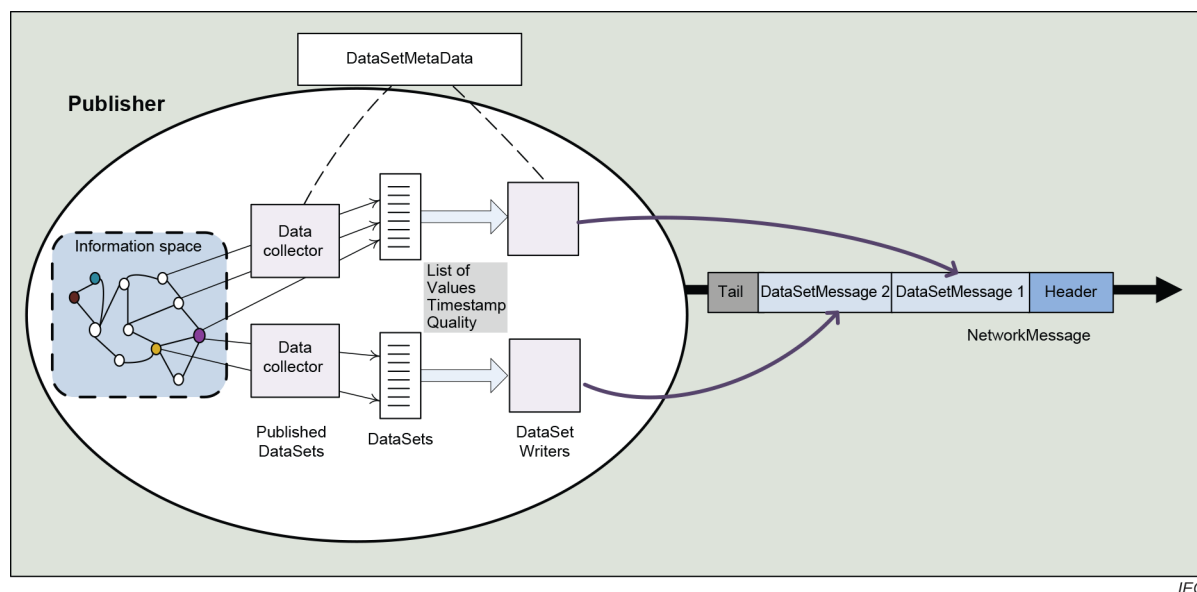


Figure 3 – DataSet in the process of publishing

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 6.2.2.2).

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 semantic 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 *DataSetMetaData* is defined in 6.2.2.1.2.

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 *ConfigurationVersionData* is defined in 6.2.2.1.5.

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 request 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.

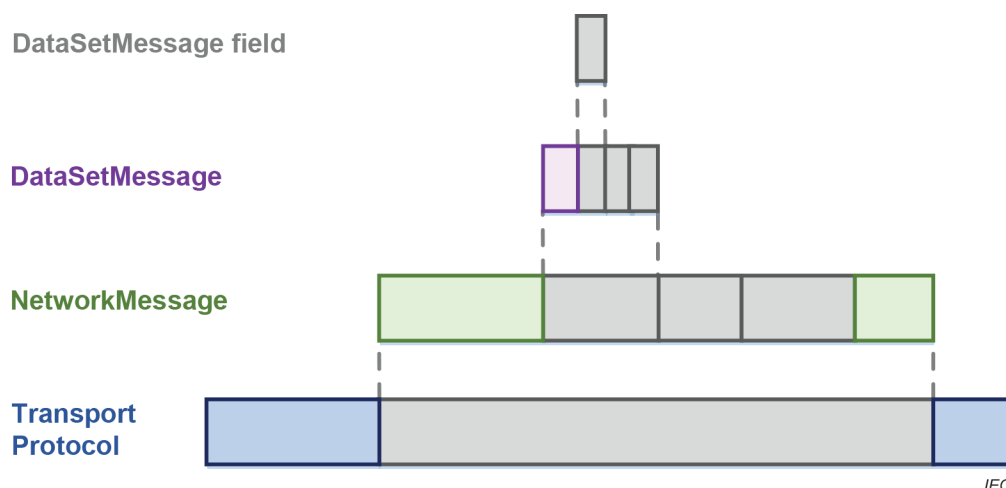


Figure 4 – OPC UA PubSub message layers

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

Subclauses 5.3.2 to 5.3.4 give an abstract definition of *DataSetMessage* and *NetworkMessage*. The concrete structure depends on the message mapping and is described 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 6.2.3.2.

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 *DataSetWriter* and indirectly the *PublishedDataSet*.

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> .
Keep alive	When no <i>DataSetMessages</i> are sent for a configured time period, a keep alive <i>DataSetMessage</i> is sent to signal to the <i>Subscribers</i> that the <i>Publisher</i> is still alive.

Some encodings differentiate between key frame *DataSetMessages* and delta frame *DataSetMessages*. A key frame *DataSetMessage* includes values for all fields of the *DataSet*. A delta frame *DataSetMessage* only contains the subset that changed since the previous *DataSetMessage*.

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

5.3.4 NetworkMessage

The *NetworkMessage* is a container for *DataSetMessages* and includes information shared between *DataSetMessages*. This information consists of:

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

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

5.3.5 Message security

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

- no security;
- signing but no encryption;
- signing and encryption.

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

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

This document defines a general distribution framework for cryptographic keys. This framework is introduced in 5.4.3.

All parameters that are relevant for message security are described in 6.2.4. These parameters are independent of any *Broker* level transport security.

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

5.3.6 Transport security

The transport security is specific to the transport protocol mapping.

When using a broker-based middleware (see 5.4.4.2.2), confidentiality and integrity can be ensured with the transport security between *Publishers* and the *Broker* as well as *Subscribers* and the *Broker*. The *Broker* level security in addition requires all *Publishers* and *Subscribers* to have credentials that grant them access to a *Broker* resource.

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

5.3.7 SecurityGroup

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

A *Security Key Service* (SKS) manages *SecurityGroups* and maintains a mapping between *Roles* and their access *Permissions* for a *SecurityGroup*. This mapping defines if a *Publisher* or *Subscriber* has access to the security keys of a *SecurityGroup*. The SKS is described in more detail in 5.4.3.

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

5.4 Entities

5.4.1 Publisher

5.4.1.1 General

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

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

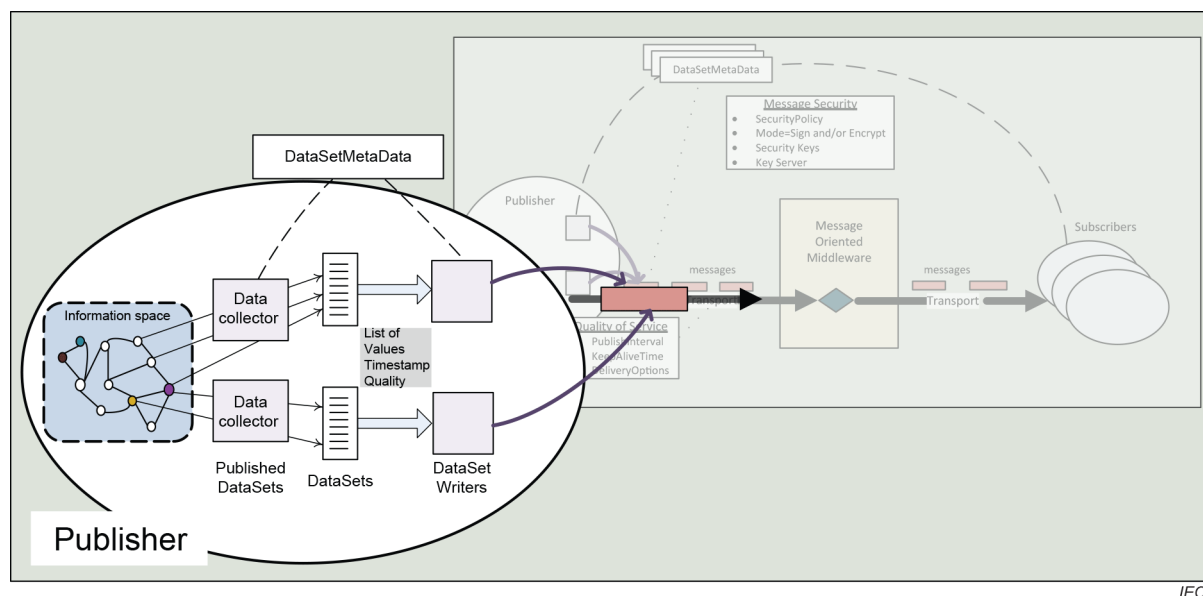


Figure 5 – Publisher details

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.

