IEC 62541-14

Edition 1.0 2020-07

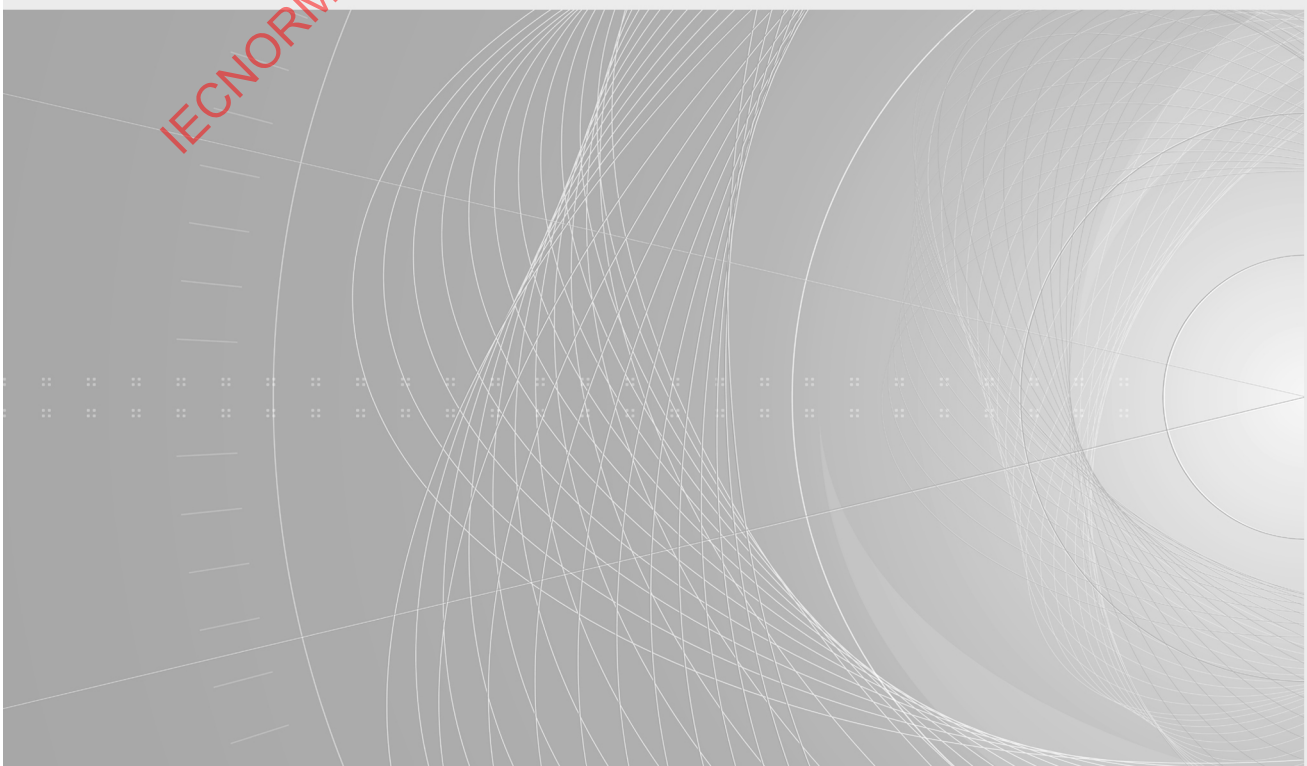
INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 14: PubSub

Architecture unifiée OPC – Partie 14: PubSub



IEC 62541-14:2020-07(en-fr)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
info@iec.ch
www.iec.ch

About the IEC
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigendum or an amendment might have been published.

IEC publications search - webstore.iec.ch/advsearchform
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and once a month by email.

IEC Customer Service Centre - webstore.iec.ch/csc
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: sales@iec.ch.

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 000 terminological entries in English and French, with equivalent terms in 16 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
67 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

A propos de l'IEC
La Commission Electrotechnique Internationale (IEC) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications IEC
Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Recherche de publications IEC - webstore.iec.ch/advsearchform
La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,…). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished
Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et une fois par mois par email.

Service Clients - webstore.iec.ch/csc
Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: sales@iec.ch.

Electropedia - www.electropedia.org
Le premier dictionnaire d'électrotechnologie en ligne au monde, avec plus de 22 000 articles terminologiques en anglais et en français, ainsi que les termes équivalents dans 16 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary
67 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

IEC 62541-14

Edition 1.0 2020-07

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 14: PubSub

Architecture unifiée OPC – Partie 14: PubSub

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

ICS 25.040.40; 35.100.05

ISBN 978-2-8322-8577-0

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

OPC UNIFIED ARCHITECTURE –

Part 14: PubSub

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

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

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

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

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

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

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

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

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

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

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

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

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

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

OPC UNIFIED ARCHITECTURE –

Part 14: PubSub

1 Scope

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

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

This document consists of

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

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

2 Normative references

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

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

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

IEC 62541-3, *OPC Unified Architecture – Part 3: Address Space Model*

IEC 62541-4, *OPC Unified Architecture – Part 4: Services*

IEC 62541-5, *OPC Unified Architecture – Part 5: Information Model*

IEC 62541-6, *OPC Unified Architecture – Part 6: Mappings*

IEC 62541-7, *OPC Unified Architecture – Part 7: Profiles*

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

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

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

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

IETF RFC 7159, *The JavaScript Object Notation (JSON) Data Interchange Format*
http://www.ietf.org/rfc/rfc7159.txt

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

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

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

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

3.1.1
DataSetClass
template declaring the content of a *DataSet*

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

3.1.2
DataSetMetaData
data describing the content and semantic of a *DataSet*

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

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

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

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

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

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

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

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

3.1.7
SubscribedDataSet
configuration for dispatching of received *DataSets*

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

3.2 Abbreviated terms

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

4 Overview

4.1 Fields of application

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

The following are some example uses for *PubSub*:

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

4.2 Abstraction layers

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

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

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

4.3 Decoupling by use of middleware

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

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

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

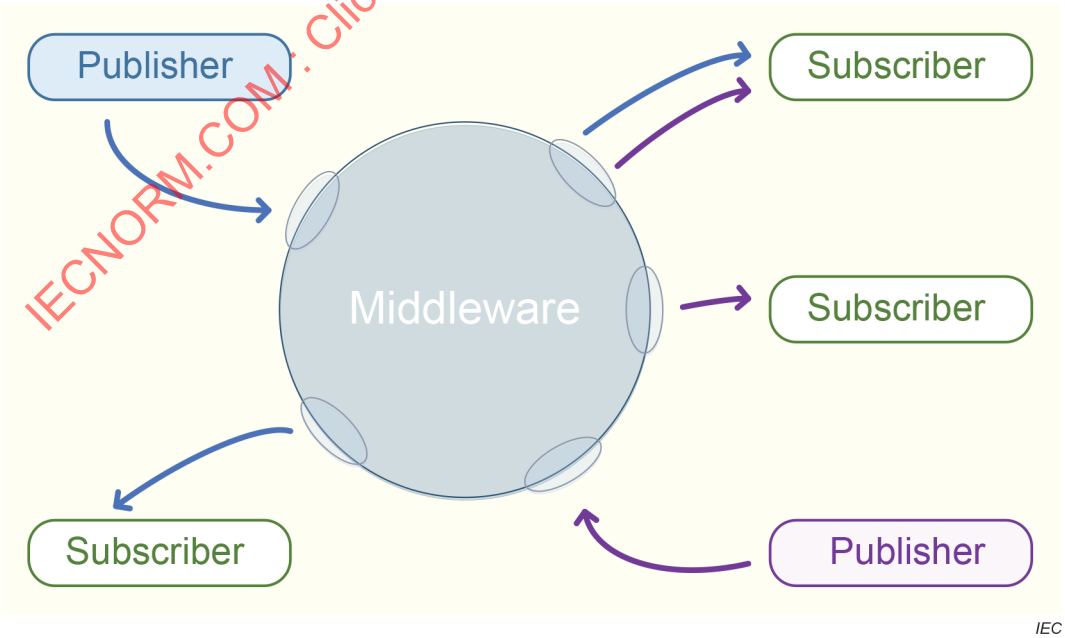


Figure 1 – Publish Subscribe Model overview

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

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

4.4 Synergy of models

PubSub and *Client Server* are both based on the OPC UA *Information Model*. *PubSub* therefore can easily be integrated into OPC UA *Servers* and OPC UA *Clients*. Quite typically, a *Publisher* will be an OPC UA *Server* (the owner of information) and a *Subscriber* is often an OPC UA *Client*. Above all, the *PubSub Information Model* for configuration (see 6.2.2) promotes the configuration of *Publishers* and *Subscribers* using the OPC UA *Client Server* model.

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

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

5 PubSub Concepts

5.1 General

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

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

The abstract communication parameters are described in Clause 6.

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

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

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

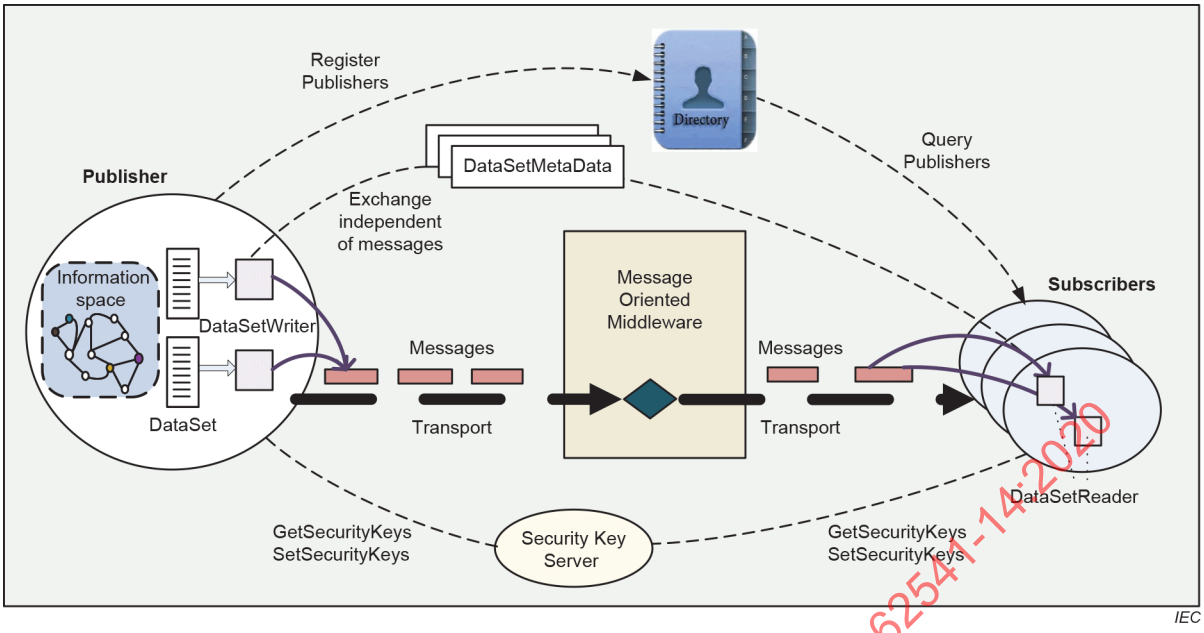


Figure 2 – Publisher and Subscriber entities

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

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

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

NOTE The PubSub directory is an optional entity that allows *Publishers* to advertise their *PublishedDataSets* and their communication parameters. This directory functionality is planned for a future edition of IEC 62541-14.

5.2 DataSet

5.2.1 General

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

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

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

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

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

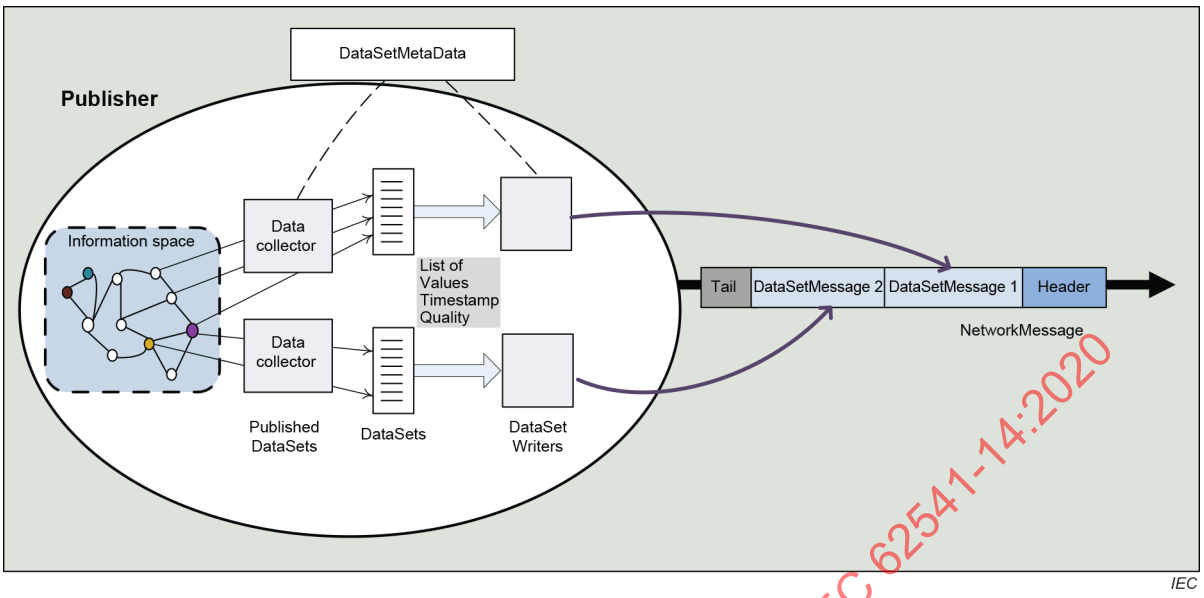


Figure 3 – DataSet in the process of publishing

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

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

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

5.2.2 DataSetClass

DataSets can be individual for a *Publisher* or they can be derived from a *DataSetClass*. Such a *DataSetClass* acts as template declaring the content of a *DataSet*. The *DataSetClass* is identified by a globally unique id – the *DataSetClassId* (see 6.2.2.2).

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

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

5.2.3 DataSetMetaData

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

The *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 9.14.2.1) provided by the *Publisher* or from a configuration server.
- The *Subscriber* supports the OPC UA configuration *Methods* defined in the *PubSub* configuration model.
- The *Subscriber* receives the *DataSetMetaData* as *NetworkMessage* from the *Publisher*. This may require an option for the *Subscriber* to request this *NetworkMessage* from the *Publisher*.
- The *Subscriber* is configured with product-specific configuration means.

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

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

5.3 Messages

5.3.1 General

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

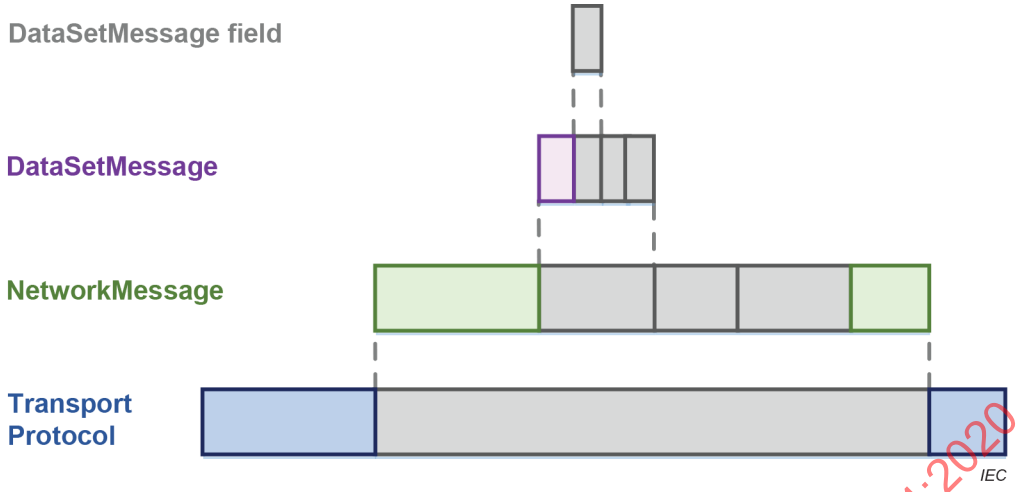


Figure 4 – OPC UA PubSub message layers

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

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

5.3.2 DataSetMessage field

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

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

A *DataSet* field can be represented as a *DataValue*, as a *Variant* or as a *RawData* in the *DataSetMessage* field. The representation depends on the *DataSetFieldContentMask* defined in 6.2.3.2.

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

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

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

5.3.3 DataSetMessage

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

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

Additional header information includes:

DataSetWriterId Identifies the *DataSetWriter* and indirectly the *PublishedDataSet*.

Sequence number	A number that is incremented for each *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 to 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 base concepts for OPC UA security are defined in IEC TR 62541-2. The level of security can be:

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

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

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

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

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

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

5.3.6 Transport security

The transport security is specific to the transport protocol mapping.

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

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

5.3.7 SecurityGroup

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

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

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

5.4 Entities

5.4.1 Publisher

5.4.1.1 General

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

Commonly, a *Publisher* is also an OPC UA *Server*. For the abstract *PubSub* concepts, however, it is an arbitrary entity and should not be assumed to be an individual or even a specific network node (an IP or a MAC address) or a specific application. Figure 5 illustrates a *Publisher* with data collection, encoding and message sending.

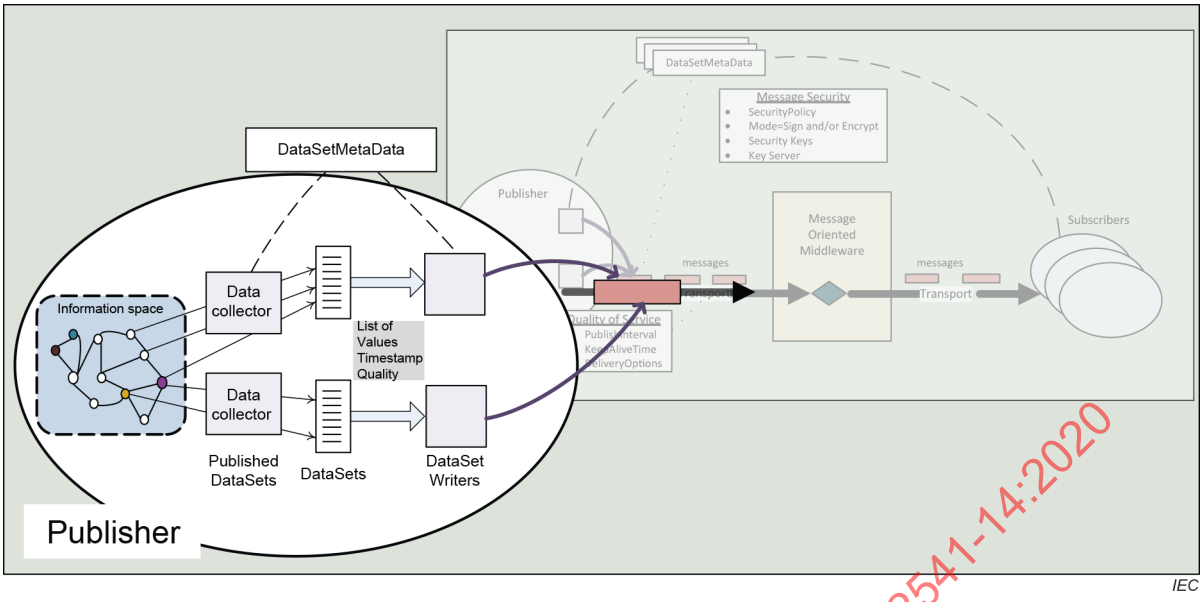


Figure 5 – Publisher details

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

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

5.4.1.2 Message sending

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

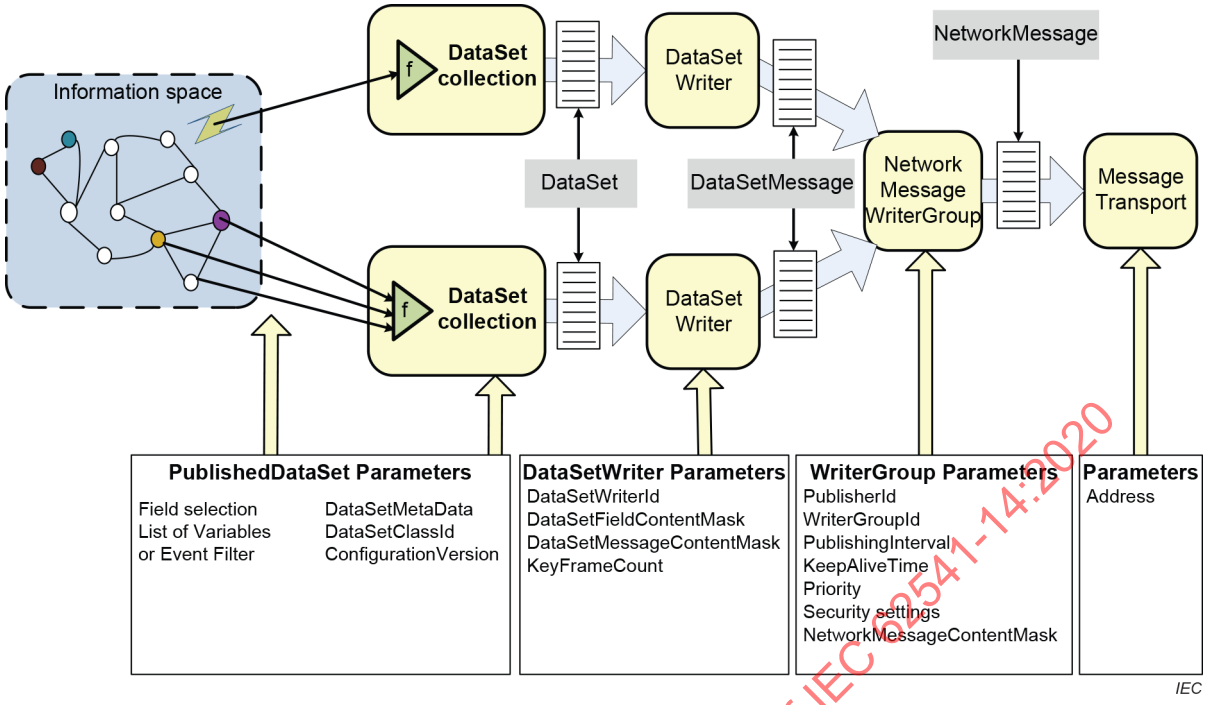


Figure 6 – Publisher message sending sequence

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

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

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

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

The resulting *DataSetMessage* is passed on to the next step together with the *DataSetWriterId* (see 6.2.3.1), the *DataSetClassId* (see 6.2.2.2), the *ConfigurationVersion* of the *DataSetMetaData* (see 6.2.2.1.5), and a list of values that match the configured propagated fields.

Next is the creation of the *NetworkMessage*. It uses the data provided from the previous step together with the *PublisherId* (see 6.2.6.1) defined on the *WriterGroup*. The structure of this message is protocol specific. If the *SecurityMode* (see 6.2.4.2) requires message security, the *SecurityGroupId* (see 6.2.4.3) is used to fetch the *SecurityPolicy* and the security keys from the SKS (see 5.4.3). This information is used to encrypt and/or sign the *NetworkMessage* as required by the *SecurityMode*.

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

5.4.2 Subscriber

5.4.2.1 General

*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*.

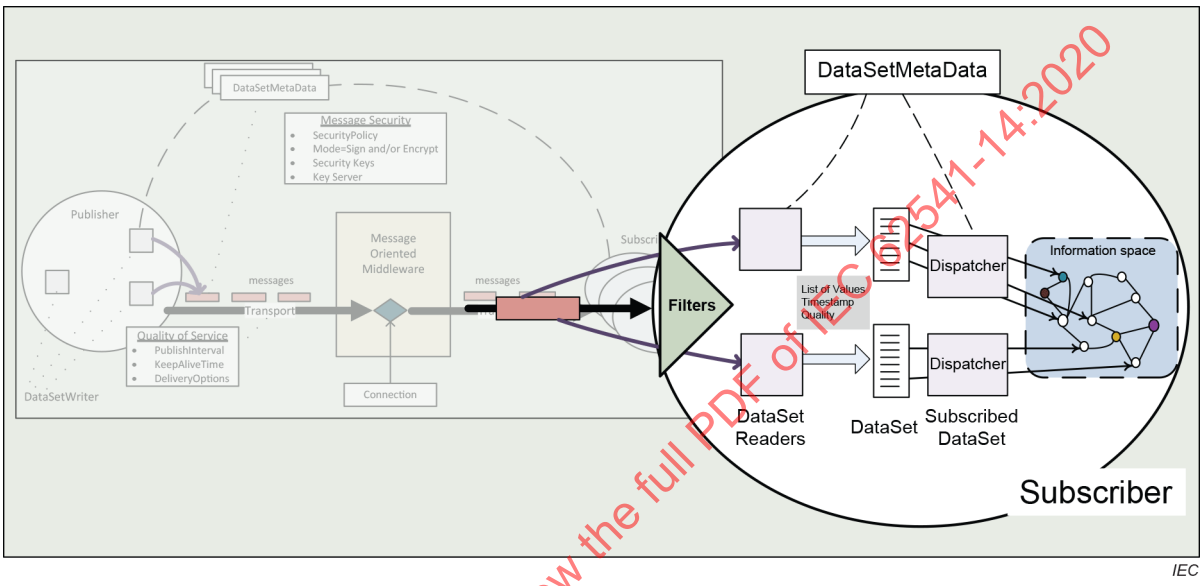


Figure 7 – Subscriber details

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

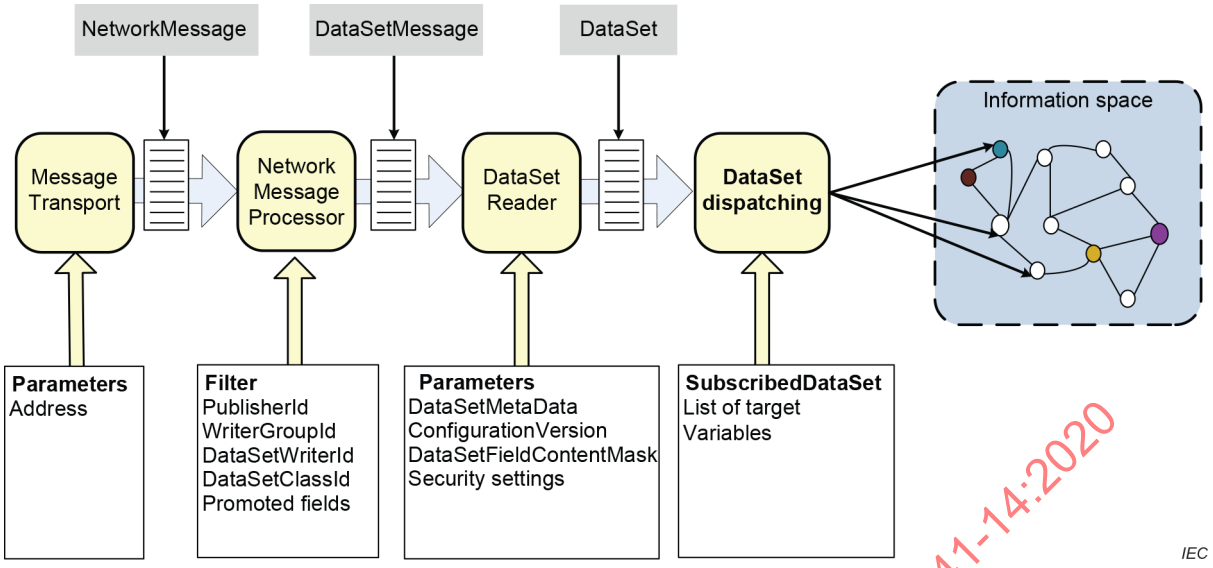


Figure 8 – Subscriber message reception sequence

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

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

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

Any further processing is application-specific. For example, an additional dispatching step may map the received values to *Nodes* in the *Subscribers* OPC UA *AddressSpace*. The configuration for such a dispatching is called *SubscribedDataSet*.

5.4.3 Security Key Service

5.4.3.1 General

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

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

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

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

5.4.3.2 SecurityGroup Management

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

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

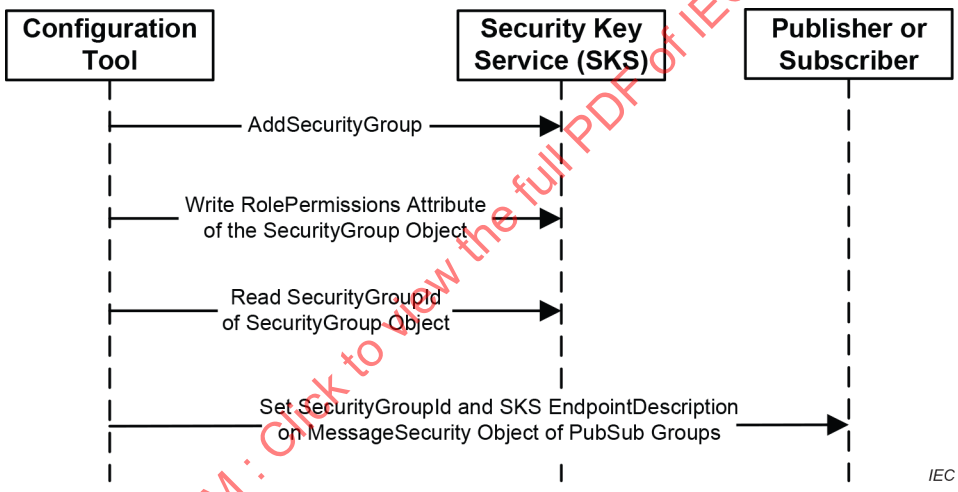


Figure 9 – SecurityGroup management sequence

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

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

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

5.4.3.3 Key acquisition handshakes

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

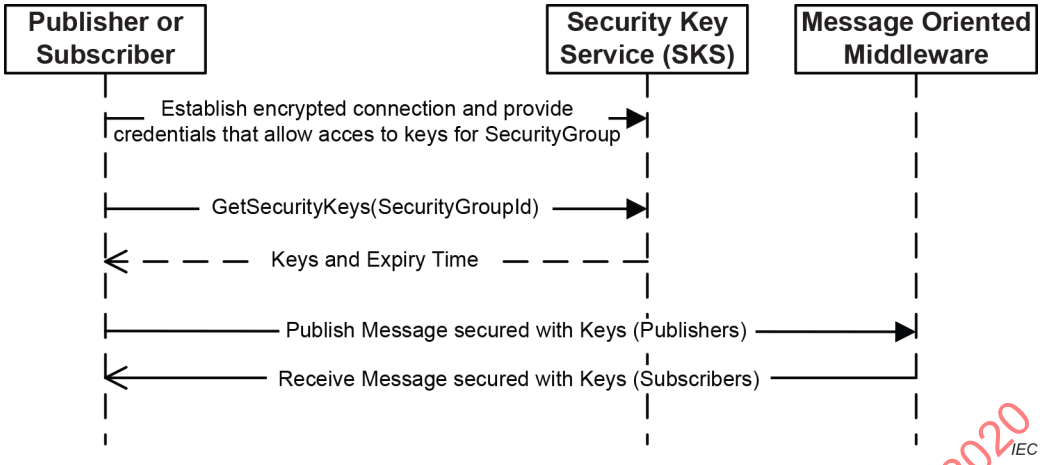


Figure 10 – Handshake used to pull keys from SKS

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.

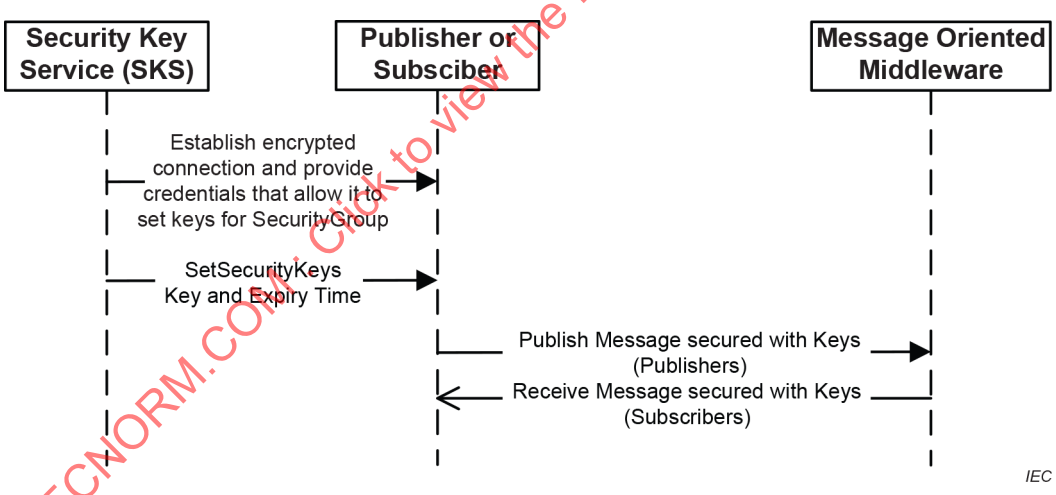


Figure 11 – Handshake used to push keys to Publishers and Subscribers

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.

