DIN EN IEC 62541-14

DIN

ICS 25.040.40; 35.200; 35.240.50 Einsprüche bis 2019-10-09

Entwurf

OPC Unified Architecture – Teil 14: Festlegungen zur Umsetzung des Publisher-Subscriber-Models (IEC 65E/617/CDV:2018); Englische Fassung prEN IEC 62541-14:2018

OPC Unified Architecture –
Part 14: PubSub
(IEC 65E/617/CDV:2018);
English version prEN IEC 62541-14:2018

Architecture unifiée OPC –
Partie 14: PubSub
(IEC 65E/617/CDV:2018);
Version anglaise prEN IEC 62541-14:2018

Anwendungswarnvermerk

Dieser Norm-Entwurf mit Erscheinungsdatum 2019-08-09 wird der Öffentlichkeit zur Prüfung und Stellungnahme vorgelegt.

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

Stellungnahmen werden erbeten

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

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

Gesamtumfang 167 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE

Documents
Document Sharing Hub

Full document details: E DIN EN IEC 62541-14:2019-09

Nationales Vorwort

Die Englische Fassung des europäischen Dokuments prEN IEC 62541-14:2018 „OPC Unified Architecture – Teil 14: Festlegungen zur Umsetzung des Publisher-Subscriber-Models“ (Entwurf in der Umfrage) ist unverändert in diesen Norm-Entwurf übernommen worden.

Die Internationale Elektrotechnische Kommission (IEC) und das Europäische Komitee für Elektrotechnische Normung (CENELEC) haben vereinbart, dass ein auf IEC-Ebene erarbeiteter Entwurf für eine Internationale Norm zeitgleich (parallel) bei IEC und CENELEC zur Umfrage (CDV-Stadium) und Abstimmung als FDIS (en: Final Draft International Standard) bzw. Schluss-Entwurf für eine Europäische Norm gestellt wird, um eine Beschleunigung und Straffung der Normungsarbeit zu erreichen. Dem entsprechend ist das internationale Dokument IEC 65E/617/CDV:2018 „OPC Unified Architecture – Part 14: PubSub“ unverändert in den Entwurf prEN IEC 62541-14:2018 übernommen worden.

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

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

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

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

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

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

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

Da sich die Benutzer des vorliegenden Norm-Entwurfs der englischen Sprache als Fachsprache bedienen, wird die englische Fassung der prEN IEC 62541-14 veröffentlicht. Zu deren Abschnitt 1, der den Anwendungsbereich festlegt, und Unterabschnitt 3.1, der die Begriffe festlegt, wurde eine Übersetzung angefertigt und als informativer Nationaler Anhang NA des vorliegenden Norm-Entwurfs hinzugefügt. Für einige der verwendeten Begriffe existieren keine gebräuchlichen deutschen Benennungen, da sich die deutschen Anwender in der Regel ebenfalls der englischen Benennungen bedienen. Dieser Norm-Entwurf steht nicht in unmittelbarem Zusammenhang mit Rechtsvorschriften und ist nicht als Sicherheitsnorm anzusehen.

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

Es wird auf die Möglichkeit hingewiesen, dass einige Elemente dieses Dokuments Patentrechte berühren können. IEC ist nicht dafür verantwortlich, einige oder alle diesbezüglichen Patentrechte zu identifizieren.

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

Documents
Document Sharing Hub

Full document details: E DIN EN IEC 62541-14:2019-09

Nationaler Anhang NA
(informativ)

Auszugsweise Übersetzung

Reihenfolge und Inhalt der folgenden Abschnitte sind identisch mit denen in den jeweiligen Abschnitten der Englischen Fassung.

1 Anwendungsbereich

Dieses Dokument definiert das *PubSub*-Kommunikationsmodell der OPC Unified Architecture (OPC UA). Es legt das OPC UA-Publish/Subscribe-Muster fest, das das durch die *Dienste* in IEC 62541-4 definierte Client-Server-Muster ergänzt. Für eine Übersicht der beiden Modelle und ihre unterschiedlichen Anwendungsmöglichkeiten siehe IEC TR 62541-1

PubSub ermöglicht das Verteilen von Daten und Ereignissen aus einer OPC UA-Informationsquelle an interessierte Beobachter in einem Gerätenetzwerk sowie in IT- und Analyse-Cloud-Systemen.

Die Spezifikation enthält die folgenden Elemente:

- eine allgemeine Einführung in die PubSub-Konzepte;
- eine Definition der PubSub-Konfigurationsparameter;
- eine Zuordnung des PubSub-Konzepts und der Konfigurationsparameter zu Mitteilungs- und Transportprotokollen;
- sowie ein PubSub-Konfigurationsmodell.

Nicht alle OPC UA-*Anwendungen* müssen alle definierten Mitteilungs- und Transportprotokoll-Zuordnungen implementieren. IEC 62541-7 legt das *Profil* fest, das vorgibt, welche Zuordnungen implementiert werden müssen, um mit einem bestimmten *Profil* übereinzustimmen.

3 Begriffe und Konventionen

3.1 Begriffe

Für die Anwendung dieses Dokuments gelten die Begriffe nach IEC TR 62541-1, IEC TR 62541-2, IEC 62541-3, IEC 62541-4 und die folgenden Begriffe.

3.1.1
DataSetClass
(en: DataSetClass)
Vorlage, die den Inhalt eines *DataSet* festlegt

Anmerkung 1 zum Begriff: Eine *DataSetClass* wird verwendet, um *DataSets* zur Verwendung in mehreren *Publishern* und zum Filtern in *Subscribern* einzugeben.

3.1.2
DataSetMetaData
(en: DataSetMetaData)
beschreibt die Inhalte und Semantik eines *DataSet*

3.1.3
DataSetReader
(en: DataSetReader)
Einheit, die *DataSetMessages* von einer *Message Oriented Middleware* erhält

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

Anmerkung 1 zum Begriff: Ein *DataSetReader* ist eine Komponente, die eine *DataSetMessage* aus einer *NetworkMessage* extrahiert, die von der *Message Oriented Middleware* empfangen wurde und die *DataSetMessage* zur weiteren Verarbeitung in dem *Subscsriber* in einen *DataSet* umwandelt.

3.1.4
DataSetWriter
(en: DataSetWriter)
Einheit, die *DataSetMessages* aus *DataSets* erstellt und diese über eine *Message Oriented Middleware* veröffentlicht

Anmerkung 1 zum Begriff: Ein *DataSetWriter* wandelt einen *DataSet* in eine *DataSetMessage* um und fügt die *DataSetMessage* in eine *NetworkMessage* ein, um sie über eine *Message Oriented Middleware* zu veröffentlichen.

3.1.5
PublishedDataSet
(en: PublishedDataSet)
Konfiguration der Anwendungsdaten, die als *DataSet* veröffentlicht werden

Anmerkung 1 zum Begriff: Ein *PublishedDataSet* kann eine Liste überwachter *Variablen* oder eine *Ereignis*-Auswahl (en: *Event*) sein.

3.1.6
SecurityGroup
(en: SecurityGroup)
Gruppierung von Sicherheitseinstellungen und Sicherheitsschlüsseln für den Zugriff auf Mitteilungen eines *Publishers*

Anmerkung 1 zum Begriff: Eine *SecurityGroup* ist eine Abstraktion, die die Sicherheitseinstellungen und Sicherheitsschlüssel repräsentiert, die für den Zugriff auf Mitteilungen eines *Publishers* verwendet werden. Eine *SecurityGroup* verfügt über einen eindeutigen Bezeichner, der *SecurityGroupId* genannt wird. Die *SecurityGroupId* ist innerhalb des Sicherheitsschlüsseldienstes einzigartig.

3.1.7
SubscribedDataSet
(en: SubscribedDataSet)
Konfiguration für den Versand von empfangenen *DataSets*

Anmerkung 1 zum Begriff: Ein *SubscribedDataSet* kann eine Zuordnung von *DataSet*-Feldern zu Variablen im *Subscriber-AddressSpace* sein.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

CONTENTS

FIGURES 5
TABLES 6
FOREWORD 11
1 Scope 13
2 Normative references 13
3 Terms, definitions and conventions 14
3.1 Terms and definitions 14
3.2 Abbreviations and symbols 14
4 Overview 15
4.1 Fields of application 15
4.2 Abstraction layers 15
4.3 Decoupling by use of middleware 16
4.4 Synergy of models 16
5 PubSub Concepts 18
5.1 Introduction 18
5.2 DataSet 19
5.2.1 General 19
5.2.2 DataSetClass 20
5.2.3 DataSetMetaData 20
5.3 Messages 21
5.3.1 General 21
5.3.2 DataSetMessage field 21
5.3.3 DataSetMessage 22
5.3.4 NetworkMessage 22
5.3.5 Message Security 22
5.3.6 Transport Security 23
5.3.7 SecurityGroup 23
5.4 Entities 23
5.4.1 Publisher 23
5.4.2 Subscriber 26
5.4.3 Security Key Service 27
5.4.4 Message Oriented Middleware 30
6 PubSub Communication Parameters 34
6.1 Overview 34
6.2 Common Configuration Parameters 35
6.2.1 PubSubState State Machine 35
6.2.2 PublishedDataSet Parameters 36
6.2.3 DataSetWriter Parameters 42
6.2.4 Shared PubSubGroup Parameters 46
6.2.5 WriterGroup Parameters 47
6.2.6 PubSubConnection Parameters 49
6.2.7 ReaderGroup Parameters 51
6.2.8 DataSetReader Parameters 52

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

6.2.9 SubscribedDataSet Parameters 55
6.2.10 Information flow and status handling 57
6.2.11 PubSubConfigurationDataType 58
6.3 Message Mapping Configuration Parameters 59
6.3.1 UADP Message Mapping 59
6.3.2 JSON Message Mapping 66
6.4 Transport Protocol Mapping Configuration Parameters 68
6.4.1 Datagram Transport Protocol 68
6.4.2 Broker Transport Protocol 69
7 PubSub Mappings 74
7.1 General 74
7.2 Message Mappings 74
7.2.1 General 74
7.2.2 UADP Message Mapping 75
7.2.3 JSON Message Mapping 88
7.3 Transport Protocol Mappings 91
7.3.1 General 91
7.3.2 OPC UA UDP 91
7.3.3 OPC UA Ethernet 92
7.3.4 AMQP 93
7.3.5 MQTT 97
8 PubSub Security Key Service Model 99
8.1 Overview 99
8.2 PublishSubscribe Object 100
8.3 PubSubKeyServiceType 100
8.4 GetSecurityKeys Method 100
8.5 GetSecurityGroup Method 102
8.6 SecurityGroupType 102
8.7 SecurityGroupFolderType 103
8.8 AddSecurityGroup Method 103
8.9 RemoveSecurityGroup Method 104
9 PubSub Configuration Model 105
9.1 Common Configuration Model 105
9.1.1 General 105
9.1.2 Configuration behaviours 107
9.1.3 Types for the PublishSubscribe Object 108
9.1.4 Published DataSet Model 112
9.1.5 Connection Model 125
9.1.6 Group Model 128
9.1.7 DataSetWriter Model 134
9.1.8 DataSetReader Model 136
9.1.9 Subscribed DataSet Model 140
9.1.10 PubSub Status Object 142
9.1.11 PubSub Diagnostics Objects 143
9.1.12 PubSub Status Events 150
9.2 Message Mapping Configuration Model 151
9.2.1 UADP Message Mapping 151
9.2.2 JSON Message Mapping 152
9.3 Transport Protocol Mapping Configuration Model 154