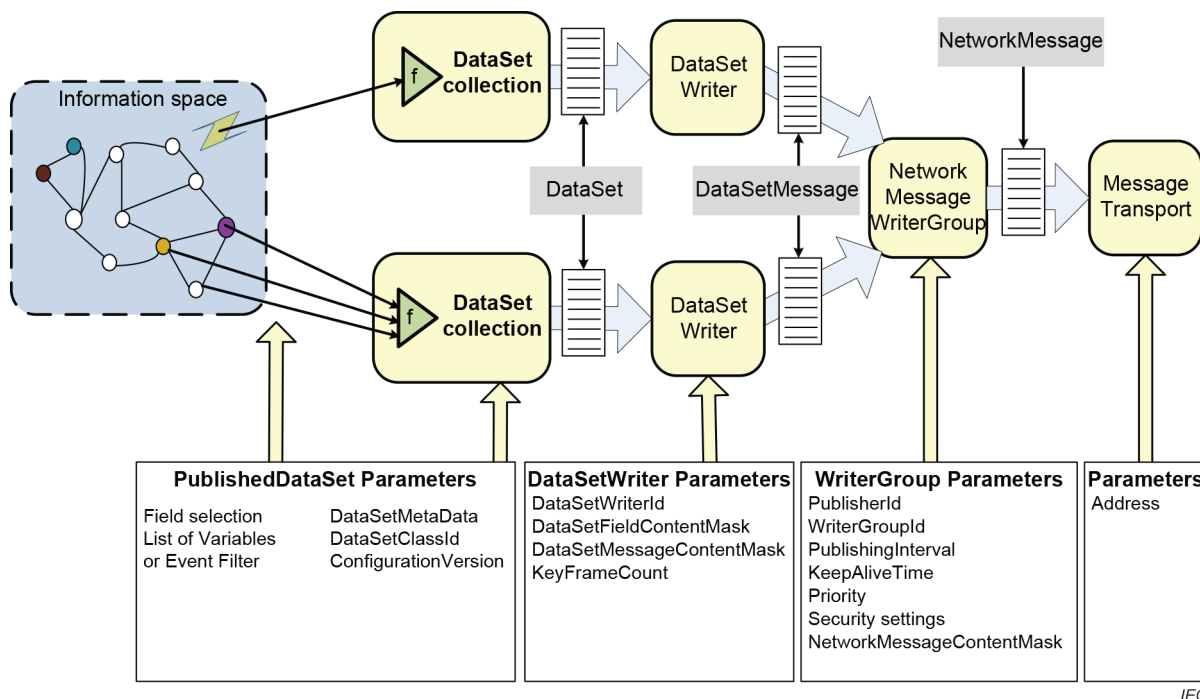


Figure 6 – Publisher message sending sequence

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*.

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 6.2.3.2) controls which attributes of a value shall be encoded.
- The *DataSetMessageContentMask* (see 6.3.1.2.2) controls which header fields shall be encoded.
- The *KeyFrameCount* controls whether a key frame or a delta frame *DataSetMessage* is to be created.

The resulting *DataSetMessage* is passed on to the next step together with the *DataSetWriterId* (see 6.2.3.1), the *DataSetClassId* (see 6.2.2.2), the *ConfigurationVersion* of the *DataSetMetaData* (see 6.2.2.1.5), 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 6.2.6.1) defined on the *WriterGroup*. The structure of this message is protocol specific. If the *SecurityMode* (see 6.2.4.2) requires message security, the *SecurityGroupId* (see 6.2.4.3) is used to fetch the *SecurityPolicy* and the security keys from the SKS (see 5.4.3). 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*.

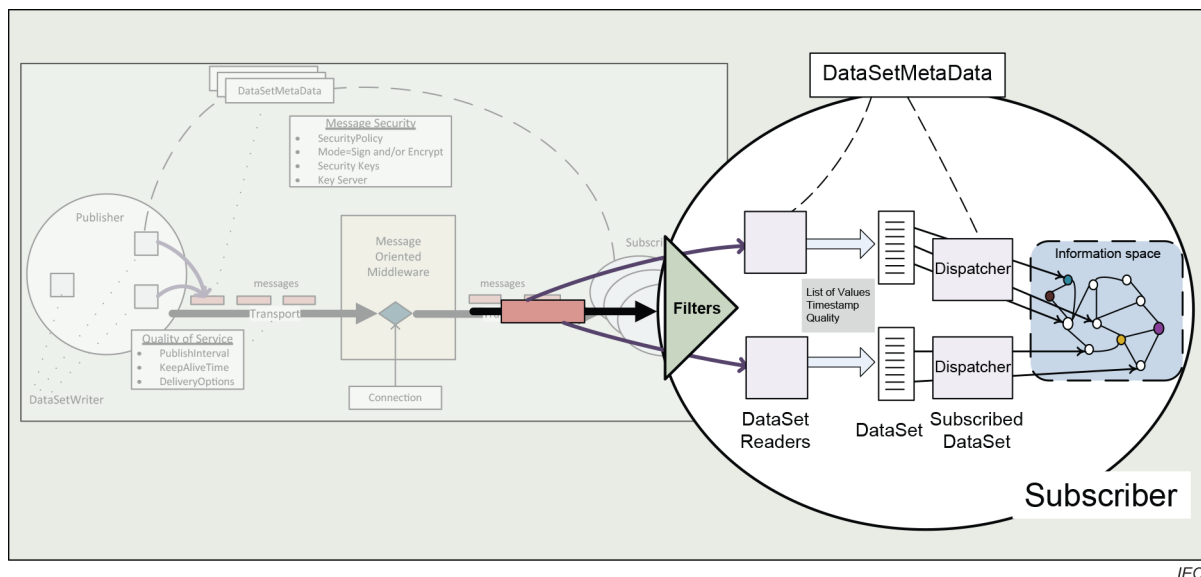


Figure 7 – Subscriber details

To determine for which *DataSetMessages* and on which *Message Oriented Middleware* to subscribe, the *Subscriber* needs 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).