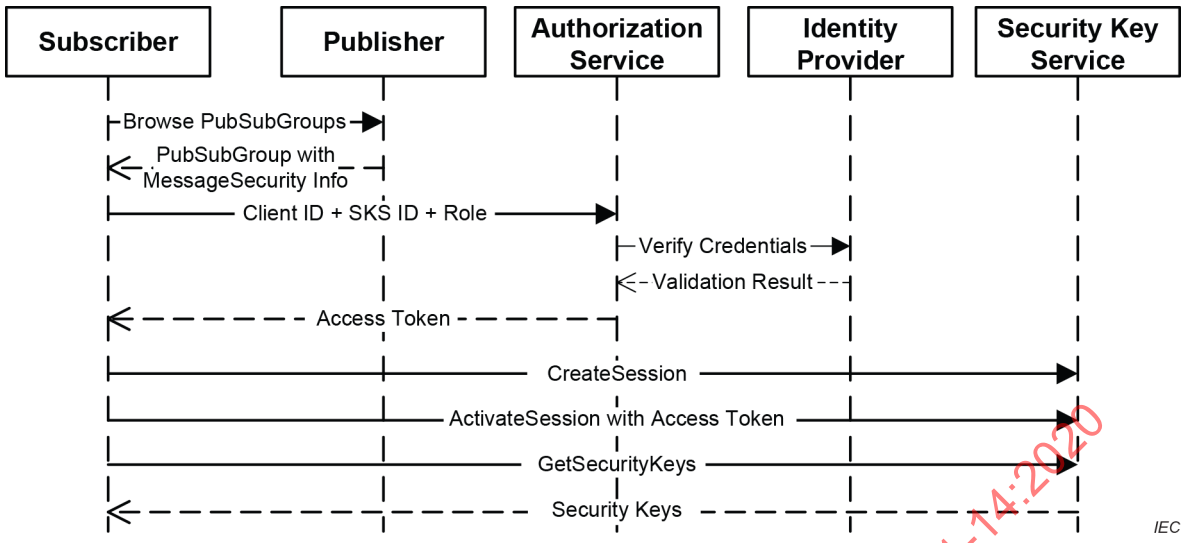


Figure 12 – Handshake with a Security Key Service

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

5.4.4 Message Oriented Middleware

5.4.4.1 General

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

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

5.4.4.2 Broker-less Middleware

5.4.4.2.1 General

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

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

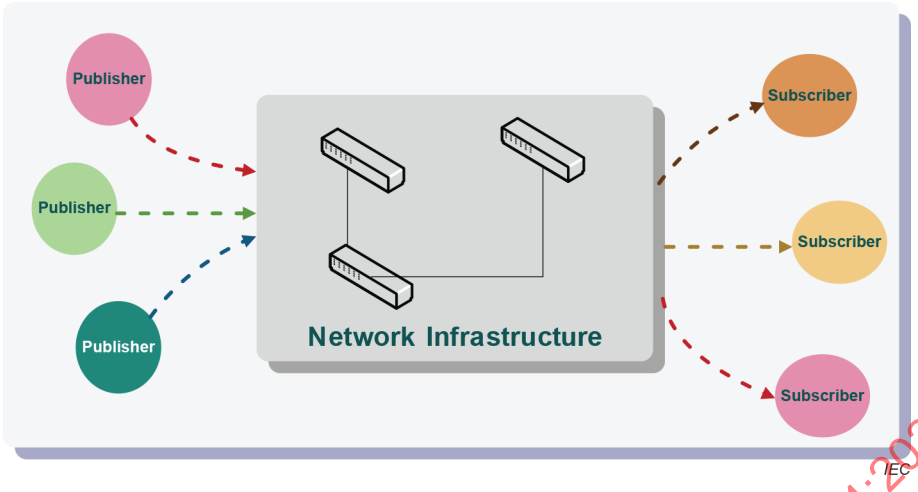


Figure 13 – PubSub using network infrastructure

Advantages of this model include:

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

5.4.4.2.2 Broker-less model with OPC UA UDP

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

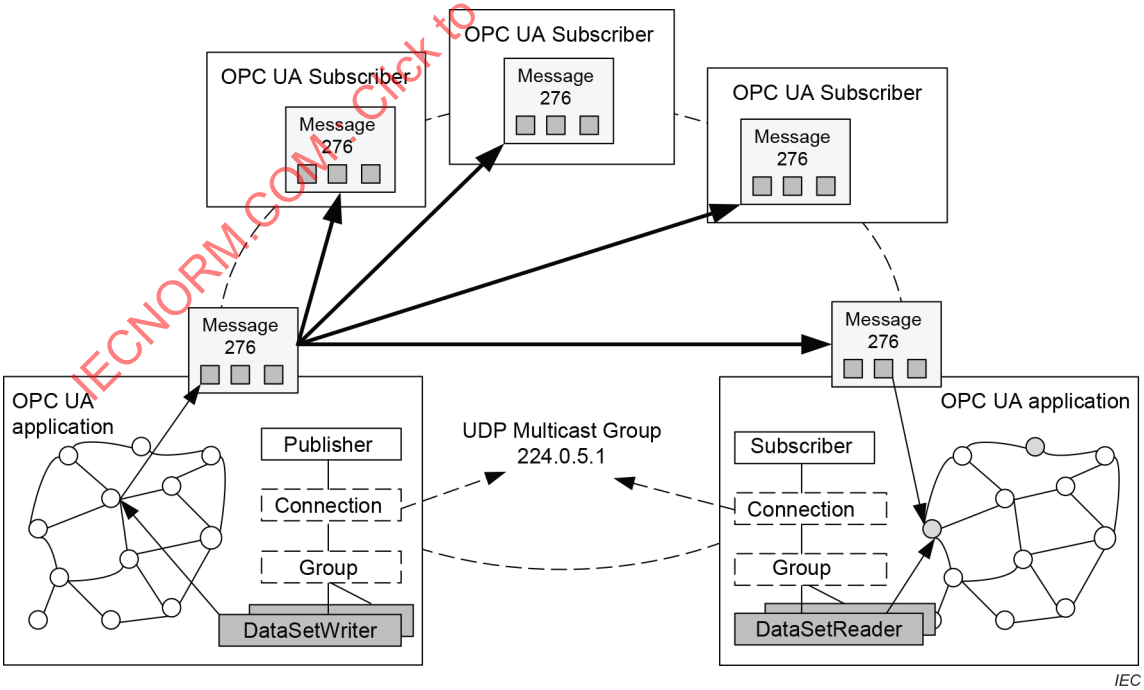


Figure 14 – UDP Multicast overview

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

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

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

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

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

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

5.4.4.3 Broker-based Middleware

5.4.4.3.1 General

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

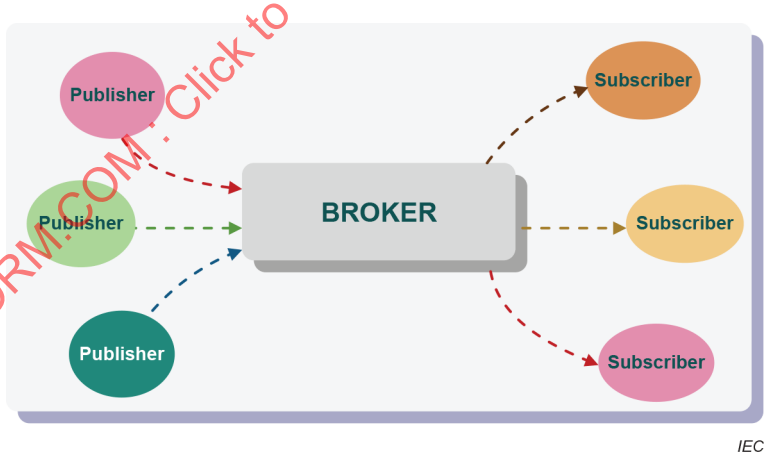


Figure 15 – PubSub using broker

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

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

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

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

5.4.4.3.2 Broker-based model

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

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

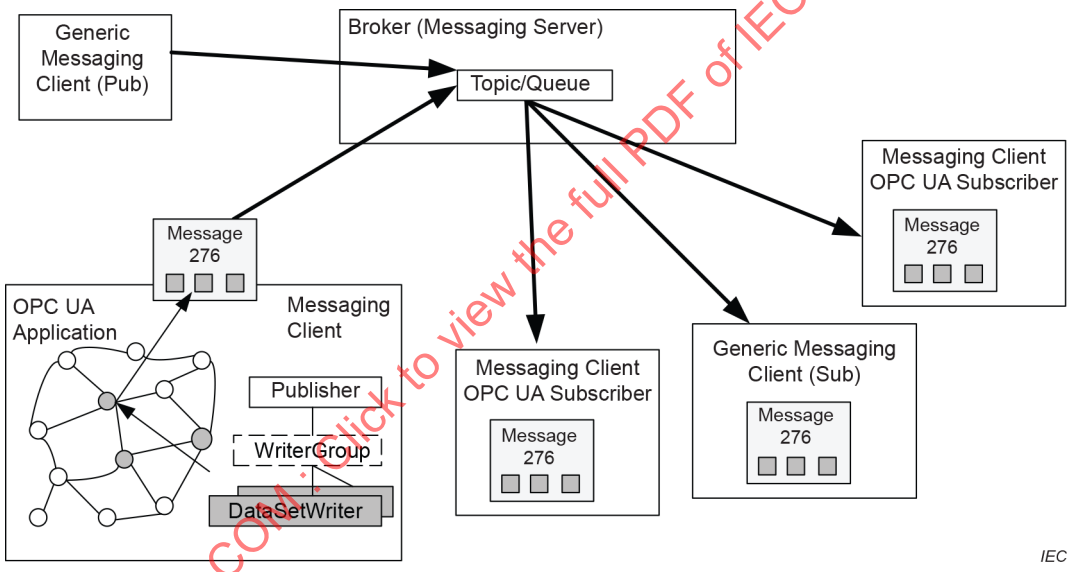


Figure 16 – Broker overview

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

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

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

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

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

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

6 PubSub communication parameters

6.1 Overview

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

The common parameters are defined in 6.2. The parameters for the different message mappings are defined in 6.3. The parameters for the different transport protocol mappings are defined in 6.4. Common types that are not only used with PubSub are defined in Annex A.

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

Configuration of these parameters can be performed through the OPC UA *Information Model* for *PubSub* configuration defined in Clause 9 or through vendor-specific mechanisms. The parameter groupings in Clause 6 define the parameters and also define *Structures* used to represent the parameters of the groupings. These *Structures* are used in the *PubSub* configuration model described in Clause 9 but they can also be used for offline configuration or vendor-specific configuration mechanisms.

Figure 17 depicts the different components and their relation to each other. The *WriterGroup*, *DataSetWriter* and *PublishedDataSet* components define the data acquisition for the *DataSets*, the message generation and the sending on the *Publisher* side. These parameters need to be known on the *Subscriber* side to configure *DataSetReaders* and to filter and process *DataSetMessages*.

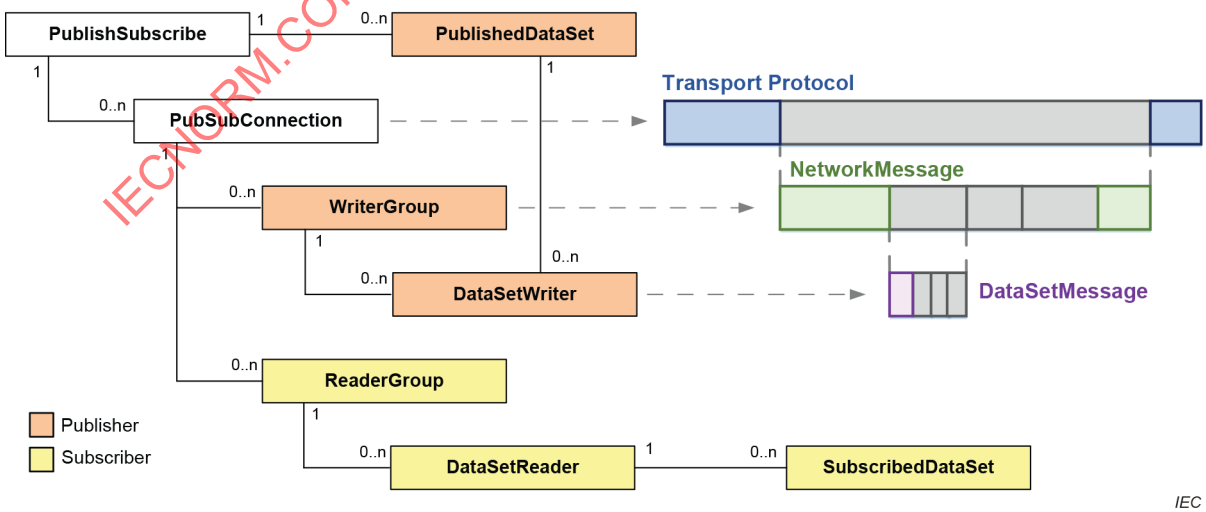


Figure 17 – PubSub component overview

Figure 17 shows the following components:

- *PublishedDataSet* contains the *DataSetMetaData* describing the content of the *DataSets* produced by the *PublishedDataSet* and the corresponding data acquisition parameters.
- *DataSetWriter* parameters are necessary for creating *DataSetMessages*. Each *DataSetWriter* is bound to a single *PublishedDataSet*. A *PublishedDataSet* can have multiple *DataSetWriters*.
- *WriterGroup* parameters are necessary for creating a *NetworkMessage*. Each writer group can have one or more *DataSetWriters*. Some of these parameters are used for creating the *DataSetMessages*. They are grouped here since they are the same for all *DataSetMessages* in a single *NetworkMessage*.
- *PubSubConnection* parameters represent settings needed for the transport protocol. One connection can have a number of writer groups and reader groups.
- *ReaderGroup* is used to group a list of *DataSetReaders* and contains a few shared settings for them. It is not symmetric to a *WriterGroup* and it is not related to a particular *NetworkMessage*. The *NetworkMessage* related filter settings are on the *DataSetReaders*.
- *DataSetReader* parameters represent settings for filtering of received *NetworkMessages* and *DataSetMessages* as well as settings for decoding of the *DataSetMessages* of interest.
- *SubscribedDataSet* parameters define the processing of the decoded *DataSet* in the *Subscriber* for one *DataSetReader*.
- *PublishSubscribe* is the overall management of the *PubSub* groupings. It contains a list of *PublishedDataSets* and a list of *PubSubConnections*.

The different PubSub mapping specific parameter groupings are shown in Figure 18.

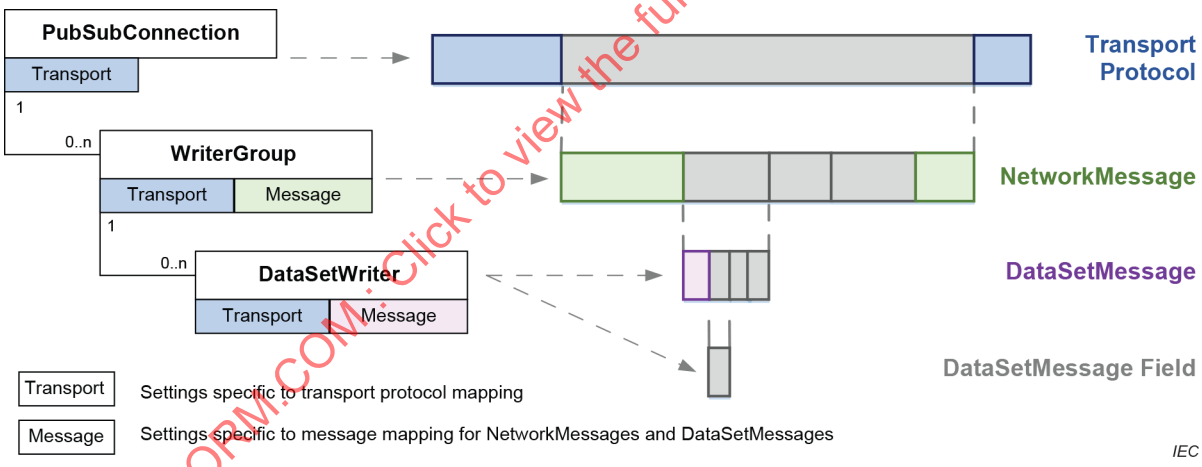


Figure 18 – PubSub mapping specific parameters overview

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.

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.

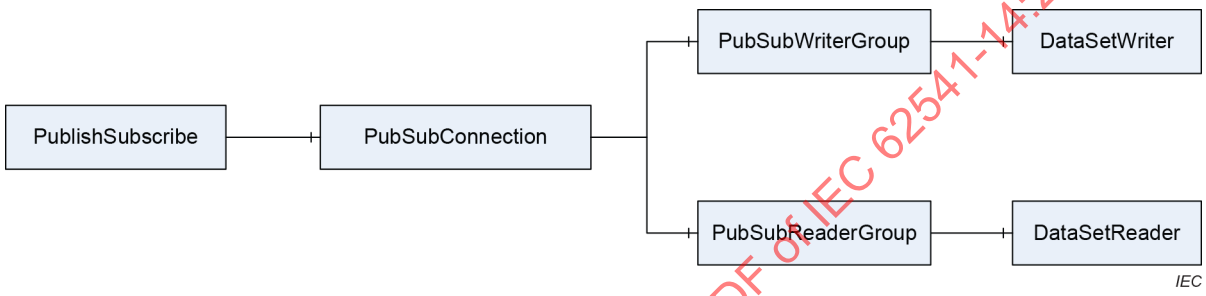


Figure 19 – PubSub component state dependencies

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

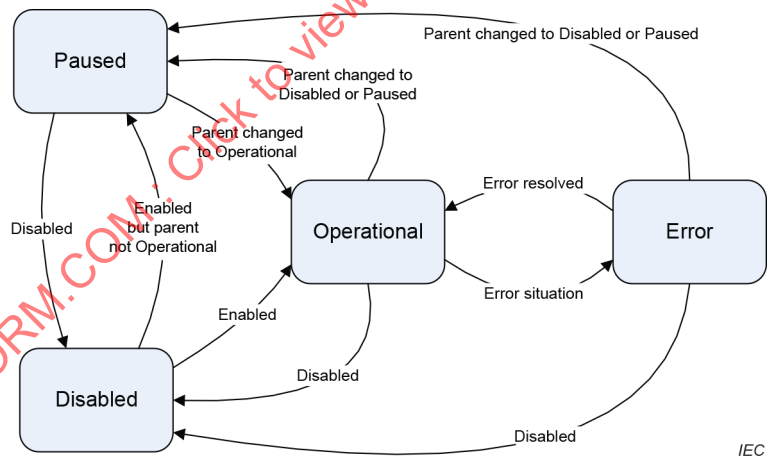


Figure 20 – PubSubState state machine

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.

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 need to 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 *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*.

Name	Type	Description
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 2 147 483 647 (max Int32). It is the total number of elements in all dimensions based on the UA Binary encoding rules for arrays.
maxStringLength	UInt32	If the *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:

Table 7 – DataSetFieldFlags definition

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

6.2.2.1.5 ConfigurationVersionDataType

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

Table 8 – ConfigurationVersionDataType structure

Name	Type	Description
ConfigurationVersionDataType	Structure	
majorVersion	VersionTime	The *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* shall 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 document.
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*.

6.2.2.2 DataSetClassId

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

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

6.2.2.3 ExtensionFields

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

6.2.2.4 PublishedDataSetDataType

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

Table 9 – PublishedDataSetDataType structure

Name	Type	Description
PublishedDataSetDataType	Structure	
name	String	Name of the *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.

6.2.2.5 PublishedDataSetSourceDataType

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

Table 10 – PublishedDataSetSourceDataType definition

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

6.2.2.6 Published Data Items

6.2.2.6.1 PublishedData

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

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

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

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

Table 11 – PublishedVariableDataType structure

Name	Type	Description
PublishedVariableDataType	Structure	
publishedVariable	NodeId	The *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 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.

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.

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.

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

Table 15 – DataSetFieldContentMask definition

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

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

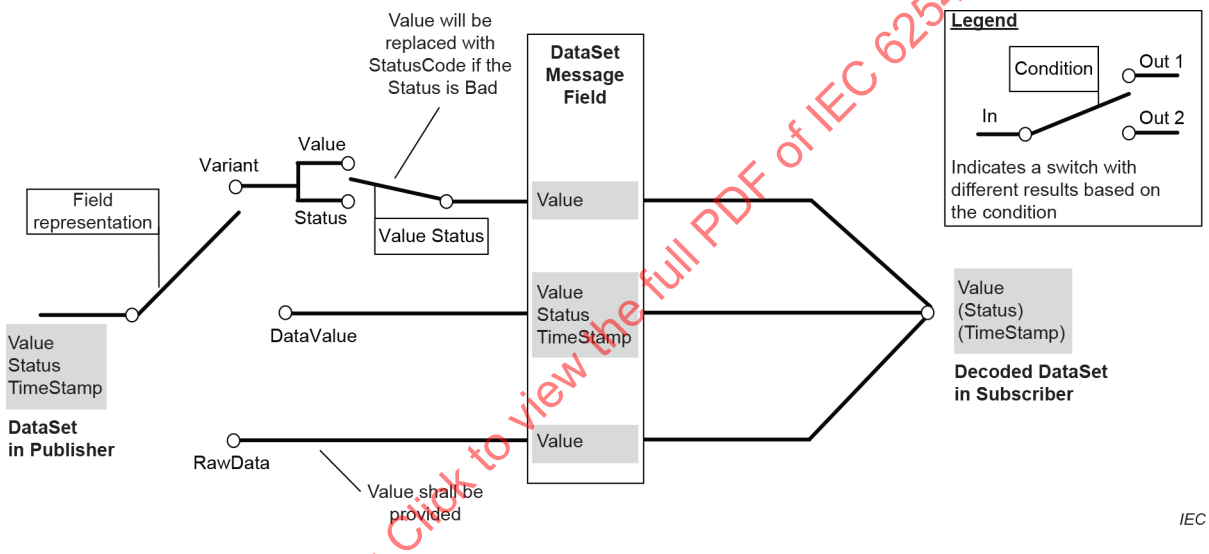


Figure 21 – PubSub Information Flow dependency to field representation

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

Table 16 – DataSetMessage field representation options

DataSet Publisher		Field	DataSetMessage		DataSet Subscriber	
Value	**Status**[d]		**Value**	**Status**[d]	**Value**	**Status**[d]
Value 1	Good_*	Variant	Value 1	N/A [a]	Value 1	N/A [a]
Value 1	Uncertain_*		Value 1		Value 1	
Null	Bad_*		Bad_* [a]		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 [b]		Value 1	
Null	Bad_*		DefaultValue [c]		DefaultValue	

[a] 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.

[b] If the worst status for one or more fields is uncertain, the *DataSetMessage* status shall be set to *Uncertain*.

[c] If the worst status for one or more fields is bad, the *DataSetMessage* status shall be set to *Bad*.

[d] 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*.

6.2.3.3 KeyFrameCount

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

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

6.2.3.4 DataSetWriterProperties

The *DataSetWriterProperties* parameter is an array of *DataType KeyValuePair* that specifies additional properties for the configured *DataSetWriter*. 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 editions of IEC 62541-14 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 subclauses 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 subclauses 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.

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 Clause 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 than or equal to 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 editions of IEC 62541-14 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.

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 6.2.5.7.1.
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.

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 62541-3 for more detail on *LocaleId*.

6.2.5.6 HeaderLayoutUri

The *HeaderLayoutUri*, with *DataType String*, defines the selection of a well defined configuration for a subset of the PubSub communication parameters. The affected subset is defined by the header layout.

A null *String* is defined as no layout selected.

If a layout is selected, all affected parameters shall be set to the values defined for the layout.

6.2.5.7 WriterGroup Structures

6.2.5.7.1 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	LocaleId[]	Defined in 6.2.5.5.
headerLayoutUri	String	Defined in 6.2.5.6.
transportSettings	WriterGroupTransportDataType	Transport mapping specific *WriterGroup* parameters. The abstract base type is defined in 6.2.5.7.2. The concrete subtypes are defined in the subclauses for transport mapping specific parameters. If no concrete subtype is defined for the transport mapping, the field shall be null.
messageSettings	WriterGroupMessageDataType	*NetworkMessage* mapping specific *WriterGroup* parameters. The abstract base type is defined in 6.2.5.7.3. The concrete subtypes are defined in the subclauses for message mapping specific parameters. If no concrete subtype is defined for the transport mapping, the field shall be null.
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.			

6.2.5.7.2 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.

Table 24 – WriterGroupTransportDataType definition

Attributes	Value			
BrowseName	WriterGroupTransportDataType			
IsAbstract	True			
References	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Subtype of Structure defined in IEC 62541-5.				
HasSubtype	DataType	DatagramWriterGroupTransportDataType	FALSE	Defined in 6.4.1.2.3.
HasSubtype	DataType	BrokerWriterGroupTransportDataType	FALSE	Defined in 6.4.2.2.6.

6.2.5.7.3 WriterGroupMessageDataType

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

The *WriterGroupMessageDataType Structure* representation in the *AddressSpace* is defined in Table 25.

Table 25 – WriterGroupMessageDataType structure

Attributes	Value			
BrowseName	WriterGroupMessageDataType			
IsAbstract	True			
References	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Subtype of Structure defined in IEC 62541-5.				
HasSubtype	DataType	UadpWriterGroupMessageDataType	FALSE	Defined in 6.3.1.1.7.
HasSubtype	DataType	JsonWriterGroupMessageDataType	FALSE	Defined in 6.3.2.1.2.

6.2.6 PubSubConnection Parameters

6.2.6.1 PublisherId

The *PublisherId* is a unique identifier for a *Publisher* within a *Message Oriented Middleware*. It can be included in sent *NetworkMessage* for identification or filtering. The value of the *PublisherId* is typically shared between *PubSubConnections* but the assignment of the *PublisherId* is vendor specific.

The *PublisherId* parameter is only relevant for the *Publisher* functionality inside a *PubSubConnection*. The filter setting on the *Subscriber* side is contained in the *DataSetReader* parameters.

Valid *DataTypes* are *UInteger* and *String*.

6.2.6.2 TransportProfileUri

The *TransportProfileUri* parameter with *DataType String* indicates the transport protocol mapping and the message mapping used.

The possible *TransportProfileUri* values are defined as URI of the transport protocols defined as *PubSub* transport *Facet* in IEC 62541-7.

6.2.6.3 Address

The *Address* parameter contains the network address information for the communication middleware. The different Structure DataTypes used to represent the Address are defined in 6.2.6.5.3.

6.2.6.4 ConnectionProperties

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

The mapping of the namespace, name, and value to concrete functionality may be defined by transport protocol mappings, future editions of IEC 62541-14 or vendor-specific extensions.

6.2.6.5 PubSubConnection Structure

6.2.6.5.1 PubSubConnectionDataType

This *Structure DataType* is used to represent the configuration parameters for *PubSubConnections*. The *PubSubConnectionDataType* is formally defined in Table 26.

Table 26 – PubSubConnectionDataType structure

Name	Type	Description
PubSubConnectionDataType	Structure	
name	String	The name of the *PubSubConnection*.
enabled	Boolean	The enabled state of the *PubSubConnection*.
publisherId	BaseDataType	Defined in 6.2.6.1.
transportProfileUri	String	Defined in 6.2.6.2.
address	NetworkAddressDataType	Defined in 6.2.6.3. The *NetworkAddressDataType* is defined in 6.2.6.5.3.
connectionProperties	KeyValuePair[]	Defined in 6.2.6.4.
transportSettings	ConnectionTransportDataType	Transport mapping specific *PubSubConnection* parameters. The abstract base type is defined in 6.2.6.5.2. The concrete subtypes are defined in the subclauses for transport mapping specific parameters.
writerGroups	WriterGroupDataType[]	The *WriterGroups* contained in the *PubSubConnection*. The *WriterGroup* is defined in 6.2.5.
readerGroups	ReaderGroupDataType[]	The *ReaderGroups* contained in the *PubSubConnection*. The *ReaderGroup* is defined in 6.2.7.

6.2.6.5.2 ConnectionTransportDataType

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

The *ConnectionTransportDataType Structure* representation in the *AddressSpace* is defined in Table 27.

Table 27 – ConnectionTransportDataType definition

Attributes	Value		
BrowseName	ConnectionTransportDataType		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**IsAbstract**
Subtype of Structure defined in IEC 62541-5.			

6.2.6.5.3 NetworkAddressDataType

Subtypes of this abstract *Structure DataType* are used to represent network address information. The *NetworkAddressDataType* is formally defined in Table 28.

Table 28 – NetworkAddressDataType structure

Name	Type	Description
NetworkAddressDataType	Structure	
networkInterface	String	The name of the network interface used for the communication relation.

The *NetworkAddressDataType Structure* representation in the *AddressSpace* is defined in Table 29.

Table 29 – NetworkAddressDataType definition

Attributes	Value			
BrowseName	NetworkAddressDataType			
IsAbstract	True			
References	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Subtype of Structure defined in IEC 62541-5.				
HasSubtype	DataType	NetworkAddressUrlDataType	False	Defined in 6.2.6.5.4.

6.2.6.5.4 NetworkAddressUrlDataType

This *Structure DataType* is used to represent network address information in the form of an URL *String*. The *NetworkAddressUrlDataType* is formally defined in Table 30.

Table 30 – NetworkAddressUrlDataType structure

Name	Type	Description
NetworkAddressUrlDataType	Structure	
url	String	The address string for the communication relation in the form on an URL *String*.

The *NetworkAddressUrlDataType Structure* representation in the *AddressSpace* is defined in Table 31.

Table 31 – NetworkAddressUrlDataType definition

Attributes	Value		
BrowseName	NetworkAddressUrlDataType		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**IsAbstract**
Subtype of NetworkAddressDataType defined in 6.2.6.5.3.			

6.2.7 ReaderGroup parameters

6.2.7.1 General

The *ReaderGroup* does not add parameters to the shared PubSubGroup parameters.

The *ReaderGroup* is used to group a list of *DataSetReaders*. 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*.

6.2.7.2 ReaderGroup structures

6.2.7.2.1 ReaderGroupDataType

This *Structure DataType* is used to represent the configuration parameters for *ReaderGroups*. The *ReaderGroupDataType* is formally defined in Table 32.

Table 32 – ReaderGroupDataType structure

Name	Type	Description
ReaderGroupDataType	Structure	
transportSettings	ReaderGroupTransportDataType	Transport mapping specific *ReaderGroup* parameters. The abstract base type is defined in 6.2.7.2.2. The concrete subtypes are defined in the subclauses for transport mapping specific parameters.
messageSettings	ReaderGroupMessageDataType	*NetworkMessage* mapping specific *ReaderGroup* parameters. The abstract base type is defined in 6.2.7.2.3. The concrete subtypes are defined in the subclauses for message mapping specific parameters.
dataSetReaders	DataSetReaderDataType[]	The DataSetReaders contained in the ReaderGroup. The DataSetReader is defined in 6.2.8.

The *ReaderGroupDataType Structure* representation in the *AddressSpace* is defined in Table 33.

Table 33 – ReaderGroupDataType definition

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

6.2.7.2.2 ReaderGroupTransportDataType

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

The *ReaderGroupTransportDataType Structure* representation in the *AddressSpace* is defined in Table 34.

Table 34 – ReaderGroupTransportDataType definition

Attributes	**Value**		
BrowseName	ReaderGroupTransportDataType		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**IsAbstract**
Subtype of Structure defined in IEC 62541-5.			

6.2.7.2.3 ReaderGroupMessageDataType

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

The *ReaderGroupMessageDataType Structure* representation in the *AddressSpace* is defined in Table 35.

Table 35 – ReaderGroupMessageDataType structure

Attributes	**Value**		
BrowseName	ReaderGroupMessageDataType		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**IsAbstract**
Subtype of Structure defined in IEC 62541-5.			

6.2.8 DataSetReader Parameters

6.2.8.1 PublisherId

The parameter *PublisherId* defines the *Publisher* to receive *NetworkMessages* from.

If the value is null, the parameter shall be ignored and all received *NetworkMessages* pass the *PublisherId* filter.

Valid *DataTypes* are *UInteger* and *String*.

6.2.8.2 WriterGroupId

The parameter *WriterGroupId* with *DataType UInt16* defines the identifier of the corresponding *WriterGroup*.

The default value 0 is defined as null value, and means this parameter shall be ignored.

6.2.8.3 DataSetWriterId

The parameter *DataSetWriterId* with *DataType UInt16* defines the *DataSet* selected in the *Publisher* for the DataSetReader.

If the value is 0 (null), the parameter shall be ignored and all received *DataSetMessages* pass the *DataSetWriterId* filter.

6.2.8.4 DataSetMetaData

The parameter *DataSetMetaData* provides the information necessary to decode *DataSetMessages* from the *Publisher*. If the *DataSetMetaData* changes in the *Publisher* and the *MajorVersion* was changed, the *DataSetReader* needs an update of the *DataSetMetaData* for further operation. If the update cannot be retrieved in the duration of the *MessageReceiveTimeout*, the *State* of the *DataSetReader* shall change to Error_3. The related *PublishedDataSet* is defined in 6.2.2. The *DataSetMetaDataType* is defined in 6.2.2.1.2. The options for retrieving the update of the *DataSetMetaData* are described in 5.2.3.

6.2.8.5 DataSetFieldContentMask

The parameter DataSetFieldContentMask with DataType DataSetFieldContentMask indicates the fields of a DataValue included in the DataSetMessages.

The *DataSetFieldContentMask* DataType is defined in 6.2.3.2.

6.2.8.6 MessageReceiveTimeout

The parameter *MessageReceiveTimeout* is the maximum acceptable time between two *DataSetMessages*. If there is no *DataSetMessage* received within this period, the *DataSetReader State* shall be changed to Error_3 until the next *DataSetMessage* is received. The *DataSetMessages* can be data or keep alive messages.

The *MessageReceiveTimeout* is related to the *Publisher* side parameters *PublishingInterval*, *KeepAliveTime* and *KeyFrameCount*.

6.2.8.7 KeyFrameCount

The *KeyFrameCount* with *DataType UInt32* is the multiplier of the *PublishingInterval* that defines the maximum number of times the *PublishingInterval* expires before a key frame message, with all field values, is received.

For *DataSets* that provide cyclic updates, the value shall be greater than or equal to 1. For non-cyclic *DataSets,* like *PublishedEvents,* that provide event based *DataSets*, the value shall be 0.

6.2.8.8 HeaderLayoutUri

The *HeaderLayoutUri*, with *DataType String*, defines the selection of a well defined configuration for a subset of the PubSub communication parameters. The affected subset is defined by the header layout.

A null *String* is defined as no layout selected.

If a layout is selected, all affected parameters shall be set to the values defined for the layout.

6.2.8.9 SecurityMode

The parameter is defined in 6.2.4.2.

This parameter overwrites the corresponding setting on the *ReaderGroup* if the value is not INVALID_0.

6.2.8.10 SecurityGroupId

The parameter is defined in 6.2.4.3.

The parameter shall be null if the *SecurityMode* is INVALID_0.

6.2.8.11 SecurityKeyServices

The parameter is defined in 6.2.4.4.

The parameter shall be null if the *SecurityMode* is INVALID_0.

6.2.8.12 DataSetReaderProperties

The *DataSetReaderProperties* parameter is an array of *DataType KeyValuePair* that specifies additional properties for the configured *DataSetReader*. 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 editions of IEC 62541-14 or vendor-specific extensions.

6.2.8.13 DataSetReader Structure

6.2.8.13.1 DataSetReaderDataType

This *Structure DataType* is used to represent the *DataSetReader* parameters. The *DataSetReaderDataType* is formally defined in Table 36.