AVA Documents Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

9.3.1 Datagram Transport Protocol Mapping 154
9.3.2 Broker Transport Protocol Mapping 155
Annex A (normative) Common Types 158
A.1 DataType Schema Header Structures 158
A.1.1 DataTypeSchemaHeader 158
A.1.2 DataTypeDescription 158
A.1.3 StructureDescription 159
A.1.4 EnumDescription 159
A.1.5 SimpleTypeDescription 160
A.2 UABinaryFileDataType 161
A.3 NetworkAddress Model 161
A.3.1 NetworkAddressType 161
A.3.2 NetworkAddressUrlType 162
Annex B (informative) Client Server vs. Publish Subscribe 163
B.1 Overview 163
B.2 Client Server Subscriptions 163
B.3 Publish-Subscribe 164
B.4 Synergy of models 165

FIGURES

Figure 1 – Publish Subscribe Model Overview 16
Figure 2 – Publisher and Subscriber entities 18
Figure 3 – DataSet in the process of publishing 19
Figure 4 – OPC UA PubSub Message Layers 21
Figure 5 – Publisher details 24
Figure 6 – Publisher message sending sequence 25
Figure 7 – Subscriber details 26
Figure 8 – Subscriber message reception sequence 27
Figure 9 – SecurityGroup Management Sequence 28
Figure 10 – Handshake used to pull keys from SKS 29
Figure 11 – Handshake used to push keys to Publishers and Subscribers 29
Figure 12 – Handshake with a Security Key Service 30
Figure 13 – PubSub using network infrastructure 31
Figure 14 – UDP Multicast Overview 31
Figure 15 – PubSub using broker 32
Figure 16 – Broker Overview 33
Figure 17 – PubSub Component Overview 34
Figure 18 – PubSub Mapping Specific Parameters Overview 35
Figure 19 – PubSub Component State Dependencies 36
Figure 20 – PubSubState State Machine 36
Figure 21 – PubSub Information Flow dependency to field representation 44
Figure 22 – PubSub Information Flow 57
Figure 23 – Start of the periodic publisher execution 59
Figure 24 – Timing offsets in a PublishingInterval 60

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

Figure 25 – DataSetOrdering and MaxNetworkMessageSize 61
Figure 26 – PublishingOffset options for multiple *NetworkMessages* 63
Figure 27 – UADP NetworkMessage 75
Figure 28 – UADP DataSet Payload 80
Figure 29 – DataSetMessage Header Structure 81
Figure 30 – Data Key Frame DataSetMessage Data 83
Figure 31 – Data Delta Frame DataSetMessage 84
Figure 32 – Event DataSetMessage 85
Figure 33 – KeepAlive Message 86
Figure 34 – PublishSubscribe Object Types Overview 99
Figure 35 – PubSub Configuration Model Overview 105
Figure 36 – PubSub Example Objects 106
Figure 37 – PubSub Information Flow 106
Figure 38 – PublishSubscribe Object Types Overview 108
Figure 39 – Published DataSet Overview 112
Figure 40 – PubSubConnectionType Overview 126
Figure 41 – PubSubGroupType Overview 129
Figure 42 – DataSet Writer Model Overview 135
Figure 43 – DataSet Reader Model Overview 137
Figure 44 – PubSub Diagnostics Overview 144
Figure 45 – PubSubDiagnosticsCounterType 144
Figure B.46 – Subscriptions in OPC UA Client Server Model 164
Figure B.47 – Publish Subscribe Model Overview 165

TABLES

Table 1 – PubSubState Values 35
Table 2 – PubSubState State Machine 36
Table 3 – DataSetMetaDataType Structure 37
Table 4 – DataSetMetaDataType Definition 37
Table 5 – FieldMetaData Structure 37
Table 6 – DataSetFieldFlags Values 38
Table 7 – DataSetFieldFlags Definition 39
Table 8 – ConfigurationVersionDataType Structure 39
Table 9 – PublishedDataSetDataType Structure 40
Table 10 – PublishedDataSetSourceDataType Definition 40
Table 11 – PublishedVariableDataType Structure 41
Table 12 – PublishedDataItemsDataType Structure 41
Table 13 – PublishedEventsDataType Structure 42
Table 14 – DataSetFieldContentMask Values 43
Table 15 – DataSetFieldContentMask Definition 43
Table 16 – DataSetMessage field representation options 44
Table 17 – DataSetWriterDataType Structure 46

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

Table 18 – DataSetWriterTransportDataType Definition 45
Table 19 – DataSetWriterMessageDataType Structure 45
Table 20 – PubSubGroupDataType Structure 47
Table 21 – PubSubGroupDataType Definition 47
Table 22 – WriterGroupDataType Structure 48
Table 23 – WriterGroupDataType Definition 48
Table 24 – WriterGroupTransportDataType Definition 49
Table 25 – WriterGroupMessageDataType Structure 49
Table 26 – PubSubConnectionDataType Structure 50
Table 27 – ConnectionTransportDataType Definition 50
Table 28 – NetworkAddressDataType Structure 50
Table 29 – NetworkAddressDataType Definition 51
Table 30 – NetworkAddressUrlDataType Structure 51
Table 31 – NetworkAddressUrlDataType Definition 51
Table 32 – ReaderGroupDataType Structure 51
Table 33 – ReaderGroupDataType Definition 52
Table 34 – ReaderGroupTransportDataType Definition 52
Table 35 – ReaderGroupMessageDataType Structure 52
Table 36 – DataSetReaderDataType Structure 54
Table 37 – DataSetReaderTransportDataType Structure 54
Table 38 – DataSetReaderTransportDataType Definition 54
Table 39 – DataSetReaderMessageDataType Structure 54
Table 40 – DataSetReaderMessageDataType Definition 55
Table 41 – SubscribedDataSetDataType Structure 55
Table 42 – SubscribedDataSetDataType Definition 55
Table 43 – TargetVariablesDataType Structure 55
Table 44 – FieldTargetDataType Structure 56
Table 45 – OverrideValueHandling Values 56
Table 46 – SubscribedDataSetMirrorDataType Structure 57
Table 47 – Source to message input mapping 58
Table 48 – Message output to target mapping 58
Table 49 – PubSubConfigurationDataType Structure 58
Table 50 – PubSubConfiguration File Content 59
Table 51 – DataSetOrderingType Values 61
Table 52 – UadpNetworkMessageContentMask Values 62
Table 53 – UadpNetworkMessageContentMask Definition 62
Table 54 – UadpWriterGroupMessageDataType Structure 63
Table 55 – UadpDataSetMessageContentMask Values 64
Table 56 – UadpDataSetMessageContentMask Definition 64
Table 57 – UadpDataSetWriterMessageDataType Structure 65
Table 58 – UadpDataSetReaderMessageDataType Structure 66
Table 59 – JsonNetworkMessageContentMask Values 66
Table 60 – JsonNetworkMessageContentMask Definition 67

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

Table 61 – JsonWriterGroupMessageDataType Structure 67
Table 62 – JsonDataSetMessageContentMask Values 67
Table 63 – JsonDataSetMessageContentMask Definition 67
Table 64 – JsonDataSetWriterMessageDataType Structure 68
Table 65 – JsonDataSetReaderMessageDataType Structure 68
Table 66 – DatagramConnectionTransportDataType Structure 68
Table 67 – DatagramWriterGroupTransportDataType Structure 69
Table 68 – BrokerConnectionTransportDataType Structure 70
Table 69 – BrokerTransportQualityOfService Values 70
Table 70 – BrokerWriterGroupTransportDataType Structure 71
Table 71 – BrokerDataSetWriterTransportDataType Structure 72
Table 72 – BrokerDataSetReaderTransportDataType Structure 73
Table 73 – UADP NetworkMessage 76
Table 74 – Layout of the key data for UADP message security 78
Table 75 – Layout of the MessageNonce for AES-CTR 78
Table 76 – Layout of the counter block for UADP message security 79
Table 77 – Chunked NetworkMessage Payload Header 79
Table 78 – Chunked NetworkMessage Payload Fields 79
Table 79 – UADP DataSet Payload Header 80
Table 80 – UADP DataSet Payload 80
Table 81 – DataSetMessage Header Structure 82
Table 82 – Data Key Frame DataSetMessage Structure 83
Table 83 – Data Delta Frame DataSetMessage Structure 84
Table 84 – Event DataSetMessage Structure 85
Table 85 – Discovery Request Header Structure 87
Table 86 – Publisher Information Request Message Structure 87
Table 87 – Discovery Response Header Structure 87
Table 88 – Publisher Endpoints Message Structure 88
Table 89 – DataSetMetaData Message Structure 88
Table 90 – DataSetWriter Configuration Message Structure 88
Table 91 – JSON NetworkMessage Definition 89
Table 92 – JSON DataSetMessage Definition 90
Table 93 – JSON DataSetMetaData Definition 91
Table 94 – UADP message transported over UDP 91
Table 95 – UADP message transported over Ethernet 92
Table 96 – AMQP Standard Header Fields 94
Table 97 - OPC UA AMQP Standard Header QualifiedName Name mappings 95
Table 98 – OPC UA AMQP Header Field Conversion Rules 96
Table 99 – PublishSubscribe Object Definition 100
Table 100 – PubSubKeyServiceType Definition 100
Table 101 – SecurityGroupType Definition 102
Table 102 – SecurityGroupFolderType Definition 103
Table 103 – PublishSubscribeType Definition 103

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

Table 104 – HasPubSubConnection ReferenceType 111
Table 105 – PublishedDataSetType Definition 113
Table 106 – ExtensionFieldsType Definition 114
Table 107 – Well-Known Extension Field Names 114
Table 108 – DataSetToWriter ReferenceType 116
Table 109 – PublishedDataItemsType Definition 116
Table 110 – PublishedEventsType Definition 118
Table 111 – DataSetFolderType Definition 120
Table 112 – PubSubConnectionType Definition 126
Table 113 – ConnectionTransportType Definition 128
Table 114 – PubSubGroupType Definition 129
Table 115 – WriterGroupType Definition 130
Table 116 – HasDataSetWriter ReferenceType 132
Table 117 – WriterGroupTransportType Definition 132
Table 118 – WriterGroupMessageType Definition 132
Table 119 – ReaderGroupType Definition 132
Table 120 – HasDataSetReader ReferenceType 134
Table 121 – ReaderGroupTransportType Definition 134
Table 122 – ReaderGroupMessageType Definition 134
Table 123 – DataSetWriterType Definition 135
Table 124 – DataSetWriterTransportType Definition 136
Table 125 – DataSetWriterMessageType Definition 136
Table 126 – DataSetReaderType Definition 137
Table 127 – DataSetReaderTransportType Definition 138
Table 128 – DataSetReaderMessageType Definition 138
Table 129 – SubscribedDataSetType Definition 140
Table 130 – TargetVariablesType Definition 140
Table 131 – SubscribedDataSetMirrorType Definition 142
Table 132 – PubSubStatusType Definition 142
Table 133 – Status Object Definition 143
Table 134 – PubSubDiagnosticsType 144
Table 135 – Counters for PubSubDiagnosticsType 145
Table 136 – DiagnosticsLevel Values 146
Table 137 – PubSubDiagnosticsCounterType 146
Table 138 – PubSubDiagnosticsCounterClassification Values 147
Table 139 – PubSubDiagnosticsRootType 147
Table 140 – LiveValues for PubSubDiagnosticsRootType 147
Table 141 – PubSubDiagnosticsConnectionType 147
Table 142 – LiveValues for PubSubDiagnosticsConnectionType 148
Table 143 – PubSubDiagnosticsWriterGroupType 148
Table 144 – Counters for PubSubDiagnosticsWriterGroupType 148
Table 145 – LiveValues for PubSubDiagnosticsWriterGroupType 148
Table 146 – PubSubDiagnosticsReaderGroupType 148