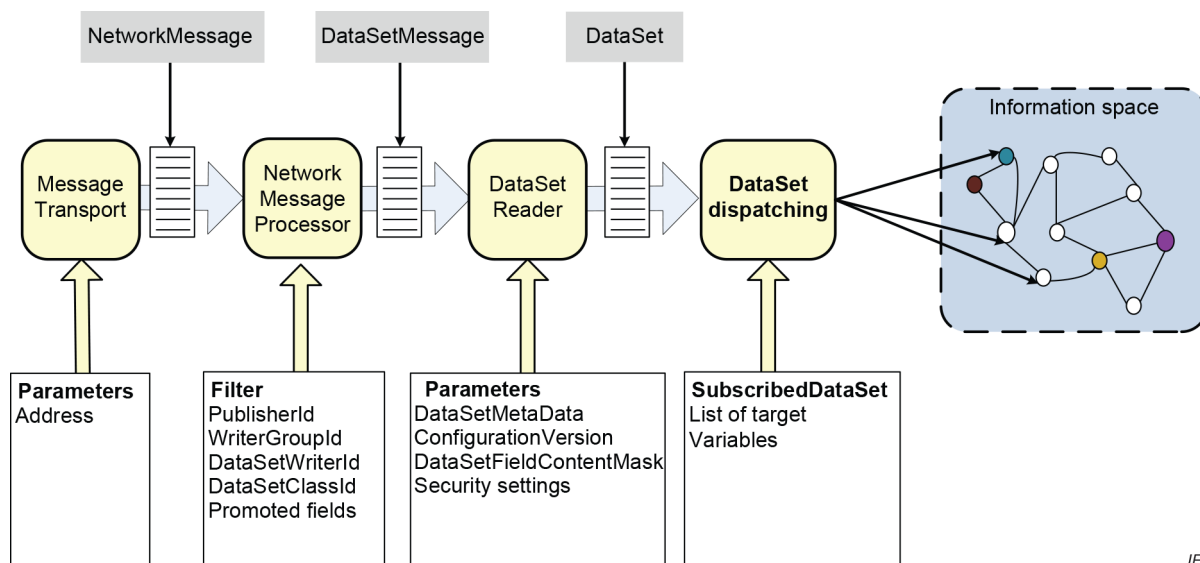
If a *NetworkMessage* is signed or signed and encrypted, the *Subscriber* will need the proper security keys (see 5.3.5) to verify the signature and decrypt the relevant *DataSetMessages*.

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.



IEC

Figure 8 – Subscriber message reception sequence

The *Subscriber* needs 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*.

5.4.3 Security Key Service

5.4.3.1 General

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

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

The *GetSecurityKeys Method* can be implemented by a *Publisher* or by a central SKS. In both cases, the well-known *NodeIds* for the *PublishSubscribe Object* and the related *GetSecurityKeys Method* are used to call the *GetSecurityKeys Method*. The *PublishSubscribe Object* is defined in 8.4.

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

5.4.3.2 SecurityGroup Management

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

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

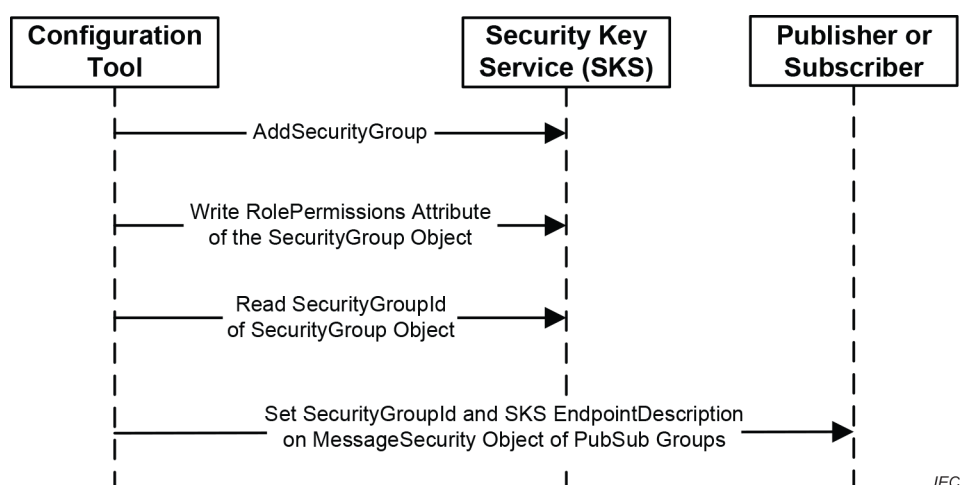


Figure 9 – SecurityGroup management sequence

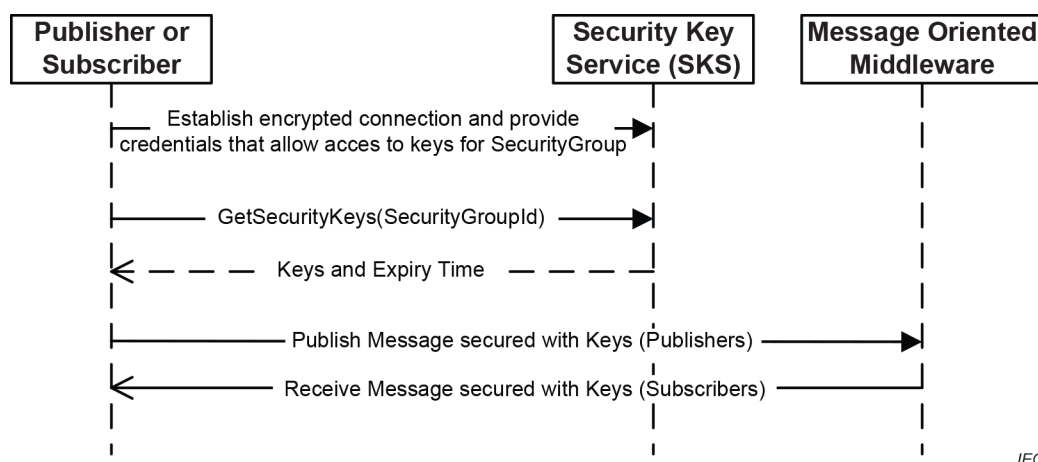
To secure *NetworkMessages*, the *NetworkMessages* shall be secured with keys provided in the context of a *SecurityGroup*. A *SecurityGroup* is created on a SKS using the *Method AddSecurityGroup*.

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

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

5.4.3.3 Key acquisition handshakes

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

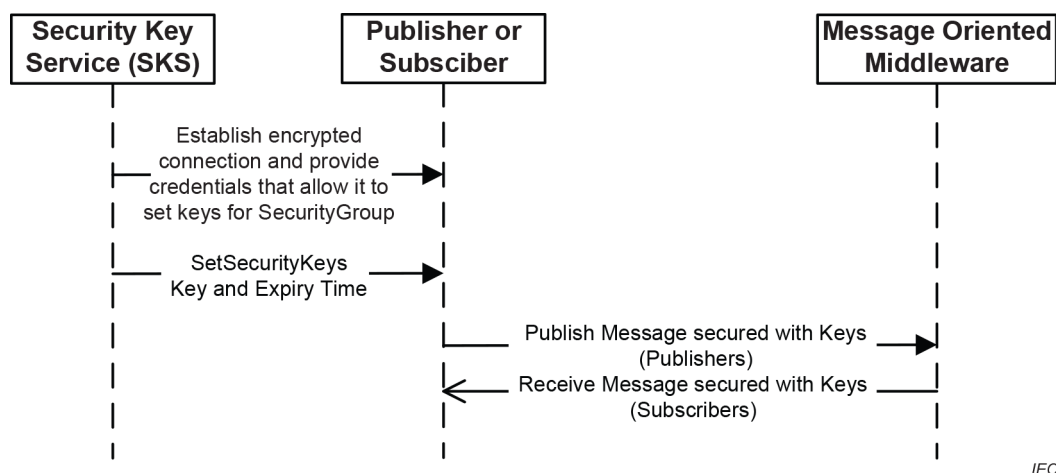


IEC

Figure 10 – Handshake used to pull keys from SKS

To pull keys, the *Publisher* or *Subscriber* creates an encrypted connection and provides credentials that allow it access to the *SecurityGroup*. Then it passes the identifier of the *SecurityGroup* to the *GetSecurityKeys Method* that verifies the *identity* and returns the keys used to secure messages for the *PubSubGroup*. The *GetSecurityKeys Method* is defined in 8.4.