Table 36 – DataSetReaderDataType structure

Name	Type	Description
DataSetReaderDataType	Structure	
name	String	The name of the DataSetReader. The name shall be unique across a *ReaderGroup*. It is recommended to use a human readable name.
enabled	Boolean	The enabled state of the DataSetReader.
publisherId	BaseDataType	Defined in 6.2.8.1.
writerGroupId	UInt16	Defined in 6.2.8.2.
dataSetWriterId	UInt16	Defined in 6.2.8.3.
dataSetMetaData	DataSetMetaDataType	Defined in 6.2.8.4.
dataSetFieldContentMask	DataSetFieldContentMask	Defined in 6.2.8.5.
messageReceiveTimeout	Duration	Defined in 6.2.8.6.
keyFrameCount	UInt32	Defined in 6.2.8.7.
headerLayoutUri	String	Defined in 6.2.8.8.
securityMode	MessageSecurityMode	Defined in 6.2.8.9.
securityGroupId	String	Defined in 6.2.8.10.
securityKeyServices	EndpointDescription[]	Defined in 6.2.8.11.
dataSetReaderProperties	KeyValuePair[]	Defined in 6.2.8.12.
transportSettings	DataSetReaderTransportDataType	Transport-specific DataSetReader parameters. The abstract base type is defined in 6.2.8.13.2. The concrete subtypes are defined in the subclauses for transport mapping specific parameters. If no concrete subtype is defined for the message mapping, the field shall be null.
messageSettings	DataSetReaderMessageDataType	DataSetMessage mapping specific DataSetReader parameters. The abstract base type is defined in 6.2.8.13.3. The concrete subtypes are defined in the subclauses for message mapping specific parameters. If no concrete subtype is defined for the message mapping, the field shall be null.
subscribedDataSet	SubscribedDataSetDataType	The SubscribedDataSet specific parameters. The abstract base type and the concrete subtypes are defined in 6.2.9.

6.2.8.13.2 DataSetReaderTransportDataType

This *Structure DataType* is an abstract base type for transport-specific *DataSetReader* parameters. The *DataSetReaderTransportDataType* is formally defined in Table 37.

Table 37 – DataSetReaderTransportDataType structure

Name	Type	Description
DataSetReaderTransportDataType	Structure	

The *DataSetReaderTransportDataType Structure* representation in the *AddressSpace* is defined in Table 38.

Table 38 – DataSetReaderTransportDataType definition

Attributes	Value			
BrowseName	DataSetReaderTransportDataType			
IsAbstract	True			
References	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Subtype of Structure defined in IEC 62541-5.				
HasSubtype	DataType	BrokerDataSetReaderTransportDataType	FALSE	Defined in 6.4.2.4.6.

6.2.8.13.3 DataSetReaderMessageDataType

This *Structure DataType* is an abstract base type for message mapping specific *DataSetReader* parameters. The *DataSetReaderMessageDataType* is formally defined in Table 39.

Table 39 – DataSetReaderMessageDataType structure

Name	Type	Description
DataSetReaderMessageDataType	Structure	

The *DataSetReaderMessageDataType Structure* representation in the *AddressSpace* is defined in Table 40.

Table 40 – DataSetReaderMessageDataType definition

Attributes	Value			
BrowseName	DataSetReaderMessageDataType			
IsAbstract	True			
References	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Subtype of Structure defined in IEC 62541-5.				
HasSubtype	DataType	UadpDataSetReaderMessageDataType	FALSE	Defined in 6.3.1.3.10.
HasSubtype	DataType	JsonDataSetReaderMessageDataType	FALSE	Defined in 6.3.2.3.3.

6.2.9 SubscribedDataSet Parameters

6.2.9.1 SubscribedDataSetDataType

This *Structure DataType* is an abstract base type for *SubscribedDataSet* parameters. The *SubscribedDataSetDataType* is formally defined in Table 41.

Table 41 – SubscribedDataSetDataType structure

Name	Type	Description
SubscribedDataSetDataType	Structure	

The *SubscribedDataSetDataType Structure* representation in the *AddressSpace* is defined in Table 42.

Table 42 – SubscribedDataSetDataType Definition

Attributes	Value			
BrowseName	SubscribedDataSetDataType			
IsAbstract	True			
References	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Subtype of Structure defined in IEC 62541-5.				
HasSubtype	DataType	TargetVariablesDataType	FALSE	Defined in 6.2.9.2.2.
HasSubtype	DataType	SubscribedDataSetMirrorDataType	FALSE	Defined in 6.2.9.3.3.

6.2.9.2 TargetVariables

6.2.9.2.1 General

The *SubscribedDataSet* option *TargetVariables* defines a list of *Variable* mappings between received *DataSet* fields and target *Variables* in the *Subscriber AddressSpace*. The *FieldTargetDataType* is defined in 6.2.9.2.3. Target *Variables* shall only be used once within the same *TargetVariables* list.

6.2.9.2.2 TargetVariablesDataType

This *Structure DataType* is used to represent *TargetVariables* specific parameters. It is a subtype of the *SubscribedDataSetDataType* defined in 6.2.9.1.

The *TargetVariablesDataType* is formally defined in Table 43.

Table 43 – TargetVariablesDataType structure

Name	Type	Description
TargetVariablesDataType	Structure	
targetVariables	FieldTargetDataType[]	Defined in 6.2.9.2.1.

6.2.9.2.3 FieldTargetDataType

This *DataType* is used to provide the metadata for the relation between a field in a *DataSetMessage* and a target *Variable* in a *DataSetReader*. The *FieldTargetDataType* is formally defined in Table 44.

Table 44 – FieldTargetDataType structure

Name	Type	Description
FieldTargetDataType	Structure	
dataSetFieldId	Guid	The unique ID of the field in the *DataSet*. The fields and their unique IDs are defined in the *DataSetMetaData Structure*.
receiverIndexRange	NumericRange	Index range used to extract parts of an array out of the received data. It is used to identify a single element of an array, or a single range of indexes for arrays for the received *DataSet* field. If a range of elements is specified, the values are returned as a composite. The first element is identified by index 0 (zero). The *NumericRange* type is defined in IEC 62541-4. This parameter is null if the specified Attribute is not an array. However, if the specified Attribute is an array, and this parameter is null, then the complete array is used. The resulting data array size of this *NumericRange* shall match the resulting data array size of the *writeIndexRange NumericRange* setting.
targetNodeId	NodeId	The *NodeId* of the *Variable* to which the received *DataSetMessage* field value is written.
attributeId	IntegerId	Id of the *Attribute* to write, e.g. the *Value Attribute*. This shall be a valid *AttributeId*. The Attributes are defined in IEC 62541-3. The *IntegerId DataType* is defined in IEC 62541-4. The *IntegerIds* for the *Attributes* are defined in IEC 62541-6.
writeIndexRange	NumericRange	The index range used for writing received data to the target node. It is used to identify a single element of an array, or a single range of indexes for arrays for the write operation to the target *Node*. If a range of elements is specified, the values are written as a composite. The first element is identified by index 0 (zero). The *NumericRange* type is defined in IEC 62541-4. This parameter is null if the specified *Attribute* is not an array. However, if the specified *Attribute* is an array, and this parameter is null, then the complete array is used.
overrideValueHandling	OverrideValueHandling	The value is used to define the override value handling behaviour if the State of the *DataSetReader* is not Operational_2 or if the corresponding field in the *DataSet* contains a *Bad StatusCode*. The handling of the *OverrideValue* in different scenarios is defined in 6.2.10. The *OverrideValueHandling* enumeration *DataType* is defined in 6.2.9.2.4.
overrideValue	Variant	This value is used if the *OverrideValueHandling* is set to *OverrideValue_2* and the State of the *DataSetReader* is not Operational_2 or if the corresponding field in the *DataSet* contains a *Bad StatusCode*. The handling of the *OverrideValue* in different scenarios is defined in 6.2.10. This Value shall match the *DataType* of the target *Node*.

6.2.9.2.4 OverrideValueHandling

The *OverrideValueHandling* is an enumeration that specifies the possible options for the handling of Override values. The possible enumeration values are described in Table 45.

Table 45 – OverrideValueHandling values

Value	Description
Disabled_0	The override value handling is disabled.
LastUsableValue_1	In the case of an error, the last usable value is used. If no last usable value is available, the default value for the data type is used.
OverrideValue_2	In the case of an error, the configured override value is used.

6.2.9.3 SubscribedDataSetMirror

6.2.9.3.1 ParentNodeName

This parameter with *DataType String* defines the *BrowseName* and *DisplayName* of the parent *Node* for the *Variables* representing the fields of the subscribed *DataSet*.

6.2.9.3.2 RolePermissions

This parameter with *DataType RolePermissionType* defines the value of the *RolePermissions* Attribute to be set on the parent Node. This value is also used as *RolePermissions* for all *Variables* of the *DataSet* mirror.

6.2.9.3.3 SubscribedDataSetMirrorDataType

This *Structure DataType* is used to represent *SubscribedDataSetMirror* specific parameters. It is a subtype of the *SubscribedDataSetDataType* defined in 6.2.9.1.

The *SubscribedDataSetMirrorDataType* is formally defined in Table 46.

Table 46 – SubscribedDataSetMirrorDataType structure

Name	Type	Description
SubscribedDataSetMirrorDataType	Structure	
parentNodeName	String	Defined in 6.2.9.3.1.
rolePermissions	RolePermissionType[]	Defined in 6.2.9.3.2.

6.2.10 Information flow and status handling

The configuration model defines different parameters that influence the information flow from *Publisher* to *Subscriber* in the case of a Bad Value Status or other error situations. Figure 22 depicts the parameters and the information flow inside a *Publisher* and inside a *Subscriber*.

The parameters and behaviour relevant for the encoding of a *DataSetMessage* on the *Publisher* side and the decoding of the *DataSetMessage* on the *Subscriber* side are defined in 6.2.3.1 together with the *DataSetFieldContentMask*.

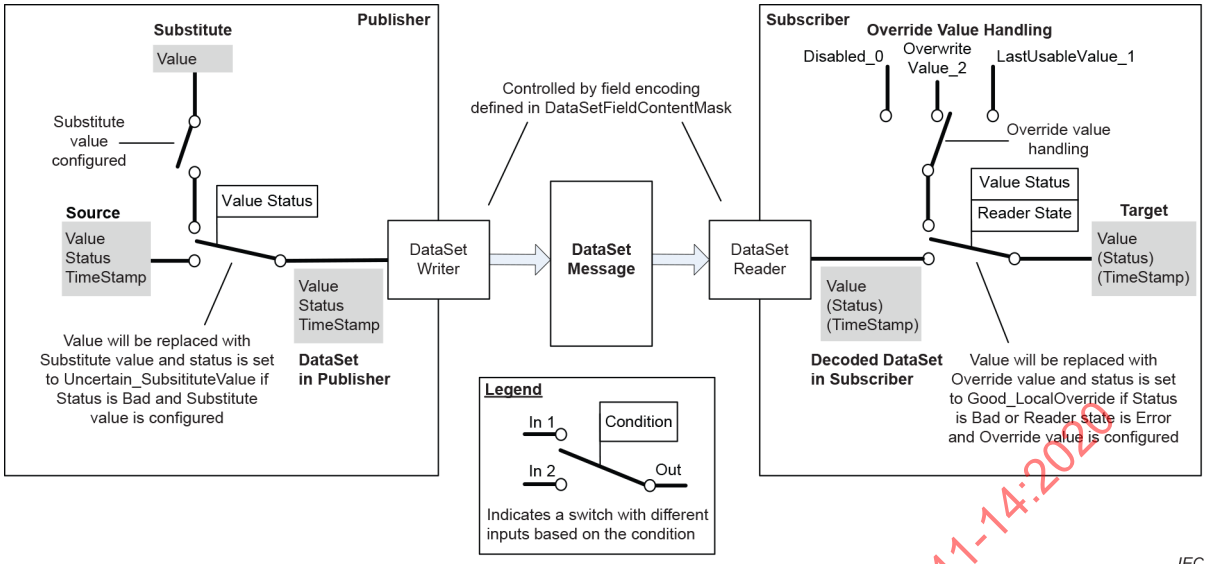


Figure 22 – PubSub information flow

The mapping of source value and status to the *DataSet* in the *Publisher* depends on the substitute value. The dependencies are defined in Table 47.

Table 47 – Source to message input mapping

Source		Substitute Value	DataSet Publisher side	
Value	**Status**[a]		**Value**	**Status**[a]
Value 1	Good_*	Value 2	Value 1	Good_*
Value 1	Uncertain_*		Value 1	Uncertain_*
Null	Bad_*		Value 2	Uncertain_SubstituteValue
Value 1	Good_*	Null	Value 1	Good_*
Value 1	Uncertain_*		Value 1	Uncertain_*
Null	Bad_*		Null	Bad_*

[a] 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.

The mapping of the decoded *DataSet* on the *Subscriber* side to the value and status of the target *Variable* depends on the override value. The dependencies are defined in Table 48.

Table 48 – Message output to target mapping

Decoded DataSet Subscriber		Override Value Handling Enum	Override Value	Reader State	Target	
Value	Status[a]				Value	Status[a]
Value 1	Good_*	OverrideValue_2	Value 2	Operational_2	Value 1	Good_*
Value 1	Uncertain_*				Value 1	Uncertain_*
Null	Bad_*				Value 2	Good_LocalOverride
Value 1	Good_*	LastUsableValue_1	Null		Value 1	Good_*
Value 1	Uncertain_*				Value 1	Uncertain_*
Null	Bad_*				LastValue[b]	Uncertain_LastUsableValue
Value 1	Good_*	Disabled_0	Null		Value 1	Good_*
Value 1	Uncertain_*				Value 1	Uncertain_*
Null	Bad_*				Null	Bad_*
No message received. The target values are updated once after a reader state change.		OverrideValue_2	Value 2	Diabled_0 Paused_1	Value 2	Good_LocalOverride
		LastUsableValue_1	Null		LastValue[b]	Uncertain_LastUsableValue
		Disabled_0	Null		Null	Bad_OutOfService
		OverrideValue_2	Value 2	Error_3	Value 2	Good_LocalOverride
		LastUsableValue_1	Null		LastValue[b]	Uncertain_LastUsableValue
		Disabled_0	Null		Null	Bad_NoCommunication

[a] 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.

[b] The last value is either the last received value or the default value for the data type if there was never a value received before.

6.2.11 PubSubConfigurationDataType

This *Structure DataType* is used to represent the *PubSub* configuration of an OPC UA *Application*. The *PubSubConfigurationDataType* is formally defined in Table 49.

Table 49 – PubSubConfigurationDataType structure

Name	Type	Description
PubSubConfigurationDataType	Structure	
publishedDataSets	PublishedDataSetDataType[]	The *PublishedDataSets* contained in the configuration. The *PublishedDataSet* is defined in 6.2.2.
connections	PubSubConnectionDataType[]	The *PubSubConnections* contained in the configuration. The *PubSubConnection* is defined in 6.2.6. The connection includes *WriterGroups* and *ReaderGroups*.
enabled	Boolean	The enabled state of the *PubSub* configuration.

If the *PubSub* configuration is stored in a file, the *UABinaryFileDataType* and the related definitions in A.2 shall be used to encode the file content. The values of the *UABinaryFileDataType* structure are described in Table 50.

Table 50 – PubSubConfiguration file content

Field	Type	Value
namespaces	String[]	null The *DataTypes* used for configuration are defined in the OPC UA namespace.
structureDataTypes	StructureDescription[]	null *DataTypes* used for configuration are defined by OPC UA.
enumDataTypes	EnumDescription[]	null *DataTypes* used for configuration are defined by OPC UA.
simpleDataTypes	SimpleTypeDescription[]	null *DataTypes* used for configuration are defined by OPC UA.
schemaLocation	String	null
fileHeader	KeyValuePair[]	null
body	BaseDataType	*PubSubConfigurationDataType Structure* The *PubSub* configuration represented by the *PubSubConfigurationDataType*.

6.3 Message mapping configuration parameters

6.3.1 UADP message mapping

6.3.1.1 UADP NetworkMessage writer

6.3.1.1.1 Relationship of Timing parameters

The *PublishingInterval*, the *SamplingOffset* the *PublishingOffset* and the timestamp in the *NetworkMessage* header shall use the same time base.

If an underlying network provides a synchronized global clock, this clock shall be used as the time base for the *Publisher* and *Subscriber*.

The beginning of a *PublishingInterval* shall be a multiple of the *PublishingInterval* relative to the start of the time base. The reference start time of the *PublishsingInterval* can be calculated by using the following formula:

Start of periodic execution =

current time + PublishingInterval − (current time / PublishingInterval)

Current time is the number of nanoseconds since the start of epoch used by the reference clock.

PublishingInterval is the duration in nanoseconds.

Start of periodic execution is the number of nanoseconds since the start of epoch which is the next possible start of a *PublishingInterval.*

Figure 23 shows an example how to select the possible start of a *PublishingInterval*.

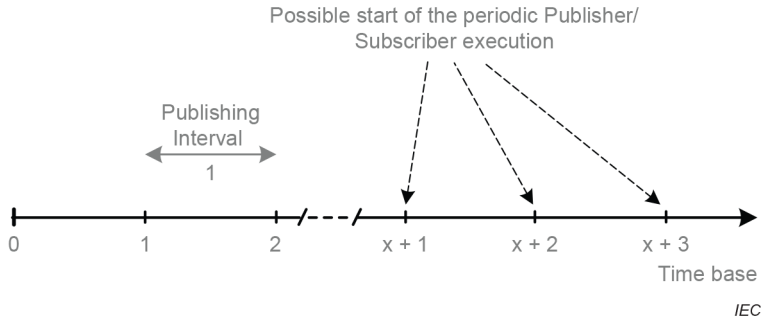


Figure 23 – Start of the periodic publisher execution

The different timing offsets inside a *PublishingInterval* cycle on *Publisher* and *Subscriber* side are shown in Figure 24. The *SamplingOffset* and *PublishingOffset* are defined as parameters of the UADP *WriterGroup*. The *ReceiveOffset* and the *ProcessingOffset* are defined as parameters of the UADP *DataSetReader* in 6.3.1.3.

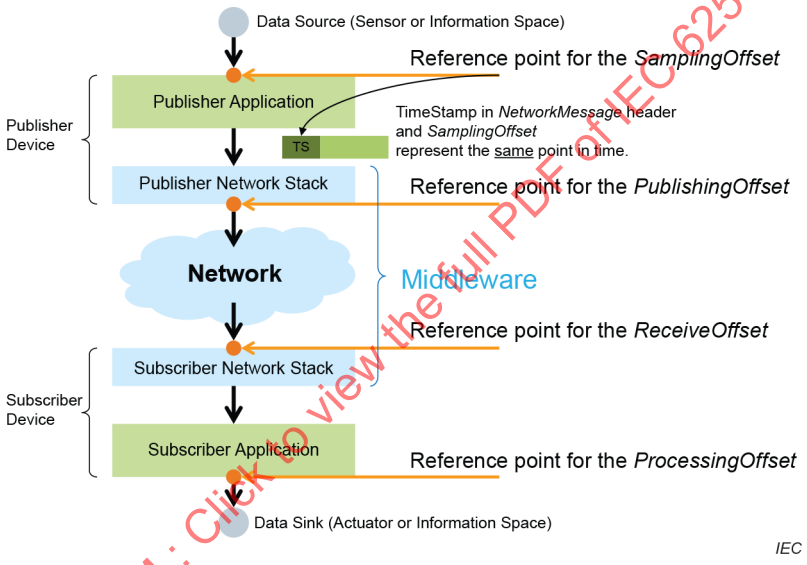


Figure 24 – Timing offsets in a PublishingInterval

6.3.1.1.2 GroupVersion

The *GroupVersion* with *DataType VersionTime* reflects the time of the last layout change of the content of the *NetworkMessages* published by the *WriterGroup*. The *VersionTime DataType* is defined in IEC 62541-4. The *GroupVersion* changes when one of the following parameters is modified:

- *NetworkMessageContentMask* of this *WriterGroup*;
- *Offset* of any *DataSetWriter* in this *WriterGroup*;
- *MinorVersion* of the *DataSet* of any *DataSetWriter* in this *WriterGroup*;
- *DataSetFieldContentMask* of any *DataSetWriter* in this *WriterGroup*;
- *DataSetMessageContentMask* of any *DataSetWriter* in this *WriterGroup*;
- *DataSetWriterId* of any *DataSetWriter* in this *WriterGroup*.

The *GroupVersion* is valid for all *NetworkMessages* resulting from this *WriterGroup*.

6.3.1.1.3 DataSetOrdering

The *DataSetOrdering* defines the ordering of the *DataSetMessages* in the *NetworkMessages*. Possible values for *DataSetOrdering* are described in Table 51. The default value is Undefined_0.

The *DataSetOrderingType* is an enumeration that specifies the possible options for the ordering of *DataSetMessages* inside *NetworkMessages*. The possible enumeration values are described in Table 51.

Table 51 – DataSetOrderingType values

Value	Description
Undefined_0	The ordering of *DataSetMessages* is not specified.
AscendingWriterId_1	*DataSetMessages* are ordered ascending by the value of their corresponding *DataSetWriterIds*.
AscendingWriterIdSingle_2	*DataSetMessages* are ordered ascending by the value of their corresponding *DataSetWriterIds* and only one *DataSetMessage* is sent per *NetworkMessage*.

If *DataSetOrdering* is *Undefined_0* any ordering between DataSets and their distribution into *NetworkMessages* is allowed. Ordering and distribution even may change between each *PublishingInterval*. If *DataSetOrdering* is set to *AscendingWriterId_1*, the *Publisher* shall fill up each *NetworkMessage* with *DataSets* with an ascending order of the related *DataSetWriterIds* as long as the accumulated *DataSet* sizes will not exceed the *MaxNetworkMessageSize*. The different options are shown in Figure 25.

Figure 25 – DataSetOrdering and MaxNetworkMessageSize

6.3.1.1.4 NetworkMessageContentMask

The parameter *NetworkMessageContentMask* defines the optional header fields to be included in the *NetworkMessages* produced by the *WriterGroup*. The *DataType* for the UADP *NetworkMessage* mapping is *UadpNetworkMessageContentMask*.

The *DataType UadpNetworkMessageContentMask* is formally defined in Table 52.

Table 52 – UadpNetworkMessageContentMask values

Value	Bit No.	Description
PublisherId	0	The *PublisherId* is included in the *NetworkMessage*s.
GroupHeader	1	The GroupHeader is included in the *NetworkMessages*.
WriterGroupId	2	The *WriterGroupId* field is included in the *GroupHeader*. The flag is only valid if Bit 1 is set.
GroupVersion	3	The *GroupVersion* field is included in the *GroupHeader*. The flag is only valid if Bit 1 is set.
NetworkMessageNumber	4	The *NetworkMessageNumber* field is included in the *GroupHeader*. The field is required if more than one NetworkMessage is needed to transfer all DataSets of the group. The flag is only valid if Bit 1 is set.
SequenceNumber	5	The *SequenceNumber* field is included in the *GroupHeader*. The flag is only valid if Bit 1 is set.
PayloadHeader	6	The *PayloadHeader* is included in the *NetworkMessages*.
Timestamp	7	The sender timestamp is included in the *NetworkMessage*s.
PicoSeconds	8	The sender *PicoSeconds* portion of the timestamp is included in the *NetworkMessage*s.
DataSetClassId	9	The *DataSetClassId* is included in the *NetworkMessage*s.
PromotedFields	10	The *PromotedFields* are included in the *NetworkMessage*s.

The *UadpNetworkMessageContentMask* representation in the *AddressSpace* is defined in Table 53.

Table 53 – UadpNetworkMessageContentMask definition

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

6.3.1.1.5 SamplingOffset

The *SamplingOffset* with the *DataType Duration* defines the time in milliseconds for the offset of creating the *NetworkMessage* in the *PublishingInterval* cycle.

Any negative value indicates that the optional parameter is not configured. In this case the *Publisher* shall calculate the time before the *PublishingOffset* that is necessary to create the *NetworkMessage* in time for sending at the *PublishingOffset*.

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

6.3.1.1.6 PublishingOffset

The *PublishingOffset* is an array of *DataType Duration* that defines the time in milliseconds for the offset in the *PublishingInterval* cycle of sending the *NetworkMessage* to the network.

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

Figure 26 depicts how the different variations of *PublishingOffset* settings affect sending of multiple *NetworkMessages*.

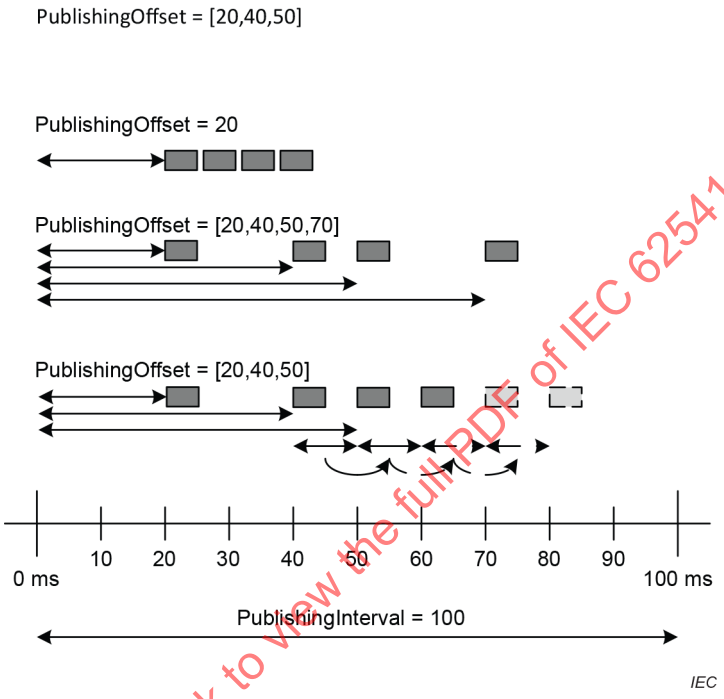


Figure 26 – PublishingOffset options for multiple *NetworkMessages*

If all *DataSets* of a group are transferred with a single *NetworkMessage*, the scalar value or the first value in the array defines the offset for sending the *NetworkMessage* relative to the start of the *PublishingInterval* cycle. If the *DataSets* of a group are sent in a series of *NetworkMessages* the values in the array define the offsets of sending the *NetworkMessages* relative to the start of the *PublishingInterval* cycle. If a scalar value is configured, the first *NetworkMessage* is sent at the offset and the following *NetworkMessages* are sent immediately after each other. If more *NetworkMessages* are available for sending than offset values in the array, the offset for the remaining *NetworkMessages* is extrapolated from the last two offset values in the array.

The *PublishingInterval*, the *SamplingOffset* the *PublishingOffset* and the timestamp in the *NetworkMessage* header shall use the same time base.

6.3.1.1.7 UadpWriterGroupMessageDataType Structure

This *Structure DataType* is used to represent the UADP *NetworkMessage* mapping specific WriterGroup parameters. It is a subtype of *WriterGroupMessageDataType* defined in 6.2.5.7.3.

The *UadpWriterGroupMessageDataType* is formally defined in Table 54.

Table 54 – UadpWriterGroupMessageDataType structure

Name	Type	Description
UadpWriterGroupMessageDataType	Structure	
groupVersion	UInt32	Defined in 6.3.1.1.2.
dataSetOrdering	DataSetOrderingType	Defined in 6.3.1.1.3.
networkMessageContentMask	UadpNetworkMessageContentMask	Defined in 6.3.1.1.4.
samplingOffset	Duration	Defined in 6.3.1.1.5.
publishingOffset	Duration[]	Defined in 6.3.1.1.6.

6.3.1.2 UADP DataSetMessage Writer

6.3.1.2.1 General

The configuration of the *DataSetWriters* in a *WriterGroup* can result in a fixed *NetworkMessage* layout where all *DataSets* have a static position between *NetworkMessages*.

In this case, the parameters *NetworkMessageNumber* and *DataSetOffset* provide information about the static position of the *DataSetMessage* in a *NetworkMessage Subscribers* can rely on. If the value of one of the two parameters is 0, the position is not guaranteed to be static.

NOTE A *Publisher* can only provide valid values for the parameters *NetworkMessageNumber* and *DataSetOffset* if the message mapping allows keeping the value for these *Properties* constant unless the configuration of the *WriterGroup* is changed.

6.3.1.2.2 DataSetMessageContentMask

The *DataSetMessageContentMask* defines the flags for the content of the *DataSetMessage* header. The UADP message mapping specific flags are defined by the *UadpDataSetMessageContentMask DataType*.

The *UadpDataSetMessageContentMask DataType* is formally defined in Table 55.

Table 55 – UadpDataSetMessageContentMask values

Value	Bit No.	Description
Timestamp	0	If this flag is set, a timestamp shall be included in the *DataSetMessage* header.
PicoSeconds	1	If this flag is set, a *PicoSeconds* timestamp field shall be included in the *DataSetMessage* header. This flag is ignored if the *HeaderTimestamp* flag is not set.
Status	2	If this flag is set, the *DataSetMessage* status is included in the *DataSetMessage* header. The rules for creating the *DataSetMessage* status are defined in Table 16.
MajorVersion	3	If this flag is set, the *ConfigurationVersion.MajorVersion* is included in the *DataSetMessage* header.
MinorVersion	4	If this flag is set, the *ConfigurationVersion.MinorVersion* is included in the *DataSetMessage* header.
SequenceNumber	5	If this flag is set, the DataSetMessageSequenceNumber is included in the DataSetMessage header.

The *UadpDataSetMessageContentMask* representation in the *AddressSpace* is defined in Table 56.

Table 56 – UadpDataSetMessageContentMask definition

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

6.3.1.2.3 ConfiguredSize

The parameter *ConfiguredSize* with the *DataType UInt16* defines the fixed size in bytes a *DataSetMessage* uses inside a *NetworkMessage*. The default value is 0 and it indicates a dynamic length. If a *DataSetMessage* would be smaller in size (e.g. because of the current values that are encoded) the *DataSetMessage* is padded with bytes with value zero. In case it would be larger, the *Publisher* shall set bit 0 of the *DataSetFlags1* to false to indicate that the *DataSetMessage* is not valid.

NOTE The parameter *ConfiguredSize* can be used for different reasons. One reason is the reservation of space inside a *NetworkMessage* by setting *ConfiguredSize* to a higher value than the assigned *DataSet* actually requires. Modifications (e.g. extensions) of the *DataSet* would then not change the required bandwidth on the network which reduces the risk of side effects. Another reason would be to maintain predictable network behaviour even when using a volatile field *DataTypes* like *String* or *ByteString*.

6.3.1.2.4 NetworkMessageNumber

The parameter *NetworkMessageNumber* with the *DataType UInt16* is a read-only parameter set by the *Publisher* in the case of a fixed *NetworkMessage* layout. The default value is 0 and indicates that the position of the *DataSetMessage* in a *NetworkMessage* is not fixed.

If the *NetworkMessage* layout is fixed and all *DataSetMessages* of a *WriterGroup* fit into one single *NetworkMessage*, the value of *NetworkMessageNumber* shall be 1. If the *DataSetMessages* of a *WriterGroup* are distributed or chunked over more than one *NetworkMessage*, the first *NetworkMessage* in a *PublishingInterval* shall be generated with the value 1; the following *NetworkMessages* shall be generated with incrementing *NetworkMessageNumbers*. To avoid a roll-over, the number of *NetworkMessages* generated from one *WriterGroup* within one *PublishingInterval* is limited to 65 535.

6.3.1.2.5 DataSetOffset

The parameter *DataSetOffset* with the *DataType UInt16* is a read-only parameter set by the *Publisher* that specifies the offset in bytes inside a *NetworkMessage* at which the *DataSetMessage* is located, relative to the beginning of the *NetworkMessage*. The default value 0 indicates that the position of the *DataSetMessage* in a *NetworkMessage* is not fixed.

6.3.1.2.6 UadpDataSetWriterMessageDataType structure

This *Structure DataType* is used to represent UADP DataSetMessage mapping specific *DataSetWriter* parameters. It is a subtype of the *DataSetWriterMessageDataType* defined in 6.2.3.5.3.

The *UadpDataSetWriterMessageDataType* is formally defined in Table 57.

Table 57 – UadpDataSetWriterMessageDataType structure

Name	Type	Description
UadpDataSetWriterMessageDataType	Structure	
dataSetMessageContentMask	UadpDataSetMessageContentMask	Defined in 6.3.1.2.2.
configuredSize	UInt16	Defined in 6.3.1.2.3.
networkMessageNumber	UInt16	Defined in 6.3.1.2.4.
dataSetOffset	UInt16	Defined in 6.3.1.2.5.

6.3.1.3 UADP DataSetMessage reader

6.3.1.3.1 GroupVersion

The parameter *GroupVersion* with *DataType VersionTime* defines the expected value in the field *GroupVersion* in the header of the *NetworkMessage*. The default value 0 is defined as null value, and means this parameter shall be ignored.

6.3.1.3.2 NetworkMessageNumber

The parameter *NetworkMessageNumber* with *DataType UInt16* is the number of the *NetworkMessage* inside a *PublishingInterval* in which this *DataSetMessage* is published. The default value 0 is defined as null value, and means this parameter shall be ignored.

6.3.1.3.3 DataSetOffset

The parameter *DataSetOffset* with *DataType UInt16* defines the offset for the *DataSetMessage* inside the corresponding *NetworkMessage*. The default value 0 is defined as null value, and means this parameter shall be ignored.

6.3.1.3.4 DataSetClassId

The parameter *DataSetClassId* with *DataType Guid* defines a *DataSet* class related filter. If the value is null, the *DataSetClassId* filter is not applied.

6.3.1.3.5 NetworkMessageContentMask

The *NetworkMessageContentMask* with *DataType UadpNetworkMessageContentMask* indicates the optional header fields included in the received *NetworkMessages*. The *UadpNetworkMessageContentMask DataType* is defined in 6.3.1.1.4.

6.3.1.3.6 DataSetMessageContentMask

The *DataSetMessageContentMask* with the *DataType UadpDataSetMessageContentMask* indicates the optional header fields included in the *DataSetMessages*.

The *UadpDataSetMessageContentMask* DataType is defined in 6.3.1.2.2.

6.3.1.3.7 PublishingInterval

The *PublishingInterval* with *DataType Duration* indicates the rate the *Publisher* sends *NetworkMessages* related to the *DataSet*. The start time for the periodic execution of the *Subscriber* shall be calculated according to 6.3.1.1.1.

6.3.1.3.8 ReceiveOffset

The *ReceiveOffset* with *DataType Duration* defines the time in milliseconds for the offset in the *PublishingInterval* cycle for the expected receive time of the *NetworkMessage* for the *DataSet* from the network.