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

Table 147 – Counters for PubSubDiagnosticsReaderGroupType 149
Table 148 – LiveValues for PubSubDiagnosticsReaderGroupType 149
Table 149 – PubSubDiagnosticsDataSetWriterType 149
Table 150 – Counters for PubSubDiagnosticsDataSetWriterType 149
Table 151 – LiveValues for PubSubDiagnosticsDataSetWriterType 149
Table 152 – PubSubDiagnosticsDataSetReaderType 150
Table 153 – Counters for PubSubDiagnosticsDataSetReaderType 150
Table 154 – LiveValues for PubSubDiagnosticsDataSetReaderType 150
Table 155 – PubSubStatusEventType Definition 150
Table 156 – PubSubTransportLimitsExceedEventType Definition 151
Table 157 – PubSubCommunicationFailureEventType Definition 151
Table 158 – UadpWriterGroupMessageType Definition 152
Table 159 – UadpDataSetWriterMessageType Definition 152
Table 160 – UadpDataSetReaderMessageType Definition 153
Table 161 – JsonWriterGroupMessageType Definition 153
Table 162 – JsonDataSetWriterMessageType Definition 154
Table 163 – JsonDataSetReaderMessageType Definition 154
Table 164 – DatagramConnectionTransportType Definition 154
Table 165 – DatagramWriterGroupTransportType Definition 155
Table 166 – BrokerConnectionTransportType Definition 155
Table 167 – BrokerWriterGroupTransportType Definition 155
Table 168 – BrokerDataSetWriterTransportType Definition 156
Table 169 – Broker Writer Well-Known Extension Field Names 156
Table 170 – BrokerDataSetReaderTransportType Definition 156
Table A.1 – DataTypeSchemaHeader Structure 158
Table A.2 – DataTypeSchemaHeader Definition 158
Table A.3 – DataTypeDescription Structure 159
Table A.4 – DataTypeDescription Definition 159
Table A.5 – StructureDescription Structure 159
Table A.6 – StructureDescription Definition 159
Table A.7 – EnumDescription Structure 160
Table A.8 – EnumDescription Definition 160
Table A.9 – SimpleTypeDescription Structure 160
Table A.10 – UABinaryFileDataType Structure 161
Table A.11 – UABinaryFileDataType Definition 161
Table A.12 – NetworkAddressType Definition 161
Table A.13 – NetworkAddressUrlType Definition 162

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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.

The main task of IEC technical committees is to prepare International Standards. However, a technical committee may propose the publication of a technical report when it has collected data of a different kind from that which is normally published as an International Standard, for example "state of the art".

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 technical report is based on the following documents:

Enquiry draft	Report on voting
65E/XX/DTR	65E/XX/RVC

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

This is the first edition of edition of IEC 62541-14.

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

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

Throughout this document and the referenced other Parts of the series, certain document conventions are used:

Italics are used to denote a defined term or definition that appears in the “Terms and definition” clause in one of the parts of the 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 often 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 publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

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

The National Committees are requested to note that for this publication the stability date is 2021.

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

A bilingual version of this publication may be issued at a later date.

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.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

OPC Unified Architecture Specification

Part 14: PubSub

1 Scope

This specification 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. See IEC TR 62541-1 for an overview of the two models and their distinct uses.

PubSub allows distributing 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.

The specification consists of

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

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

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application.

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) Version 1.0

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

RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

http://www.ietf.org/rfc/rfc7159.txt

3 Terms, definitions and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC TR 62541-1, IEC TR 62541-2, *OPC Unified Architecture – Part 2: Security Model*

IEC 62541-3, and IEC 62541-4, as well as the following apply.

3.1.1
DataSetClass
template declaring the content of a *DataSet*

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

3.1.2
DataSetMetaData
describes the content and semantic of a *DataSet*

3.1.3
DataSetReader
entity receiving *DataSetMessages* from a *Message Oriented Middleware*

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

3.1.4
DataSetWriter
entity creating *DataSetMessages* from *DataSets* and publishing them through a *Message Oriented Middleware*

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

3.1.5
PublishedDataSet
configuration of application-data to be published as *DataSet*

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

3.1.6
SecurityGroup
grouping of security settings and security keys used to access messages from a *Publisher*

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

3.1.7
SubscribedDataSet
configuration for dispatching of received *DataSets*

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

3.2 Abbreviations and symbols

AMQP Advanced Message Queuing Protocol
AS Authorization Service
CA Certificate Authority
CRL Certificate Revocation List
CTL Certificate Trust List

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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
STS	Security Token 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 this clause and do not represent a specification for implementation. The concrete communication parameters are specified in 6. The concrete transport protocol mappings and message mappings are later specified in 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.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

- The use of established standard messaging protocols (e.g. AMQP or MQTT) with JSON data encoding supports the cloud integration path and readily allows handling of the information in modern stream and batch analytics systems.

4.3 Decoupling by use of middleware

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

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

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

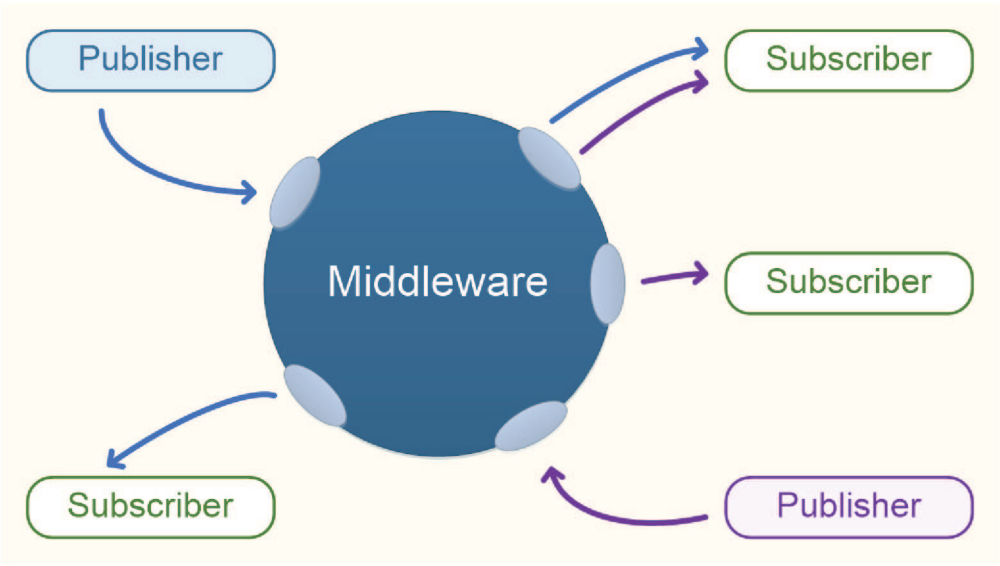


Figure 1 – Publish Subscribe Model Overview

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

- 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)

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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.

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

5 PubSub Concepts

5.1 Introduction

This clause 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*.

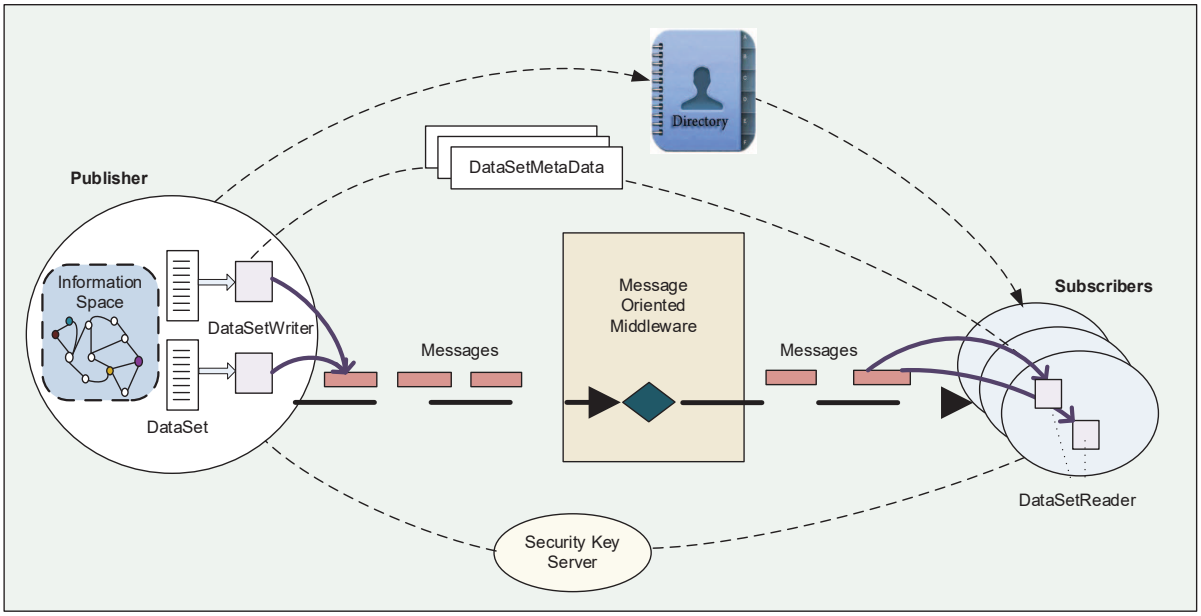


Figure 2 – Publisher and Subscriber entities

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

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

the *Publisher* and the data acquisition parameters are called *PublishedDataSet*. *DataSet*, *DataSetMetaData* and *PublishedDataSet* are detailed in 5.2.

Note 1: 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 release of this specification.

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.

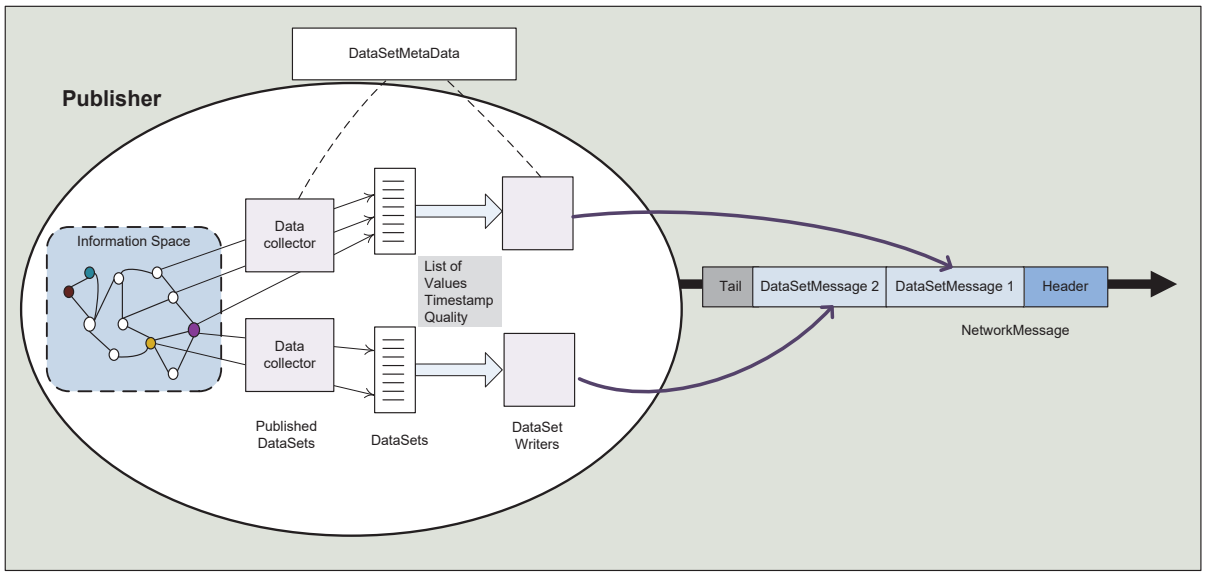


Figure 3 – DataSet in the process of publishing

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 clause 8.2 or with vendor specific configuration tools.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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 *DataSetMetaDataType* 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 *ConfigurationVersionDataType* 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*, from a configuration server or from a directory 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

AVA Documents Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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 specification 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.