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



IEC

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

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

5.4.3.4 Authorization Services and Security Key Service

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

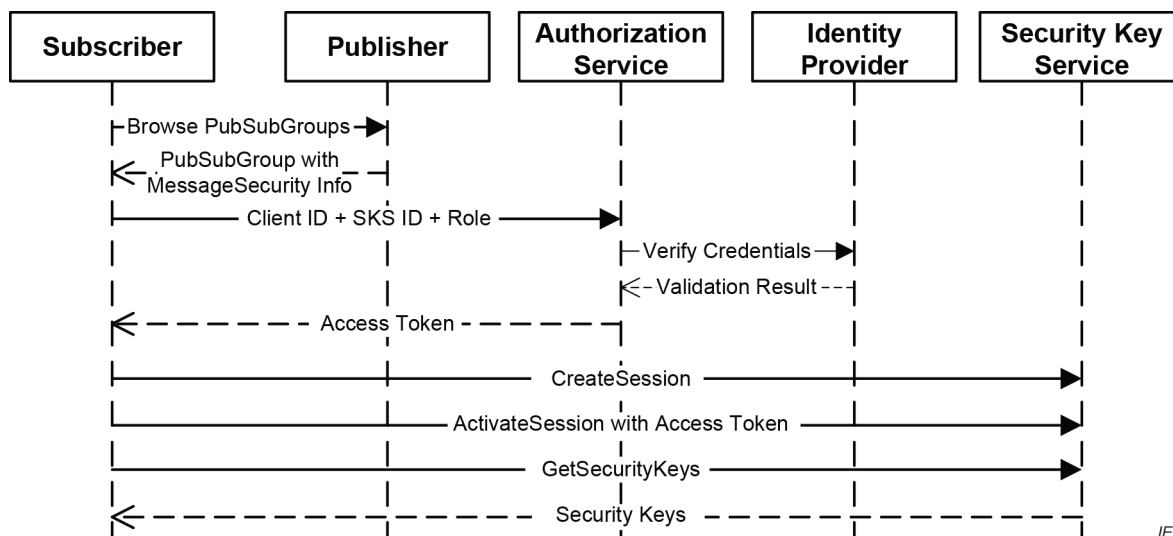


Figure 12 – Handshake with a Security Key Service

The SKS is a *Server* that exposes a *Method* called *GetSecurityKeys*. The *Access Token* is used 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*.

5.4.4 Message Oriented Middleware

5.4.4.1 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 5.4.4.2 and 5.4.4.3.

5.4.4.2 Broker-less Middleware

5.4.4.2.1 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.

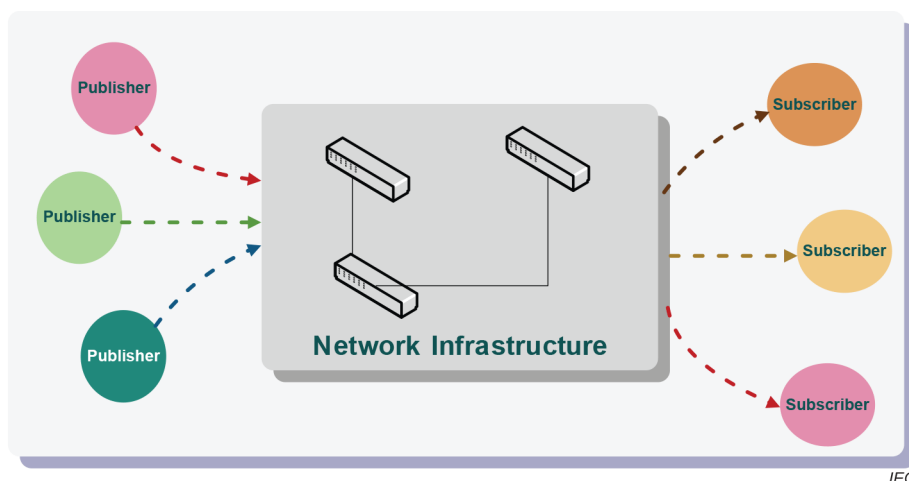


Figure 13 – PubSub using network infrastructure

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.

5.4.4.2.2 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*.

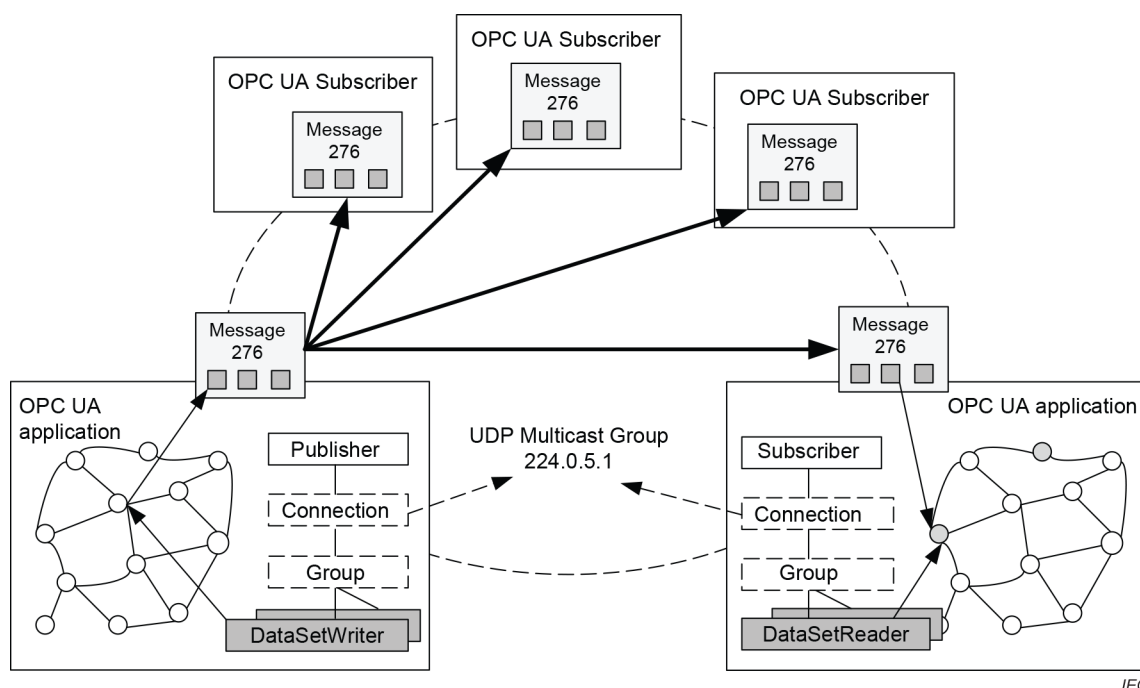


Figure 14 – UDP Multicast overview

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.

5.4.4.3 Broker-based Middleware

5.4.4.3.1 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).

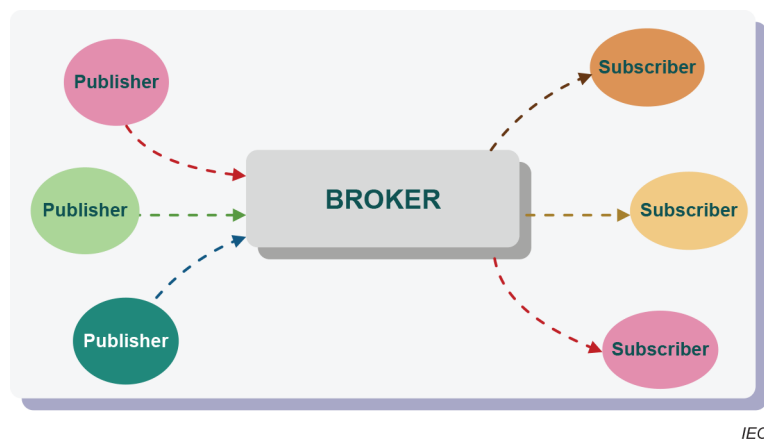


Figure 15 – PubSub using broker

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.