6.3.1.3.9 ProcessingOffset

The *ProcessingOffset* with *DataType Duration* defines the time in milliseconds for the offset in the *PublishingInterval* cycle when the received DataSet needs to be processed by the application in the *Subscriber*.

The different timing offsets inside a *PublishingInterval* cycle on the *Publisher* and *Subscriber* sides are shown in Figure 24.

6.3.1.3.10 UadpDataSetReaderMessageDataType

This *Structure DataType* is used to represent UADP message mapping specific *DataSetReader* parameters. It is a subtype of the *DataSetReaderMessageDataType* defined in 6.2.8.13.3.

The *UadpDataSetReaderMessageDataType* is formally defined in Table 58.

Table 58 – UadpDataSetReaderMessageDataType structure

Name	Type	Description
UadpDataSetReaderMessageDataType	Structure	
groupVersion	VersionTime	Defined in 6.3.1.3.1.
networkMessageNumber	UInt16	Defined in 6.3.1.3.2.
dataSetOffset	UInt16	Defined in 6.3.1.3.3.
dataSetClassId	Guid	Defined in 6.3.1.3.4.
networkMessageContentMask	UadpNetworkMessageContentMask	Defined in 6.3.1.3.5.
dataSetMessageContentMask	UadpDataSetMessageContentMask	Defined in 6.3.1.3.6.
publishingInterval	Duration	Defined in 6.3.1.3.7.
receiveOffset	Duration	Defined in 6.3.1.3.8.
processingOffset	Duration	Defined in 6.3.1.3.9.

6.3.2 JSON message mapping

6.3.2.1 JSON NetworkMessage Writer

6.3.2.1.1 NetworkMessageContentMask

The parameter *NetworkMessageContentMask* defines the optional header fields to be included in the *NetworkMessages* produced by the *WriterGroup*. The *DataType* for the JSON *NetworkMessage* mapping is *JsonNetworkMessageContentMask*.

The *DataType JsonNetworkMessageContentMask* is formally defined in Table 59.

Table 59 – JsonNetworkMessageContentMask values

Value	Bit No.	Description
NetworkMessageHeader	0	The JSON *NetworkMessage* header is included in the *NetworkMessages*. If this bit is false, bits 2 to 4 shall be 0.
DataSetMessageHeader	1	The JSON *DataSetMessage* header is included in each *DataSetMessage*. If this bit is false then the *DataSetMessageContentMask* for the *DataSetWriters* is ignored (see 6.3.2.2.1).
SingleDataSetMessage	2	Each JSON *NetworkMessage* contains only one *DataSetMessage*.
PublisherId	3	The *PublisherId* is included in the *NetworkMessages*.
DataSetClassId	4	The *DataSetClassId* is included in the *NetworkMessages*.
ReplyTo	5	The *ReplyTo* is included in the *NetworkMessages*.

The *JsonNetworkMessageContentMask* representation in the *AddressSpace* is defined in Table 60.

Table 60 – JsonNetworkMessageContentMask definition

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

6.3.2.1.2 JsonWriterGroupMessageDataType structure

This *Structure DataType* is used to represent the JSON *NetworkMessage* mapping specific *WriterGroup* parameters. It is a subtype of *WriterGroupMessageDataType* defined in 6.2.5.7.3.

The *JsonWriterGroupMessageDataType* is formally defined in Table 61.

Table 61 – JsonWriterGroupMessageDataType structure

Name	Type	Description
JsonWriterGroupMessageDataType	Structure	
networkMessageContentMask	JsonNetworkMessageContentMask	Defined in 6.3.2.1.1.

6.3.2.2 JSON DataSetMessage writer

6.3.2.2.1 DataSetMessageContentMask

The *DataSetMessageContentMask* defines the flags for the content of the *DataSetMessage* header. The JSON message mapping specific flags are defined by the *JsonDataSetMessageContentMask DataType.*

The *JsonDataSetMessageContentMask DataType* is formally defined in Table 62.

Table 62 – JsonDataSetMessageContentMask values

Value	Bit No.	Description
DataSetWriterId	0	If this flag is set, a *DataSetWriterId* shall be included in the *DataSetMessage* header.
MetaDataVersion	1	If this flag is set, the *ConfigurationVersion* is included in the *DataSetMessage* header.
SequenceNumber	2	If this flag is set, the *DataSetMessageSequenceNumber* is included in the *DataSetMessage* header.
Timestamp	3	If this flag is set, a timestamp shall be included in the *DataSetMessage* header.
Status	4	If this flag is set, an overall status is included in the *DataSetMessage* header.

The *JsonDataSetMessageContentMask* representation in the *AddressSpace* is defined in Table 63.

Table 63 – JsonDataSetMessageContentMask definition

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

6.3.2.2.2 JsonDataSetWriterMessageDataType structure

This *Structure DataType* is used to represent JSON *DataSetMessage* mapping specific *DataSetWriter* parameters. It is a subtype of the *DataSetWriterMessageDataType* defined in 6.2.3.5.3.

The *JsonDataSetWriterMessageDataType* is formally defined in Table 64.

Table 64 – JsonDataSetWriterMessageDataType structure

Name	Type	Description
JsonDataSetWriterMessageDataType	Structure	
dataSetMessageContentMask	JsonDataSetMessageContentMask	Defined in 6.3.2.2.1.

6.3.2.3 JSON DataSetMessage reader

6.3.2.3.1 NetworkMessageContentMask

The *NetworkMessageContentMask* with *DataType JsonNetworkMessageContentMask* indicates the optional header fields included in the received *NetworkMessages*. The *JsonNetworkMessageContentMask DataType* is defined in 6.3.2.1.1.

6.3.2.3.2 DataSetMessageContentMask

The *DataSetMessageContentMask* with the *DataType JsonDataSetMessageContentMask* indicates the optional header fields included in the *DataSetMessages*.

The *JsonDataSetMessageContentMask* DataType is defined in 6.3.2.2.1.

6.3.2.3.3 JsonDataSetReaderMessageDataType structure

This *Structure DataType* is used to represent JSON *DataSetMessage* mapping specific *DataSetReader* parameters. It is a subtype of the *DataSetReaderMessageDataType* defined in 6.2.8.13.3.

The *JsonDataSetReaderMessageDataType* is formally defined in Table 65.

Table 65 – JsonDataSetReaderMessageDataType structure

Name	Type	Description
JsonDataSetWriterMessageDataType	Structure	
networkMessageContentMask	JsonNetworkMessageContentMask	Defined in 6.3.2.3.1.
dataSetMessageContentMask	JsonDataSetMessageContentMask	Defined in 6.3.2.3.2.

6.4 Transport Protocol mapping configuration parameters

6.4.1 Datagram Transport Protocol

6.4.1.1 Datagram PubSubConnection

6.4.1.1.1 DiscoveryAddress

The *DiscoveryAddress* parameter contains the network address information used for the discovery request and response messages. The different *Structure DataTypes* used to represent the Address are defined in 6.2.6.5.3.

6.4.1.1.2 DatagramConnectionTransportDataType Structure

This *Structure DataType* is used to represent the configuration parameters for the Datagram transport protocol-specific settings of *PubSubConnections*. It is a subtype of the *ConnectionTransportDataType* defined in 6.2.6.5.2.

The *DatagramConnectionTransportDataType* is formally defined in Table 66.

Table 66 – DatagramConnectionTransportDataType structure

Name	Type	Description
DatagramConnectionTransportDataType	Structure	
discoveryAddress	NetworkAddressDataType	Defined in 6.4.1.1.1. The *NetworkAddressDataType* is defined in 6.2.6.5.3.

6.4.1.2 Datagram WriterGroup

6.4.1.2.1 MessageRepeatCount

The *MessageRepeatCount* with *DataType Byte* defines how many times every *NetworkMessage* is repeated. The default value is 0 and disables the repeating.

6.4.1.2.2 MessageRepeatDelay

The *MessageRepeatDelay* with *DataType Duration* defines the time between *NetworkMessage* repeats in milliseconds. The parameter shall be ignored if the parameter *MessageRepeatCount* is set to 0.

6.4.1.2.3 DatagramWriterGroupTransportDataType structure

This *Structure DataType* is used to represent the datagram specific transport mapping parameters for *WriterGroups*. It is a subtype of the *WriterGroupTransportDataType* defined in 6.2.5.7.2.

The *DatagramWriterGroupTransportDataType* is formally defined in Table 67.

Table 67 – DatagramWriterGroupTransportDataType structure

Name	Type	Description
DatagramWriterGroupTransportDataType	Structure	
messageRepeatCount	Byte	Defined in 6.4.1.2.1.
messageRepeatDelay	Duration	Defined in 6.4.1.2.2.

6.4.1.3 Datagram DataSetWriter parameters

There are no datagram-specific transport mapping parameters defined for the *DataSetWriter*.

6.4.1.4 Datagram DataSetReader

There are no datagram-specific transport mapping parameters defined for the *DataSetReader*.

6.4.2 Broker Transport Protocol

6.4.2.1 Broker PubSubConnection

6.4.2.1.1 ResourceUri

The *ResourceUri* parameter of *DataType String* enables the transport implementation to look up a configured key from the corresponding *KeyCredentialConfigurationType* instance defined in IEC 62541-12 to use for authenticating access to the broker at the connection level or for queues configured below the connection.

If null, no authentication or anonymous authentication shall be assumed as default unless authentication settings are provided on a subordinated *WriterGroup* or a *DataSetWriter* to authenticate access to individual queues.

6.4.2.1.2 AuthenticationProfileUri

The parameter *AuthenticationProfileUri* of *DataType String* allows the selection of the authentication protocol used by the transport implementation. This maps to the *ProfileUri Property* in the *KeyCredentialConfigurationType* instance selected through the *ResourceUri* and *AuthenticationProfileUri Strings*.

This parameter is optional. If more than one *ProfileUri* describing the protocol to use for authentication is configured and this value is null, the transport will choose one. If the transport cannot fine a suitable authentication mechanism in the *ProfileUri* array, the transport sets the *State* of the *PubSubConnection* is set to *Error_3*.

6.4.2.1.3 BrokerConnectionTransportDataType structure

This *Structure DataType* is used to represent the Broker-specific transport mapping parameters for the *PubSubConnection*. It is a subtype of the *ConnectionTransportDataType* defined in 6.2.6.5.2.

The *BrokerConnectionTransportDataType* is formally defined in Table 68.

Table 68 – BrokerConnectionTransportDataType structure

Name	Type	Description
BrokerConnectionTransportDataType	Structure	
resourceUri	String	Defined in 6.4.2.1.1.
authenticationProfileUri	String	Defined in 6.4.2.1.2.

6.4.2.2 Broker WriterGroup

6.4.2.2.1 QueueName

The *QueueName* parameter with *DataType String* specifies the queue in the *Broker* that receives *NetworkMessages* sent by the *Publisher*. This could be the name of a queue or topic defined in the *Broker*.

6.4.2.2.2 ResourceUri

The *ResourceUri* property of *DataType String* allows the transport implementation to look up the configured key from the corresponding *KeyCredentialConfigurationType* instance defined in IEC 62541-12 to use for authenticating access to the specified queue.

If this *String* is not null, it overrides the *ResourceUri* of the *PubSubConnection* authentication settings.

6.4.2.2.3 AuthenticationProfileUri

The parameter *AuthenticationProfileUri* of *DataType String* allows the selection of the authentication protocol used by the transport implementation for authenticating access to the specified queue.

If this *String* is not null, it overrides the *AuthenticationProfileUri* of the *PubSubConnection* transport settings defined in 6.4.2.1.2.

6.4.2.2.4 RequestedDeliveryGuarantee

The *RequestedDeliveryGuarantee* parameter with *DataType BrokerTransportQualityOfService* specifies the delivery guarantees that shall apply to all *NetworkMessages* published by the *WriterGroup* unless otherwise specified on the *DataSetWriter* transport settings. The *DataType BrokerTransportQualityOfService* is defined in 6.4.2.2.5.

The value *NotSpecified_0* is not allowed on the *WriterGroup*. If the selected delivery guarantee cannot be applied, the *WriterGroup* shall set the state to *Error_3*.

6.4.2.2.5 BrokerTransportQualityOfService Enumeration

The *BrokerTransportQualityOfService* Enumeration *DataType* is formally defined in Table 69.

The mapping of quality of service to the broker transport specific implementation is defined in 7.3.4.5 for AMQP and 7.3.5.5 for MQTT.

Table 69 – BrokerTransportQualityOfService values

Value	Description
NotSpecified_0	The value is not specified and the value of the parent object shall be used.
BestEffort_1	The transport shall make the best effort to deliver a message. Worst case this means data loss or data duplication are possible.
AtLeastOnce_2	The transport guarantees that the message shall be delivered at least once, but duplication is possible. Readers shall de-duplicate based on message id or sequence number.
AtMostOnce_3	The transport guarantees that the message shall be sent once, but if it is lost it is not sent again.
ExactlyOnce_4	The transport handshake guarantees that the message shall be delivered to the broker exactly once and not more or less.

6.4.2.2.6 BrokerWriterGroupTransportDataType structure

This *Structure DataType* is used to represent the Broker-specific transport mapping parameters for *WriterGroups*. It is a subtype of the *WriterGroupTransportDataType* defined in 6.2.5.7.2.

The *BrokerWriterGroupTransportDataType* is formally defined in Table 70.

Table 70 – BrokerWriterGroupTransportDataType structure

Name	Type	Description
BrokerWriterGroupTransportDataType	Structure	
queueName	String	Defined in 6.4.2.2.1.
resourceUri	String	Defined in 6.4.2.2.2.
authenticationProfileUri	String	Defined in 6.4.2.2.3.
requestedDeliveryGuarantee	BrokerTransportQualityOfService	Defined in 6.4.2.2.4.

6.4.2.3 Broker DataSetWriter

6.4.2.3.1 QueueName

The *QueueName* parameter with *DataType String* specifies the queue in the *Broker* that receives *NetworkMessages* sent by the *Publisher* for the DataSetWriter. This could be the name of a queue or topic defined in the *Broker*. This parameter is only valid if the *NetworkMessages* from the *WriterGroup* contain only one *DataSetMessage*.

If this *String* is not null, it overrides the *QueueName* of the *WriterGroup* transport settings.

6.4.2.3.2 ResourceUri

The *ResourceUri* property of *DataType String* allows the transport implementation to look up the configured key from the corresponding *KeyCredentialConfigurationType* instance defined in IEC 62541-12 to use for authenticating access to the specified queue.

If this *String* is not null, it overrides the *ResourceUri* of the *WriterGroup* authentication settings.

6.4.2.3.3 AuthenticationProfileUri

The parameter *AuthenticationProfileUri* of *DataType String* allows the selection of the authentication protocol used by the transport implementation for authenticating access to the specified queue.

If this *String* is not null, it overrides the *AuthenticationProfileUri* of the *WriterGroup* transport settings.

6.4.2.3.4 RequestedDeliveryGuarantee

The *RequestedDeliveryGuarantee* parameter with *DataType BrokerTransportQualityOfService* specifies the delivery guarantees that shall apply to all messages published by the *DataSetWriter*. The *DataType BrokerTransportQualityOfService* is defined in 6.4.2.2.5.

If the value is not *NotSpecified*_0, it overrides the *RequestedDeliveryGuarantee* of the *WriteGroup* transport settings.

If the selected delivery guarantee cannot be applied, the *DataSetWriter* shall set the state to *Error_3*.

6.4.2.3.5 MetaDataQueueName

For message mappings like UADP, the *Subscriber* needs access to the *DataSetMetaData* to process received *DataSetMessages*. The Publisher can provide the *DataSetMetaData* through a dedicated queue.

The parameter *MetaDataQueueName* with the *DataType String* specifies the *Broker* queue that receives messages with *DataSetMetaData* sent by the *Publisher* for this *DataSetWriter*. This could be the name of a queue or topic defined in the *Broker*.

6.4.2.3.6 MetaDataUpdateTime

Specifies the interval in milliseconds with DataType Duration at which the Publisher shall send the DataSetMetaData to the MetaDataQueueName. A value of 0 or any negative value shall be interpreted as infinite interval.

The broker transport shall publish all messages with an expiration time that is equal to or greater than this value.

If the update time is infinite, a broker transport shall attempt to negotiate message retention if possible. In this case the *DataSetMetaData* is only sent if the *ConfigurationVersion* of the corresponding *DataSetMetaData* is changed and *DataSetWriters* shall try to negotiate *AtLeastOnce*_2 or *ExactlyOnce*_4 delivery guarantees with the broker for any *DataSetMetaData* sent to ensure metadata is available to readers.

The *DataSetWriterProperties* settings apply also to *DataSetMetaData* sent to the queue named through the *MetaDataQueueName* parameter.

6.4.2.3.7 BrokerDataSetWriterTransportDataType structure

This *Structure DataType* is used to represent the Broker-specific transport mapping parameters for *DataSetWriters*. It is a subtype of the *DataSetWriterTransportDataType* defined in 6.2.3.5.2.

The *BrokerDataSetWriterTransportDataType* is formally defined in Table 71.

Table 71 – BrokerDataSetWriterTransportDataType structure

Name	Type	Description
BrokerDataSetWriterTransportDataType	Structure	
queueName	String	Defined in 6.4.2.3.1.
resourceUri	String	Defined in 6.4.2.3.2.
authenticationProfileUri	String	Defined in 6.4.2.3.3.
requestedDeliveryGuarantee	BrokerTransportQualityOfService	Defined in 6.4.2.3.4.
metaDataQueueName	String	Defined in 6.4.2.3.5.
metaDataUpdateTime	Duration	Defined in 6.4.2.3.6.

6.4.2.4 Broker DataSetReader

6.4.2.4.1 QueueName

The *QueueName* parameter with *DataType String* specifies the queue in the *Broker* where the *DataSetReader* can receive *NetworkMessages* with the DataSet of interest sent by the *Publisher*. This could be the name of a queue or topic defined in the *Broker*. This parameter is only valid if the *NetworkMessages* from the *WriterGroup* contain only one *DataSetMessage*.

6.4.2.4.2 ResourceUri

The *ResourceUri* property of *DataType String* allows the transport implementation to look up the configured key from the corresponding *KeyCredentialConfigurationType* instance defined in IEC 62541-12 to use for authenticating access to the specified queue.

If this *String* is not null, it overrides the *ResourceUri* of the *PubSubConnection* authentication settings.

6.4.2.4.3 AuthenticationProfileUri

The parameter *AuthenticationProfileUri* of *DataType String* allows the selection of the authentication protocol used by the transport implementation for authenticating access to the specified queue.

If this *String* is not null, it overrides the *AuthenticationProfileUri* of the *PubSubConnection* transport settings defined in 6.4.2.1.2.

6.4.2.4.4 RequestedDeliveryGuarantee

The *RequestedDeliveryGuarantee* parameter with *DataType BrokerTransportQualityOfService* specifies the delivery guarantees the *DataSetReader* negotiates with the broker for all messages received. The *DataType BrokerTransportQualityOfService* is defined in 6.4.2.2.5.

The value *NotSpecified_0* is not allowed on the *DataSetReader*. If the selected delivery guarantee cannot be applied, the *DataSetReader* shall set the state to *Error_3*.

6.4.2.4.5 MetaDataQueueName

The parameter *MetaDataQueueName* with the *DataType String* specifies the *Broker* queue that provides messages with *DataSetMetaData* sent by the *Publisher* for the *DataSet* of interest. This could be the name of a queue or topic defined in the *Broker*.

6.4.2.4.6 BrokerDataSetReaderTransportDataType structure

This *Structure DataType* is used to represent the Broker-specific transport mapping parameters for *DataSetWriters*. It is a subtype of the *DataSetReaderTransportDataType* defined in 6.2.8.13.2.

The *BrokerDataSetReaderTransportDataType* is formally defined in Table 72.

Table 72 – BrokerDataSetReaderTransportDataType structure

Name	Type	Description
BrokerDataSetReaderTransportDataType	Structure	
queueName	String	Defined in 6.4.2.4.1.
resourceUri	String	Defined in 6.4.2.4.2.
authenticationProfileUri	String	Defined in 6.4.2.4.3.
requestedDeliveryGuarantee	BrokerTransportQualityOfService	Defined in 6.4.2.4.4.
metaDataQueueName	String	Defined in 6.4.2.4.5.

7 PubSub mappings

7.1 General

Clause 7 specifies the mapping between the *PubSub* concepts described in Clause 5 and the *PubSub* configuration parameters defined in Clause 6 to concrete message mappings and transport protocol mappings that can be used to implement them.

DataSetMessage mappings, *NetworkMessage* mappings and transport protocol mappings are combined together to create transport profiles defined in IEC 62541-7. All *PubSub* applications shall implement at least one transport profile.

7.2 Message mappings

7.2.1 General

Message mappings specify a specific structure and encoding for *NetworkMessages*. Such a structure represents the payload for transport protocol mappings like UDP, MQTT or AMQP.

Different mappings are defined for different use cases.

7.2.2 UADP message mapping

7.2.2.1 General

The UADP message mapping uses optimized UA Binary encoding and provides message security for OPC UA PubSub. The available protocol mappings are defined in 7.3.

The UADP message mapping defines different optional header fields, variations of field settings and different message types and data encodings.

A *Publisher* shall support all variations it allows through configuration. The required set of features is defined through profiles in IEC 62541-7.

A *Subscriber* shall be able to process all possible *NetworkMessages* and shall be able to skip information the *Subscriber* is not interested in. The *Subscriber* may not support all security policies. The capabilities related to processing different *DataSet* encodings is defined in IEC 62541-7.

7.2.2.2 NetworkMessage

7.2.2.2.1 General

The UADP *NetworkMessage* header and other parts of the *NetworkMessage* are shown in Figure 27.

When using security, the payload and the *Padding* field are encrypted and after that, the whole *NetworkMessage* is signed if signing and encryption is active. The *NetworkMessage* shall be signed without being encrypted if only the signing is active.

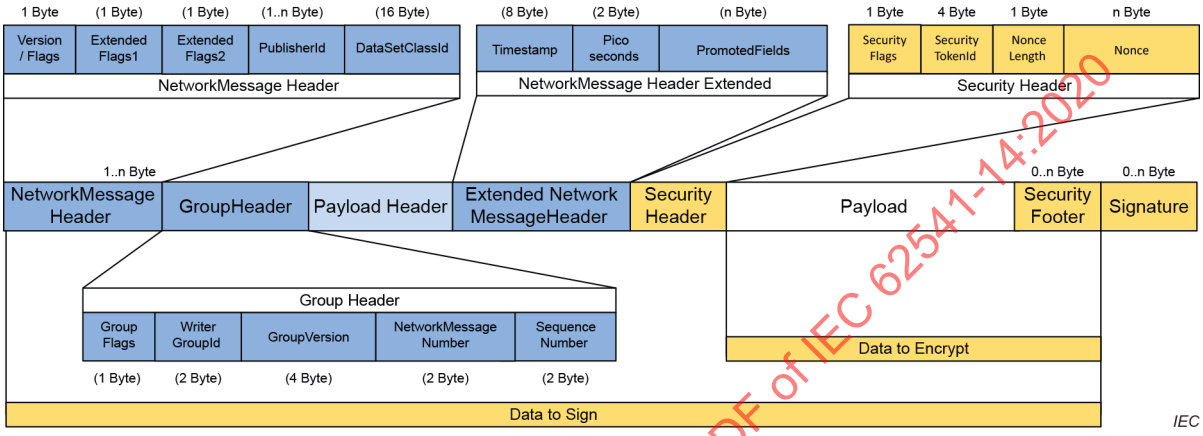


Figure 27 – UADP NetworkMessage

7.2.2.2.2 NetworkMessage layout

The encoding of the UADP *NetworkMessage* is specified in Table 73.

The *NetworkMessageContentMask* setting of the *Publisher* controls the flags in the fields *UADPFlags* and *ExtendedFlags1*. The *SecurityMode* setting of the *Publisher* controls the security enabled flag of the *ExtendedFlags1*. The setting of the flags shall not change until the configuration of the *Publisher* is changed.

Table 73 – UADP NetworkMessage

Name	Type	Description
UADPVersion	Bit[0-3]	Bit range 0-3: Version of the UADP NetworkMessage. The *UADPVersion* for this specification version is 1.
UADPFlags	Bit[4-7]	Bit 4: *PublisherId* enabled If the PublisherId is enabled, the type of PublisherId is indicated in the ExtendedFlags1 field. Bit 5: *GroupHeader* enabled Bit 6: *PayloadHeader* enabled Bit 7: *ExtendedFlags1* enabled The bit shall be false, if ExtendedFlags1 is 0.

Name	Type	Description
ExtendedFlags1	Byte	The *ExtendedFlags1* shall be omitted if bit 7 of the *UADPFlags* is false. If the field is omitted, the *Subscriber* shall handle the related bits as false. Bit range 0-2: *PublisherId* Type 000 The *PublisherId* is of *DataType Byte* This is the default value if *ExtendedFlags1* is omitted 001 The *PublisherId* is of *DataType UInt16* 010 The *PublisherId* is of *DataType UInt32* 011 The *PublisherId* is of *DataType UInt64* 100 The *PublisherId* is of *DataType String* 101 Reserved 11x Reserved 111 Reserved Bit 3: *DataSetClassId* enabled Bit 4: Security enabled If the *SecurityMode* is SIGN_1 or SIGNANDENCRYPT_2, this flag is set, message security is enabled and the *SecurityHeader* is contained in the NetworkMessage header. If this flag is not set, the *SecurityHeader* is omitted. Bit 5: *Timestamp* enabled Bit 6: PicoSeconds enabled Bit 7: *ExtendedFlags2* enabled The bit shall be false, if ExtendedFlags2 is 0.
ExtendedFlags2	Byte	The *ExtendedFlags2* shall be omitted if bit 7 of the *ExtendedFlags1* is false. If the field is omitted, the *Subscriber* shall handle the related bits as false. Bit 0: Chunk message defined in in 7.2.2.2.4. Bit 1: *PromotedFields* enabled Promoted fields can only be sent if the *NetworkMessage* contains only one *DataSetMessage*. Bit range 2-4: UADP *NetworkMessage* type 000 *NetworkMessage* with *DataSetMessage* payload defined in 7.2.2.2.4. If the *ExtendedFlags2* field is not provided, this is the default *NetworkMessage* type. 001 *NetworkMessage* with discovery request payload defined in 7.2.2.3.4. 010 *NetworkMessage* with discovery response payload defined in 7.2.2.4.2. 011 Reserved 1xx Reserved Bit 5: Reserved Bit 6: Reserved Bit 7: Reserved for further extended flag fields
PublisherId	Byte[*]	The *PublisherId* shall be omitted if bit 4 of the *UADPFlags* is false. The Id of the *Publisher* that sent the data. Valid *DataTypes* are *UInteger* and *String*. The DataType is indicated by bits 0-2 of the *ExtendedFlags1*. A *Subscriber* can skip *NetworkMessages* from *Publishers* it does not expect *NetworkMessages* from.
DataSetClassId	Guid	The *DataSetClassId* associated with the *DataSets* in the NetworkMessage. All DataSetMessages in the NetworkMessage shall have the same DataSetClassId. The *DataSetClassId* shall be omitted if bit 3 of the *ExtendedFlags1* is false.

Name	Type	Description
GroupHeader		The group header shall be omitted if bit 5 of the *UADPFlags* is false.
GroupFlags	Byte	Bit 0: Writer*GroupId* enabled Bit 1: *GroupVersion* enabled Bit 2: *NetworkMessageNumber* enabled Bit 3: *SequenceNumber* enabled Bits 4-6: Reserved Bit 7: Reserved for further extended flag fields
WriterGroupId	UInt16	Unique id for the *WriterGroup* in the Publisher. A *Subscriber* can skip *NetworkMessages* from *WriterGroups* it does not expect *NetworkMessages* from. This field shall be omitted if bit 0 of the *GroupFlags* is false.
GroupVersion	VersionTime	Version of the header and payload layout configuration of the *NetworkMessages* sent for the group. This field shall be omitted if bit 1 of the *GroupFlags* is false.
NetworkMessage Number	UInt16	Unique number of a *NetworkMessage* across the combination of *PublisherId* and *WriterGroupId* within one *PublishingInterval*. The number is needed if the *DataSetMessages* for one group are split into more than one *NetworkMessage* in a *PublishingInterval*. The value 0 is invalid. This field shall be omitted if bit 2 of the *GroupFlags* is false.
SequenceNumber	UInt16	Sequence number for the *NetworkMessage*. This field shall be omitted if bit 3 of the *GroupFlags* is false.
PayloadHeader	Byte [*]	The payload header depends on the UADP *NetworkMessage* Type flags defined in the *ExtendedFlags2* bit range 2-4. The default is *DataSetMessage* if the *ExtendedFlags2* field is not enabled. The PayloadHeader shall be omitted if bit 6 of the *UADPFlags* is false. The *PayloadHeader* is not contained in the payload but it is contained in the unencrypted *NetworkMessage* header since it contains information necessary to filter *DataSetMessages* on the *Subscriber* side.
Timestamp	DateTime	The time the NetworkMessage was created. The *Timestamp* shall be omitted if bit 5 of *ExtendedFlags1* is false. The *PublishingInterval*, the *SamplingOffset* the *PublishingOffset* and the *Timestamp* and *PicoSeconds* in the *NetworkMessage* header shall use the same time base.
PicoSeconds	UInt16	Specifies the number of 10 picoseconds ($1,0 \times 10^{-11}$ seconds) intervals which shall be added to the *Timestamp*. The *PicoSeconds* shall be omitted if bit 6 of *ExtendedFlags1* is false.
PromotedFields		The *PromotedFields* shall be omitted if bit 1 of the *ExtendedFlags2* is false. If the *PromotedFields* are provided, the number of *DataSetMessages* in the *Network* Message shall be one.
Size	UInt16	Total size in *Bytes* of the *Fields* contained in the *PromotedFields*.
Fields	BaseDataType[ ]	Array of promoted fields. The size, order and *DataTypes* of the fields depend on the settings in the *FieldMetaData* of the *DataSetMetaData* associated with the *DataSetMessage* contained in the *NetworkMessage*.

Name	Type	Description
SecurityHeader		The security header shall be omitted if bit 4 of the *ExtendedFlags1* is false.
SecurityFlags	Byte	Bit 0: *NetworkMessage* Signed Bit 1: *NetworkMessage* Encrypted Bit 2: SecurityFooter enabled Bit 3: Force key reset This bit is set if all keys will be made invalid. It is set until the new key is used. The publisher needs to give subscribers a reasonable time to request new keys. The minimum time is five times the *KeepAliveTime* configured for the corresponding PubSub group. This flag is typically set if all keys are invalidated to exclude Subscribers, who no longer have access to the keys. Bit range 4-7: Reserved
SecurityTokenId	IntegerId	The ID of the security token that identifies the security key in a *SecurityGroup*. The relation to the *SecurityGroup* is done through *DataSetWriterIds* contained in the *NetworkMessage*.
NonceLength	Byte	The length of the Nonce used to initialize the encryption algorithm.
MessageNonce	Byte [NonceLength]	A number used exactly once for a given security key. For a given security key a unique nonce shall be generated for every *NetworkMessage*. The rules for constructing the *MessageNonce* are defined for the UADP Message Security in 7.2.2.2.3.
SecurityFooterSize	UInt16	The size of the *SecurityFooter*. The security footer size shall be omitted if bit 2 of the *SecurityFlags* is false.
Payload	Byte [*]	The payload depends on the UADP *NetworkMessage* Type flags defined in the *ExtendedFlags2* bit range 2-4.
SecurityFooter	Byte [*]	Optional security footer shall be omitted if bit 2 of the *SecurityFlags* is false. The content of the security footer is defined by the *SecurityPolicy*.
Signature	Byte [*]	The signature of the *NetworkMessage*.

7.2.2.2.3 UADP message security

7.2.2.2.3.1 General

The algorithm and nonce length used for the UADP *NetworkMessage* security depend on the selected *SecurityPolicy*. They are defined by *SymmetricPubSubEncryptionAlgorithm* and *SymmetricPubSubNonceLength*.

The keys used to encrypt and sign messages are returned from the *GetSecurityKeys* method (see 8.4). This *Method* returns a sequence of random data with a length that depends on the *SecurityPolicyUri*, which is also returned by the *Method*. The layout of the random data is defined in Table 74.

Table 74 – Layout of the key data for UADP message security

Name	Type	Description
SigningKey	Byte [SymmetricSignatureAlgorithm Key Length]	Signing key part of the key data returned from *GetSecurityKeys*. The SymmetricSignatureAlgorithm is defined in the SecurityPolicy.
EncryptingKey	Byte [SymmetricEncryptionAlgorithm KeyLength]	Encryption key part of the key data returned from *GetSecurityKeys*. The SymmetricEncryptionAlgorithm is defined in the SecurityPolicy.
KeyNonce	Byte [SymmetricPubSubNonceLength]	Nonce part of the key data returned from *GetSecurityKeys*.

7.2.2.2.3.2 AES-CTR

The layout of the MessageNonce for AES-CTR mode is defined in Table 75.

Table 75 – Layout of the MessageNonce for AES-CTR

Name	Type	Description
Random	Byte [4]	The random part of the MessageNonce. This number does not need to be a cryptographically random number, it can be pseudo-random.
SequenceNumber	UInt32	A strictly monotonically increasing sequence number assigned by the publisher to each *NetworkMessage* sent for a *SecurityTokenId* and *PublisherId* combination. The sequence number is reset to 1 after the key and *SecurityTokenId* are updated in the *Publisher*. A receiver should ignore older *NetworkMessages* than the last sequence processed if it does not handle reordering of *NetworkMessages*. Receivers need to be aware of sequence numbers roll over (change from 4 294 967 295 to 0). To determine whether a received *NetworkMessage* is newer than the last processed *NetworkMessage* the following formula shall be used: (4 294 967 295 + received sequence number – last processed sequence number) modulo 4 294 967 296. Results below 1 073 741 824 indicate that the received *NetworkMessage* is newer than the last processed *NetworkMessage* and the received *NetworkMessage* is processed. Results above 3 221 225 472 indicate that the received message is older (or same) than the last processed *NetworkMessage* and the received *NetworkMessage* should be ignored if reordering of *NetworkMessages* is not necessary. Other results are invalid and the *NetworkMessages* shall be ignored. The key lifetime should be selected in a way that a new key is used before a rollover for the *SequenceNumber* happens. Subscribers shall reset the records they keep for sequence numbers if they do not receive messages for two times the keep alive time to deal with Publishers that are out of service and were not able to continue from the last used *SequenceNumber*.