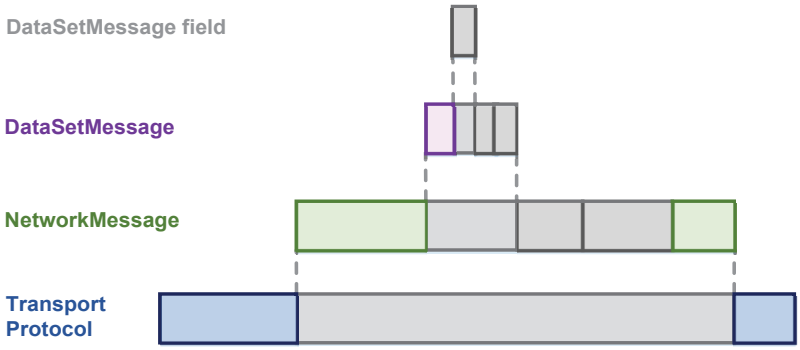


Figure 4 – OPC UA PubSub Message Layers

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

Following is 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 should be included in the *DataSetMessage*.

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

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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 *DataSetMessage*. Can be used to verify the ordering and to detect missing messages.
Timestamp	A timestamp describing when the data in this *DataSetMessage* was obtained.
Version	Version information about the configuration of the *DataSetMetaData*.
Status	Status information about the data in this *DataSetMessage*.
Keep alive	When no *DataSetMessages* are sent for a configured time period, a keep alive *DataSetMessage* is sent to signal the *Subscribers* that the *Publisher* 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 *Publisher*.
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 *DataSet* also sent in the header.
Payload	One or more *DataSetMessages*.

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 level of security can be:

- No security
- Signing but no encryption
- Signing and encryption

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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 standard 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* must 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.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

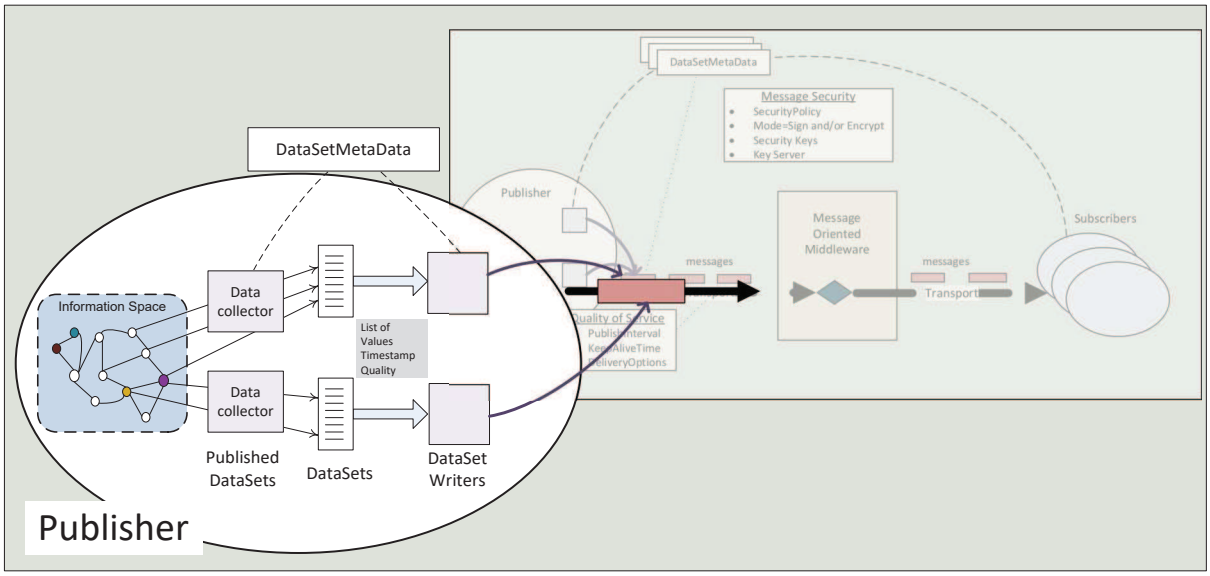


Figure 5 – Publisher details

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 have to exist to generate the OPC UA *PubSub* messages.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

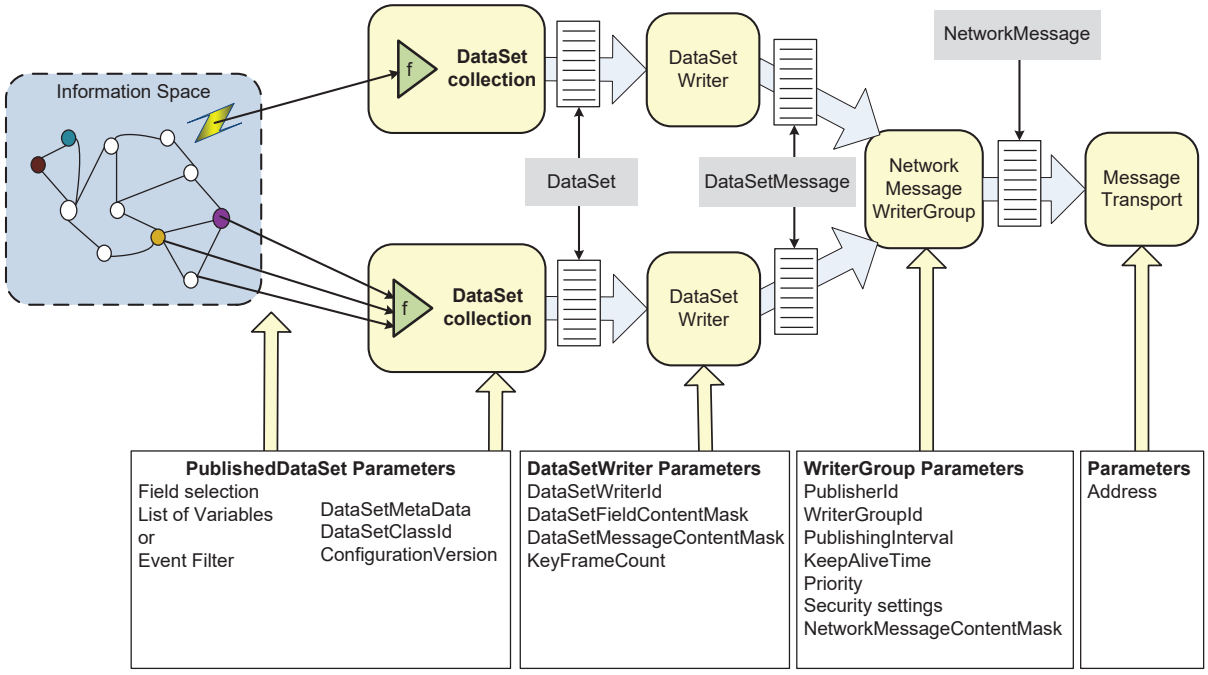


Figure 6 – Publisher message sending sequence

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 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*.

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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

*Subscriber*s 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*.

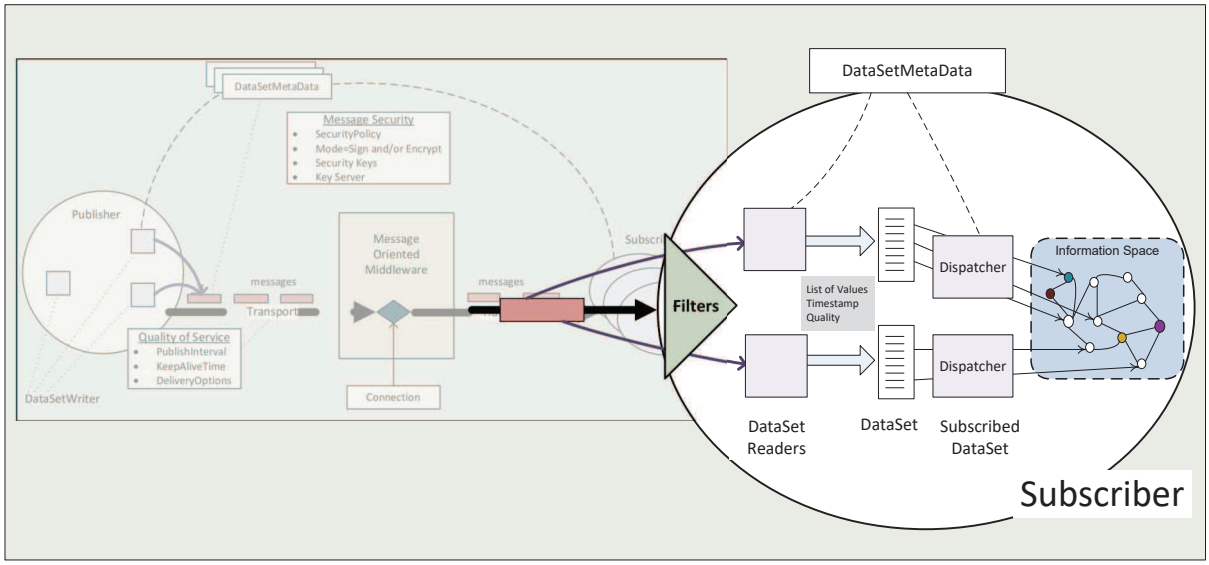


Figure 7 – Subscriber details

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

*Subscriber*s 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.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

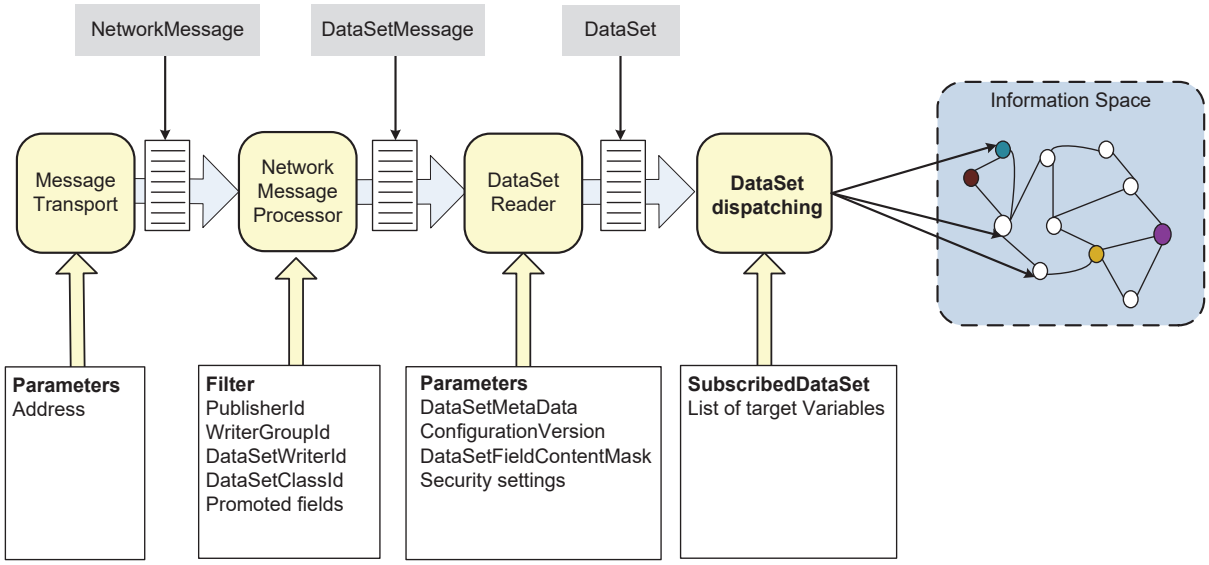


Figure 8 – Subscriber message reception sequence

The *Subscriber* has 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 has to be 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.