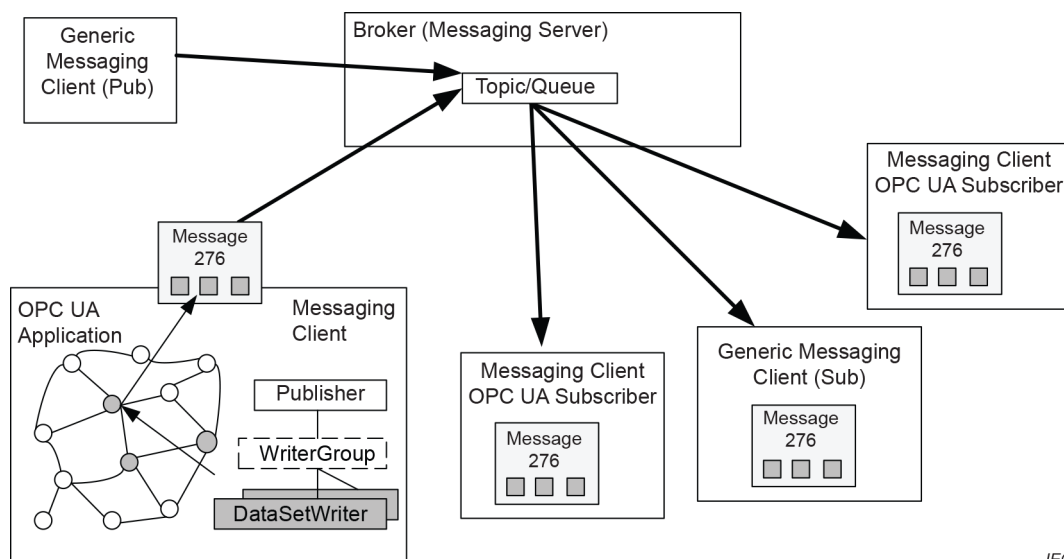
- *Publisher* and *Subscriber* can use different messaging protocols to communicate with the *Broker*.

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.

5.4.4.3.2 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.



IEC

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.

6 PubSub communication parameters

6.1 Overview

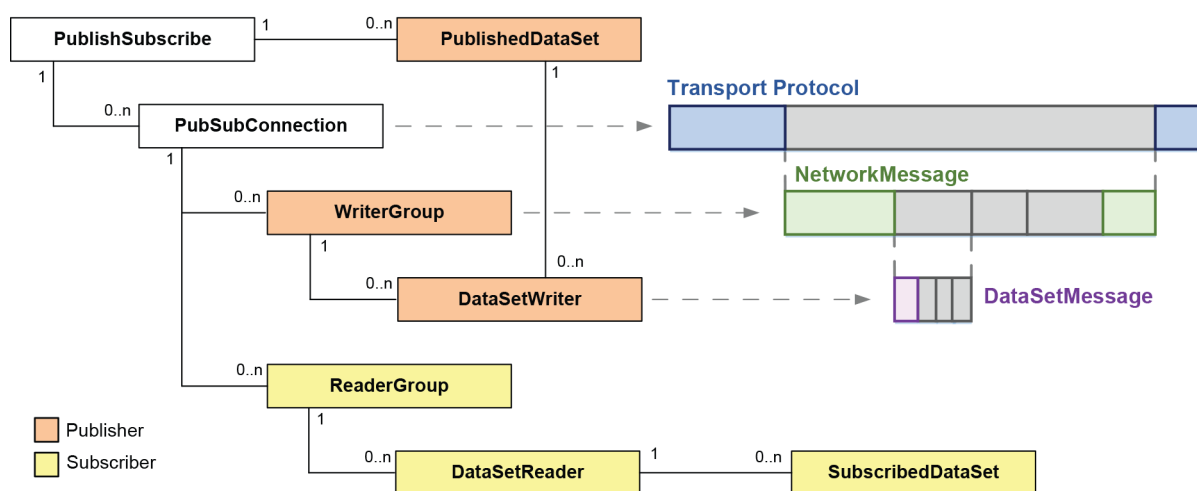
PubSub defines different configuration parameters for the various *PubSub* components. They define the behaviour of *Publisher* and *Subscriber*. The parameters are grouped by component and are partitioned into 'common', 'message mapping', and 'transport protocol mapping'.

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. Common types that are not only used with *PubSub* are defined in Annex A.

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 Clause 6 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 17 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*.



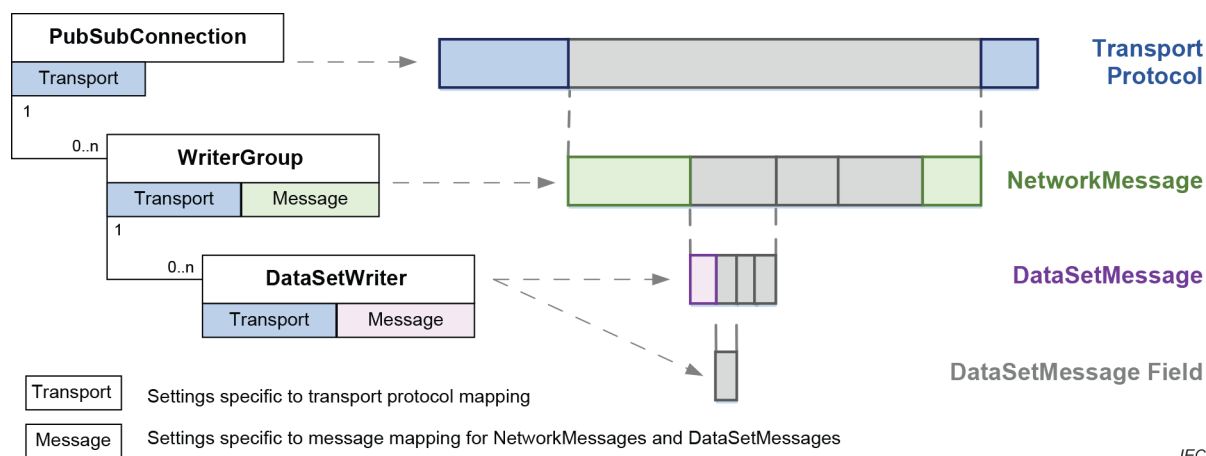
IEC

Figure 17 – PubSub component overview

Figure 17 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 18.



IEC

Figure 18 – 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

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_0</i> or <i>Paused_1</i> .
Operational_2	The <i>PubSub</i> component is operational.
Error_3	The <i>PubSub</i> component is in an error state.

Figure 19 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.

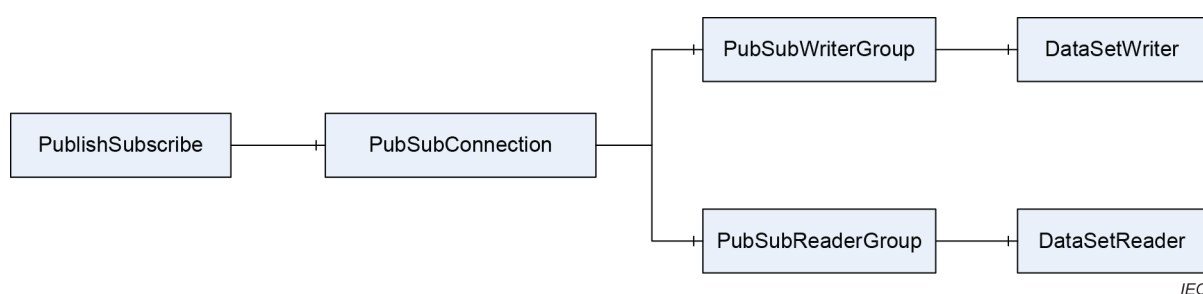
**Figure 19 – PubSub component state dependencies**

Figure 20 describes the formal state machine with the possible transitions.