The message encryption and decryption with AES-CTR mode uses a secret and a counter block. The secret is the *EncryptingKey* from the key data defined in Table 74. The layout and content of the counter block is defined in Table 76.

Table 76 – Layout of the counter block for UADP message security

Name	Type	Description
KeyNonce	Byte [4]	The KeyNonce portion of the key data returned from *GetSecurityKeys*.
MessageNonce	Byte [8]	The first 8 bytes of the *Nonce* in the *SecurityHeader* of the *NetworkMessage*. For AES-CTR mode the length of the *SecurityHeader Nonce* shall be 8 Bytes.
BlockCounter	Byte [4]	The counter for each encrypted block of the NetworkMessage. The counter is a 32-bit big endian integer (the opposite of the normal encoding for UInt32 values in OPC UA. This convention comes from the AES-CTR RFC). The counter starts with 0 at the first block. The counter is incremented by 1 for each block.

AES-CTR mode takes the counter block and encrypts it using the encrypting key. The encrypted key stream is then logically XORed with the data to encrypt or decrypt. The process is repeated for each block in plain text. No padding is added to the end of the plain text. AES-CTR does not change the size of the plain text data and can be applied directly to a memory buffer containing the message.

The signature is calculated on the entire *NetworkMessage* including any encrypted data. The signature algorithm is specified by the *SecurityPolicyUri* in IEC 62541-7.

When a Subscriber receives a *NetworkMessage*, it shall verify the signature first. If verification fails, it drops the *NetworkMessage*.

Other *SecurityPolicy* may specify different key lengths or cryptography algorithms.

7.2.2.2.4 UADP Chunk NetworkMessage

If a *NetworkMessage* payload like a *DataSetMessage* or a discovery response message needs to be split across multiple *NetworkMessages*, the chunks are sent with the payload header defined in Table 77 and the payload defined in Table 78. A chunk *NetworkMessage* can only contain chunked payload of one *DataSetMessage*.

Table 77 – Chunked NetworkMessage payload header

Name	Type	Description
DataSetWriterId	UInt16	DataSetWriterId contained in the *NetworkMessage*. The *DataSetWriterId* identifies the *PublishedDataSet* and the *DataSetWriter* responsible for sending Messages for the *DataSet*. A *Subscriber* can skip *DataSetMessages* from *DataSetWriters* it does not expect *DataSetMessages* from. The *DataSetWriterId* shall be set to 0 for discovery response messages.

Table 78 – Chunked NetworkMessage payload fields

Name	Type	Description
MessageSequenceNumber	UInt16	Sequence number of the payload as defined for the *NetworkMessage* type like *DataSetMessageSequenceNumber* in a *DataSetMessage*. *NetworkMessages* may be received out of order. In this case, a chunk for the next payload can be received before the last chunk of the previous payload was received. If the next sequence number is received by a *Subscriber* that can handle only one payload, the chunks of the previous payload are skipped if they are not completely received yet.
ChunkOffset	UInt32	The byte offset position of the chunk in the complete *NetworkMessage* payload. The last chunk is detected if *ChunkOffset* plus the size of the current chunk equals *TotalSize*. All chunks, except for the last one, shall have the same size. The size of all chunks other than the last one can be used to calculate the number of expected chunks. The reassembled *NetworkMessage* payload can be processed after all chunks are received. Depending on the transport protocol mapping, the chunks may be received out of order and the last chunk may be received before all other chunks are received.
TotalSize	UInt32	Total size of the *NetworkMessage* payload in bytes.
ChunkData	ByteString	The pieces of the original *DataSetMessage*, are copied into the chunk until the maximum size allowed for a single *NetworkMessage* is reached minus space for the signature. The data copied into next chunk starts with the byte after the last byte copied into current chunk. A *DataSetMessage* is completely received when all chunks are received and the *DataSetMessage* can be processed completely.

7.2.2.3 DataSetMessage

7.2.2.3.1 General

The UADP *DataSet* payload header and other parts of the *NetworkMessage* are shown in Figure 28.

Different types of *DataSetMessage* can be combined in one *NetworkMessage*.

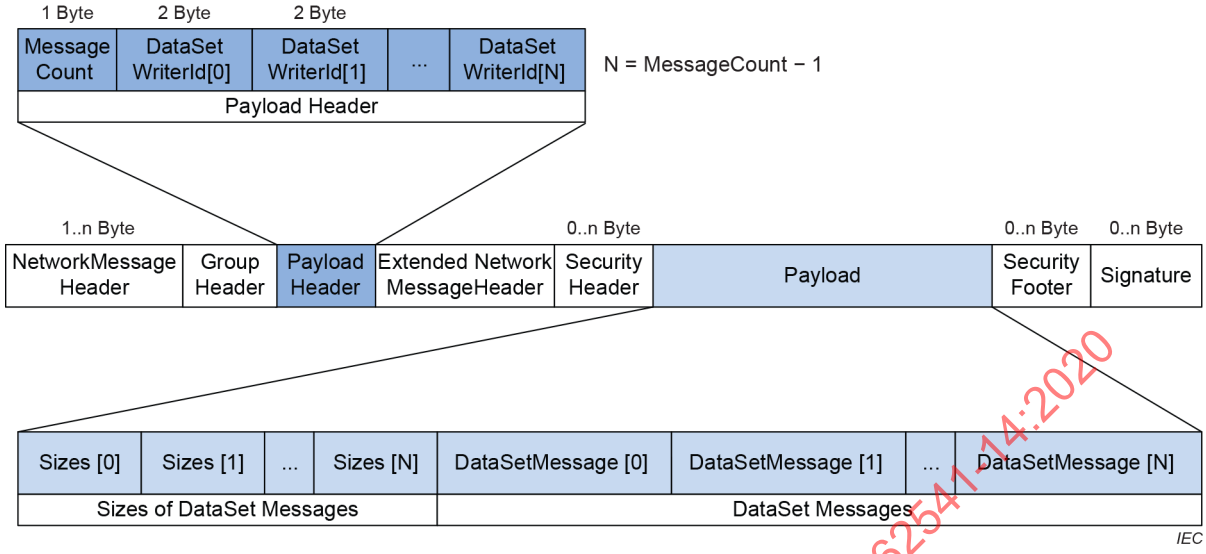


Figure 28 – UADP DataSet payload

7.2.2.3.2 DataSet payload header

The encoding of the UADP *DataSet* payload header is specified in Table 79. The payload header is unencrypted. This header shall be omitted if bit 6 of the *UADPFlags* is false.

Table 79 – UADP DataSet payload header

Name	Type	Description
Count	Byte	Number of *DataSetMessages* contained in the *NetworkMessage*. The *NetworkMessage* shall contain at least one *DataSetMessage* if the *NetworkMessage* type is *DataSetMessage* payload.
DataSetWriterIds	UInt16 [Count]	List of *DataSetWriterIds* contained in the *NetworkMessage*. The size of the list is defined by the *Count*. The *DataSetWriterId* identifies the *PublishedDataSet* and the *DataSetWriter* responsible for sending Messages for the *DataSet*. A *Subscriber* can skip *DataSetMessages* from *DataSetWriters* it does not expect *DataSetMessages* from.

7.2.2.3.3 DataSet payload

The *DataSet* payload is defined in Table 80. The payload is encrypted.

Table 80 – UADP DataSet payload

Name	Type	Description
Sizes	UInt16 [Count]	List of byte sizes of the *DataSetMessages*. The size of the list is defined by the *Count* in the *DataSet* payload header. If the payload size exceeds 65 535, the *DataSetMessages* shall be allocated to separate *NetworkMessages*. If a single *DataSetMessage* exceeds the payload size it shall be split into *Chunk NetworkMessages*. This field shall be omitted if count is one or if bit 6 of the *UADPFlags* is false.
DataSetMessages	DataSetMessage [Count]	*DataSetMessages* contained in the *NetworkMessage*. The size of the list is defined by the *Count* in the *DataSet* payload header. The type of encoding used for the *DataSetMessages* is defined by the *DataSetWriter*. The encodings for the *DataSetMessage* are defined in 7.2.2.3.4.

7.2.2.3.4 DataSetMessage header

The *DataSetMessage* header structure and the relation to other parts in a *NetworkMessage* is shown in Figure 29.

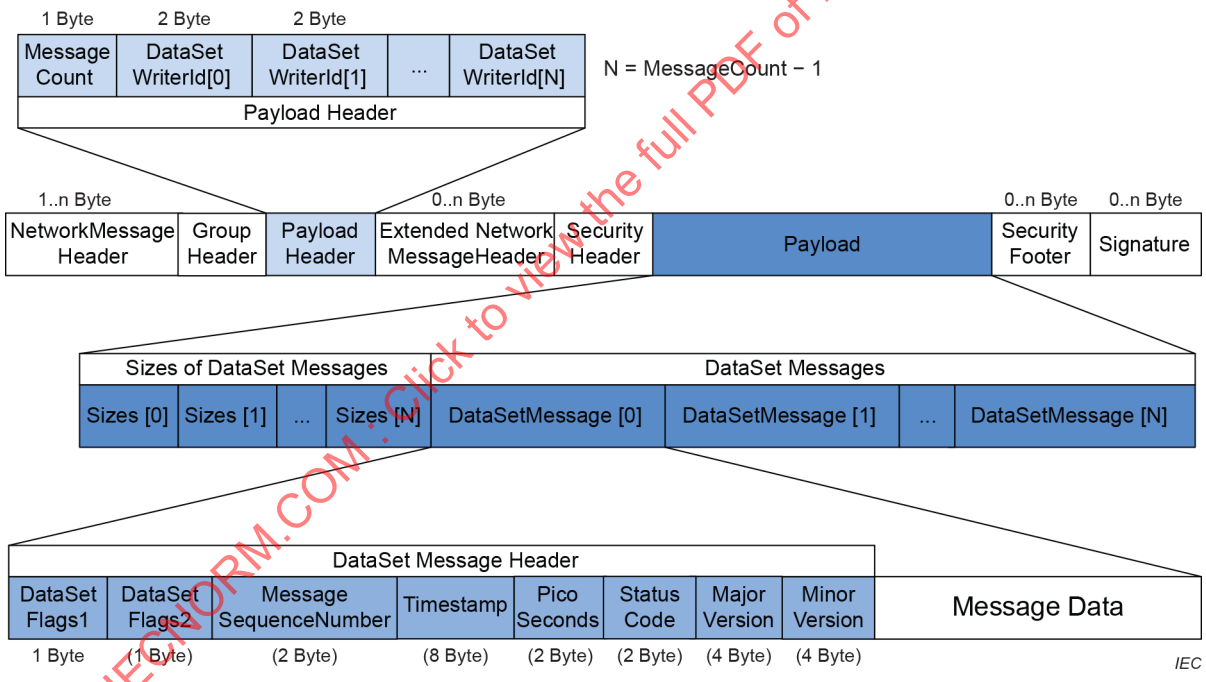


Figure 29 – DataSetMessage header structure

The encoding of the *DataSetMessage* header structure is specified in Table 81.

The *DataSetFieldContentMask* and the *DataSetMessageContentMask* settings of the *DataSetWriter* control the flags in the fields *DataSetFlags1* and *DataSetFlags2*. The setting of the flags shall not change until the configuration of the *DataSetWriter* is changed.

Table 81 – DataSetMessage header structure

Name	Type	Description
DataSetFlags1	Byte	Bit 0: DataSetMessage is valid. If this bit is set to false, the rest of this DataSetMessage is considered invalid, and shall not be processed by the *Subscriber*. Bit range 1-2: Field Encoding 00 The *DataSet* fields are encoded as Variant The *Variant* can contain a *StatusCode* instead of the expected *DataType* if the status of the field is Bad. The *Variant* can contain a DataValue with the value and the statusCode if the status of the field is Uncertain. 01 RawData Field Encoding The RawData field encoding is defined in 7.2.2.3.9. 10 DataValue Field Encoding The *DataSet* fields are encoded as DataValue. This option is set if the *DataSet* is configured to send more than the Value. 11 Reserved Bit 3: *DataSetMessageSequenceNumber* enabled Bit 4: *Status* enabled Bit 5: *ConfigurationVersionMajorVersion* enabled Bit 6: *ConfigurationVersionMinorVersion* enabled Bit 7: *DataSetFlags2* enabled The bit shall be false, if DataSetFlags2 is 0.
DataSetFlags2	Byte	The *DataSetFlags2* shall be omitted if bit 7 of the *DataSetFlags1* is false. If the field is omitted, the *Subscriber* shall handle the related bits as false. Bit range 0-3: UADP *DataSetMessage* type 0000 Data Key Frame (see 7.2.2.3.5) If the *DataSetFlags2* field is not provided, this is the default *DataSetMessage* type. 0001 Data Delta Frame (see 7.2.2.3.6) 0010 Event (see 7.2.2.3.7) 0011 Keep Alive (see 7.2.2.3.8) 01xx Reserved 1xxx Reserved Bit 4: *Timestamp* enabled Bit 5: *PicoSeconds* included in the *DataSetMessage* header Bit 6: Reserved Bit 7: Reserved for further extended flag fields
DataSetMessage SequenceNumber	UInt16	A strictly monotonically increasing sequence number assigned by the publisher to each *DataSetMessage* sent. A receiver should ignore older *DataSetMessage* than the last sequence processed if it does not handle reordering of *DataSetMessages*. Receivers need to be aware of sequence numbers roll over (change from 65 535 to 0). To determine whether a received *DataSetMessage* is newer than the last processed *DataSetMessage* the following formula shall be used: (65 535 + received sequence number – last processed sequence number) modulo 65 536 Results below 16 384 indicate that the received *DataSetMessage* is newer than the last processed *DataSetMessage* and the received *DataSetMessage* is processed. Results above 49 162 indicate that the received message is older than (or same as) the last processed *DataSetMessage* and the received *DataSetMessage* should be ignored if reordering of *DataSetMessages* is not necessary. Other results are invalid and the *DataSetMessage* shall be ignored. The field shall be omitted if Bit 3 of *DataSetFlags1* is false.

Name	Type	Description
Timestamp	UtcTime	The time the Data was collected. The *Timestamp* shall be omitted if Bit 4 of *DataSetFlags1* is false.
PicoSeconds	UInt16	Specifies the number of 10 picoseconds ($1,0 \times 10^{-11}$ seconds) intervals which shall be added to the *Timestamp*. The field shall be omitted if Bit 5 of *DataSetFlags2* is false.
Status	UInt16	The overall status of the DataSet. This is the high order 16 bits of the *StatusCode DataType* representing the numeric value of the *Severity* and *SubCode* of the *StatusCode DataType*. The field shall be omitted if Bit 4 of *DataSetFlags1* is false.
ConfigurationVersion MajorVersion	VersionTime	The major version of the configuration version of the DataSet used as consistency check with the *DataSetMetaData* available on the *Subscriber* side. The field shall be omitted if Bit 5 of *DataSetFlags1* is false.
ConfigurationVersion MinorVersion	VersionTime	The minor version of the configuration version of the DataSet used as consistency check with the *DataSetMetaData* available on the *Subscriber* side. The field shall be omitted if Bit 6 of *DataSetFlags1* is false.

7.2.2.3.5 Data Key Frame DataSetMessage

The data key frame *DataSetMessage* data and related headers are shown in Figure 30.

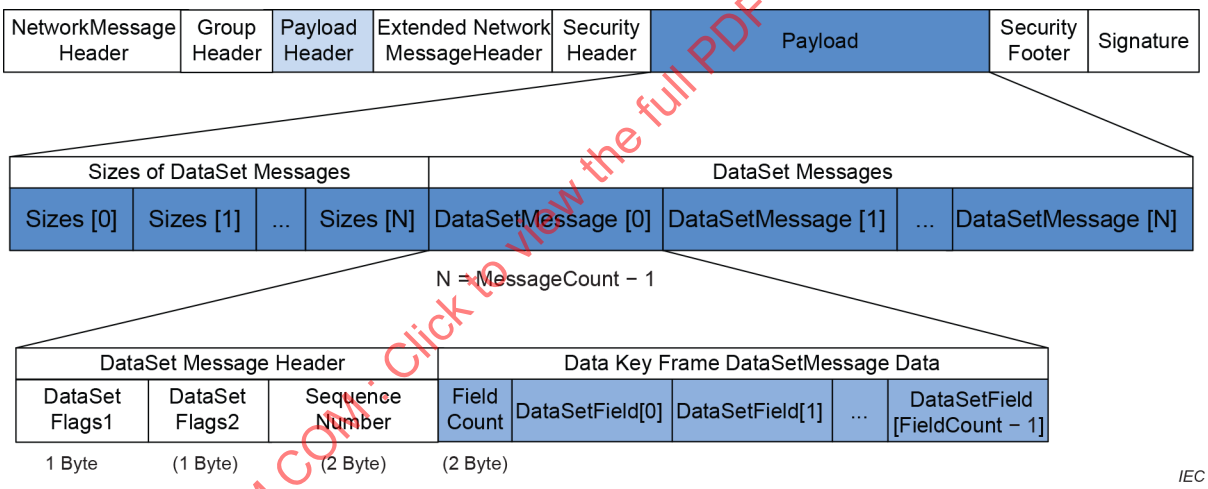


Figure 30 – Data Key Frame DataSetMessage data

The encoding of the data key *DataSetMessage* structure is specified in Table 82.

Table 82 – Data Key Frame DataSetMessage structure

Name	Type	Description
FieldCount	UInt16	Number of fields of the *DataSet* contained in the *DataSetMessage*. The *FieldCount* shall be omitted if *RawData* field encoding is set in the *EncodingFlags* defined in 7.2.2.3.4.
DataSetFields	BaseDataType[]	The field values of the DataSet. The field encoding depends on the *EncodingFlags* of the *DataSetMessage* Header defined in 7.2.2.3.4. The default encoding is *Variant* if bit 1 and 2 are not set.

7.2.2.3.6 Data Delta Frame DataSetMessage

The data delta frame *DataSetMessage* data and the related headers are shown in Figure 31.

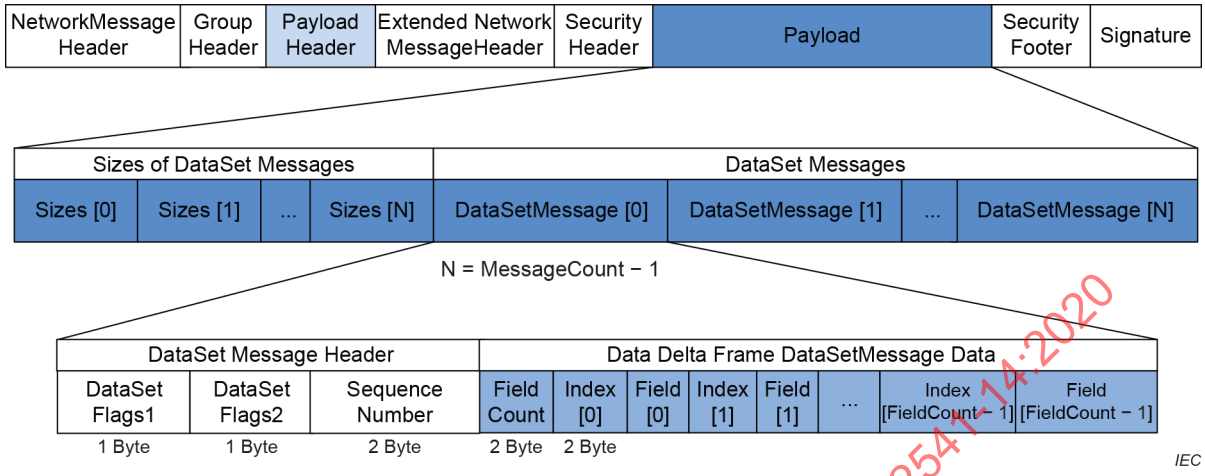


Figure 31 – Data Delta Frame DataSetMessage

The information for a single value in delta frame messages is larger because of the additional index necessary for sending just changed data. The *Publisher* shall send a key frame message if the delta frame message is larger than a key frame message.

The encoding of the data delta frame *DataSetMessage* structure is specified in Table 83.

Table 83 – Data Delta Frame DataSetMessage structure

Name	Type	Description
FieldCount	UInt16	Number of fields of the DataSet contained in the *DataSetMessage*.
DeltaFrameFields	Structure[]	The subset of field values of the DataSet contained in the delta frame.
FieldIndex	UInt16	The index of the Field in the DataSet. The index is based on the field position in the *DataSetMetaData* with the configuration version defined in the *ConfigurationVersion* field.
FieldValue	BaseDataType	The field values of the DataSet. The field encoding depends on the EncodingFlags of the *DataSetMessage* Header defined in 7.2.2.3.4. The default encoding is Variant if bit 1 and 2 are not set.

7.2.2.3.7 Event *DataSetMessage*

The *Event DataSetMessage* data and the related headers are shown in Figure 32.

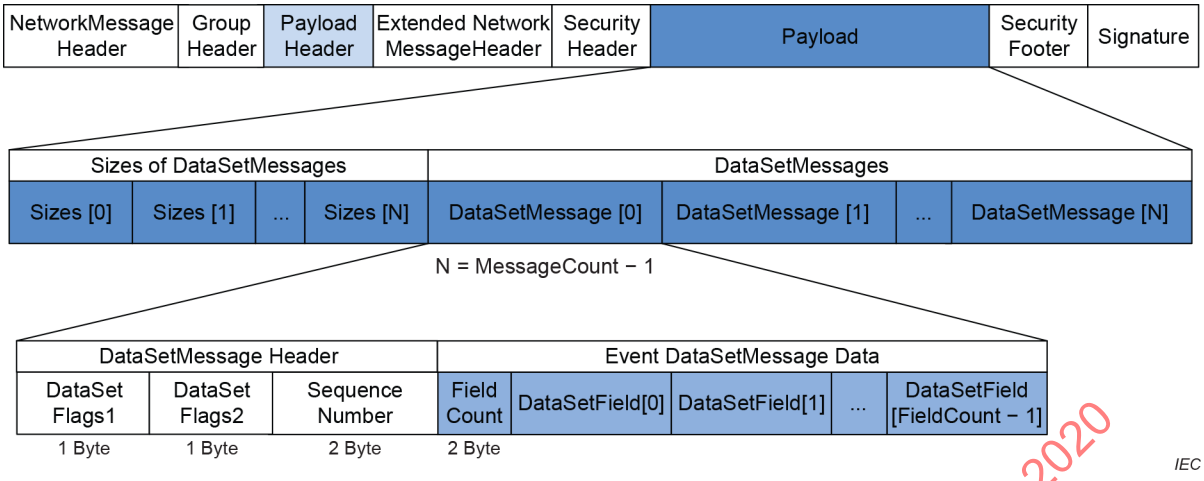


Figure 32 – Event DataSetMessage

The encoding of the *Event DataSetMessage* structure is specified in Table 84.

Table 84 – Event DataSetMessage structure

Name	Type	Description
FieldCount	UInt16	Number of fields of the *DataSet* contained in the *DataSetMessage*.
DataSetFields	BaseDataType[]	The field values of the DataSet. The fields of Event *DataSetMessages* shall be encoded as Variant. The Field Encoding *DataSetFlags1* of the *DataSetMessage* header (bit 1 and 2) defined in 7.2.2.3.4 shall be set to false.

7.2.2.3.8 KeepAlive message

The keep alive message does not add any additional fields. The message and the related headers are shown in Figure 33.

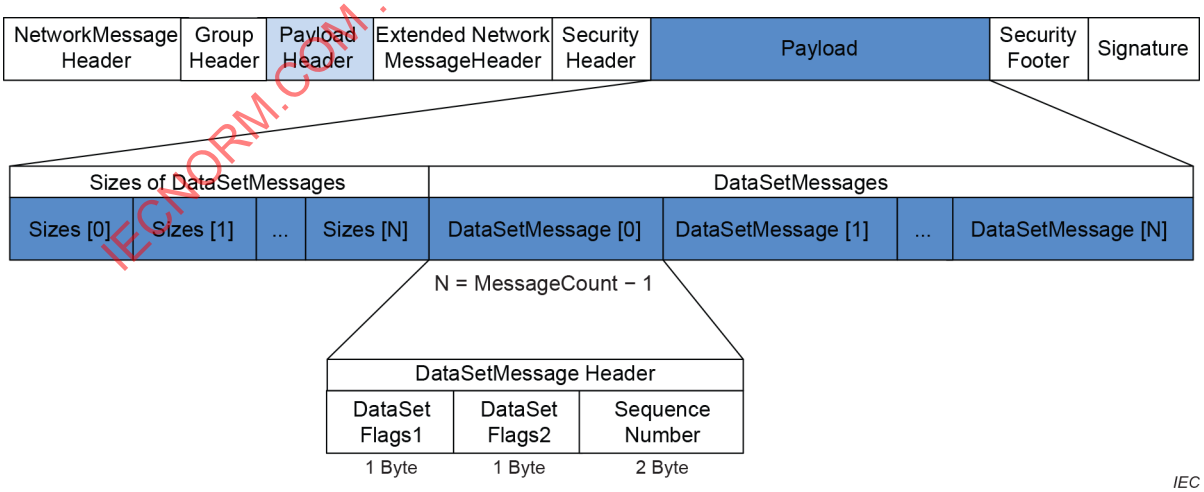


Figure 33 – KeepAlive message

The sequence number contains the next expected sequence number for the *DataSetWriter*.

7.2.2.3.9 RawData Field Encoding

The encoding of the *DataSetMessage* fields 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*.

A *DataSet* field is encoded in the *DataType* and *ValueRank* specified in the *DataSetMetaData* for the *DataSet*. The following special handling shall be applied.

- If the *DataType* of a *DataSet* field or a *Structure* field is *String* or *ByteString* and the actual size is smaller than the maximum possible size indicated by the dimensions, the field shall be padded with bytes with value zero.
- If the *ValueRank* is *OneDimension* (1) or n > 1 and the actual size is smaller than the maximum possible size indicated by the dimensions, the field shall be padded with bytes with value zero.

The following restrictions apply to the *RawData* field encoding.

- Fields shall have dimensions defined if the *DataType* is *String* or *ByteString* or if it is an array. This includes *Structure* fields with such *DataTypes* or *ValueRank*.
- *DataSet* fields and *Structure* fields shall have a concrete valueRank with values −1 or n > 0.
- *DataSet* fields and *Structure* fields shall not have the *BuiltInType NodeId*, *ExpandedNodeId*, *QualifiedName*, *LocalizedText*, *XmlElement* or *DataValue*.
- *Structure* fields shall not have the *builtInType ExtensionObject* or *Variant*.

The *DataSetMessage* valid bit 0 in *DataSetFlags1* shall be set to false if the fields do not fulfil these requirements at the time the *DataSetMessage* is created.

7.2.2.4 Discovery messages

7.2.2.4.1 UADP Discovery Request NetworkMessage

7.2.2.4.1.1 General

The NetworkMessage flags used with the discovery request messages shall use the following bit values.

- *UADPFlags* bits 5 and 6 shall be false, bits 4 and 7 shall be true.
- *ExtendedFlags1* bits 3, 5 and 6 shall be false, bits 4 and 7 shall be true.
- *ExtendedFlags2* bit 2 shall be true, all other bits shall be false.

The setting of the flags ensures a known value for the first five fields in the *NetworkMessage* on the *Publisher* as receiver. The actual security settings for the *NetworkMessage* are indicated by the *SecurityHeader*.

7.2.2.4.1.2 Traffic reduction

A variety of rules are used to reduce the amount of traffic on the network in the case of multicast or broadcast communication.

A *Subscriber* should cache configuration information for *PublisherId* and *DataSetWriterIds* of interest.

If a *Subscriber* requires information from *Publishers* after a startup or version change detection, discovery requests shall be randomly delayed in the range of 100 ms to 500 ms. The request shall be skipped if the information is already received during this time or another *Subscriber* sent already a request and the response to this request is used.

Discovery requests for different *DataSetWriters* in one *WriterGroup* shall be aggregated into one discovery response.

A *Publisher* shall delay subsequent responses for a combination of request type and identifier like the *DataSetWriterId* for at least 500 ms. Duplicate requests that have not yet been responded to shall be discarded by the *Publisher*.

A *Subscriber* shall wait for a response at least 500 ms. As long as not all responses are received, the Subscriber requests the missing information. It shall double the time period between follwing requests until all needed response are received or denied.

7.2.2.4.1.3 Discovery request header

The encoding of the discovery request header structure is specified in Table 85.

Table 85 – Discovery request header structure

Name	Type	Description
RequestType	Byte	The following types of discovery request messages are defined. 0 Reserved 1 *Publisher* information request message (see 7.2.2.4.1.4)

7.2.2.4.1.4 Publisher information request message

The encoding of the *Publisher* information request message structure is specified in Table 86.

Table 86 – Publisher information request message structure

Name	Type	Description
InformationType	Byte	The following types of Publisher information requests are defined. 0 Reserved 1 *Publisher Server Endpoints* 2 *DataSetMetaData* 3 *DataSetWriter* configuration
DataSetWriterIds	UInt16[]	List of *DataSetWriterIds* the information is requested for. If the request is not related to DataSet, the array shall be null.

7.2.2.4.2 UADP Discovery response NetworkMessage

7.2.2.4.2.1 General

The NetworkMessage flags used with the discovery response messages shall use the following bit values.

- *UADPFlags* bits 5 and 6 shall be false, bits 4 and 7 shall be true.
- *ExtendedFlags1* bits 3, 5 and 6 shall be false, bit 7 shall be true.
- *ExtendedFlags2* bit 1 shall be false and the *NetworkMessage* type shall be discovery response.

The setting of the flags ensures a known value for the first five fields in the *NetworkMessage* for *Publishers* expected by the *Subscriber*. The actual security settings for the *NetworkMessage* are indicated by the *SecurityHeader*.

7.2.2.4.2.2 Discovery response Header

The encoding of the discovery response header structure is specified in Table 87.

Table 87 – Discovery response header structure

Name	Type	Description
ResponseType	Byte	The following types of discovery response messages are defined. 0 Reserved 1 Publisher Endpoint message (see 7.2.2.4.2.3) 2 DataSet Metadata message (see 7.2.2.4.2.4) 3 DataSetWriter configuration message (see 7.2.2.4.2.5)
SequenceNumber	UInt16	A strictly monotonically increasing sequence number assigned to each discovery response sent in the scope of a *PublisherId*.

7.2.2.4.2.3 Publisher Endpoints message

The encoding of the available *Endpoints* of a *Publisher* is specified in Table 88.

Table 88 – Publisher Endpoints message structure

Name	Type	Description
Endpoints	EndpointDescription[]	The OPC UA *Server Endpoints* of the *Publisher*. The *EndpointDescription* is defined in IEC 62541-4.
statusCode	StatusCode	Status code indicating the capability of the *Publisher* to provide *Endpoints*.

7.2.2.4.2.4 DataSetMetaData message

The encoding of the *DataSet* metadata message structure is specified in Table 89. It contains the current layout and *DataSetMetaData* for the *DataSet*.

The *ConfigurationVersion* in the *DataSetMessage* header shall match the *ConfigurationVersion* in the *DataSetMetaData*.

The *Publisher* shall send this message without a corresponding discovery request if the *DataSetMetaData* changed for the *DataSet*.

Table 89 – DataSetMetaData message structure

Name	Type	Description
DataSetWriterId	UInt16	*DataSetWriterId* of the *DataSet* described with the *MetaData*.
MetaData	DataSetMetaDataType	The current *DataSet* metadata for the *DataSet* related to the *DataSetWriterId*. The *DataSetMetaDataType* is defined in 6.2.2.1.2.
statusCode	StatusCode	Status code indicating the capability of the *Publisher* to provide *MetaData* for the DataSetWriterId.

7.2.2.4.2.5 DataSetWriter configuration message

The encoding of the *DataSetWriter* configuration data message structure is specified in Table 90. It contains the current configuration of the *WriterGroup* and the *DataSetWriter* for the *DataSet*.

The *Publisher* shall send this message without a corresponding discovery request if the configuration of the WriterGroup changed.

Table 90 – DataSetWriter configuration message structure

Name	Type	Description
DataSetWriterIds	UInt16[]	*DataSetWriterIds* contained in the configuration information.
DataSetWriterConfig	WriterGroupDataType	The current *WriterGroup* and *DataSetWriter* settings for the *DataSet* related to the *DataSetWriterId*. The *WriterGroupDataType* is defined in 6.2.5.6. The field *DataSetWriters* of the *WriterGroupDataType* shall contain only the entry for the requested or changed *DataSetWriters* in the *WriterGroup*.
statusCodes	StatusCode[]	Status codes indicating the capability of the *Publisher* to provide configuration information for the *DataSetWriterIds*. The size of the array shall match the size of the *DataSetWriterIds* array.

7.2.3 JSON message mapping

7.2.3.1 General

JSON is a format that uses human readable text. It is defined in IETF RFC 7159.

The JSON based message mapping allows OPC UA *Applications* to interoperate with web and enterprise software that use this format.

7.2.3.2 NetworkMessage

Each JSON *NetworkMessage* contains one or more JSON *DataSetMessages*. The JSON *NetworkMessage* is a JSON object with the fields defined in Table 91.

Table 91 – JSON NetworkMessage definition

Name	Type	Description
MessageId	String	A globally unique identifier for the message. This value is mandatory.
MessageType	String	This value shall be "ua-data". This value is mandatory.
PublisherId	String	A unique identifier for the *Publisher*. It identifies the source of the message. This value is optional. The presence of the value depends on the setting in the *JsonNetworkMessageContentMask*. The source is the *PublisherId* on a *PubSubConnection* (see 6.2.6.1).
DataSetClassId	String	The *DataSetClassId* associated with the *DataSets* in the *NetworkMessage*. This value is optional. The presence of the value depends on the setting in the *JsonNetworkMessageContentMask*. If specified, all *DataSetMessages* in the *NetworkMessage* shall have the same *DataSetClassId*. The source is the *DataSetClassId* on the *PublishedDataSet* (see 6.2.2.2) associated with the *DataSetWriters* that produced the *DataSetMessages*.
Messages	*	A JSON array of JSON *DataSetMessages* (see 7.2.3.3). This value is mandatory.

All fields with a concrete *DataType* defined are encoded using reversible OPC UA JSON *Data Encoding* defined in IEC 62541-6.

The fields in the JSON *NetworkMessage* are controlled by the *NetworkMessageContentMask* of the JSON *NetworkMessage* mapping (see 6.3.2.1.1).

If the *NetworkMessageHeader* bit of the *NetworkMessageContentMask* is not set, the *NetworkMessage* is the contents of the *Messages* field (e.g. a JSON array of *DataSetMessages*).