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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

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.

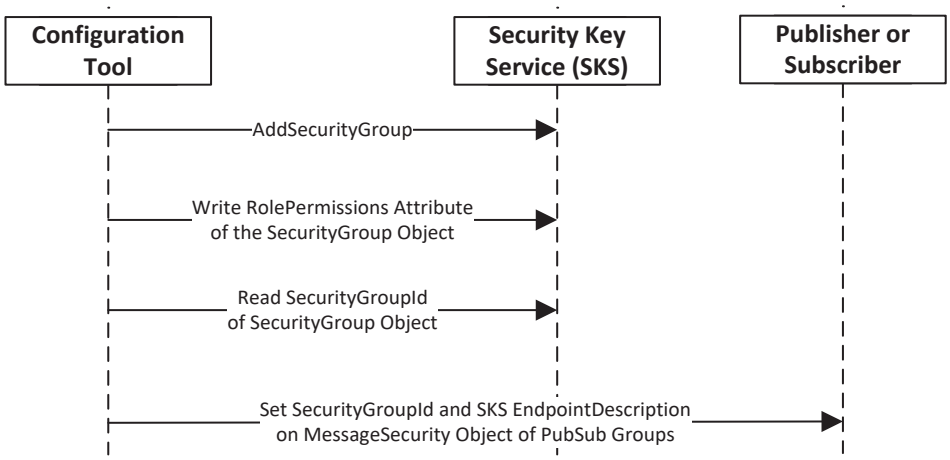


Figure 9 – SecurityGroup Management Sequence

To secure *NetworkMessages*, the *NetworkMessages* must 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* must 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* groups.

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 a SKS to *Publishers* and *Subscribers* is shown in Figure 11.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

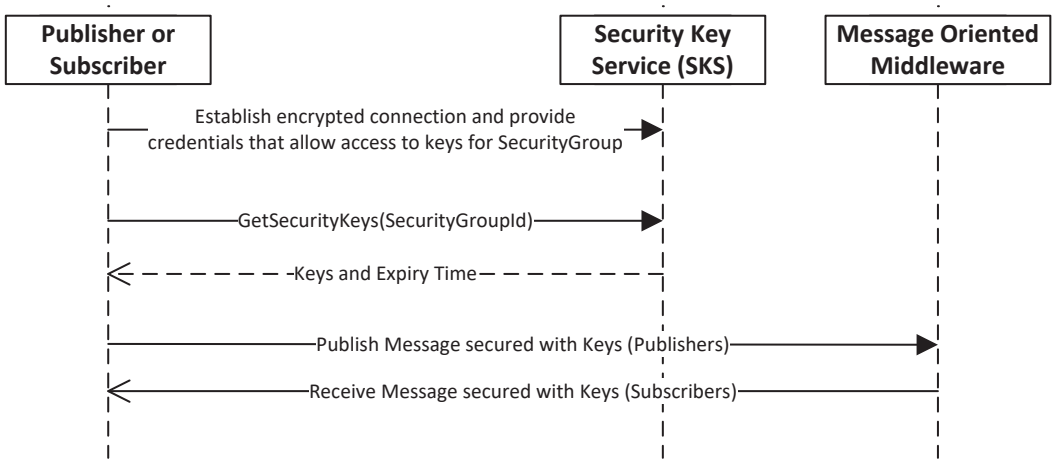


Figure 10 – Handshake used to pull keys from SKS

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.

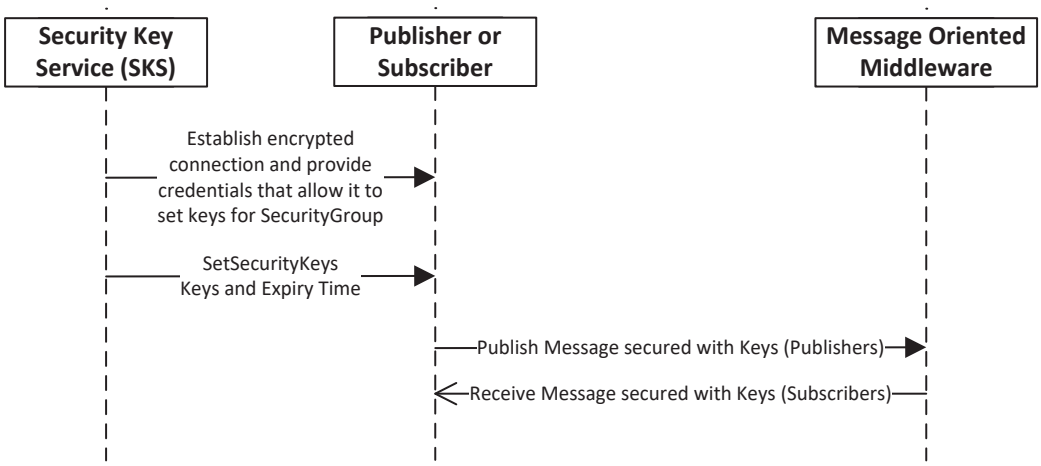


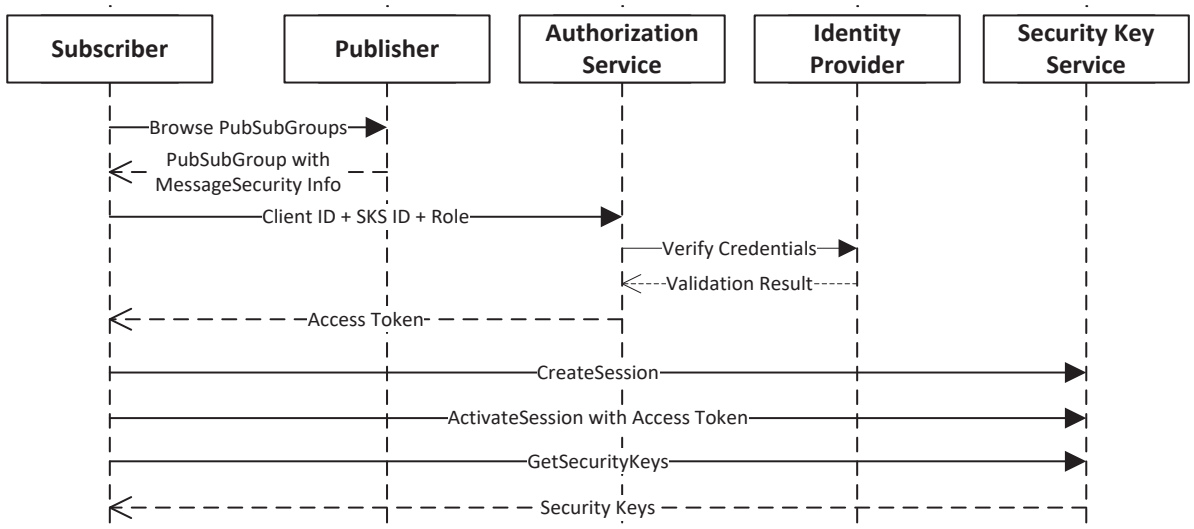
Figure 11 – Handshake used to push keys to Publishers and Subscribers

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.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09



643

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

645 The SKS is a *Server* that exposes a *Method* called *GetSecurityKeys*. The *Access Token* is
646 used to determine if the calling application is allowed to access the keys. One way to do this
647 would be to check the *Permissions* assigned to the *SecurityGroup Object* identified by the
648 *GetSecurityKeys Method* arguments. *Publishers* and *Subscribers* can request keys if the
649 *Access Token* they provide is mapped to *Roles* that have been granted *Permission* to *Browse*
650 the *SecurityGroup Object.*

651 ### 5.4.4 Message Oriented Middleware

652 #### 5.4.4.1 General

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

657 This part describes two general types of *Message Oriented Middleware* to cover a large
658 number of use cases. The two types, broker-less and broker-based middleware are described
659 in 5.4.4.2 and 5.4.4.3.

660 #### 5.4.4.2 Broker-less Middleware

661 ##### 5.4.4.2.1 General

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

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

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

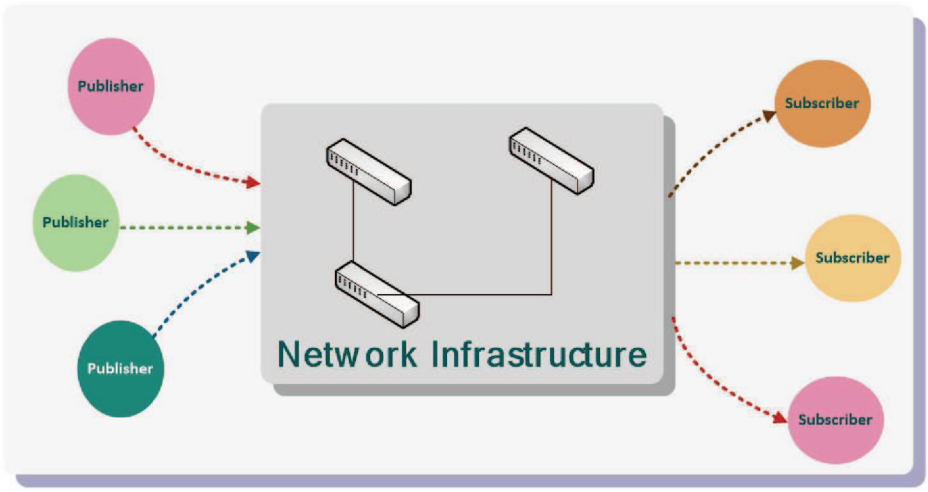


Figure 13 – PubSub using network infrastructure

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*.

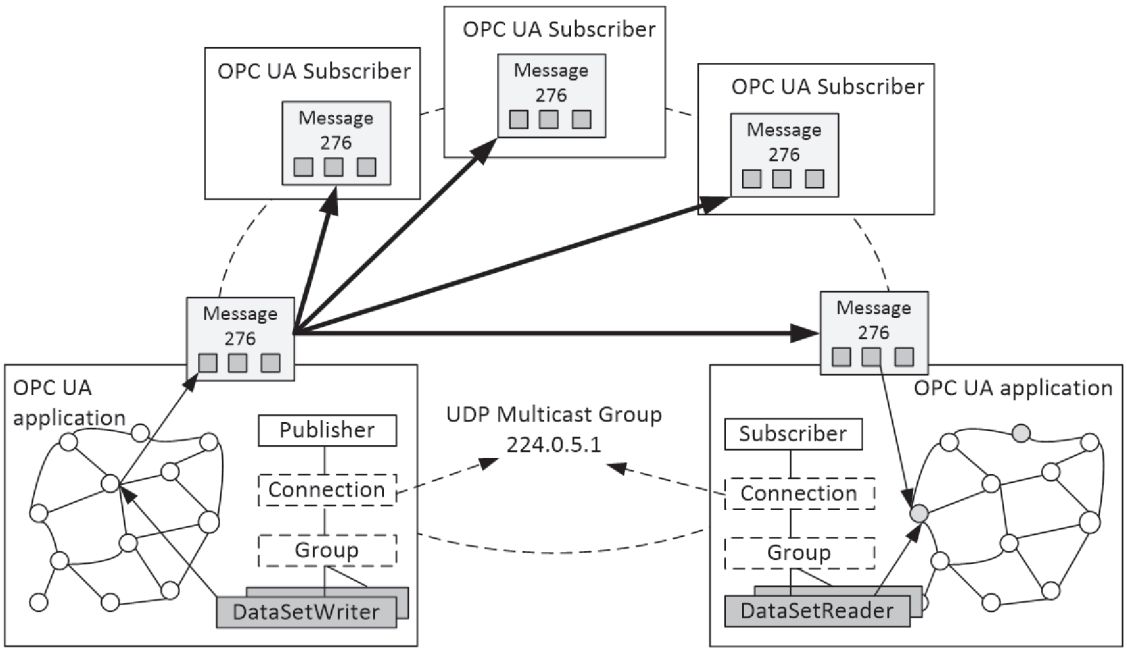


Figure 14 – UDP Multicast Overview

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*

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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).