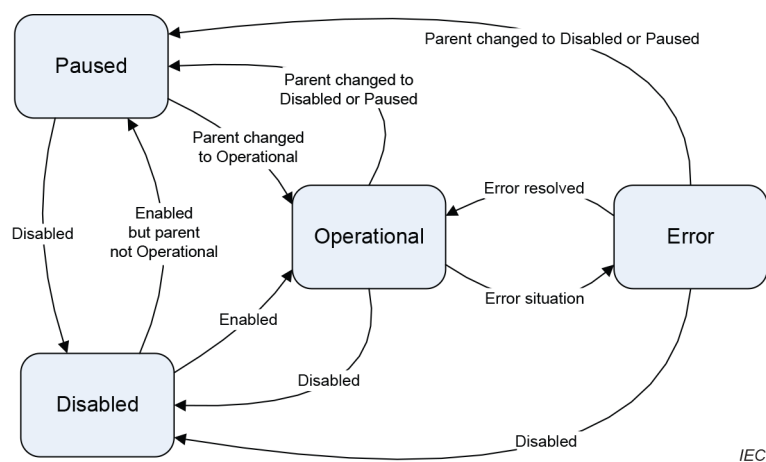
**Figure 20 – PubSubState state machine**

Table 2 formally defines the transitions of the state machine.

Table 2 – PubSubState state machine

Source State	Target State	Trigger Description
Disabled_0	Paused_1	The component was successfully enabled but the parent component is in the state Disabled_0 or Paused_1.
Disabled_0	Operational_2	The component was successfully enabled.
Paused_1	Disabled_0	The component was successfully disabled.
Paused_1	Operational_2	The state of the parent component changed to Operational_2.
Operational_2	Disabled_0	The component was successfully disabled.
Operational_2	Paused_1	The state of the parent component changed to Disabled_0 or Paused_1.
Operational_2	Error_3	There is a pending error situation for the related <i>PubSub</i> component.
Error_3	Disabled_0	The component was successfully disabled.
Error_3	Paused_1	The state of the parent component changed to Disabled_0 or Paused_1.
Error_3	Operational_2	The error situation was resolved for the related <i>PubSub</i> component.

6.2.2 PublishedDataSet parameters

6.2.2.1 DataSetMetaData

6.2.2.1.1 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 6.2.2.1.2.

6.2.2.1.2 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 3.

The *DataTypeSchemaHeader* provides OPC UA *DataType* definitions used in the *DataSetMetaData*. The *DataTypeSchemaHeader* is defined in A.1.1.

Table 3 – DataSetMetaDataType structure

Name	Type	Description
DataSetMetaDataType	Structure	
name	String	Name of the <i>DataSet</i> .
description	LocalizedText	Description of the <i>DataSet</i> . The default value is a null <i>LocalizedText</i> .
fields	FieldMetaData[]	The metadata for the fields in the <i>DataSet</i> . The FieldMetaData <i>DataType</i> is defined in 6.2.2.1.3.
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 4.

Table 4 – DataSetMetaData definition

Attributes	Value
BrowseName	DataSetMetaData
IsAbstract	False
Subtype of DataTypeSchemaHeader defined in A.1.1.	

6.2.2.1.3 FieldMetaData

This *Structure DataType* is used to provide the metadata for a field in a *DataSet*. The *FieldMetaData* is formally defined in Table 5.

Table 5 – 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 <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 6.2.3.1. (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> . 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> .

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: $n > 1$: 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>.
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 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.
maxLength	UInt32	If the <i>dataType</i> field is a <i>String</i> or <i>ByteString</i> then this field specifies the maximum supported length. If the maximum is unknown the value shall be 0. If the <i>dataType</i> field is not a <i>String</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. 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>.

6.2.2.1.4 DataSetFieldFlags

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

The *DataSetFieldFlags* is formally defined in Table 6.

Table 6 – DataSetFieldFlags values

Value	Bit No.	Description
PromotedField	0	The flag indicates if the field is promoted to the <i>NetworkMessages</i> or transport protocol header. Setting this flag increases the size of the <i>NetworkMessages</i> since information from the <i>DataSetMessage</i> body is also promoted to the header. Depending on the used security, the header including the field may be unencrypted. Promoted fields are always included in the header even if the <i>DataSetMessage</i> payload is a delta frame and the <i>DataSet</i> field is not included in the delta frame. In this case the last sent value is sent in the header. The order of the fields in the <i>DataSetMetaData</i> promoted to the header shall match the order of the fields in the header unless the header includes field names.

The *DataSetFieldFlags* representation in the *AddressSpace* is defined in Table 7.

Table 7 – DataSetFieldFlags definition

Attributes	Value		
BrowseName	DataSetFieldFlags		
IsAbstract	False		
References	NodeClass	BrowseName	DataType
Subtype of UInt16 defined in IEC 62541-5.			
HasProperty	Variable	OptionSetValues	LocalizedText []

6.2.2.1.5 ConfigurationVersionDataType

This *Structure DataType* is used to indicate configuration changes in the information published for a *DataSet*. The *ConfigurationVersionDataType* is formally defined in Table 8.

Table 8 – ConfigurationVersionDataType structure

Name	Type	Description
ConfigurationVersionDataType	Structure	
majorVersion	VersionTime	The <i>MajorVersion</i> reflects the time of the last major change of the <i>DataSet</i> content. The <i>VersionTime DataType</i> is defined in IEC 62541-4. To assure interoperability, the <i>Subscriber</i> shall use <i>DataSetMetaData</i> for decoding with a <i>MajorVersion</i> that matches the <i>MajorVersion</i> in <i>DataSetMessages</i> sent by the <i>Publisher</i>. Removing fields from the <i>DataSet</i> content, reordering fields, adding fields in between other fields or a <i>DataType</i> change in fields shall result in an update of the <i>MajorVersion</i>. If at least one <i>Property</i> value of a <i>DataSetMetaData</i> field changes, the <i>MajorVersion</i> shall be updated. There can be situations where older configurations of a <i>Publisher</i> are loaded and changed with product-specific configuration tools. In this case the <i>MajorVersion</i> shall be updated if the configuration tool is not able to verify if the change only extends the configuration and does not change the existing content. Additional criteria for changing <i>MajorVersion</i> or <i>MinorVersion</i> are defined in this document.
minorVersion	VersionTime	The <i>MinorVersion</i> reflects the time of the last change. Only the <i>MinorVersion</i> shall be updated if fields are added at the end of the <i>DataSet</i> content. If the <i>MajorVersion</i> version is updated, the <i>MinorVersion</i> is updated to the same value as <i>MajorVersion</i>.

6.2.2.2 DataSetClassId

DataSetMetaData may be specific to a single *Publisher* and a single selection of information or universal, e.g. defined by a standards organization or by a plant operator as a *DataSetClass*. *Datasets* that conform to such a *DataSetClass* are identified with a *DataSetClassId*.

The *DataSetClassId* is the globally unique identifier (*Guid*) of a *DataSetClass*. It is included in the *DataSetMetaData*. The *NetworkMessageContentMask* controls the availability of the *DataSetClassId* in the *NetworkMessage*.

6.2.2.3 ExtensionFields

The *ExtensionFields* parameter allows the configuration of fields with values to be included in the *DataSet* when the existing *AddressSpace* of the *Publisher* does not provide the necessary information. The *ExtensionFields* are represented as an array of *KeyValuePair Structures*.

6.2.2.4 PublishedDataSetDataType

This *Structure DataType* represents the *PublishedDataSet* parameters. The *PublishedDataSetDataType* is formally defined in Table 9.