If the *DataSetMessageHeader* bit of the *NetworkMessageContentMask* is not set, the content of the *Messages* field is an array of content from the *Payload* field for each *DataSetMessage* (see 7.2.3.3).

If the *SingleDataSetMessage* bit of the *NetworkMessageContentMask* is set, the content of the *Messages* field is a JSON object containing a single *DataSetMessage*.

If the *NetworkMessageHeader* and the *DataSetMessageHeader* bits are not set and *SingleDataSetMessage* bit is set, the *NetworkMessage* is a JSON object containing the set of name/value pairs defined for a single *DataSet*.

If the JSON encoded *NetworkMessage* size exceeds the *Broker* limits, the message is dropped and a *PubSubTransportLimitsExceeded Event* is reported.

7.2.3.3 DataSetMessage

A *DataSetMessage* is produced by a *DataSetWriter* and contains list of name/value pairs which are specified by the *PublishedDataSet* associated with the *DataSetWriter*. The contents of the *DataSetMessage* are formally described by a *DataSetMetData Object.* A *DataSetMessage* is a JSON object with the fields defined in Table 92.

*DataSetWriter*s may periodically provide keep-alive messages which are *DataSetMessages* without any *Payload* field.

Table 92 – JSON DataSetMessage definition

Name	Type	Description
DataSetWriterId	UInt16	An identifier for *DataSetWriter* which created the *DataSetMessage*. This value is mandatory. It is unique within the scope of a *Publisher*.
SequenceNumber	UInt32	A strictly monotonically increasing sequence number assigned to the *DataSetMessage* by the *DataSetWriter.* This value is optional. The presence of the value depends on the setting in the *JsonDataSetMessageContentMask*.
MetaDataVersion	ConfigurationVersionDataType	The version of the *DataSetMetaData* which describes the contents of the *Payload*. This value is optional. The presence of the value depends on the setting in the *JsonDataSetMessageContentMask*.
Timestamp	DateTime	A timestamp which applies to all values contained in the *DataSetMessage.* This value is optional. The presence of the value depends on the setting in the *JsonDataSetMessageContentMask*.
Status	StatusCode	A status code which applies to all values contained in the *DataSetMessage*. This value is optional. The presence of the value depends on the setting in the *JsonDataSetMessageContentMask*.
Payload	Object	A JSON object containing the name-value pairs specified by the *PublishedDataSet*. The format of the value depends on the *DataType* of the field and the flags specified by the *DataSetMessageContentMask*.

All fields with a concrete *DataType* are encoded using reversible OPC UA JSON *Data Encoding* defined in IEC 62541-6.

The fields in the *DataSetMessage* are specified by the *DataSetFieldContentMask* in the *DataSetWriter* parameters.

DataSetFieldContentMask specifies the format of the field values in the *Payload* according to the following rules.

- If the *DataSetFieldContentMask* results in a RawData representation, the field value is a *Variant* encoded using the non-reversible OPC UA JSON *Data Encoding* defined in IEC 62541-6.
- If the *DataSetFieldContentMask* results in a *DataValue* representation, the field value is a *DataValue* encoded using the non-reversible OPC UA JSON *Data Encoding* defined in IEC 62541-6.
- If the *DataSetFieldContentMask* results in a *Variant* representation, the field value is encoded as a *Variant* encoded using the reversible OPC UA JSON *Data Encoding* defined in IEC 62541-6.

7.2.3.4 Discovery Messages

7.2.3.4.1 General

The JSON message mapping defines only one optional discovery message for the exchange of the *DataSetMetaData*. The main purpose is the exchange of additional information not contained in the *DataSetMessages* like *Properties* for the *DataSet* fields.

7.2.3.4.2 DataSetMetaData

DataSetMetaData describe the content a *DataSet* published by a *DataSetWriter*. More specifically, it specifies the names and data types of the values that shall appear in the *Payload* of a *DataSetMessage.*

When the DataSetMetaData of a DataSet changes, the *DataSetWriter* may be configured to publish the updated value through the mechanism defined by the transport protocol mapping.

The *DataSetWriterId* and *Version* fields in a *DataSetMessage* are used to correlate a *DataSetMessage* with a *DataSetMetaData.*

A *DataSetMetaData* is a JSON object with the fields defined in Table 93.

Table 93 – JSON DataSetMetaData definition

Name	Type	Description
MessageId	String	A globally unique identifier for the message. This value is mandatory.
MessageType	String	This value shall be "ua-metadata". This value is mandatory.
PublisherId	String	A unique identifier for the *Publisher*. It identifies the source of the message. This value is mandatory.
DataSetWriterId	UInt16	An identifier for *DataSetWriter* which published the *DataSetMetaData*. This value is mandatory. It is unique within the scope of a *Publisher*.
MetaData	DataSetMetaDataType	The metadata as defined in 6.2.2.1.2. This value is mandatory.

All fields with a concrete *DataType* are encoded using reversible OPC UA JSON *Data Encoding* defined in IEC 62541-6.

7.3 Transport Protocol Mappings

7.3.1 General

Subclause7.3 lists the standard protocols that have been selected for this document and their possible combinations with message mappings.

7.3.2 OPC UA UDP

OPC UA UDP is a simple UDP based protocol that is used to transport UADP *NetworkMessages*.

The syntax of the UDP transporting protocol URL used in the *Address* parameter defined in 6.2.6.3 has the following form:

opc.udp://<host>[:<port>]

The host is either an IP address or a registered name like a hostname or domain name. IP addresses can be unicast, multicast or broadcast addresses. It is the destination of the UDP datagram.

The IANA registered OPC UA port for UDP communication is 4840. This is the default and recommended port for broadcast, multicast and unicast communication. Alternative ports may be used.

The transport of a UADP *NetworkMessage* in a UDP packet is defined in Table 94.

Table 94 – UADP message transported over UDP

Name	Description
Frame Header	The frame header.
IP Header	The IP header for the frame contains information like source IP address and destination IP address. The size of the IP header depends on the used version.
UDP Header	The UDP header for the frame contains the source port, destination port, length and checksum. Each field is two byte long. The total size of the UDP header is 8 byte.
UADP NetworkMessage	The UADP NetworkMessage is sent as UDP data.
Frame Footer	The frame footer.

For OPC UA UDP it is recommended to limit the *MaxNetworkMessageSize* plus additional headers to a MTU size. The number of frames used for a UADP *NetworkMessage* influences the probability that UADP *NetworkMessages* get lost.

For OPC UA UDP the *MaxNetworkMessageSize* plus additional headers shall be limited to 65 535 Byte.

It is recommended to use switches with IGMP support to limit the distribution of multicast traffic to the interested participants. OPC UA *Applications* shall issue an IGMP membership report.

NOTE The Internet Group Management Protocol (IGMP) is a standard protocol used by hosts to report their IP multicast group memberships and needs to be implemented by any host that wishes to receive IP multicast datagrams. IGMP messages are used by multicast routers to learn which multicast groups have members on their attached networks. IGMP messages are also used by switches capable of supporting "IGMP snooping" whereby the switch listens to IGMP messages and only sends the multicast *NetworkMessages* to ports that have joined the multicast group.

There are three versions of IGMP:

– IGMP V1 is defined in IETF RFC 1112.

– IGMP V2 is defined in IETF RFC 2236.

– IGMP V3 is defined in IETF RFC 3376.

IETF RFC 2236 and IETF RFC 3376 discuss host and router requirements for interoperation with older IGMP versions.

Since OPC UA devices make extensive use of IP multicast for UDP transport, consistent IGMP usage by OPC UA devices is essential in order to create well-functioning OPC UA *Application* networks.

OPC UA devices issue a Membership Report message (V1, V2 or V3 as appropriate) when opening a UDP connection on which they will receive multicast *NetworkMessages*.

7.3.3 OPC UA Ethernet

OPC UA Ethernet is a simple Ethernet based protocol using EtherType B62C that is used to transport UADP *NetworkMessages* as payload of the Ethernet II frame without IP or UDP headers.

The syntax of the Ethernet transporting protocol URL used in the *Address* parameter defined in 6.2.6.3 has the following form:

opc.eth://<host>[:<VID>[.PCP]]

The host is a MAC address, an IP address or a registered name like a hostname. The format of a MAC address is six groups of hexadecimal digits, separated by hyphens (e.g. 01-23-45-67-89-ab). A system may also accept hostnames and/or IP addresses if it provides means to resolve it to a MAC address (e.g. DNS and Reverse-ARP).

The VID is the VLAN ID as number.

The PCP is the priority code point as one digit number.

The transport of a UADP *NetworkMessage* in an Ethernet II frame is defined in Table 95.

Table 95 – UADP message transported over Ethernet

Name	Description
Frame Header	The frame header with an EtherType of 0xB62C.
UADP NetworkMessage	The UADP NetworkMessage is sent as Ethernet payload.
Frame Footer	The frame footer.

For OPC UA Ethernet the *MaxNetworkMessageSize* plus additional headers shall be limited to an Ethernet frame size of 1 522 Byte.

The IANA registered OPC UA EtherType for UADP communication is 0xB62C.

7.3.4 AMQP

7.3.4.1 General

The Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for *Message Oriented Middleware*. AMQP is often used with a *Broker* that relays messages between applications that cannot communicate directly.

Publishers send AMQP messages to AMQP endpoints. Subscribers listen to AMQP endpoints for incoming messages. If a *Broker* is involved it may persist messages so they can be delivered even if the subscriber is not online. *Brokers* may also allow messages to be sent to multiple Subscribers.

The AMQP protocol defines a binary encoding for all messages with a header and a body. The header allows applications to insert additional information as name-value pairs that are serialized using the AMQP binary encoding. The body is an opaque binary blob that can contain any data serialized using an encoding chosen by the application.

This document defines two possible message mappings for the AMQP message body: the UADP message mapping defined in 7.2.2 and a JSON message mapping defined in 7.2.3. AMQP *Brokers* have an upper limit on message size. The mechanism for handling *NetworkMessages* that exceed the *Broker* limits depends on the encoding.

Security with AMQP is primary provided by a TLS connection between the *Publisher* or *Subscriber* and the AMQP *Broker*; however, this requires that the AMQP *Broker* be trusted. For that reason, it may be necessary to provide end-to-end security. Applications that require end-to-end security with AMQP need to use the UADP *NetworkMessages* and binary message encoding defined in 7.2.2.2. JSON encoded message bodies rely on the security mechanisms provided by AMQP and the AMQP *Broker*.

7.3.4.2 Address

The syntax of the AMQP transporting protocol URL used in the *Address* parameter defined in 6.2.6.3 has the following form:

amqps://<domain name>[:<port>][/<path>]

The default port is 5671.

The syntax for an AMQP URL over Web Sockets has the following form:

wss://<domain name>[:<port>][/<path>]

The default port is 443.

7.3.4.3 Authentication

Authentication shall be performed according to the configured *AuthenticationProfileUri* of the *PubSubConnection*, *DataSetWriterGroup*, *DataSetWriter* or *DataSetReader* entities.

If no authentication information is provided in the form of *ResourceUri* and *AuthenticationProfileUri*, SASL Anonymous is implied.

If the authentication profile specifies SASL PLAIN authentication, a separate connection for each new Authentication setting is required.

7.3.4.4 Connection properties

AMQP allows sending properties as part of opening the connection, session establishment and link attach.

The connection properties apply to any connection, session or link created as part of the *PubSubConnection*, or subordinate configuration entities, such as *WriterGroup* and *DataSetWriter*.

The properties are defined through the *KeyValuePair* array in the ConnectionProperties *WriterGroupProperties* and *DataSetWriterProperties*. The *NamespaceIndex* of the *QualifiedName* in the *KeyValuePair* shall be 0 for AMQP standard properties. The *Name* of the *QualifiedName* is constructed from a prefix and the AMQP property name with the following syntax.

Name = <target prefix>-<AMQP property name>

The target prefix can have the following values:

- connection;
- session;
- link.

The *Value* of the *KeyValuePair* is converted to an AMQP data type using the rules defined in Table 98. If there is no rule defined for a data type, the property shall not be included.

The connection properties are intended to be used sparingly to optimize interoperability with existing broker endpoints.

7.3.4.5 RequestedDeliveryGuarantee

A writer negotiates the delivery guarantees for its link using the snd-settle-mode settlement policy (settled, unsettled, mixed) it will use, and the desired rcv-settle-mode (first, second) of the broker.

Vice versa, the reader negotiates delivery guarantees using its rcv-settle-mode (first, second) and the desired snd-settle-mode (settled, unsettled) of the broker.

This matches to the *BrokerTransportQualityOfService* values as follows.

- *AtMostOnce_1* – messages are pre-settled at the sender endpoint and not sent again. Messages may be lost in transit. This is the default setting.
- *AtLeastOnce_2* – messages are received and settled at the receiver without waiting for the sender to settle.
- *ExactlyOnce_3 – messages are received, the sender settles and then the receiver settles.*

7.3.4.6 Transport Limits and Keep Alive

If the *KeepAliveTime* is set on a *WriterGroup*, a value slightly higher than the configured value of the group should be used as idle timeout of the connection ensuring that the connection is disconnected if the keep alive message was not sent by any writer. Otherwise, if no *KeepAliveTime* is specified, the implementation should set a reasonable default value.

When setting the maximum message sizes for the Link, the *MaxNetworkMessageSize* of the *PubSubGroup* shall be used. If this value is 0, the implementation chooses a reasonable maximum.

Other limits are up to the implementation and depend on the capabilities of the OS or on the capabilities of the device the *Publisher* or *Subscriber* is running on, and can be made configurable through configuration model extensions or by other means.

7.3.4.7 Message header

The AMQP message header has a number of standard fields. Table 96 describes how these fields shall be populated when an AMQP message is constructed.

Table 96 – AMQP standard header fields

Field Name	Source
message-id	A globally unique value created by the *DataSetWriter*.
subject	Valid values are ua-data or ua-metadata.
content-type	The MIME type for the message body. The MIME types are specified in the message body subclauses 7.3.4.8.1 and 7.3.4.8.2.

The subject defines the type of the message contained in the AMQP body. A value of "ua-data" specifies the body contains a UADP or JSON *NetworkMessage*. A value of "ua-metadata" specifies a body that contains a UA Binary or JSON encoded *DataSetMetaData Message*. The content-type specifies the whether the message is binary or JSON data.

The AMQP message header shall include additional fields defined on the *WriterGroup* or *DataSetWriter* through the *KeyValuePair* array in the *WriterGroupProperties* and *DataSetWriterProperties*. The *NamespaceIndex* of the *QualifiedName* in the *KeyValuePair* shall be 0 for AMQP standard message properties. The *Name* of the *QualifiedName* is constructed from a message prefix and the AMQP property name with the following syntax.

Name = message-<AMQP property name>

Table 97 defines the AMQP standard message properties.

Table 97 – OPC UA AMQP standard header QualifiedName Name mappings

AMQP standard property name	OPC UA DataType	AMQP data type
to	String	*
user-id	ByteString	binary
reply-to	String	string
correlation-id	ByteString	*
absolute-expiry-time	DateTime	timestamp
group-id	String	string
reply-to-group-id	String	string
creation-time	DateTime	timestamp
content-encoding	String	symbol

Any name not in the table is assumed to be an application property. In this case the namespace provided as part of the *QualifiedName* shall be the *ApplicationUri*.

The AMQP message header shall include additional promoted fields of the *DataSet* as a list of name-value pairs. *DataSet* fields with the *PromotedField* flag set in the *FieldMetaData fieldFlags* are copied into the AMQP header. The *FieldMetaData Structure* is defined in 6.2.2.1.3. Promoted fields shall always be included in the header even if the *DataSetMessage* body is a delta frame and the *DataSet* field is not included in the delta frame. In this case the last known value is sent in the header.

When a field is added to the header it is converted to an AMQP data type using the rules defined in Table 98. If there is no rule defined for the data type, the field shall not be included.

Table 98 – OPC UA AMQP header field conversion rules

OPC UA DataType	Conversion Rules to AMQP data types.
Boolean	AMQP 'boolean' type.
SByte	AMQP 'byte' type.
Byte	AMQP 'ubyte' type.
Int16	AMQP 'short' type.
UInt16	AMQP 'ushort' type.
Int32	AMQP 'int' type.
UInt32	AMQP 'uint' type.
Int64	AMQP 'long' type.
UInt64	AMQP 'ulong' type.
Float	AMQP 'float' type.
Double	AMQP 'double' type.
String	AMQP 'string' type.
ByteString	AMQP 'binary' type.
DateTime	AMQP 'timestamp' type. This conversion may result in loss of precision on some platforms. The rules for dealing with the loss of precision are described in IEC 62541-6.
Guid	AMQP 'uuid' type.
QualifiedName	The QualifiedName is encoded as an AMQP 'string' type with the format <NamespaceUri>'#'<Name>.
LocalizedText	Not supported and the related field is discarded.
NodeId	If the NamespaceIndex = 0 the value is encoded as an AMQP 'string' type using the format for a NodeId defined in IEC 62541-6. If the NamespaceIndex > 0 the value is converted to an ExpandedNodeId with a NamespaceUri and is encoded as an AMQP 'string' type using the format for an ExpandedNodeId defined in IEC 62541-6.
ExpandedNodeId	If the NamespaceUri is not provided the rules for the NodeId are used. If the NamespaceUri is provided the value is encoded as an AMQP 'string' type using the format for an ExpandedNodeId defined in IEC 62541-6.
StatusCode	AMQP 'uint' type.
Variant	If the value has a supported datatype it uses that conversion; otherwise it is not supported and the related field is discarded.
Structure	Not supported and the related field is discarded.
Structure with option fields	Not supported and the related field is discarded.
Array	Not supported and the related field is discarded.
Union	Not supported and the related field is discarded.

7.3.4.8 Message body

7.3.4.8.1 JSON

A JSON body is encoded as defined for the JSON message mapping defined in 7.2.3.

The corresponding MIME type is application/json.

7.3.4.8.2 UADP

A UADP body is encoded as defined for the UADP message mapping defined in 7.2.2.

The corresponding MIME type is application/opcua+uadp.

If the encoded AMQP message size exceeds the *Broker* limits it shall be broken into multiple chunks as described in 7.2.2.2.4.

It is recommended that the *MetaDataQueueName* as described in 6.4.2.3.6 is configured as a sub-topic of the related *QueueName* with the name $Metadata.

7.3.5 MQTT

7.3.5.1 General

The Message Queue Telemetry Transport (MQTT) is an open standard application layer protocol for *Message Oriented Middleware*. MQTT is often used with a *Broker* that relays messages between applications that cannot communicate directly.

Publishers send MQTT messages to MQTT brokers. Subscribers subscribe to MQTT brokers for messages. A *Broker* may persist messages so they can be delivered even if the subscriber is not online. *Brokers* may also allow messages to be sent to multiple *Subscribers*.

The MQTT protocol defines a binary protocol used to send and receive messages from and to topics. The body is an opaque binary blob that can contain any data serialized using an encoding chosen by the application.

This document defines two possible encodings for the message body: the binary encoded *DataSetMessage* defined in 7.2.2 and a JSON encoded *DataSetMessage* defined in 7.2.3. MQTT does not provide a mechanism for specifying the encoding of the MQTT message which means the *Subscribers* shall be configured in advance with knowledge of the expected encoding. *Publishers* should only publish *NetworkMessages* using a single encoding to a unique MQTT topic name.

Security with MQTT is primarily provided by a TLS connection between the *Publisher* or *Subscriber* and the MQTT server; however, this requires that the MQTT server be trusted. For that reason, it may be necessary to provide end-to-end security. Applications that require end-to-end security with MQTT need to use the UADP *NetworkMessages* and binary message encoding defined in 7.2.2. JSON encoded message bodies need to rely on the security mechanisms provided by MQTT and the MQTT server.

7.3.5.2 Address

The syntax of the MQTT transporting protocol URL used in the *Address* parameter defined in 6.2.6.3 has the following form:

mqtts://<domain name>[:<port>][/<path>]

The default port is 8883.

The syntax for an MQTT URL over Web Sockets has the following form:

wss://<domain name>[:<port>][/<path>]

The default port is 443.

7.3.5.3 Authentication

The current MQTT transport mapping only supports simple Username/Password authentication.

7.3.5.4 ConnectionProperties

The current MQTT transport mapping does not support the concept of connection properties and any configured setting in the connection properties shall be silently discarded.

Implementations should attempt to reconnect to existing sessions (CleanSession=0) and attempt to resume message transfer at the specified QoS level.

7.3.5.5 RequestedDeliveryGuarantee

The *BrokerTransportQualityOfService* values map to MQTT publish and subscribe QoS settings as follows.

- AtMostOnce_1 is mapped to MQTT QoS 0.
- AtLeastOnce_2 is mapped to MQTT QoS 1.
- ExactlyOnce_3 is mapped to MQTT Qos 2.

7.3.5.6 Transport Limits and Keep Alive

If the *KeepAliveTime* is set on a *WriterGroup*, a value slightly higher than the configured value of the group in seconds should be set as MQTT Keep Alive ensuring that the connection is disconnected if the keep alive message was not sent by any writer in the specified time.

The implmentation chooses packet and message size limits depending on the capabilities of the OS or or the capabilities of the device the application is running on. They can be made configurable through configuration model extensions or by other means.

7.3.5.7 Message header

The current MQTT transport mapping does not support message headers. Any promoted field or additional fields defined on the *WriterGroup* or *DataSetWriter* shall be silently discarded. Implementations shall not set the MQTT RETAIN flag, except for metadata messages published to the *MetaDataQueueName* as described in 6.4.2.3.6.

7.3.5.8 Message body

7.3.5.8.1 JSON

A JSON body is encoded as defined for the JSON message mapping defined in 7.2.3.

7.3.5.8.2 UADP

A UADP body is encoded as defined for the UADP message mapping defined in 7.2.2.

It is expected that the software used to receive UADP *NetworkMessage* can process the body without needing to know how it was transported.

If the encoded MQTT message size exceeds the *Broker* limits, it is broken into multiple chunks as described in 7.2.2.2.4.

It is recommended that the *MetaDataQueueName* as described in 6.4.2.3.6 is configured as a sub-topic of the related *QueueName* with the name $Metadata. The MQTT RETAIN flag shall be set for metadata messages.

8 PubSub security key service model

8.1 Overview

Clause 8 specifies the OPC UA *Information Model* for a *Security Key Service* (SKS). The functionality and behaviour of an SKS is described in 5.4.3. It defines the distribution framework for cryptographic keys used for message security.

The SKS can be a network service used to manage keys for all *Publishers* and *Subscribers* or it can be part of a *Publisher* to manage the keys for the *NetworkMessages* sent by this *Publisher*.

Figure 34 depicts the *ObjectTypes* and their components used to represent the *PublishSubscribe* Object.

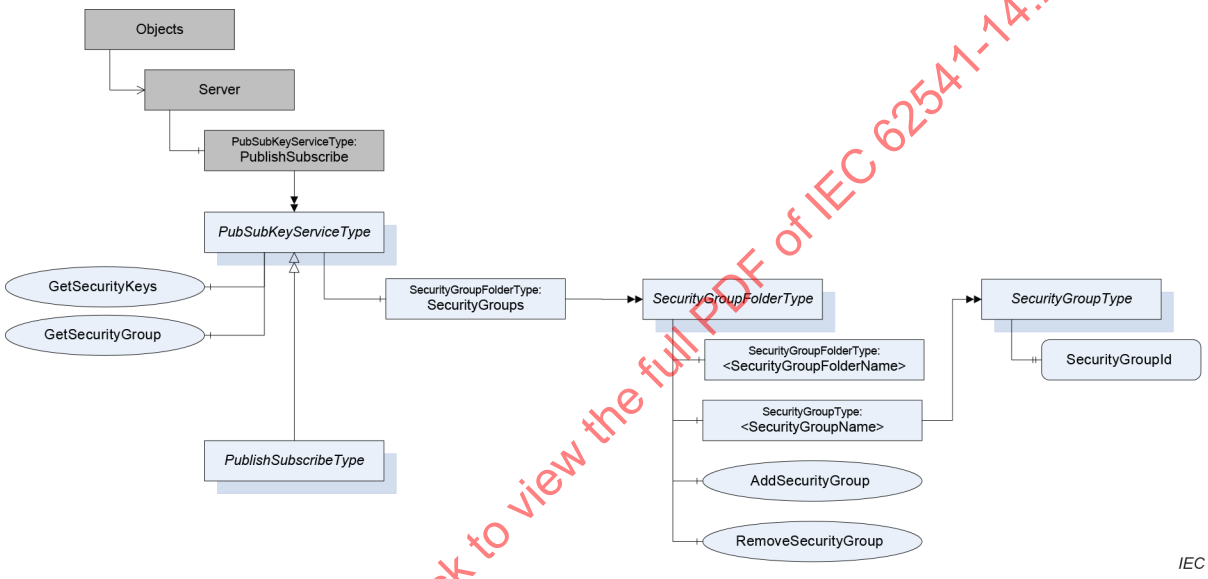


Figure 34 – PublishSubscribe Object Types overview

The *PublishSubscribe Object* is the root node for all *PubSub* related configuration *Objects*. It is an instance of the *PubSubKeyServiceType* or the *PublishSubscribeType* and a component of the *Server Object*.

The *PubSubKeyServiceType* defines the *Method* for access to security keys and the related management of *SecurityGroups*. This *ObjectType* is used for the *PublishSubscribe Object* if only the *Security Key Service* functionality is provided. If the *PubSub* configuration functionality is provided, the *PublishSubscribeType* is used instead.

The *SecurityGroups* are organized by the *SecurityGroupFolderType* and represented by instances of the *SecurityGroupType*.

The *PublishSubscribeType* contains the entry points for the PubSub configuration model defined in Clause 9.

8.2 PublishSubscribe Object

To provide interoperability between *Publishers, Subscribers, Security Key Services* and configuration tools, all *PubSub* related *Objects* shall be exposed through an *Object* called "PublishSubscribe" that is of the type *PubSubKeyServiceType* or a subtype. This *Object* shall be a component of the *Server Object*. It is formally defined in Table 99.

Table 99 – PublishSubscribe Object definition

Attribute	Value				
BrowseName	PublishSubscribe				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
ComponentOf the *Server Object* defined in IEC 62541-5.					
HasTypeDefinition	ObjectType	PubSubKeyServiceType			

8.3 PubSubKeyServiceType

The *PubSubKeyServiceType* is formally defined in Table 100.

Table 100 – PubSubKeyServiceType definition

Attribute	Value				
BrowseName	PubSubKeyServiceType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5.					
HasComponent	Method	GetSecurityKeys	Defined in 8.4.		Optional
HasComponent	Method	GetSecurityGroup	Defined in 8.7.		Optional
HasComponent	Object	SecurityGroups		SecurityGroupFolderType	Optional

The *PubSubKeyServiceType ObjectType* is a concrete type and can be used directly.

The *SecurityGroups* folder organizes the *Objects* representing the *SecurityGroup* configuration.

8.4 GetSecurityKeys method

This *Method* is used to retrieve the security keys for a *SecurityGroup*.

This *Method* is required to access the security keys of a *PubSubGroup* where the *SecurityGroup* manages the security keys for *PubSubGroups*. The *MessageSecurity Object* of the *PubSubGroup Object* contains the *SecurityGroupId* that shall be passed to this *Method* in order to access the keys for the *PubSubGroup*. Note that multiple *PubSubGroups* can share a *SecurityGroupId*.

The *Permission* of the *SecurityGroupType Object* for the *SecurityGroupId* controls the access to the security keys for the *SecurityGroupId*. If the user used to call this *Method* does not have the *Call Permission* set for the related *SecurityGroupType Object*, the *Server* shall return *Bad_UserAccessDenied* for this *Method*. The *SecurityGroupType* is defined in 8.6. Encryption is required for this *Method*. The *Method* shall return *Bad_SecurityModeInsufficient* if the communication is not encrypted.

The information necessary to access the *Server* that implements the *GetSecurityKeys Method* for the *SecurityGroup* is also contained in the *MessageSecurity Object* of the *PubSubGroup Object*.

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*.

If the *Publisher* implements the *GetSecurityKeys Method* and the related *SecurityGroup* management, the keys are made invalid immediately after a *SecurityGroup* is removed or keys for a *SecurityGroup* are revoked.

If a central SKS implements the *GetSecurityKeys Method* and the related *SecurityGroup* management, the keys are no longer valid after a *SecurityGroup* is removed or keys for a *SecurityGroup* are revoked. However, *Subscribers* shall be prepared for *Publishers* using invalid keys until they have called the *GetSecurityKeys Method*. *Publishers* using a central SKS shall call *GetSecurityKeys* at a period of half the *KeyLifetime*.

Signature

```
GetSecurityKeys (
    [in]  String           SecurityGroupId
    [in]  UInt32           StartingTokenId
    [in]  UInt32           RequestedKeyCount
    [out] String           SecurityPolicyUri
    [out] IntegerId        FirstTokenId
    [out] ByteString[]     Keys
    [out] Duration         TimeToNextKey
    [out] Duration         KeyLifetime
    );
```

Argument	Description
SecurityGroupId	The identifier for the *SecurityGroup*. It shall be unique within the *Security Key Service*.
StartingTokenId	The current token is requested by passing 0. It can be a *SecurityTokenId* from the past to get a key valid for previously sent messages. If the *StartingTokenId* is unknown, the oldest available tokens are returned.
RequestedKeyCount	The number of requested keys which should be returned in the response. If 0 is requested, no future keys are returned. If the caller requests a number larger than the *Security Key Service* permits, then the SKS shall return the maximum it allows.
SecurityPolicyUri	The URI for the set of algorithms and key lengths used to secure the messages. The *SecurityPolicies* are defined in IEC 62541-7.
FirstTokenId	The *SecurityTokenId* of the first key in the array of returned keys. The *SecurityTokenId* appears in the header of messages secured with a *Key*. It starts at 1 and is incremented by 1 each time the *KeyLifetime* elapses even if no keys are requested. If the *SecurityTokenId* increments past the maximum value of *UInt32* it restarts at 1. If the caller has key material from previous *GetSecurityKeys Method* calls, the *FirstTokenId* is used to match the existing list with the fetched list and to eliminate duplicates. If the *FirstTokenId* is unknown, the existing list shall be discarded and replaced.
Keys	An ordered list of keys that are used when the *KeyLifetime* elapses. If the current key was requested, the first key in the array is used to secure the messages. This key is not used directly since the protocol associated with the *PubSubGroup(s)* specifies an algorithm to generate distinct keys for different types of cryptography operations. Further details are defined in 7.2.2.2.3. The *SecurityTokenId* associated with the first key in the list is the *FirstTokenId*. All following keys have a *SecurityTokenId* that is incremented by 1 for every key returned.
TimeToNextKey	The time, in milliseconds, before the *CurrentKey* is expected to expire. If a *Publisher* uses this *Method* to get the keys from a SKS, the *TimeToNextKey* and *KeyLifetime* are used to calculate the time the *Publisher* shall use the next key. The *TimeToNextKey* defines the time when to switch from *CurrentKey* to *FutureKeys* and the *KeyLifetime* defines when to switch from one future key to the next future key. For a *Subscriber* the *TimeToNextKey* and *KeyLifetime* are used to calculate the time the *Subscriber* expects that the *Publishers* use the next key. Due to network latency, out of order delivery and the use of keys for several *Publishers*, a *Subscriber* needs to expect some overlap time where *NetworkMessages* are received that are using the previous or the next key. *TimeToNextKey* and *KeyLifetime* are also used to calculate the time until *Publisher* and *Subscriber* shall fetch new keys.
KeyLifetime	The lifetime of a key in milliseconds. The returned keys may expire earlier if the keys are discarded for some reason. An unplanned key rotation is indicated in the *NetworkMessage* header before the next key is used to give the *Subscriber* some time to fetch new keys. If the *CurrentTokenId* in the message is not recognized the receiver shall call this *Method* again to get new keys.

Method Result codes

ResultCode	Description
Bad_NotFound	The *SecurityGroupId* is unknown.
Bad_UserAccessDenied	The caller is not allowed to request the keys for the SecurityGroup.
Bad_SecurityModeInsufficient	The communication channel is not using encryption.

8.5 GetSecurityGroup method

This *Method* provides a direct lookup of the *NodeId* of a *SecurityGroupType Object* based on a *SecurityGroupId*. It is used by a security administration tool to get the *SecurityGroup Object* for configuration of access permissions for the keys.

The *SecurityGroupId* is the identifier for the *SecurityGroup* in *Publishers*, *Subscribers* and the key *Server*. This *Method* returns the *NodeId* of the corresponding *SecurityGroup Object Node* providing the configuration and diagnostic options for a *SecurityGroup*.

Signature

```
GetSecurityGroup (
    [in]  String      SecurityGroupId
    [out] NodeId      SecurityGroupNodeId
    );
```

Argument	Description
SecurityGroupId	The *SecurityGroupId* of the *SecurityGroup* to lookup.
SecurityGroupNodeId	The *NodeId* of the *SecurityGroupType Object*.

Method Result codes

ResultCode	Description
Bad_NoMatch	The SecurityGroupId cannot be found in the *Server*.

8.6 SecurityGroupType

The *SecurityGroupType* is formally defined in Table 101.

The *Permission* of the *SecurityGroupType Objects* controls the access to the security keys for the *SecurityGroup* through the *Method GetSecurityKeys*. The *GetSecurityKeys Method* is defined in 8.4.

Table 101 – SecurityGroupType definition

Attribute	Value				
BrowseName	SecurityGroupType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5.					
HasProperty	Variable	SecurityGroupId	String	PropertyType	Mandatory
HasProperty	Variable	KeyLifetime	Duration	PropertyType	Mandatory
HasProperty	Variable	SecurityPolicyUri	String	PropertyType	Mandatory
HasProperty	Variable	MaxFutureKeyCount	UInt32	PropertyType	Mandatory
HasProperty	Variable	MaxPastKeyCount	UInt32	PropertyType	Mandatory

The *Property SecurityGroupId* contains the identifier for the *SecurityGroup* used in the key exchange *Methods GetSecurityKeys* and *SetSecurityKeys* in the *PubSubGroupType*.

The *Property KeyLifetime* defines the lifetime of a key in milliseconds.

The *Property SecurityPolicyUri* is the identifier for a *SecurityPolicy*. *SecurityPolicies* define the set of algorithms and key lengths used to secure the messages exchanged in the context of the *SecurityGroup*. The *SecurityPolicies* are defined in IEC 62541-7.

The *Property MaxFutureKeyCount* defines the maximum number of future keys returned by the *Method GetSecurityKeys*.