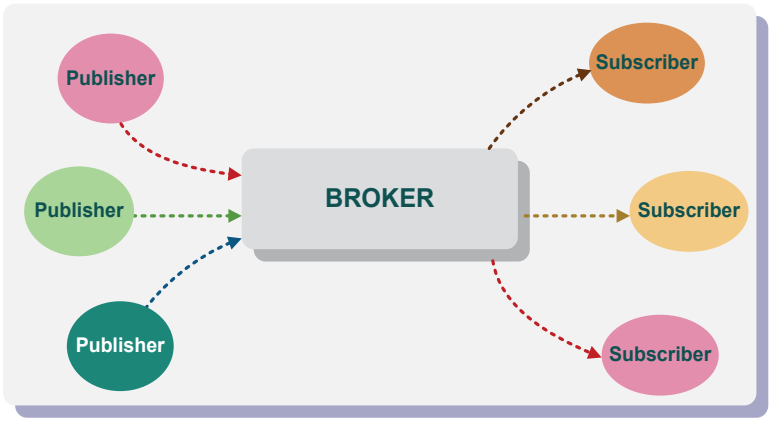


Figure 15 – PubSub using broker

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.

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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

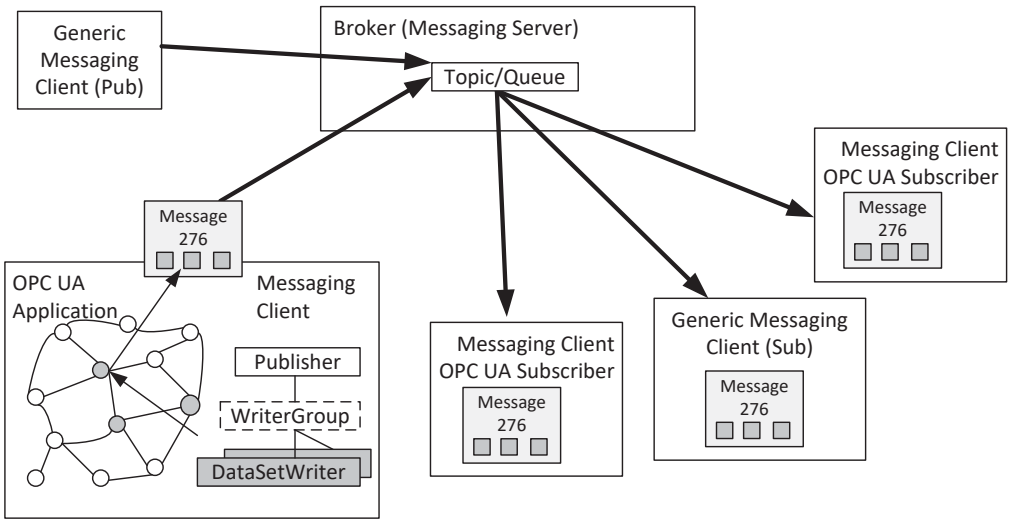


Figure 16 – Broker Overview

An OPC UA *Publisher* that publishes data may be configured through the *PubSub* configuration model. It contains a connection *Object* per *Broker*. The *Broker* is configured through an URL in the connection. The connection can have one or more groups which identity 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.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

6 PubSub Communication Parameters

6.1 Overview

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

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

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

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

Figure 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*.

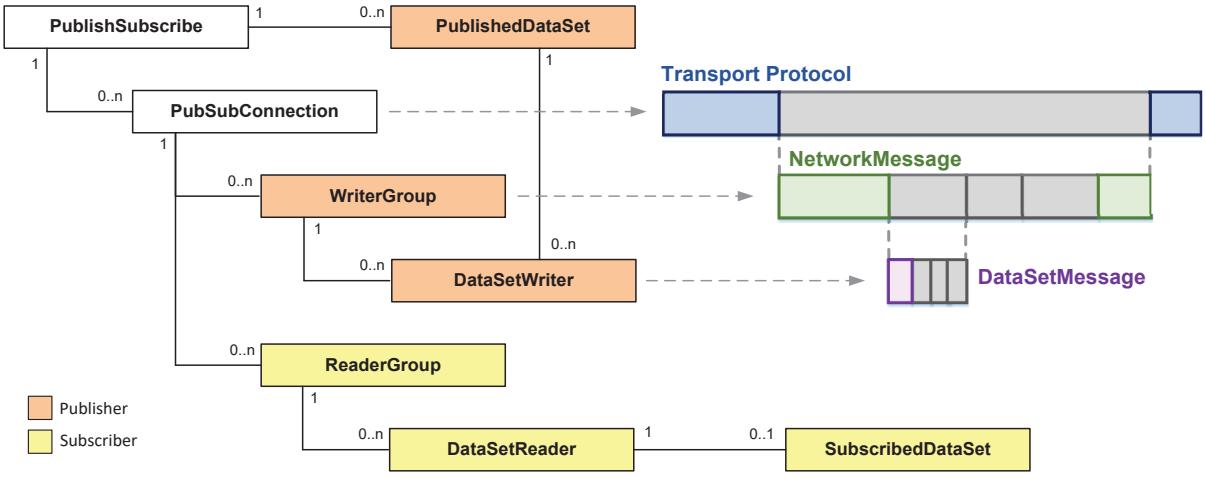


Figure 17 – PubSub Component Overview

The figure 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.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

- *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.

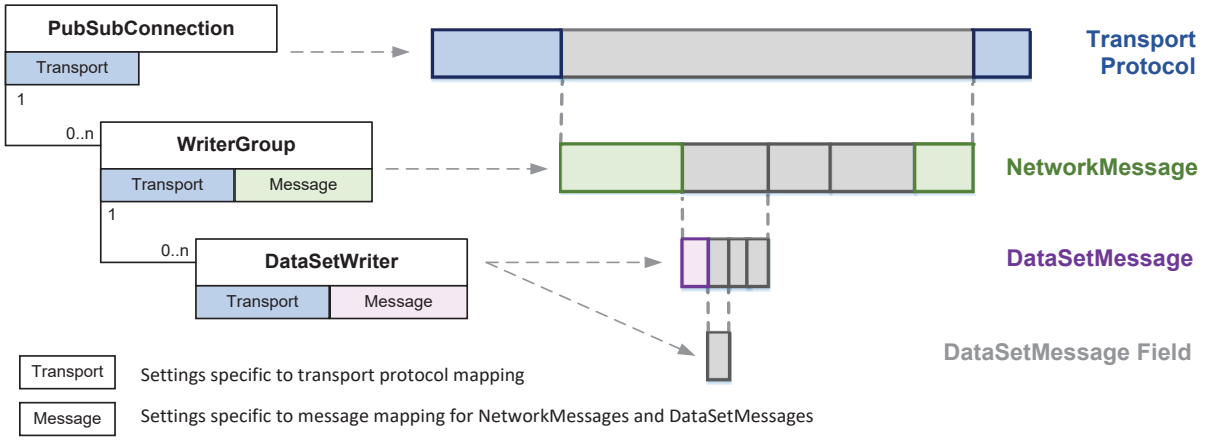


Figure 18 – PubSub Mapping Specific Parameters Overview

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 *PubSub* component is configured but currently disabled.
Paused_1	The *PubSub* component is enabled but currently paused by a parent component. The parent component is either *Disabled_0* or *Paused_1*.
Operational_2	The *PubSub* component is operational.
Error_3	The *PubSub* component is in an error state.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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.

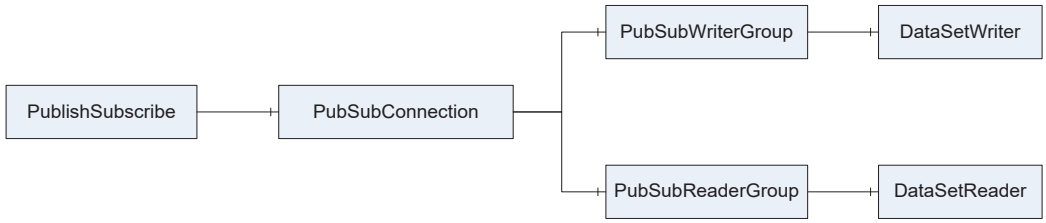


Figure 19 – PubSub Component State Dependencies

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

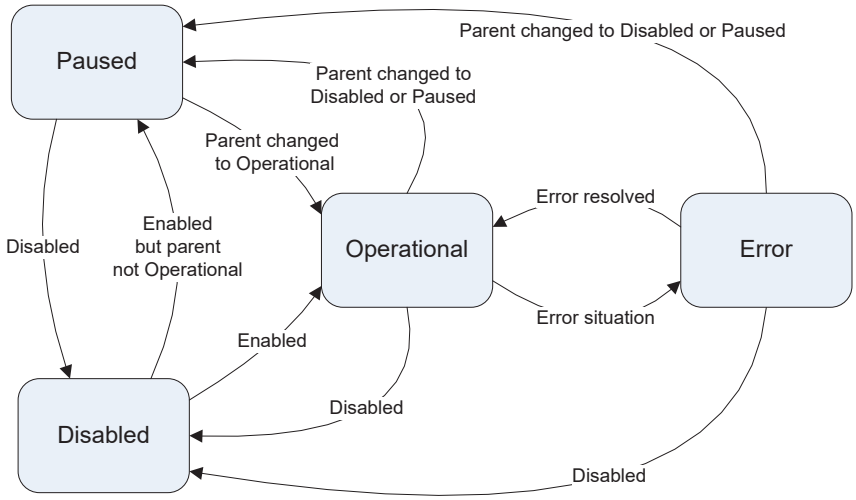


Figure 20 – PubSubState State Machine

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 *PubSub* 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 *PubSub* 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.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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 *DataSet*.
description	LocalizedText	Description of the *DataSet*. The default value is a null *LocalizedText*.
fields	FieldMetaData[]	The metadata for the fields in the *DataSet*. The FieldMetaData DataType is defined in 6.2.2.1.3.
dataSetClassId	Guid	This field provides the globally unique identifier of the class of *DataSet* if the *DataSet* is based on a *DataSetClass*. In this case, this field shall match the *DataSetClassId* of the concrete *DataSet* configuration. If the *DataSets* are not created from a class, this field is null.
configurationVersion	Configuration VersionDataType	The configuration version for the current configuration of the *DataSet*.

Its representation in the AddressSpace is defined in Table 4.