Table 9 – PublishedDataSetDataType structure

Name	Type	Description
PublishedDataSetDataType	Structure	
name	String	Name of the <i>PublishedDataSet</i> . The name of the <i>PublishedDataSet</i> shall be unique in the <i>Publisher</i> .
dataSetFolder	String[]	Optional path of the <i>DataSet</i> folder used to group <i>PublishedDataSets</i> where each entry in the <i>String</i> array represents one level in a <i>DataSet</i> folder hierarchy. If no grouping is needed the parameter is a null <i>String</i> array.
dataSetMetaData	DataSetMetaData	Defined in 6.2.2.1.
extensionFields	KeyValuePair[]	Defined in 6.2.2.3.
dataSetSource	PublishedDataSetSourceDataType	Defined in 6.2.2.5.

6.2.2.5 PublishedDataSetSourceDataType

The *PublishedDataSetSourceDataType Structure* is an abstract base type without fields for the definition of the *PublishedDataSet* source. Its representation in the *AddressSpace* is defined in Table 10.

Table 10 – PublishedDataSetSourceDataType definition

Attributes	Value			
BrowseName	PublishedDataSetSourceDataType			
IsAbstract	True			
References	NodeClass	BrowseName	IsAbstract	Description
Subtype of Structure defined in IEC 62541-5.				
HasSubtype	DataType	PublishedDataItemsDataType	FALSE	Defined in 6.2.2.6.2.
HasSubtype	DataType	PublishedEventsDataType	FALSE	Defined in 6.2.2.7.4.

6.2.2.6 Published Data Items

6.2.2.6.1 PublishedData

The parameter *PublishedData* defines the content of a *DataSet* created from *Variable Values* and therefore the content of the *DataSetMessage* sent by a *DataSetWriter*. The sources of the *DataSet* fields are defined through an array of *PublishedVariableDataType*.

The index into the array has an important role for *Subscribers* and for configuration tools. It is used as a handle to reference the *Value* in *DataSetMessages* received by *Subscribers*. The index may change after configuration changes. Changes are indicated by the *ConfigurationVersion* of the *DataSet* and applications working with the index shall always check the *ConfigurationVersion* before using the index.

If an entry of the *PublishedData* references one of the *ExtensionFields*, the *substituteValue* shall contain the *QualifiedName* of the *ExtensionFields* entry. All other fields of this *PublishedVariableDataType* array element shall be null.

The *DataType* *PublishedVariableDataType* represents the configuration information for one *Variable*. The *PublishedVariableDataType* is formally defined in Table 11.

Table 11 – PublishedVariableDataType structure

Name	Type	Description								
PublishedVariableDataType	Structure									
publishedVariable	NodeId	The <i>NodeId</i> of the published <i>Variable</i> . Some transport protocols require knowledge on the message receiver side about the <i>DataType</i> , <i>ValueRank</i> and <i>ArrayDimensions</i> to be able to decode the message content. This information is provided through the <i>DataSetMetaData</i> provided for the <i>DataSet</i> .								
attributeId	IntegerId	Id of the <i>Attribute</i> to publish e.g. the <i>Value Attribute</i> . This shall be a valid <i>Attribute</i> id. The <i>Attributes</i> are defined in IEC 62541-3. The <i>IntegerId DataType</i> is defined in IEC 62541-4. The <i>IntegerIds</i> for the <i>Attributes</i> are defined in IEC 62541-6.								
samplingIntervalHint	Duration	A recommended rate of acquiring new values for change or deadband evaluation. A <i>Publisher</i> should use this value as hint for setting the internal sampling rate. The value 0 indicates that the <i>Server</i> should use the fastest practical rate. The value -1 indicates that the default sampling interval defined by the <i>PublishingInterval</i> of the <i>WriterGroup</i> is requested. Any negative number is interpreted as -1.								
deadbandType	UInt32	A value that defines the <i>Deadband</i> type and behaviour. <table><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>None_0</td><td>No <i>Deadband</i> calculation should be applied.</td></tr><tr><td>Absolute_1</td><td>AbsoluteDeadband (This type is specified in IEC 62541-4)</td></tr><tr><td>Percent_2</td><td>PercentDeadband (This type is specified in IEC 62541-8).</td></tr></tbody></table>	Value	Description	None_0	No <i>Deadband</i> calculation should be applied.	Absolute_1	AbsoluteDeadband (This type is specified in IEC 62541-4)	Percent_2	PercentDeadband (This type is specified in IEC 62541-8).
Value	Description									
None_0	No <i>Deadband</i> calculation should be applied.									
Absolute_1	AbsoluteDeadband (This type is specified in IEC 62541-4)									
Percent_2	PercentDeadband (This type is specified in IEC 62541-8).									
deadbandValue	Double	The deadband value for the corresponding <i>DeadbandType</i> . The meaning of the value depends on <i>DeadbandType</i> .								
indexRange	NumericRange	This parameter is used to identify a single element of an array, or a single range of indexes for arrays. The <i>NumericRange</i> type and the logic for <i>IndexRange</i> are defined in IEC 62541-4.								
substituteValue	BaseDataType	The value that is included in the <i>DataSet</i> if the <i>StatusCode</i> of the <i>DataValue</i> is Bad. In this case the <i>StatusCode</i> is set to Uncertain_SubstituteValue. This Value shall match the <i>DataType</i> of the <i>PublishedVariable</i> since <i>DataSetWriters</i> may depend on a valid <i>Value</i> with the right <i>DataType</i> that matches the <i>ConfigurationVersion</i> . If the <i>SubstituteValue</i> is Null, the <i>StatusCode</i> of the <i>DataValue</i> is processed. The handling of the <i>SubstituteValue</i> is defined in 6.2.10.								
metaDataProperties	QualifiedName []	This parameter specifies an array of <i>Properties</i> to be included in the <i>FieldMetaData</i> created for this <i>Variable</i> . It shall be used to populate the <i>properties</i> element of the resulting field in the <i>DataSetMetaData</i> .								

6.2.2.6.2 PublishedDataItemsDataType

This *Structure DataType* is used to represent *PublishedDataItems* specific parameters. It is a subtype of the *PublishedDataSetSourceDataType* defined in 6.2.2.5.

The *PublishedDataItemsDataType* is formally defined in Table 12.

Table 12 – PublishedDataItemsDataType structure

Name	Type	Description
PublishedDataItemsDataType	Structure	
publishedData	PublishedVariableDataType[]	Defined in 6.2.2.6.1.

6.2.2.7 Published Events

6.2.2.7.1 EventNotifier

The parameter *EventNotifier* defines the *NodeId* of the *Object* in the event notifier tree of the OPC UA Server from which *Events* are collected.

6.2.2.7.2 SelectedFields

The parameter *SelectedFields* defines the selection of *Event* fields contained in the *DataSet* generated for an *Event* and sent through the *DataSetWriter*. The *SimpleAttributeOperand DataType* is defined in IEC 62541-4. The *DataType* of the selected *Event* field in the *EventType* defines the *DataType* of the *DataSet* field. *Event* fields can be null or the field value can be a *StatusCode*. The encoding of *Event* based *DataSetMessages* shall be able to handle these cases. *ExtensionFields* defined for the instance of the *PublishedEventsType* can be included in the *SelectedFields* by specifying the *PublishedEventsType NodeId* as *typeld* in the *SimpleAttributeOperand* and the *BrowseName* of the extension field in the *browsePath* of the *SimpleAttributeOperand*.