The *Property MaxPastKeyCount* defines the maximum number of historical keys stored by the SKS. The historical keys are necessary to allow *Subscribers* to request keys for older *NetworkMessages*.

8.7 SecurityGroupFolderType

The *SecurityGroupFolderType* is formally defined Table 102.

Table 102 – SecurityGroupFolderType definition

Attribute	Value				
BrowseName	SecurityGroupFolderType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**		**TypeDefinition**	**ModellingRule**
Subtype of FolderType defined in IEC 62541-5.					
Organizes	Object	<SecurityGroupFolderName>		SecurityGroup FolderType	OptionalPlaceholder
HasComponent	Object	<SecurityGroupName>		SecurityGroupType	OptionalPlaceholder
HasComponent	Method	AddSecurityGroup	Defined in 8.8.		Mandatory
HasComponent	Method	RemoveSecurityGroup	Defined in 8.9.		Mandatory

The *SecurityGroupFolderType ObjectType* is a concrete type and can be used directly.

Instances of the *SecurityGroupFolderType* can contain *SecurityGroup Objects* or other instances of the *SecurityGroupFolderType*. This can be used to build a tree of folder *Objects* used to organize the configured *SecurityGroups*.

The *SecurityGroup Objects* are added as components to the instance of the *SecurityGroupFolderType*. A *SecurityGroup Object* is referenced only from one folder. If the folder is deleted, all referenced *SecurityGroup Objects* are deleted with the folder.

8.8 AddSecurityGroup Method

This *Method* is used to add *a SecurityGroupType Object* to the *SecurityGroupFolderType Object*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddSecurityGroup (
    [in]  String      SecurityGroupName
    [in]  Duration    KeyLifetime
    [in]  String      SecurityPolicyUri
    [in]  UInt32      MaxFutureKeyCount
    [in]  UInt32      MaxPastKeyCount
    [out] String      SecurityGroupId
    [out] NodeId      SecurityGroupNodeId
    );
```

Argument	Description
SecurityGroupName	Name of the *SecurityGroup* to add.
KeyLifetime	The lifetime of a key in milliseconds
SecurityPolicyUri	The *SecurityPolicy* used for the *SecurityGroup*.
MaxFutureKeyCount	The maximum number of future keys returned by the *Method GetSecurityKeys*.
MaxPastKeyCount	The maximum number of historical keys stored by the SKS
SecurityGroupId	The identifier for the *SecurityGroup*.
SecurityGroupNodeId	The *NodeId* of the added *SecurityGroupType Object*.

Method Result codes

ResultCode	Description
Bad_NodeIdExists	A *SecurityGroup* with the name already exists.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the object.

8.9 RemoveSecurityGroup Method

This *Method* is used to remove a *SecurityGroupType Object* from the *SecurityGroupFolderType Object*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality and for the *SecurityGroup* to delete when invoking this *Method* on the *Server*.

See 8.4 for details on the lifetime of keys previously issued for this *SecurityGroup*.

Signature

```
RemoveSecurityGroup (
   [in]  NodeId      SecurityGroupNodeId
   );
```

Argument	Description
SecurityGroupNodeId	*NodeId* of the *SecurityGroupType Object* to remove from the *Server*

Method Result codes

ResultCode	Description
Bad_NodeIdUnknown	The *SecurityGroupNodeId* is unknown.
Bad_NodeIdInvalid	The *SecurityGroupNodeId* is not a *NodeId* of a *SecurityGroupType Object*.
Bad_UserAccessDenied	The *Session* user is not allowed to delete the *SecurityGroupType Object*.

9 PubSub configuration model

9.1 Common configuration model

9.1.1 General

Figure 35 depicts the *ObjectTypes* of the message and transport protocol mapping independent part of the *PubSub* configuration model, their main components and their relations.

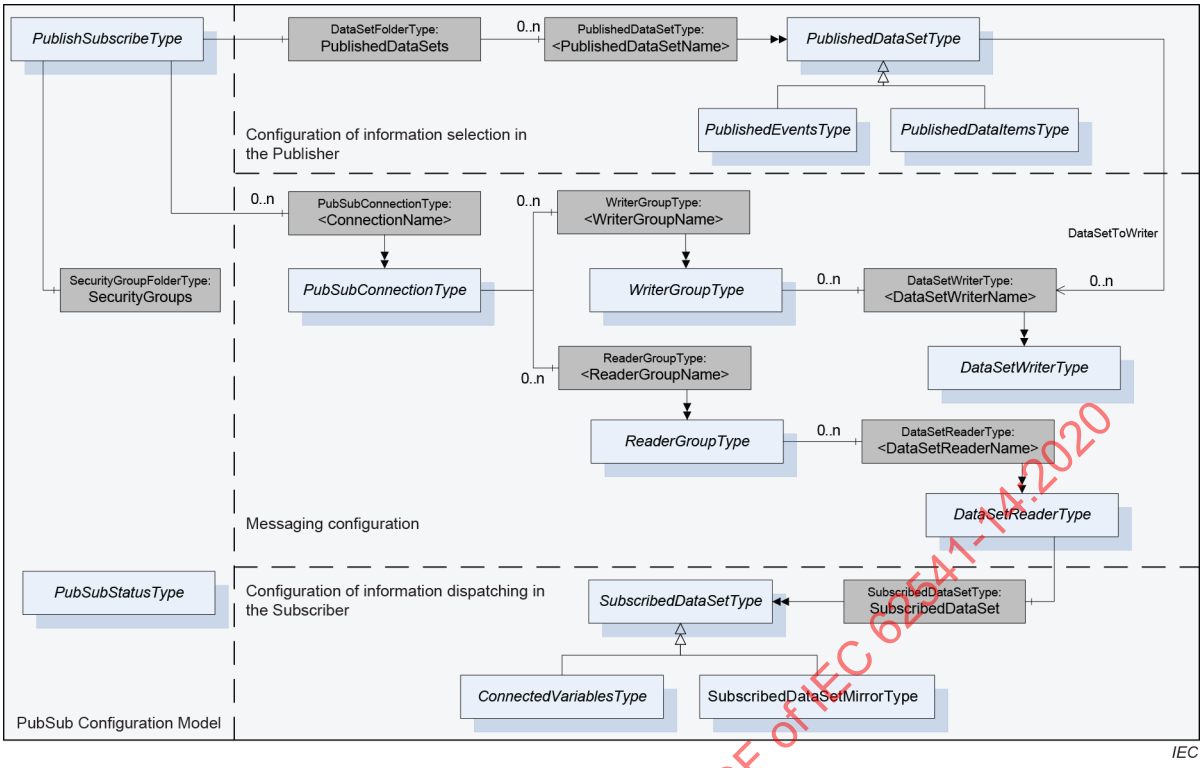


Figure 35 – PubSub configuration model overview

An instance of the *PublishSubscribeType* with the name *PublishSubscribe* represents the root *Object* for all *PubSub* related *Objects*. It manages a list of *PubSubConnectionType Objects* and the *PublishedDataSetType Objects* through the *PublishedDataSets* folder.

On the *Publisher* side, a *PublishedDataSet* represents the information to publish and the *DataSetWriter* represents the transport settings for creating *DataSetMessages* for delivery through a *Message Oriented Middleware*.

On the *Subscriber* side, a *DataSetReader* represents the transport settings for receiving *DataSetMessages* from a *Message Oriented Middleware* and the *SubscribedDataSet* represents the information to dispatch the received *DataSets* in the *Subscriber*.

The configuration can be done through *Methods* or product-specific configuration tools. The *DataSetFolderType* can be used to organize the *PublishedDataSetType Objects* in a tree of folders.

Figure 36 shows an example configuration with the root *Object PublishSubscribe* that is a component of the *Server Object*.

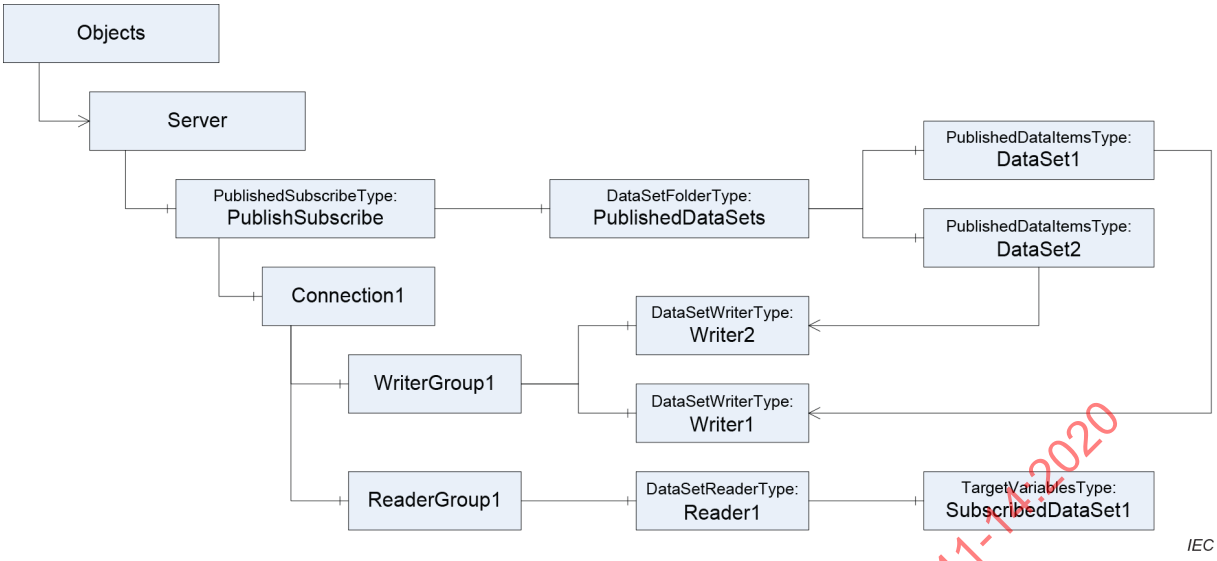


Figure 36 – PubSub example Objects

The example defines two *PublishedDataSets* published through one connection and one group and one *DataSetReader* used to subscribe one *DataSet*.

Figure 37 depicts the information flow and the related *ObjectTypes* from the *PubSub Information Model*. The boxes in the lower part of the figure are examples for blocks necessary to implement the information flow in a *Publisher*.

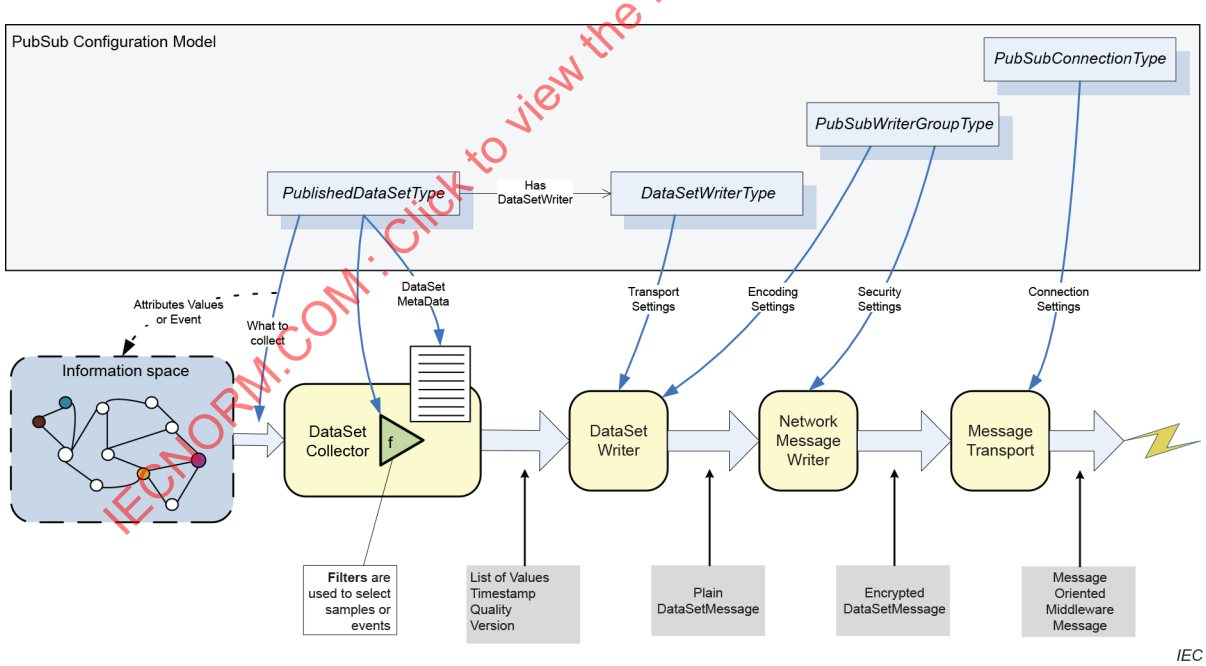


Figure 37 – PubSub information flow

The *PublishedDataSetType* represents the selection and configuration of *Variables* or *Events*. An *Event* notification or a snapshot of the *Variables* comprises a *DataSet*. A *DataSet* is the content of a *DataSetMessage* created by a *DataSetWriter*. Examples of concrete *PublishedDataSetTypes* are *PublishedEventsType* and *PublishedDataItemsType*. An instance of *PublishedDataSetType* has a list of *DataSetWriters* used to produce *DataSetMessages* sent via the *Message Oriented Middleware*. The *DataSetMetaData* describes the content of a *DataSet*.

Instances of the *PubSubConnectionType* represent settings associated with *Message Oriented Middleware*. A connection manages a list of *WriterGroupType Objects* and transport protocol mapping specific parameters.

Instances of the *WriterGroupType* contain instances of *DataSetWriter Objects* that share settings such as security configuration, encoding or timing of *NetworkMessages*. A group manages a list of *DataSetWriterType* Objects that define the payload of the *NetworkMessages* created from the group settings.

DataSetWriters represent the configuration necessary to create *DataSetMessages* contained as payload in *NetworkMessages*.

DataSetReaders represent the configuration necessary to receive and process *DataSetMessages* on the *Subscriber* side.

NetworkMessages are sent through a transport like AMQP, MQTT or OPC UA UDP. Other transport protocols can be added as subtypes without changing the base model.

The definition of the *PubSub* related *ObjectTypes* does not prescribe how the instances are created or configured or how dynamic the configuration can be. A *Publisher* may have a preconfigured number of *PublishedDataSets* and *DataSetWriters* where only protocol-specific settings can be configured. If a *Publisher* allows dynamic creation of *Objects* like *DataSets* and *DataSetWriters*, this can be done through product-specific configuration tools or through the standardized configuration *Methods* defined in this document.

9.1.2 Configuration behaviours

Publishers and *Subscribers* may be configurable through vendor-specific engineering tools or with the configuration *Methods* and parameters described in this document. This allows a standard OPC UA Client based configuration tool to configure an OPC UA *Server* that is a *Publisher* and/or *Subscriber*.

Configuration parameters are exposed as *Variables* of the configurable *Objects*. *Methods* for creation of *Objects* have input arguments for mandatory *Variables*. Optional *Variables* are not contained in the input arguments of *Methods* for *Object* creation. *Optional Variables* are created with a default value if they are supported for the *Object* or required for the current configuration. The default value can be changed by writing to the *Variable* after creation. Newly created *Objects* shall have the *Status Disabled_0* if they are created with the standard *Methods*.

Variables that can be configured shall have the *CurrentWrite* flag set in the *AccessLevel Attribute*. The *UserAccessLevel* may be limited based on the rights of the user of the OPC UA *Client*.

Configuration changes shall be applied in a batch to avoid inconsistencies between different configuration parameters. The mechanism to apply changes in a batch operation is to allow changes only when the related *Object* has the *Status Disabled_0* and to apply the new configuration settings when the *Status* is changed to *Operational_2*. Therefore write operations to configuration parameters shall be rejected with *Bad_InvalidState* if the *Status* is not *Disabled_0*. Changes to *PublishedDataSet* configurations shall be rejected with *Bad_InvalidState* if not all related *DataSetWriters* have the *Status Disabled_0*.

9.1.3 Types for the PublishSubscribe Object

9.1.3.1 Overview

Figure 38 depicts the *PublishSubscribeType* and the components used to represent the *PublishSubscribe* Object.

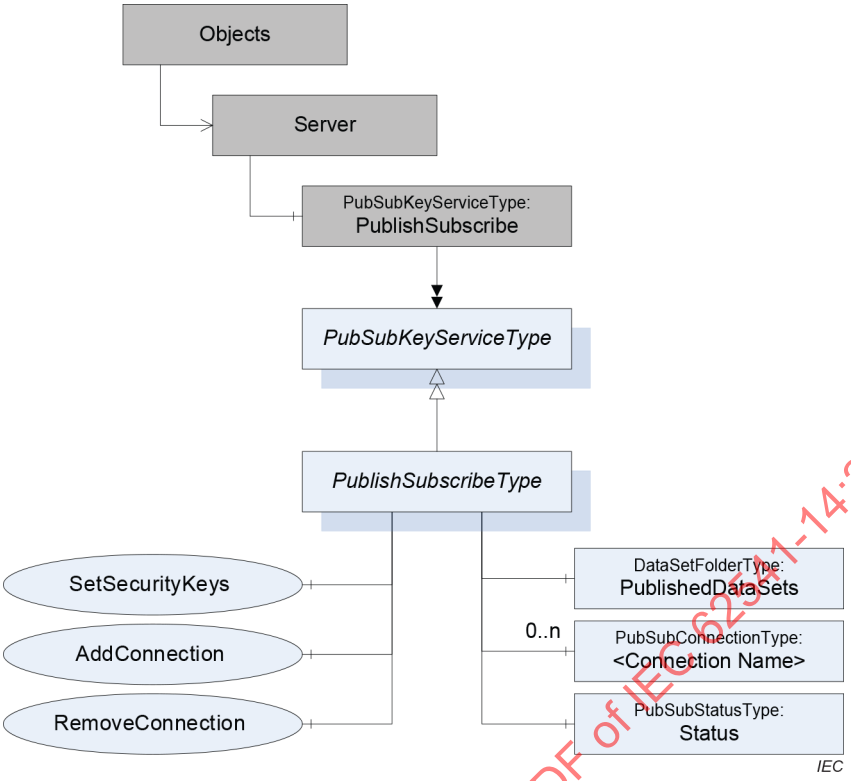


Figure 38 – PublishSubscribe Object Types overview

The *PublishSubscribe Object* is the root node for all *PubSub* related configuration *Objects*. It is an instance of the *PublishSubscribeType* and a component of the *Server Object*.

The *PublishSubscribeType* contains the entry point for *PublishedDataSet* configuration, the entry point for *PubSub* connections. In addition, it provides *Methods* for connection management.

9.1.3.2 PublishSubscribeType

An instance of this *ObjectType* represents the root *Object* for all *PubSub* related configuration and metadata *Objects*. The one instance of this *ObjectType* that represents the root *Object* is defined in 8.4. The *ObjectType* is formally defined in Table 103.

Table 103 – PublishSubscribeType definition

Attribute	Value				
BrowseName	PublishSubscribeType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubKeyServiceType defined in 8.2.					
HasPubSub Connection	Object	<ConnectionName>		PubSubConnectionType	Optional Placeholder
HasComponent	Method	SetSecurityKeys	Defined in 9.1.3.3.		Optional
HasComponent	Method	AddConnection	Defined in 9.1.3.4.		Optional
HasComponent	Method	RemoveConnection	Defined in 9.1.3.5.		Optional
HasComponent	Object	PublishedDataSets		DataSetFolderType	Mandatory
HasComponent	Object	Status		PubSubStatusType	Mandatory
HasComponent	Object	Diagnostics		PubSubDiagnosticsRootType	Optional
HasProperty	Variable	SupportedTransportProfiles	String[]	PropertyType	Mandatory

The *PublishSubscribeType ObjectType* is a concrete type and can be used directly.

The configured connection *Objects* are added as components to the instance of the *PublishSubscribeType*. Connection *Objects* may be configured with product-specific configuration tools or added and removed through the *Methods AddUadpConnection, AddBrokerConnection* and *RemoveConnection*. The *PubSubConnectionType* is defined in 9.1.5.2. The *HasPubSubConnection ReferenceType* is defined in 9.1.3.6.

The *PublishedDataSets Object* contains the configured *PublishedDataSets*. The *DataSetFolderType* is defined in 9.1.4.5.1. The *DataSetFolderType* can be used to build a tree of *DataSetFolders*.

The *Status Object* provides the current operational status of the *PublishSubscribe* functionality. The *PubSubStatusType* is defined in 9.1.10. The state machine for the status and the relation to other *PubSub Objects* like *PubSubConnection*, *PubSubGroup*, *DataSetWriter* and *DataSetReader* are defined in 6.2.1.

The *Diagnostics Object* provides the current diagnostic information for the *PublishSubscribe Object*. The *PubSubDiagnosticsRootType* is defined in 9.1.11.7.

The *SupportedTransportProfiles Property* provides a list of *TransportProfileUris* supported by the *Server*. The TransportProfileUris are defined in IEC 62541-7.

9.1.3.3 SetSecurityKeys

This *Method* is used to push the security keys for a *SecurityGroup* into a *Publisher* or *Subscriber*. It is used if *Publisher* or *Subscriber* have no OPC UA *Client* functionality.

Encryption is required for this *Method*. The *Method* shall return *Bad_SecurityModeInsufficient* if the communication is not encrypted.

Signature

```
SetSecurityKeys (
    [in]  String           SecurityGroupId
    [in]  String           SecurityPolicyUri
    [in]  IntegerId        CurrentTokenId
    [in]  ByteString       CurrentKey
    [in]  ByteString[]     FutureKeys
    [in]  Duration         TimeToNextKey
    [in]  Duration         KeyLifetime
    );
```

Argument	Description
SecurityGroupId	The identifier for the *SecurityGroup*.
SecurityPolicyUri	The URI for the set of algorithms and key lengths used to secure the messages. The *SecurityPolicies* are defined in IEC 62541-7.
CurrentTokenId	The *SecurityTokenId* that appears in the header of messages secured with the *CurrentKey*. It starts at 1 and is incremented by 1 each time the *KeyLifetime* elapses even if no keys are requested. If the *CurrentTokenId* increments past the maximum value of *UInt32* it restarts at 1. If the *PubSub Object* has key material from previous *SetSecurityKeys Method* calls, the *CurrentTokenId* is used to match the existing list with the fetched list and to eliminate duplicates. If the CurrentTokenId is unknown, the existing list shall be discarded and replaced.
CurrentKey	The current key used to secure the messages. This key is not used directly since the protocol associated with the *PubSubGroup(s)* specifies an algorithm to generate distinct keys for different types of cryptography operations.
FutureKeys	An ordered list of future keys that are used when the *KeyLifetime* elapses. The *SecurityTokenId* associated with the first key in the list is 1 more than the *CurrentTokenId*. All following keys have a SecurityTokenId that is incremented by 1 for every key returned.
TimeToNextKey	The time, in milliseconds, before the *CurrentKey* is expected to expire. If a *Publisher* uses this *Method* to get the keys from an SKS, the *TimeToNextKey* and *KeyLifetime* are used to calculate the time the *Publisher* shall use the next key. The *TimeToNextKey* defines the time when to switch from *CurrentKey* to *FutureKeys* and the *KeyLifetime* defines when to switch from one future key to the next future key. For a *Subscriber* the *TimeToNextKey* and *KeyLifetime* are used to calculate the time the *Subscriber* expects that the *Publishers* use the next key. Due to network latency, out of order delivery and the use of keys for several *Publishers*, a *Subscriber* needs to expect some overlap time where *NetworkMessages* are received that are using the previous or the next key. *TimeToNextKey* and *KeyLifetime* are also used to calculate the time until *Publisher* and *Subscriber* shall fetch new keys.
KeyLifetime	The lifetime of a key in milliseconds. The returned keys may expire earlier if the keys are discarded for some reason. An unplanned key rotation is indicated in the *NetworkMessage* header before the next key is used to give the *Subscriber* some time to fetch new keys. If the *CurrentTokenId* in the message is not recognized, the receiver shall call this *Method* again to get new keys.

Method Result codes

ResultCode	Description
Bad_NotFound	The *SecurityGroupId* is unknown.
Bad_UserAccessDenied	The caller is not allowed to set the keys for the SecurityGroup.
Bad_SecurityModeInsufficient	The communication channel is not using encryption.

9.1.3.4 AddConnection Method

This *Method* is used to add a new *PubSubConnection Object* to the *PublishSubscribe Object*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddConnection (
    [in]  PubSubConnectionDataType    Configuration
    [out] NodeId                      ConnectionId
    );
```

Argument	Description
Configuration	Configuration parameters for the *PubSubConnection*. The parameters and the *PubSubConnectionDataType* are defined in 6.2.6.
ConnectionId	The *NodeId* of the new connection.

Method result codes

ResultCode	Description
Bad_InvalidArgument	The *Server* is not able to apply the name. The name may be too long or may contain invalid characters.
Bad_BrowseNameDuplicated	An *Object* with the name already exists.
Bad_ResourceUnavailable	The *Server* has not enough resources to add the *PubSubConnection Object*.
Bad_UserAccessDenied	The *Session* user is not allowed to create a *PubSubConnection Object*.

9.1.3.5 RemoveConnection Method

This *Method* is used to remove a *PubSubConnection Object* from the *PublishSubscribe Object*.

A successful removal of the *PubSubConnection Object* removes all associated groups, *DataSetWriter* and *DataSetReader Objects*. Before the *Objects* are removed, their state is set to Disabled_0.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
RemoveConnection (
    [in]  NodeId      ConnectionId
    );
```

Argument	Description
ConnectionId	*NodeId* of the *PubSubConnection Object* to remove from the *Server*

Method result codes

ResultCode	Description
Bad_NodeIdUnknown	The *ConnectionId* is unknown.
Bad_UserAccessDenied	The *Session* user is not allowed to delete the *PubSubConnection Object*.

9.1.3.6 HasPubSubConnection

The *HasPubSubConnection ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *HasComponent ReferenceType*.

The *SourceNode* of *References* of this type shall be the *PublishSubscribe Object* defined in 8.4.

The *TargetNode* of this *ReferenceType* shall be an *Object* of type *PubSubConnectionType* defined in 9.1.5.2.

The representation of the *HasPubSubConnection ReferenceType* in the *AddressSpace* is specified in Table 104.

Table 104 – HasPubSubConnection ReferenceType

Attributes	Value		
BrowseName	HasPubSubConnection		
InverseName	PubSubConnectionOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
Subtype of HasComponent defined in IEC 62541-5.			

9.1.4 Published DataSet Model

9.1.4.1 Overview

Figure 39 depicts the *ObjectTypes* of the published *DataSet* model and their components.

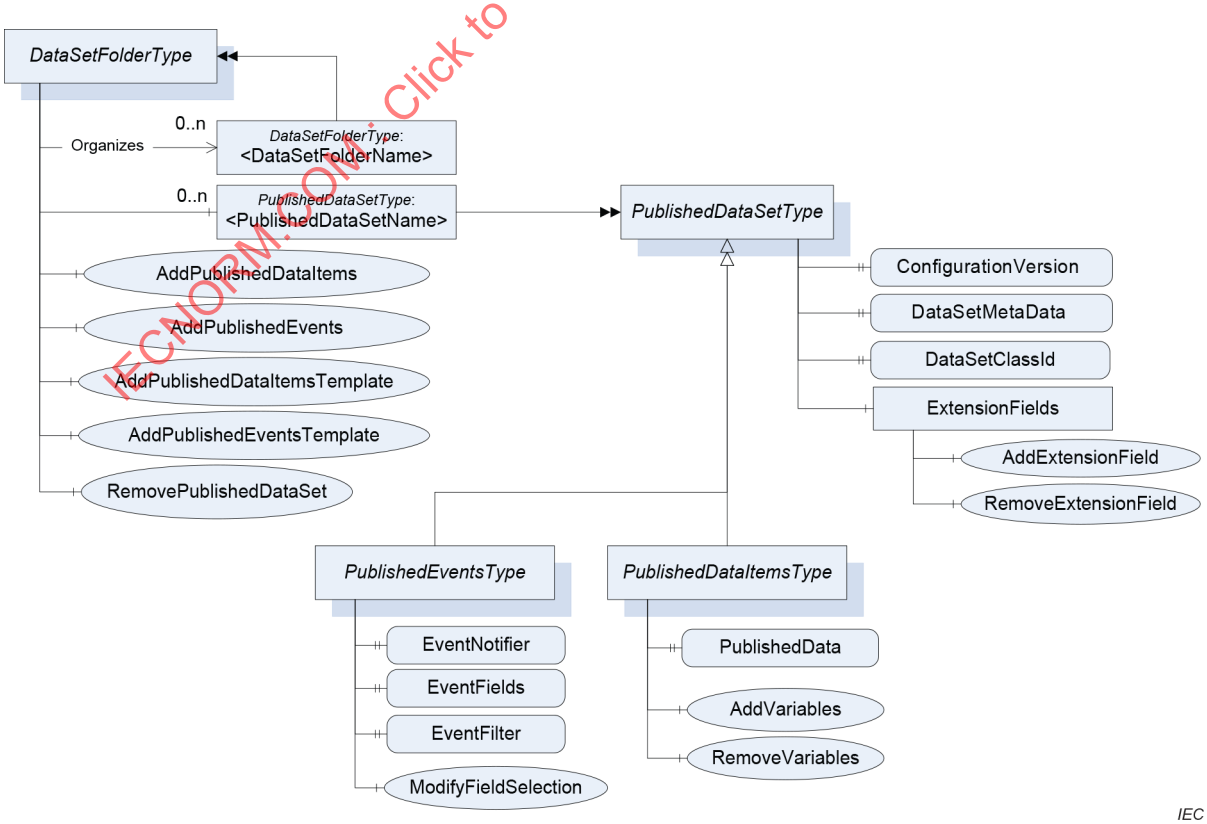


Figure 39 – Published DataSet overview

Instances of the *DataSetFolderType* are used to organize *PublishedDataSetType Objects* in a tree of *DataSetFolders*. The configuration can be made through *Methods* or can be made by product-specific configuration tools.

The *PublishedDataSetType* defines the information necessary for a *Subscriber* to understand and decode *DataSetMessages* received from the *Publisher* for a *DataSet* and to detect changes of the *DataSet* semantic and metadata.

The types derived from the *PublishedDataSetType* define the source of information for a *DataSet* in the OPC UA *Server AddressSpace* like *Variables* or *Events*.

9.1.4.2 Published DataSet

9.1.4.2.1 PublishedDataSetType

This *ObjectType* is the base type for *PublishedDataSets*. It defines the metadata and the configuration version of the *DataSets* sent as *DataSetMessages* through *DataSetWriters*.

The *PublishedDataSetType* is the base type for configurable *DataSets*. Derived types like *PublishedDataItemsType* and *PublishedEventsType* define how to collect the *DataSet* to be published. For *PublishedDataItemsType* this is a list of monitored *Variables*. For *PublishedEventsType* this is an *Event* selection. The list of monitored Variables or the list of selected *EventFields* defines the content and metadata of the *PublishedDataSetType Object*.

If the content of the *DataSet* is defined by a product-specific configuration and the source of the *DataSet* is not known, the *PublishedDataSetType* can be used directly to expose the *PublishedDataSet* in the *AddressSpace* of the *Publisher*.

The *PublishedDataSetType* is formally defined in Table 105.

Table 105 – PublishedDataSetType definition

Attribute	Value				
BrowseName	PublishedDataSetType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5.					
DataSetToWriter	Object	<DataSetWriterName>		DataSetWriterType	Optional Placeholder
HasProperty	Variable	ConfigurationVersion	ConfigurationVersionDataType	PropertyType	Mandatory
HasProperty	Variable	DataSetMetaData	DataSetMetaDataType	PropertyType	Mandatory
HasProperty	Variable	DataSetClassId	Guid	PropertyType	Optional
HasComponent	Object	ExtensionFields		ExtensionFieldsType	Optional

The *PublishedDataSetType ObjectType* is a concrete type and can be used directly. It can be used to expose a *PublishedDataSet* where the data collection is not visible in the *AddressSpace*.

The *Object* has a list of *DataSetWriters*. A *DataSetWriter* sends *DataSetMessages* created from *DataSets* through a *Message Oriented Middleware*. The link between the *PublishedDataSet Object* and a *DataSetWriter* shall be created when an instance of the *DataSetWriterType* is created. The *DataSetWriterType* is defined in 9.1.7.2. If a *DataSetWriter* is created for the *PublishedDataSet*, it is added to the list using the *ReferenceType DataSetToWriter*. The *DataSetToWriter ReferenceType* is defined in 9.1.4.2.5. If a

DataSetWriter for the *PublishedDataSet* is removed from a group, the *Reference* to this *DataSetWriter* shall also be removed from this list. The group model is defined in 9.1.6.

The *Property ConfigurationVersion* is related to configuration of the *DataSet* produced by the *PublishedDataSet Object*. The *PublishedDataSet* parameters affecting the version are defined in the concrete types derived from this base type. The *ConfigurationVersionDataType* and the rules for setting the version are defined in 6.2.2.1.5.

The *Property DataSetMetaData* provides the information necessary to decode *DataSetMessages* on the *Subscriber* side if the *DataSetMessages* are not self-describing. The information in this *Property* is automatically updated if the *ConfigurationVersion* is changed based on *DataSet* configuration change. The *DataSetMetaDataType* is defined in 6.2.2.1.2. The *Name* field in the *DataSetMetaDataType* shall match the name of the *PublishedDataSetType Object* if the *DataSetMetaData* is not based on a *DataSetClass*.

The *MajorVersion* part of the *ConfigurationVersion* contained in the *DataSetMessage* needs to match the *ConfigurationVersion* of the *DataSetMetaData available on the Subscriber* side.

The *DataSetClassId* is the globally unique identifier for a *DataSetClass*. The optional *Property* shall be present if the *DataSetClassId* of the *DataSetMetaData* is not null. If the *DataSetClassId* is set, the *Publisher* shall reject any configuration changes that change the *DataSetMetaData*.

The *ExtensionFields Object* allows the configuration of fields with values to be included in the *DataSet* in case the existing *AddressSpace* of the *Publisher* does not provide the necessary information. The extension fields are added as *Properties* to the *ExtensionFields Object.* For *PublishedDataItemsType* base *PublishedDataSets*, an extension field is included as a *Variable* in the published *DataSet*. For *PublishedEventsType* base *PublishedDataSets*, an extension field is included into the *SelectedFields* for the *DataSet*.

9.1.4.2.2 ExtensionFieldsType

The *ExtensionFieldsType* is formally defined in Table 106. It allows the configuration of fields with values to be included in the *DataSet* in case the existing *AddressSpace* of the *Publisher* does not provide the necessary information.

Table 106 – ExtensionFieldsType definition

Attribute	Value				
BrowseName	ExtensionFieldsType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5.					
HasProperty	Variable	<ExtensionFieldName>	BaseDataType	PropertyType	OptionalPlaceholder
HasComponent	Method	AddExtensionField	Defined in 9.1.4.2.3.		Mandatory
HasComponent	Method	RemoveExtensionField	Defined in 9.1.4.2.4.		Mandatory

The *ExtensionFieldsType ObjectType* is a concrete type and can be used directly.

The configured list of extension fields is exposed through *Properties* and managed through the *Methods AddExtensionField* and *RemoveExtensionField*. An *ExtensionField* is not automatically included in the *DataSet*. The *ExtensionField* shall be added to the *DataSet* after creation.

Metadata that normally appear in message headers can be included in the body by adding extension fields with well-known *QualifiedNames*. These well-known *QualifiedNames* are shown in Table 107. The qualifying namespace is the OPC UA namespace.

Table 107 – Well-Known Extension Field Names

Name	Type	Description
PublisherId	BaseDataType	The *PublisherId* from the *Connection Object*.
DataSetName	String	The *Name* from the *DataSetMetaData*.
DataSetClassId	Guid	The *DataSetClassId* from the *DataSetMetaData*.
MajorVersion	UInt32	The MajorVersion from the ConfigurationVersion
MinorVersion	UInt32	The MinorVersion from the ConfigurationVersion
DataSetWriterId	BaseDataType	The *DataSetWriterId* from the *DataSetWriterTransport Object*.
MessageSequenceNumber	UInt16	The sequence number from the *DataSetMessage*.

If a well-known name is used, the value placed in the message body is dynamically generated from the current settings. The value set in the *AddExtensionField Method* is ignored. Subtypes of *DataSetWriterTransportType* may extend this list.

9.1.4.2.3 AddExtensionField Method

This *Method* is used to add *a Property* to the *Object ExtensionFields*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddExtensionField (
   [in]  QualifiedName   FieldName
   [in]  BaseDataType    FieldValue
   [out] NodeId          FieldId
   );
```

Argument	Description
FieldName	Name of the field to add.
FieldValue	The value of the field to add.
FieldId	The *NodeId* of the added field *Property*.