Table 4 – DataSetMetaDataType Definition

Attributes	Value
BrowseName	DataSetMetaDataType
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 *DataSet*.
description	LocalizedText	Description of the field. The default value shall be a null *LocalizedText*.
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 *DataSetMessages* as one of the built-in data types. In most cases the identifier of the *DataType NodeId* matches the built-in type. The following special cases must be handled in addition: (1) Abstract types always have the built-in type *Variant* since they can result in different concrete types in a *DataSetMessage*. The *dataType* field may provide additional restrictions e.g. if the abstract type is *Number*. Abstract types shall not be used if the field is represented as *RawData* set by the *DataSetFieldContentMask* defined in 6.2.3.1. (2) *Enumeration DataTypes* are encoded as *Int32*. The *Enumeration* strings are defined through a *DataType* referenced through the *dataType* field. (3) *Structure* and *Union DataTypes* are encoded as *ExtensionObject*. The encoding rules are defined through a *DataType* referenced through the *dataType* field. (4) *DataTypes* derived from built-in types have the *BuiltInType* of the corresponding base DataType. The concrete subtype is defined through the *dataType* field. (5) *OptionSet DataTypes* are either encoded as one of the concrete *UInteger DataTypes* or as an instance of an *OptionSetType* in an *ExtensionObject*.
dataType	NodeId	The *NodeId* of the *DataType* of this field. If the *DataType* is an *Enumeration* or an *OptionSet*, the semantic of the *Enumeration DataType* is provided through the *enumDataTypes* field of the

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

Name	Type	Description
		DataSetMetaData. If the *DataType* is a *Structure* or *Union*, the encoding and decoding description of the *Structure DataType* is provided through the *structureDataTypes* field of the *DataSetMetaData*.
valueRank	Int32	Indicates whether the *dataType* is an array and how many dimensions the array has. It may have the following values: $n > 1$: the *dataType* is an array with the specified number of dimensions. OneDimension (1): The *dataType* is an array with one dimension. OneOrMoreDimensions (0): The *dataType* is an array with one or more dimensions. Scalar (−1): The *dataType* is not an array. Any (−2): The *dataType* can be a scalar or an array with any number of dimensions. ScalarOrOneDimension (−3): The *dataType* can be a scalar or a one dimensional array. NOTE All *DataTypes* are considered to be scalar, even if they have array-like semantics like *ByteString* and *String*.
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 v*alueRank field*. This field shall be null if v*alueRank* ≤ 0. The maximum number of elements of an array transferred on the wire is 2147483647 (max Int32). It is the total number of elements in all dimensions based on the UA Binary encoding rules for arrays.
maxStringLength	UInt32	If the *dataType* field is a *String* or *ByteString* then this field specifies the maximum supported length. If the maximum is unknown the value shall be 0. If the dataType field is not a *String* or *ByteString* the value shall be 0. If the *valueRank* is greater than 0 this field applies to each element of the array.
dataSetFieldId	Guid	The unique ID for the field in the *DataSet*. 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 *Property* values providing additional semantic for the field. If at least one *Property* value changes, the *MajorVersion* of the *ConfigurationVersion* shall be updated. If the *Property* is *EngineeringUnits*, the unit of the *Field Value* shall match the unit of the *FieldMetaData*. The *KeyValuePair* DataType is defined in IEC 62541-5. For this field the key in the *KeyValuePair* structure is the *BrowseName* of the *Property* and the value in the *KeyValuePair* structure is the *Value* of the *Property*.

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 *NetworkMessages* or transport protocol header. Setting this flag increases the size of the *NetworkMessages* since information from the *DataSetMessage* 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 *DataSetMessage* payload is a delta frame and the *DataSet* 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 *DataSetMetaData* 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.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

853 **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 [ ]

854

855 #### 6.2.2.1.5 ConfigurationVersionDataType

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

858 **Table 8 – ConfigurationVersionDataType Structure**

Name	Type	Description
ConfigurationVersionDataType	Structure	
majorVersion	VersionTime	The *MajorVersion* reflects the time of the last major change of the *DataSet* content. The *VersionTime DataType* is defined in IEC 62541-4. To assure interoperability, the *Subscriber* has to use *DataSetMetaData* for decoding with a *MajorVersion* that matches the *MajorVersion* in *DataSetMessages* sent by the *Publisher*. Removing fields from the *DataSet* content, reordering fields, adding fields in between other fields or a DataType change in fields shall result in an update of the *MajorVersion*. If at least one *Property* value of a *DataSetMetaData* field changes, the *MajorVersion* shall be updated. There can be situations where older configurations of a *Publisher* are loaded and changed with product specific configuration tools. In this case the *MajorVersion* 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 *MajorVersion* or *MinorVersion* are defined in this specification.
minorVersion	VersionTime	The *MinorVersion* reflects the time of the last change. Only the *MinorVersion* shall be updated if fields are added at the end of the *DataSet* content. If the *MajorVersion* version is updated, the *MinorVersion* is updated to the same value as *MajorVersion*.

859

860 #### 6.2.2.2 DataSetClassId

861 *DataSetMetaData* may be specific to a single *Publisher* and a single selection of information
862 or universal e.g. defined by a standard organisation or by a plant operator as a *DataSetClass*.
863 *DataSets* that conform to such a *DataSetClass* are identified with a *DataSetClassId*.

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

867 #### 6.2.2.3 ExtensionFields

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

871 #### 6.2.2.4 PublishedDataSetDataType

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

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

874 **Table 9 – PublishedDataSetDataType Structure**

Name	Type	Description
PublishedDataSetDataType	Structure	
name	String	Name of the *PublishedDataSet*. The name of the *PublishedDataSet* shall be unique in the *Publisher*.
dataSetFolder	String[]	Optional path of the *DataSet* folder used to group *PublishedDataSets* where each entry in the *String* array represents one level in a *DataSet* folder hierarchy. If no grouping is needed the parameter is a null *String* 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.

875

876 ### 6.2.2.5 PublishedDataSetSourceDataType

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

880 **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.

881

882 ### 6.2.2.6 Published Data Items

883 #### 6.2.2.6.1 PublishedData

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

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

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

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

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

Table 11 – PublishedVariableDataType Structure

Name	Type	Description
PublishedVariableDataType	Structure	
publishedVariable	NodeId	The *NodeId* of the published *Variable*. Some transport protocols require knowledge on the message receiver side about the DataType, ValueRank and ArrayDimensions to be able to decode the message content. This information is provided through the *DataSetMetaData* provided for the *DataSet*.
attributeId	IntegerId	Id of the *Attribute* to publish e.g. the *Value Attribute*. This shall be a valid *Attribute* id. The *Attributes* are defined in IEC TR 62541-2, *OPC Unified Architecture – Part 2: Security Model* IEC 62541-3. The *IntegerId DataType* is defined in IEC 62541-4. The *IntegerIds* for the *Attributes* are defined in IEC 62541-6.
samplingIntervalHint	Duration	A recommended rate of acquiring new values for change or deadband evaluation. A *Publisher* should use this value as hint for setting the internal sampling rate. The value 0 indicates that the *Server* should use the fastest practical rate. The value -1 indicates that the default sampling interval defined by the *PublishingInterval* of the *WriterGroup* is requested. Any negative number is interpreted as -1.
deadbandType	UInt32	A value that defines the *Deadband* type and behaviour. Value – Description None_0 – No *Deadband* 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 *DeadbandType*. The meaning of the value depends on *DeadbandType*.
indexRange	NumericRange	This parameter is used to identify a single element of an array, or a single range of indexes for arrays. The *NumericRange* type and the logic for *IndexRange* are defined in IEC 62541-4.
substituteValue	BaseDataType	The value that is included in the *DataSet* if the *StatusCode* of the *DataValue* is Bad. In this case the *StatusCode* is set to Uncertain_SubstituteValue. This Value shall match the *DataType* of the *PublishedVariable* since *DataSetWriters* may depend on a valid *Value* with the right *DataType* that matches the *ConfigurationVersion*. If the *SubstituteValue* is Null, the *StatusCode* of the *DataValue* is processed. The handling of the *SubstituteValue* is defined in 6.2.10.
metaDataProperties	QualifiedName [ ]	This parameter specifies an array of *Properties* to be included in the *FieldMetaData* created for this *Variable*. It shall be used to populate the *properties* element of the resulting field in the *DataSetMetaData*.

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.

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

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 typeId 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.

AVA Documents Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

953 **Table 14 – DataSetFieldContentMask Values**

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

954

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

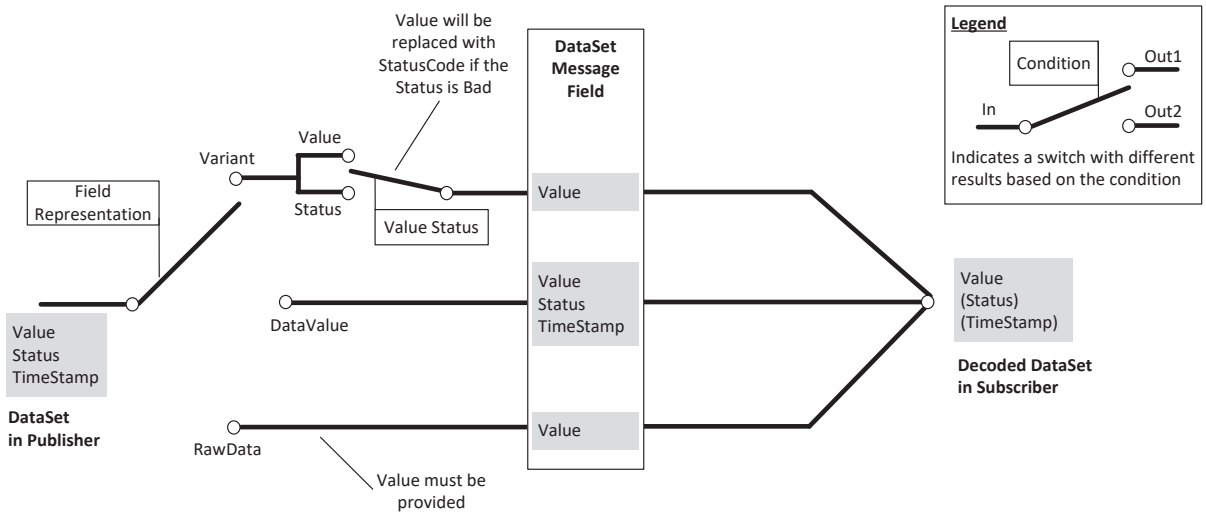
956 **Table 15 – DataSetFieldContentMask Definition**

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

957

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

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09



964

965 **Figure 21 – PubSub Information Flow dependency to field representation**

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

970 **Table 16 – DataSetMessage field representation options**

DataSet Publisher		Field	DataSetMessage		DataSet Subscriber	
Value	**Status**[4]		**Value**	**Status**[4]	**Value**	**Status**[4]
Value 1	Good_*	Variant	Value 1	N/A [1]	Value 1	N/A [1]
Value 1	Uncertain_*		Value 1		Value 1	
Null	Bad_*		Bad_* [1]		Null	Bad_*
Value 1	Good_*	DataValue	Value 1	Good_*	Value 1	Good_*
Value 1	Uncertain_*		Value 1	Uncertain_*	Value 1	Uncertain_*
Null	Bad_*		Null	Bad_*	Null	Bad_*
Value 1	Good_*	RawData	Value 1	N/A	Value 1	N/A
Value 1	Uncertain_*		Value 1[2]		Value 1	
Null	Bad_*		DefaultValue[3]		DefaultValue	

Note 1: A bad status is transferred instead of a value. An uncertain status is not transferred for a field. If the status field is included in the *DataSetMessage* header, the status is set to uncertain if one of the fields has an uncertain status.
Note 2: If the worst status for one or more fields is uncertain, the *DataSetMessage* status shall be set to *Uncertain*.
Note 3: If the worst status for one or more fields is bad, the *DataSetMessage* status shall be set to *Bad*.
Note 4: If no specific *StatusCode* is used, the grouping into severity *Good*, *Uncertain* or *Bad* is used. In this case, the resulting *Status* matches the input *Status*.

971

972 6.2.3.3 KeyFrameCount

973 The *KeyFrameCount* with *DataType UInt32* is the multiplier of the *PublishingInterval* that
974 defines the maximum number of times the *PublishingInterval* expires before a key frame
975 message with values for all published *Variables* is sent. The delta frame *DataSetMessages*
976 contains just the changed values. If no changes exist, the delta frame *DataSetMessage* shall
977 not be sent. If the *KeyFrameCount* is set to 1, every message contains a key frame.