The index into the list of entries in the *SelectedFields* has an important role for *Subscribers*. It is used as handle to reference the *Event* field in *DataSetMessages* received by *Subscribers*. The index may change after configuration changes. Changes are indicated by the *ConfigurationVersion* and applications working with the index shall always check the *ConfigurationVersion* before using the index. If a change of the *SelectedFields* adds additional fields, the *MinorVersion* of the *ConfigurationVersion* shall be updated. If a change of the *SelectedFields* removes fields, the *MajorVersion* of the *ConfigurationVersion* shall be updated. The *ConfigurationVersionDataType* and the rules for setting the version are defined in 6.2.2.1.5.

6.2.2.7.3 Filter

The parameter *Filter* defines the filter applied to the *Events*. It allows the reduction of the *DataSets* generated from *Events* through a filter. The *ContentFilter DataType* is defined in IEC 62541-4.

6.2.2.7.4 PublishedEventsDataType

This *Structure DataType* is used to represent *PublishedEvents* specific parameters. It is a subtype of the *PublishedDataSetSourceDataType* defined in 6.2.2.5.

The *PublishedEventsDataType* is formally defined in Table 13.

Table 13 – PublishedEventsDataType structure

Name	Type	Description
PublishedEventsDataType	Structure	
eventNotifier	NodeId	Defined in 6.2.2.7.1.
selectedFields	SimpleAttributeOperand[]	Defined in 6.2.2.7.2.
filter	ContentFilter	Defined in 6.2.2.7.3.

6.2.3 DataSetWriter Parameters

6.2.3.1 DataSetWriterId

The *DataSetWriterId* with *DataType UInt16* defines the unique ID of the *DataSetWriter* for a *PublishedDataSet*. It is used to select *DataSetMessages* for a *PublishedDataSet* on the *Subscriber* side.

It shall be unique across all *DataSetWriters* for a *PublisherId*.

All values, except for 0, are valid *DataSetWriterIds*. The value 0 is defined as null value.

6.2.3.2 DataSetFieldContentMask

A *DataSet* field consists of a value and related metadata. In most cases the value comes with status and timestamp information.

This *DataType* defines flags to include *DataSet* field related information like status and timestamp in addition to the value in the *DataSetMessage*.

The *DataSetFieldContentMask* is formally defined in Table 14.

The handling of bad status for different field representations is defined in Figure 21 and Table 16.

Table 14 – DataSetFieldContentMask values

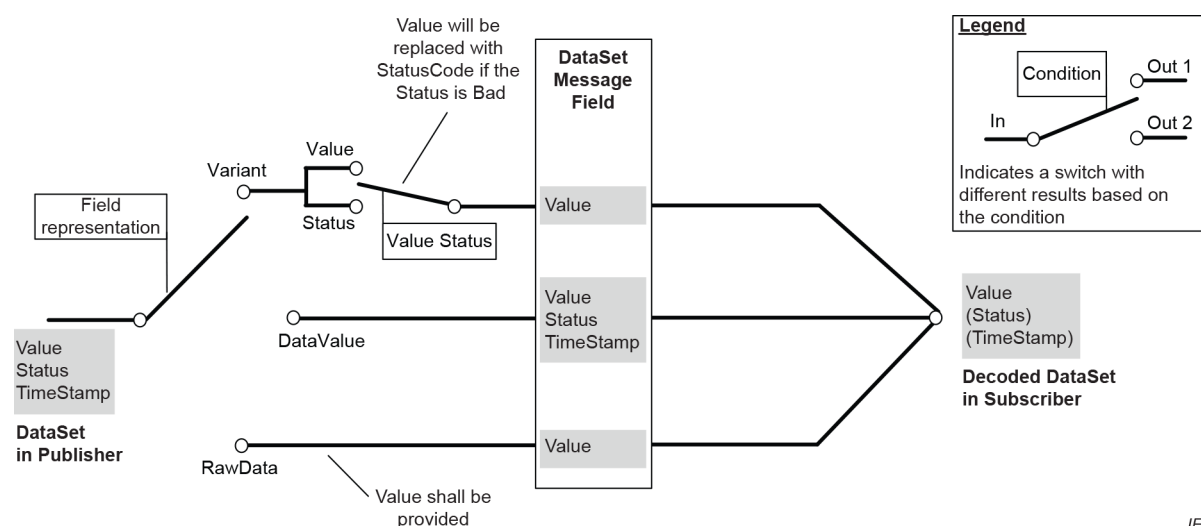
Value	Bit No.	Description
<i>DataSet</i> fields can be represented as <i>RawData</i> , <i>Variant</i> or <i>DataValue</i> as described in 5.3.2. If none of the flags are set, the fields are represented as <i>Variant</i> . If the <i>RawData</i> flag is set, the fields are represented as <i>RawData</i> and all other bits are ignored. If one of the bits 0 to 4 is set, the fields are represented as <i>DataValue</i> .		
StatusCode	0	The <i>DataValue</i> structure field <i>StatusCode</i> is included in the <i>DataSetMessages</i> . If this flag is set, the fields are represented as <i>DataValue</i> .
SourceTimestamp	1	The <i>DataValue</i> structure field <i>SourceTimestamp</i> is included in the <i>DataSetMessages</i> . If this flag is set, the fields are represented as <i>DataValue</i> .
ServerTimestamp	2	The <i>DataValue</i> structure field <i>ServerTimestamp</i> is included in the <i>DataSetMessages</i> . If this flag is set, the fields are represented as <i>DataValue</i> .
SourcePicoSeconds	3	The <i>DataValue</i> structure field <i>SourcePicoSeconds</i> is included in the <i>DataSetMessages</i> . If this flag is set, the fields are represented as <i>DataValue</i> . This flag is ignored if the <i>SourceTimestamp</i> flag is not set.
ServerPicoSeconds	4	The <i>DataValue</i> structure field <i>ServerPicoSeconds</i> is included in the <i>DataSetMessages</i> . If this flag is set, the fields are represented as <i>DataValue</i> . This flag is ignored if the <i>ServerTimestamp</i> flag is not set.
RawData	5	If this flag is set, the values of the <i>DataSet</i> are encoded as <i>Structure</i> and all other field related flags shall be ignored. The <i>RawData</i> representation is handled like a <i>Structure DataType</i> where the <i>DataSet</i> fields are handled like <i>Structure</i> fields and fields with <i>Structure DataType</i> are handled like nested structures. All restrictions for the encoding of <i>Structure DataTypes</i> also apply to the <i>RawData Field Encoding</i> . Fields shall not have an abstract <i>DataType</i> or shall have a fixed <i>ValueRank</i> . Fields shall have dimensions defined if the <i>DataType</i> is <i>String</i> or <i>ByteString</i> or if it is an array. This includes <i>Structure</i> fields with such fields. The flag shall be ignored and the fields shall be represented as <i>Variant</i> if the fields do not fulfil these requirements.

The *DataSetFieldContentMask* representation in the *AddressSpace* is defined in Table 15.

Table 15 – DataSetFieldContentMask definition

Attributes	Value		
BrowseName	DataSetFieldContentMask		
IsAbstract	False		
References	NodeClass	BrowseName	DataType
Subtype of UInt32 defined in IEC 62541-5.			
HasProperty	Variable	OptionSetValues	LocalizedText []

The *DataSetFieldContentMask* defines different options that influence the information flow from *Publisher* to *Subscriber* in the case of a Bad Value Status or other error situations. Figure 21 depicts the parameters and the information flow from *DataSet* field to *DataSetMessage* creation on the *Publisher* side and the decoded *DataSet* field on the *Subscriber* side. The *DataSetFieldContentMask* controls the representation of the *DataSet* fields in a *DataSetMessage*.



IEC

Figure 21 – PubSub Information Flow dependency to field representation

The representation of the *DataSet* fields in a *DataSetMessage* on the *Publisher* side and the decoding back to the *DataSet* fields on the *Subscriber* side is defined in Table 16. The representation on the *Publisher* side depends on the field representation defined in the *DataSetFieldContentMask*.