978 For *PublishedDataSets* like *PublishedDataItems* that provide cyclic updates of the *DataSet*,
979 the value shall be greater or equal to 1. For non-cyclic *PublishedDataSets,* like
980 *PublishedEvents,* that provide event based *DataSets*, the value shall be 0.

981 6.2.3.4 DataSetWriterProperties

982 The *DataSetWriterProperties* parameter is an array of *DataType KeyValuePair* that specifies
983 additional properties for the configured *DataSetWriter*. The *KeyValuePair DataType* is defined
984 in IEC 62541-5 and consists of a *QualifiedName* and a value of *BaseDataType*.

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

The mapping of the name and value to concrete functionality may be defined by transport protocol mappings, future versions of this specification or vendor specific extensions.

6.2.3.5 DataSetWriter Structure

6.2.3.5.1 DataSetWriterDataType

This *Structure DataType* is used to represent the *DataSetWriter* parameters. The *DataSetWriterDataType* is formally defined in Table 17.

Table 17 – DataSetWriterDataType Structure

Name	Type	Description
DataSetWriterDataType	Structure	
name	String	The name of the *DataSetWriter*.
enabled	Boolean	The enabled state of the *DataSetWriter*.
dataSetWriterId	UInt16	Defined in 6.2.3.1.
dataSetFieldContentMask	DataSetFieldContentMask	Defined in 6.2.3.2.
keyFrameCount	UInt32	Defined in 6.2.3.3.
dataSetName	String	The name of the corresponding *PublishedDataSet*.
dataSetWriterProperties	KeyValuePair[]	Defined in 6.2.3.4.
transportSettings	DataSetWriterTransportDataType	Transport mapping specific *DataSetWriter* parameters. The abstract base type is defined in 6.2.3.5.2. The concrete subtypes are defined in the sections for transport mapping specific parameters.
messageSettings	DataSetWriterMessageDataType	*DataSetMessage* mapping specific *DataSetWriter* parameters. The abstract base type is defined in 6.2.3.5.3. The concrete subtypes are defined in the sections for message mapping specific parameters.

6.2.3.5.2 DataSetWriterTransportDataType

This *Structure DataType* is an abstract base type for transport mapping specific *DataSetWriter* parameters. The abstract *DataType* does not define fields.

The *DataSetWriterTransportDataType Structure* representation in the *AddressSpace* is defined in Table 18.

Table 18 – DataSetWriterTransportDataType Definition

Attributes	Value			
BrowseName	DataSetWriterTransportDataType			
IsAbstract	True			
References	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Subtype of Structure defined in IEC 62541-5.				
HasSubtype	DataType	BrokerDataSetWriterTransportDataType	FALSE	Defined in 6.4.2.3.7.

6.2.3.5.3 DataSetWriterMessageDataType

This *Structure DataType* is an abstract base type for message mapping specific *DataSetWriter* parameters. The abstract *DataType* does not define fields.

The *DataSetWriterMessageDataType Structure* representation in the *AddressSpace* is defined in Table 19.

Table 19 – DataSetWriterMessageDataType Structure

Attributes	Value			
BrowseName	DataSetWriterMessageDataType			
IsAbstract	True			
References	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Subtype of Structure defined in IEC 62541-5.				
HasSubtype	DataType	UadpDataSetWriterMessageDataType	FALSE	Defined in 6.3.1.2.6.
HasSubtype	DataType	JsonDataSetWriterMessageDataType	FALSE	Defined in 6.3.2.2.2.

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

6.2.4 Shared PubSubGroup Parameters

6.2.4.1 General

The parameters are shared between *WriterGroup* and *ReaderGroup*.

The parameters are related to *PubSub NetworkMessage* security. See 5.4.3 for an introduction of PubSub security and 8 for the definition of the PubSub Security Key Service.

6.2.4.2 SecurityMode

The *SecurityMode* indicates the level of security applied to the *NetworkMessages* published by a *WriterGroup* or received by a *ReaderGroup*. The *MessageSecurityMode DataType* is defined in IEC 62541-4.

6.2.4.3 SecurityGroupId

The *SecurityGroupId* with *DataType String* is the identifier for a *SecurityGroup* in the *Security Key Server*. It is unique within a SKS.

The parameter is null if the *SecurityMode* is NONE_1.

If the *SecurityMode* is not NONE_1 the *SecurityGroupId* identifies the *SecurityGroup*. The *SecurityGroup defines the SecurityPolicy* and the security keys used for the *NetworkMessage* security. The *PubSubGroup* defines the *SecurityMode* for the *NetworkMessages* sent by the group.

6.2.4.4 SecurityKeyServices

SecurityKeyServices is an array of the *DataType EndpointDescription* and *defines* one or more *Security Key Servers* (SKS) that manage the security keys for the *SecurityGroup* assigned to the *PubSubGroup*. The *EndpointDescription DataType* is defined in IEC 62541-4.

The parameter is null if the *SecurityMode* is NONE_1.

Each element in the array is an *Endpoint* for an SKS that can supply the security keys for the *SecurityGroupId*. Multiple *Endpoints* exist because an SKS may support multiple transport profiles and/or may have multiple redundant instances. The *UserTokenPolicies* in each *Endpoint* specify what user credentials are required. IEC 62541-4 describes *UserTokenPolicies* in more detail.

6.2.4.5 MaxNetworkMessageSize

The *MaxNetworkMessageSize* with *DataType UInt32* indicates the maximum size in bytes for *NetworkMessages* created by the *WriterGroup*. It refers to the size of the complete *NetworkMessage* including padding and signature without any additional headers added by the transport protocol mapping. If the size of a *NetworkMessage* exceeds the *MaxNetworkMessageSize,* the behaviour depends on the message mapping.

The transport protocol mappings defined in 7.3 may define restrictions for the maximum value of this parameter.

Note 1: The value for the *MaxNetworkMessageSize* should be configured in a way that ensures that *NetworkMessages* together with additional headers added by the transport protocol are still smaller or equal than the transport protocol MTU.

6.2.4.6 GroupProperties

The *GroupProperties* parameter is an array of *DataType KeyValuePair* that specifies additional properties for the configured group. The *KeyValuePair DataType* is defined in IEC 62541-5 and consists of a *QualifiedName* and a value of *BaseDataType*.

The mapping of the name and value to concrete functionality may be defined by transport protocol mappings, future versions of this specification or vendor specific extensions.

6.2.4.7 PubSubGroup Structure

This *Structure DataType* is an abstract base type for *PubSubGroups*. The *PubSubGroupDataType* is formally defined in Table 20.

AVA Documents Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

Table 20 – PubSubGroupDataType Structure

Name	Type	Description
PubSubGroupDataType	Structure	
name	String	The name of the *PubSubGroup*.
enabled	Boolean	The enabled state of the *PubSubGroup*.
securityMode	MessageSecurityMode	Defined in 6.2.4.2.
securityGroupId	String	Defined in 6.2.4.3.
securityKeyServices	EndpointDescription[]	Defined in 6.2.4.4.
maxNetworkMessageSize	UInt32	Defined in 6.2.4.5.
groupProperties	KeyValuePair[]	Defined in 6.2.4.6.

The *PubSubGroupDataType Structure* representation in the *AddressSpace* is defined in Table 21.

Table 21 – PubSubGroupDataType Definition

Attributes	Value			
BrowseName	PubSubGroupDataType			
IsAbstract	True			
References	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Subtype of Structure defined in IEC 62541-5.				
HasSubtype	DataType	WriterGroupDataType	FALSE	Defined in 6.2.5.6.1.
HasSubtype	DataType	ReaderGroupDataType	FALSE	Defined in 6.2.7.2.1.

6.2.5 WriterGroup Parameters

6.2.5.1 WriterGroupId

The *WriterGroupId* with *DataType UInt16* is an identifier for the *WriterGroup* and shall be unique across all *WriterGroups* for a *PublisherId.* All values, except for 0, are valid. The value 0 is defined as null value.

6.2.5.2 PublishingInterval

The *PublishingInterval* with the *DataType Duration* defines the interval in milliseconds for publishing *NetworkMessages* and the embedded *DataSetMessages* created by the related *DataSetWriters*.

In the case of *Event* based *DataSets*, this may result in zero to many *DataSetMessages* produced for one *PublishedDataSet* in a *PublishingInterval*. All *Events* that occur between two *PublishingIntervals* shall be buffered until the next *NetworkMessage* is sent. If the number of Events exceeds the buffer capability of the DataSetWriter, an *Event* of type *EventQueueOverflowEventType* is inserted into the buffer.

The *Duration DataType* is a subtype of *Double* and allows configuration of intervals smaller than a millisecond.

6.2.5.3 KeepAliveTime

The *KeepAliveTime* with *DataType Duration* defines the time in milliseconds until the *Publisher* sends a keep alive *DataSetMessage* in the case where no *DataSetMessage* was sent in this period by a *DataSetWriter*. The minimum value shall equal the *PublishingInterval*.

6.2.5.4 Priority

The *Priority* with *DataType Byte* defines the relative priority of the *WriterGroup* to all other *WriterGroups* across all *PubSubConnections* of the *Publisher*.

If more than one *WriterGroup* needs to be processed, the priority number defines the order of processing. The highest priority is processed first.

The lowest priority is zero and the highest is 255.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09

6.2.5.5 LocaleIds

The *LocaleIds* with DataType *LocaleId* defines a list of locale ids in priority order for localized strings for all *DataSetWriters* in the *WriterGroup*. The first *LocaleId* in the list has the highest priority.

If the *Publisher* sends a localized *String*, the *Publisher* shall send the translation with the highest priority that it can. If it does not have a translation for any of the locales identified in this list, then it shall send the *String* value that it has and include the *LocaleId* with the *String*. If no locale id is configured, the *Publisher* shall use any that it has. See IEC TR 62541-2, *OPC Unified Architecture – Part 2: Security Model*

IEC 62541-3 for more detail on *LocaleId*.

6.2.5.6 WriterGroup Structures

6.2.5.6.1 WriterGroupDataType

This *Structure DataType* is used to represent the configuration parameters for *WriterGroups*. It is a subtype of *PubSubGroupDataType* defined in 6.2.4.7.

The *WriterGroupDataType* is formally defined in Table 22.

Table 22 – WriterGroupDataType Structure

Name	Type	Description
WriterGroupDataType	Structure	
writerGroupId	UInt16	Defined in 6.2.5.1.
publishingInterval	Duration	Defined in 6.2.5.2.
keepAliveTime	Duration	Defined in 6.2.5.3.
priority	Byte	Defined in 6.2.5.4.
localeIds	String[]	Defined in 6.2.5.5.
transportSettings	WriterGroupTransportDataType	Transport mapping specific *WriterGroup* parameters. The abstract base type is defined in 6.2.5.6.2. The concrete subtypes are defined in the sections for transport mapping specific parameters.
messageSettings	WriterGroupMessageDataType	*NetworkMessage* mapping specific *WriterGroup* parameters. The abstract base type is defined in 6.2.5.6.3. The concrete subtypes are defined in the sections for message mapping specific parameters.
dataSetWriters	DataSetWriterDataType[]	The DataSetWriters contained in the *WriterGroup*. The *DataSetWriter* parameters are defined in 6.2.3.

The *WriterGroupDataType Structure* representation in the *AddressSpace* is defined in Table 23.

Table 23 – WriterGroupDataType Definition

Attributes	Value		
BrowseName	WriterGroupDataType		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**IsAbstract**
Subtype of PubSubGroupDataType defined in 6.2.4.7.			

6.2.5.6.2 WriterGroupTransportDataType

This *Structure DataType* is an abstract base type for transport mapping specific *WriterGroup* parameters. The abstract *DataType* does not define fields.

The *WriterGroupTransportDataType Structure* representation in the *AddressSpace* is defined in Table 24.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-14:2019-09