Method Result codes

ResultCode	Description
Bad_NodeIdExists	A field with the name already exists.
Bad_InvalidArgument	The *Server* is not able to apply the *Name*. The *Name* may be too long or may contain invalid characters.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

9.1.4.2.4 RemoveExtensionField Method

This *Method* is used to remove *a Property* from the *Object ExtensionFields*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
RemoveExtensionField (
    [in]  NodeId          FieldId
    );
```

Argument	Description
FieldId	The *NodeId* field *Property* to remove.

Method Result codes

ResultCode	Description
Bad_NodeIdUnknown	A field with the *NodeId* does not exist.
Bad_NodeIdInvalid	The *FieldId* is not a *NodeId* of a *Property* of the *ExtensionFieldsType Object*.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

9.1.4.2.5 DataSetToWriter

The *DataSetToWriter ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *HierarchicalReferences ReferenceType*.

The *SourceNode* of *References* of this type shall be an *Object* of *ObjectType PublishedDataSetType* or an *ObjectType* that is a subtype of *PublishedDataSetType* defined in 9.1.4.2.1.

The *TargetNode* of this *ReferenceType* shall be an *Object* of the *ObjectType DataSetWriterType* defined in 9.1.7.1.

Each *DataSetWriter Object* shall be the *TargetNode* of exactly one *DataSetToWriter Reference*.

Servers shall provide the inverse *Reference* that relates a *DataSetWriter Object* back to a *PublishedDataSetType* Object.

The representation of the *DataSetToWriter ReferenceType* in the *AddressSpace* is specified in Table 108.

Table 108 – DataSetToWriter ReferenceType

Attributes	Value		
BrowseName	DataSetToWriter		
InverseName	WriterToDataSet		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
Subtype of HierarchicalReferences defined in IEC 62541-5.			

9.1.4.3 Published Data Items

9.1.4.3.1 PublishedDataItemsType

The *PublishedDataItemsType* is used to select a list of OPC UA *Variables* as the source for the creation of *DataSets* sent through one or more *DataSetWriters*.

The *PublishedDataItemsType* is formally defined Table 109.

Table 109 – PublishedDataItemsType definition

Attribute	Value				
BrowseName	PublishedDataItemsType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PublishedDataSetType defined in 9.1.4.2.					
HasProperty	Variable	PublishedData	PublishedVariable DataType[]	PropertyType	Mandatory
HasComponent	Method	AddVariables	Defined in 9.1.4.3.2.		Optional
HasComponent	Method	RemoveVariables	Defined in 9.1.4.3.3.		Optional

The *PublishedDataItemsType ObjectType* is a concrete type and can be used directly.

The *PublishedData* is defined in 6.2.2.6.1. Existing entries in the array can be changed by writing the new settings to the *Variable Value*. A new *Value* shall be rejected with Bad_OutOfRange if the array size would be changed. Entries in the array can be added and removed with the *Methods AddVariables* and *RemoveVariables*.

The index into the list of entries in the *PublishedData* has an important role for *Subscribers* and for configuration tools. It is used as a handle to reference the entry in configuration actions like *RemoveVariable* or the *Value* 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.

9.1.4.3.2 AddVariables Method

This *Method* is used to add *Variables* to the *PublishedData Property*. The *PublishedData* contains a list of published *Variables* of a *PublishedDataItemsType Object*. The information provided in the input Arguments and information available for the added Variables is also used to create the content of the *DataSetMetaData Property*. The mapping to the *DataSetMetaData* is described for the input *Arguments*.

Variables shall be added at the end of the list in *PublishedData*. This ensures that *Subscribers* are only affected by the change if they are interested in the added *Variables*.

If at least one *Variable* was added to the *PublishedData*, the *MinorVersion* of the *ConfigurationVersion* shall be updated. The *ConfigurationVersionDataType* and the rules for setting the version are defined in 6.2.2.1.5.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddVariables (
   [in]  ConfigurationVersionDataType      ConfigurationVersion
   [in]  String[]                          FieldNameAliases
   [in]  Boolean[]                         PromotedFields
   [in]  PublishedVariableDataType[]       VariablesToAdd
   [out] ConfigurationVersionDataType      NewConfigurationVersion
   [out] StatusCode[]                      AddResults
   );
```

Argument	Description
ConfigurationVersion	Configuration version of the *DataSet*. The configuration version shall match the entire current configuration version of the *Object* when the *Method* call is processed. If it does not match, the result Bad_InvalidState shall be returned. The *ConfigurationVersionDataType* is defined in 6.2.2.1.5.
FieldNameAliases	The names assigned to the selected *Variables* for the fields in the *DataSetMetaData* and in the *DataSetMessages* for tagged message encoding. The size and the order of the array shall match the *VariablesToAdd*. The string shall be used to set the name field in the *FieldMetaData* that is part of the *DataSetMetaData*.
PromotedFields	The flags indicating if the corresponding field is promoted to the *DataSetMessage* header. The size and the order of the array shall match the *VariablesToAdd*. The flag is used to set the *PromotedField* flag in the *fieldFlags* parameter in the *FieldMetaData*.
VariablesToAdd	Array of *Variables* to add to *PublishedData* and the related configuration settings. Successfully added variables are appended to the end of the list of published variables configured in the *PublishedData Property*. Failed variables are not added to the list. The *PublishedVariableDataType* is defined in 6.2.2.6.1. The parameters *builtInType*, *dataType*, *valueRank* and *arrayDimensions* of the *FieldMetaData* are filled from corresponding *Variable Attributes*.
NewConfigurationVersion	Returns the new configuration version of the *PublishedDataSet*.
AddResults	The result codes for the variables to add. Variables exceeding the maximum number of items in the *Object* are rejected with Bad_TooManyVariables.

Method Result codes

ResultCode	Description
Bad_NothingToDo	An empty list of variables was provided.
Bad_InvalidState	The configuration version did not match the current state of the object.
Bad_NotWritable	The *DataSet* is based on a *DataSetClass* and the size of the *PublishedData* array cannot be changed.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the object.

Operation Result codes

ResultCode	Description
Bad_NodeIdInvalid	See IEC 62541-4 for the description of this result code.
Bad_NodeIdUnknown	See IEC 62541-4 for the description of this result code.
Bad_IndexRangeInvalid	See IEC 62541-4 for the description of this result code.
Bad_IndexRangeNoData	See IEC 62541-4 for the description of this result code. If the *ArrayDimensions* have a fixed length that cannot change and no data exists within the range of indexes specified, Bad_IndexRangeNoData is returned in *AddVariables*. Otherwise, if the length of the array is dynamic, the *Publisher* shall insert this status in a *DataSet* if no data exists within the range.
Bad_TooManyVariables	The *Publisher* has reached its maximum number of items for the *PublishedDataItemsType* object.

9.1.4.3.3 RemoveVariables Method

This *Method* is used to remove *Variables* from the *PublishedData* list. It contains the list of published *Variables* of a *PublishedDataItemsType Object*.

A caller shall read the current Values of *PublishedData* and *ConfigurationVersion* prior to calling this *Method*, to ensure the use of the correct index of the *Variables* that are being removed.

If at least one *Variable* was successfully removed from the *PublishedData*, 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.

The order of the remaining *Variables* in the *PublishedData* shall be preserved.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
RemoveVariables (
    [in]  ConfigurationVersionDataType      ConfigurationVersion
    [in]  UInt32[]                          VariablesToRemove
    [out] ConfigurationVersionDataType      NewConfigurationVersion
    [out] StatusCode[]                      RemoveResults
    );
```

Argument	Description
ConfigurationVersion	Configuration version of the *DataSet*. The configuration version and the indices provided through *VariablesToRemove* shall match the entire current configuration version of the *Object* when the *Method* call is processed. If it does not match, the result Bad_InvalidState shall be returned. The *ConfigurationVersionDataType* is defined in 6.2.2.1.5.
VariablesToRemove	Array of indices of Variables to remove from the list of *Variables* configured in *PublishedData* of the *PublishedDataItemsType*. This matches the list of fields configured in the *DataSetMetaData* of the *PublishedDataSetType*.
NewConfigurationVersion	Returns the new configuration version of the *DataSet*.
RemoveResults	The result codes for each of the variables to remove.

Method Result codes

ResultCode	Description
Bad_NothingToDo	An empty list of variables was provided.
Bad_InvalidState	The configuration version did not match the current state of the *Object*.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

Operation Result Codes

ResultCode	Description
Bad_InvalidArgument	The provided index was invalid.

9.1.4.4 Published Events

9.1.4.4.1 PublishedEventsType

This *PublishedDataSetType* is used to configure the collection of OPC UA *Events*.

The *PublishedEventsType* is formally defined in Table 110.

Table 110 – PublishedEventsType definition

Attribute	Value				
BrowseName	PublishedEventsType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PublishedDataSetType defined in 9.1.4.2.1.					
HasProperty	Variable	EventNotifier	NodeId	PropertyType	Mandatory
HasProperty	Variable	SelectedFields	SimpleAttributeOperand[]	PropertyType	Mandatory
HasProperty	Variable	Filter	ContentFilter	PropertyType	Mandatory
HasComponent	Method	ModifyFieldSelection	Defined in 9.1.4.4.2.		Optional

The *PublishedEventsType ObjectType* is a concrete type and can be used directly.

The *EventNotifier* is defined in 6.2.2.7.1.

The *SelectedFields* is defined in 6.2.2.7.2.

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 *Property ConfigurationVersion* is defined in the base *ObjectType PublishedDataSetType*.

The *Filter* is defined in 6.2.2.7.3. A change of the *Filter* does not affect the *ConfigurationVersion* since the content of the *DataSet* does not change.

9.1.4.4.2 ModifyFieldSelection Method

This *Method* is used to modify the event field selection of a *PublishedEventsType Object*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
ModifyFieldSelection (
    [in]  ConfigurationVersionDataType      ConfigurationVersion
    [in]  String[]                          FieldNameAliases
    [in]  Boolean[]                         PromotedFields
    [in]  SimpleAttributeOperand[]          SelectedFields
    [out] ConfigurationVersionDataType      NewConfigurationVersion
    );
```

Argument	Description
ConfigurationVersion	Configuration version of the *DataSet*. The configuration version shall match the entire current configuration version of the *Object* when the *Method* call is processed. If it does not match, the result Bad_InvalidState shall be returned. The *ConfigurationVersionDataType* is defined in 6.2.2.1.5.
FieldNameAliases	The names assigned to the selected fields in the *DataSetMetaData* and in the *DataSetMessages* for tagged message encoding. The size and the order of the array shall match the *SelectedFields*. The string is used to set the name field in the *FieldMetaData* that is part of the *DataSetMetaData*.
PromotedFields	The flags indicating if the corresponding field is promoted to the *DataSetMessage* header. The size and the order of the array shall match the *SelectedFields*. The flag is used to set the corresponding field in the *FieldMetaData* that is part of the *DataSetMetaData*.
SelectedFields	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. A change to the selected fields requires a change of the *ConfigurationVersion*.
NewConfigurationVersion	Return the new configuration version of the *DataSet*.

Method Result codes

ResultCode	Description
Bad_InvalidState	The configuration version did not match the current state of the *Object*.
Bad_EventFilterInvalid	The event filter is not valid.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

9.1.4.5 DataSet Folder

9.1.4.5.1 DataSetFolderType

The *DataSetFolderType* is formally defined Table 111.

Table 111 – DataSetFolderType definition

Attribute	Value				
BrowseName	DataSetFolderType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of FolderType defined in IEC 62541-5.					
Organizes	Object	<DataSetFolderName>		DataSetFolderType	OptionalPlaceholder
HasComponent	Object	<PublishedDataSetName>		PublishedDataSetType	OptionalPlaceholder
HasComponent	Method	AddPublishedDataItems	Defined in 9.1.4.5.2.		Optional
HasComponent	Method	AddPublishedEvents	Defined in 9.1.4.5.3.		Optional
HasComponent	Method	AddPublishedDataItemsTemplate	Defined in 9.1.4.5.4.		Optional
HasComponent	Method	AddPublishedEventsTemplate	Defined in 9.1.4.5.5.		Optional
HasComponent	Method	RemovePublishedDataSet	Defined in 9.1.4.5.6.		Optional
HasComponent	Method	AddDataSetFolder	Defined in 9.1.4.5.7.		Optional
HasComponent	Method	RemoveDataSetFolder	Defined in 9.1.4.5.8.		Optional

The *DataSetFolderType ObjectType* is a concrete type and can be used directly.

Instances of the *DataSetFolderType* can contain *PublishedDataSets* or other instances of the *DataSetFolderType*. This can be used to build a tree of *Folder Objects* used to group the configured *PublishedDataSets*.

The *PublishedDataSetType Objects* are added as components to the instance of the *DataSetFolderType*. An instance of a *PublishedDataSetType* is referenced only from one *DataSetFolder*. If the *DataSetFolder* is deleted, all referenced *PublishedDataSetType Objects* are deleted with the folder.

PublishedDataSetType Objects may be configured with product-specific configuration tools or added and removed through the *Methods AddPublishedDataItems*, *AddPublishedEvents* and *RemovePublishedDataSet*. The *PublishedDataSetType* is defined in 9.1.4.2.1.

9.1.4.5.2 AddPublishedDataItems Method

This *Method* is used to create a *PublishedDataSets Object* of type *PublishedDataItemsType* and to add it to the *DataSetFolderType Object*. The configuration parameters provided with this *Method* are further described in the *PublishedDataItemsType* defined in 9.1.4.3.1 and the *PublishedDataSetType* defined in 9.1.4.2.

The settings in the *VariablesToAdd* are used to configure the data acquisition for the *DataSet* and are used to initialize the *PublishedData Property* of the *PublishedDataItemsType*.

The *DataSetMetaData* of the *PublishedDataSetType* is created from meta-data of the *Variables* referenced in *VariablesToAdd* and the settings in *FieldNameAliases* and *FieldFlags*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddPublishedDataItems (
    [in]  String                              Name
    [in]  String[]                            FieldNameAliases
    [in]  DataSetFieldFlags[]                 FieldFlags
    [in]  PublishedVariableDataType[]         VariablesToAdd
    [out] NodeId                              DataSetNodeId
    [out] ConfigurationVersionDataType        ConfigurationVersion
    [out] StatusCode[]                        AddResults
    );
```

Argument	Description
Name	Name of the *Object* to create.
FieldNameAliases	The names assigned to the selected *Variables* for the fields in the *DataSetMetaData* and in the *DataSetMessages* for tagged message encoding. The size and the order of the array shall match the *VariablesToAdd*. The string shall be used to set the name field in the *FieldMetaData* that is part of the *DataSetMetaData*. The name shall be unique in the *DataSet*.
FieldFlags	The field flags assigned to the selected *Variables* for the fields in the *DataSetMetaData*. The size and the order of the array shall match the *VariablesToAdd*. The flag is used to set the corresponding field in the *FieldMetaData* that is part of the *DataSetMetaData*.
VariablesToAdd	Array of Variables to add to PublishedData and the related configuration settings. Successfully added variables are appended to the end of the list of published variables configured in the *PublishedData Property*. Failed variables are not added to the list. The *PublishedVariableDataType* is defined in 6.2.2.6.1.
DataSetNodeId	*NodeId* of the created *PublishedDataSets Object*.
ConfigurationVersion	Returns the initial configuration version of the *DataSet*.
AddResults	The result codes for the variables to add. Variables exceeding the maximum number of items in the *Object* are rejected with Bad_TooManyMonitoredItems.

Method Result codes

ResultCode	Description
Bad_InvalidState	The current state of the *Object* does not allow a configuration change.
Bad_BrowseNameDuplicated	A data set *Object* with the name already exists.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.
Bad_InvalidArgument	The *Server* is not able to apply the *Name*. The *Name* may be too long or may contain invalid characters.

Operation Result codes

ResultCode	Description
Bad_NodeIdInvalid	See IEC 62541-4 for the description of this result code.
Bad_NodeIdUnknown	See IEC 62541-4 for the description of this result code.
Bad_IndexRangeInvalid	See IEC 62541-4 for the description of this result code.
Bad_IndexRangeNoData	See IEC 62541-4 for the description of this result code. If the *ArrayDimensions* have a fixed length that cannot change and no data exists within the range of indexes specified, Bad_IndexRangeNoData is returned in *AddVariables*. Otherwise if the length of the array is dynamic, the *Publisher* shall insert this status in a *DataSet* if no data exists within the range.
Bad_TooManyMonitoredItems	The *Server* has reached its maximum number of items for the PublishedDataItemsType object.
Bad_DuplicateName	The passed field name alias already exists.

9.1.4.5.3 AddPublishedEvents Method

This *Method* is used to add a *PublishedEventsType Object* to the *DataSetFolderType Object*. The configuration parameters provided with this *Method* are further described in the *PublishedEventsType* defined in 9.1.4.4.1 and the *PublishedDataSetType* defined in 9.1.4.2.

The settings in the *EventNotifier*, *SelectedFields* and *Filter* are used to configure the data acquisition for the *DataSet* and are used to initialize the corresponding *Properties* of the *PublishedEventsType*.

The *DataSetMetaData* of the *PublishedDataSetType* is created from metadata of the selected *Event* fields and the settings in *FieldNameAliases* and *FieldFlags*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddPublishedEvents (
    [in]  String                          Name
    [in]  NodeId                          EventNotifier
    [in]  String[]                        FieldNameAliases
    [in]  DataSetFieldFlags[]             FieldFlags
    [in]  SimpleAttributeOperand[]        SelectedFields
    [in]  ContentFilter                   Filter
    [out] ConfigurationVersionDataType    ConfigurationVersion
    [out] NodeId                          DataSetNodeId
    );
```

Argument	Description
Name	Name of the *DataSet Object* to create.
EventNotifier	The *NodeId* of the *Object* in the event notifier tree of the OPC UA *Server* from which *Events* are collected.
FieldNameAliases	The names assigned to the selected fields in the *DataSetMetaData* and in the *DataSetMessages* for tagged message encoding. The size and the order of the array shall match the *SelectedFields*. The string is used to set the name field in the *FieldMetaData* that is part of the *DataSetMetaData*.
FieldFlags	The field flags assigned to the selected fields in the *DataSetMetaData*. The size and the order of the array shall match the *SelectedFields*. The flag is used to set the corresponding field in the *FieldMetaData* that is part of the *DataSetMetaData*.
SelectedFields	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.
Filter	The filter applied to the *Events*. It allows the reduction of the *DataSets* generated from *Events* through a filter like filtering for a certain *EventType*. The *ContentFilter DataType* is defined in IEC 62541-4.
ConfigurationVersion	Returns the initial configuration version of the *PublishedDataSets*.
DataSetNodeId	*NodeId* of the created *PublishedDataSets Object*.

Method Result codes

ResultCode	Description
Bad_InvalidState	The current state of the *Object* does not allow a configuration change.
Bad_NodeIdExists	A data set *Object* with the name already exists.
Bad_NodeIdUnknown	The *Event* notifier node is not known in the *Server*.
Bad_EventFilterInvalid	The *Event* filter is not valid.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.
Bad_InvalidArgument	The *Server* is not able to apply the *Name*. The *Name* may be too long or may contain invalid characters.

9.1.4.5.4 AddPublishedDataItemsTemplate Method

This *Method* is used to create a *PublishedDataSets Object* of type *PublishedDataItemsType* and to add it to the *DataSetFolderType Object*. The configuration parameters provided with this *Method* are further described in the *PublishedDataItemsType* defined in 9.1.4.3.1 and the *PublishedDataSetType* defined in 9.1.4.2.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddPublishedDataItemsTemplate (
    [in]  String                              Name
    [in]  DataSetMetaDataType                 DataSetMetaData
    [in]  PublishedVariableDataType[]         VariablesToAdd
    [out] NodeId                              DataSetNodeId
    [out] StatusCode[]                        AddResults
    );
```

Argument	Description
Name	Name of the *Object* to create.
DataSetMetaData	The *DataSetMetaData* predefined by the caller. The initial setting shall not be changed by the *Publisher*. If the *dataSetClassId* of the *DataSetMetaData* is not null, the *DataSetClassId Property* of the *PublishedDataSetType* shall be created and initialized with the *dataSetClassId* value. The name of the *PublishedDataSet Object* is defined by the name in the *DataSetMetaData*.
VariablesToAdd	Array of variable settings for the data acquisition for the fields in the *DataSetMetaData*. The size of the array shall match the size of the *fields* array in the *DataSetMetaData*. The *substituteValue* in the *VariablesToAdd* entries shall be configured. For failed variables the *publishedVariable* field of entry in the resulting *PublishedData Property* shall be set to a null *NodeId*. If there is no *Variable* available for a field in the *DataSetMetaData* the *publishedVariable* field for the entry shall be set to a null NodeId. The *PublishedVariableDataType* is defined in 6.2.2.6.1.
DataSetNodeId	*NodeId* of the created *PublishedDataSets Object*.
AddResults	The result codes for the variables to add.

Method Result codes

ResultCode	Description
Bad_InvalidState	The current state of the *Object* does not allow a configuration change.
Bad_BrowseNameDuplicated	A data set *Object* with the name already exists.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.
Bad_InvalidArgument	The *VariablesToAdd* parameter does not match the array size of the fields in the *DataSetMetaData* or the configuration of the *VariablesToAdd* contains invalid settings.
Bad_TooManyMonitoredItems	The *Object* cannot be created since the number of items in the *PublishedDataSet* exceeds the capabilities of the *Publisher*.

Operation Result codes

ResultCode	Description
Bad_NodeIdInvalid	See IEC 62541-4 for the description of this result code.
Bad_NodeIdUnknown	See IEC 62541-4 for the description of this result code.
Bad_IndexRangeInvalid	See IEC 62541-4 for the description of this result code.
Bad_IndexRangeNoData	See IEC 62541-4 for the description of this result code. If the *ArrayDimensions* have a fixed length that cannot change and no data exists within the range of indexes specified, Bad_IndexRangeNoData is returned in *AddVariables*. Otherwise if the length of the array is dynamic, the *Publisher* shall insert this status in a *DataSet* if no data exists within the range.
Bad_TooManyMonitoredItems	The *Server* has reached its maximum number of items for the PublishedDataItemsType *Object*.
Bad_DuplicateName	The passed field name alias already exists.

9.1.4.5.5 AddPublishedEventsTemplate Method

This *Method* is used to add a *PublishedEventsType Object* to the *DataSetFolderType Object*. The configuration parameters provided with this *Method* are further described in the *PublishedEventsType* defined in 9.1.4.4.1 and the *PublishedDataSetType* defined in 9.1.4.2.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddPublishedEventsTemplate (
    [in]  String                        Name
    [in]  DataSetMetaDataType           DataSetMetaData
    [in]  NodeId                        EventNotifier
    [in]  SimpleAttributeOperand[]      SelectedFields
    [in]  ContentFilter                 Filter
    [out] NodeId                        DataSetNodeId
    );
```

Argument	Description
Name	Name of the *Object* to create.
DataSetMetaData	The *DataSetMetaData* predefined by the caller. The initial setting shall not be changed by the *Publisher*. If the *dataSetClassId* of the *DataSetMetaData* is not null, the *DataSetClassId Property* of the *PublishedDataSetType* shall be created and initialized with the *dataSetClassId* value. The name of the *PublishedDataSet Object* is defined by the name in the *DataSetMetaData*.
EventNotifier	The *NodeId* of the *Object* in the event notifier tree of the OPC UA *Server* from which *Events* are collected.
SelectedFields	The selection of Event Fields contained in the *DataSet* generated for an *Event* and sent through the *DataSetWriter*. The size of the array shall match the size of the fields array in the *DataSetMetaData*. If there is no *Event* field available for a field in the *DataSetMetaData*, the *browsePath* field for the *SimpleAttributeOperand* entry shall be set to null. The *SimpleAttributeOperand DataType* is defined in IEC 62541-4.
Filter	The filter applied to the *Events*. It allows the reduction of the *DataSets* generated from *Events* through a filter, like filtering for a certain *EventType*. The *ContentFilter DataType* is defined in IEC 62541-4.
DataSetNodeId	*NodeId* of the created *PublishedDataSets Object*.

Method Result codes

ResultCode	Description
Bad_InvalidState	The current state of the *Object* does not allow a configuration change.
Bad_NodeIdExists	A *DataSet Object* with the name already exists.
Bad_NodeIdUnknown	The *Event* notifier node is not known in the *Server*.
Bad_EventFilterInvalid	The *Event* filter is not valid.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.
Bad_InvalidArgument	The *Server* is not able to apply the *Name*. The *Name* may be too long or may contain invalid characters.

9.1.4.5.6 RemovePublishedDataSet Method

This *Method* is used to remove a *PublishedDataSetType Object* from the *DataSetFolderType Object*.

A successful removal of the *PublishedDataSetType Object* removes all associated *DataSetWriter Objects*. Before the *Objects* are removed, their state is changed to Disabled_0

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
RemovePublishedDataSet (
   [in]  NodeId      DataSetNodeId
   );
```

Argument	Description
DataSetNodeId	*NodeId* of the *PublishedDataSets Object* to remove from the *Server*. The *DataSetId* is either returned by the *AddPublishedDataItems* or *AddPublishedEvents Methods* or can be discovered by browsing the list of configured *PublishedDataSets* in the *PublishSubscribe Object*.

Method Result codes

ResultCode	Description
Bad_NodeIdUnknown	The *DataSetNodeId* is unknown.
Bad_UserAccessDenied	The *Session* user is not allowed to delete a *PublishedDataSetType*.

9.1.4.5.7 AddDataSetFolder Method

This *Method* is used to add a *DataSetFolderType Object* to a *DataSetFolderType Object*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddDataSetFolder (
   [in]  String      Name
   [out] NodeId      DataSetFolderNodeId
   );
```

Argument	Description
Name	Name of the *Object* to create.
DataSetFolderNodeId	*NodeId* of the created *DataSetFolderType Object*.

Method Result codes

ResultCode	Description
Bad_BrowseNameDuplicated	A folder *Object* with the name already exists.
Bad_InvalidArgument	The *Server* is not able to apply the *Name*. The *Name* may be too long or may contain invalid characters.
Bad_UserAccessDenied	The *Session* user is not allowed to add a folder.

9.1.4.5.8 RemoveDataSetFolder Method

This *Method* is used to remove a *DataSetFolderType Object* from the parent *DataSetFolderType Object*.

A successful removal of the *DataSetFolderType Object* removes all associated *PublishedDataSetType Objects* and their associated *DataSetWriter Objects*. Before the *Objects* are removed, their state is changed to Disabled_0.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
RemoveDataSetFolder (
   [in]  NodeId      DataSetFolderNodeId
   );
```

Argument	Description
DataSetFolderNodeId	*NodeId* of the *DataSetFolderType Object* to remove from the *Server*.

Method Result codes

ResultCode	Description
Bad_NodeIdUnknown	The *DataSetFolderNodeId* is unknown.
Bad_UserAccessDenied	The *Session* user is not allowed to delete a data set.

9.1.5 Connection Model

9.1.5.1 Overview

Figure 40 depicts the *ObjectType* for the *PubSub* connection model and its components and the relations to other parts of the model.

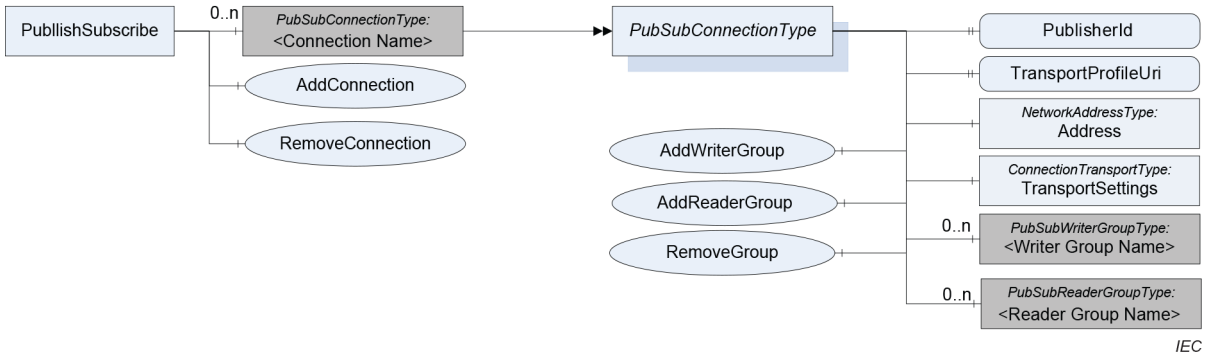


Figure 40 – PubSubConnectionType overview

9.1.5.2 PubSubConnectionType

This *ObjectType* is a concrete type for *Objects* representing *PubSubConnections*. A *PubSubConnection* is a combination of protocol selection, protocol settings and addressing information. The *PubSubConnectionType* is formally defined in Table 112.

Table 112 – PubSubConnectionType definition

Attribute	Value				
BrowseName	PubSubConnectionType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5.					
HasProperty	Variable	PublisherId	BaseDataType	PropertyType	Mandatory
HasComponent	Variable	TransportProfileUri	String	SelectionListType	Mandatory
HasProperty	Variable	ConnectionProperties	KeyValuePair[]	PropertyType	Mandatory
HasComponent	Object	Address		NetworkAddressType	Mandatory
HasComponent	Object	TransportSettings		ConnectionTransportType	Optional
HasComponent	Object	<WriterGroupName>		WriterGroupType	OptionalPlaceholder
HasComponent	Object	<ReaderGroupName>		ReaderGroupType	OptionalPlaceholder
HasComponent	Object	Status		PubSubStatusType	Mandatory
HasComponent	Object	Diagnostics		PubSubDiagnostics ConnectionType	Optional
HasComponent	Method	AddWriterGroup	Defined in 9.1.5.3.		Optional
HasComponent	Method	AddReaderGroup	Defined in 9.1.5.4.		Optional
HasComponent	Method	RemoveGroup	Defined in 9.1.5.5.		Optional

The *PublisherId* is defined in 6.2.6.1.

The *TransportProfileUri* is defined in 6.2.6.2. The *Property* is initialized with the default transport protocol for the *Address* during the creation of the connection. The *SelectionValues Property* of the *SelectionListType* shall contain the list of supported *TransportProfileUris*. The *SelectionListType* is defined in IEC 62541-5.

The *ConnectionProperties* is defined in 6.2.6.4.

The *Address* is defined in 6.2.6.3. The abstract *NetworkAddressType* is defined in A.3.1. The default type used for concrete instances is the *NetworkAddressUrlType* defined in A.3.2. It represents the *Address* in the form of a URL *String*.

The transport protocol mapping specific settings are provided in the optional *Object TransportSettings*. The *ConnectionTransportType* is defined in 9.1.5.6. The *Object* shall be present if the transport protocol mapping defines specific parameters.

The configured *WriterGroup* and *ReaderGroup Objects* are added as components to the instance of the *PubSubConnectionType*. *PubSubGroup Objects* may be configured with product-specific configuration tools or added and removed through the OPC UA *Methods AddWriterGroup, AddReaderGroup* and *RemoveGroup*.

The *Status Object* provides the current operational status of the connection. The *PubSubStatusType* is defined in 9.1.10. The state machine for the status and the relation to other *PubSub Objects* like *PublishSubscribe*, *PubSubGroup*, *DataSetWriter* and *DataSetReader* are defined in 6.2.1.

The *Diagnostics Object* provides the current diagnostic information for a *PubSubConnectionType Object*. The *PubSubDiagnosticsConnectionType* is defined in 9.1.11.8.

9.1.5.3 AddWriterGroup Method

This *Method* is used to add a new *WriterGroup Object* to an instance of the *PubSubConnection*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddWriterGroup (
    [in]  WriterGroupDataType   Configuration
    [out] NodeId                GroupId
    );
```

Argument	Description
Configuration	Configuration parameters for the *WriterGroup*. The parameters and the *WriterGroupDataType* are defined in 6.2.5.
GroupId	The *NodeId* of the new *WriterGroup Object*.

Method Result codes

ResultCode	Description
Bad_InvalidArgument	The *Server* is not able to apply the *GroupName*. The name may be too long or may contain invalid characters.
Bad_BrowseNameDuplicated	An *Object* with the name already exists in the connection.
Bad_ResourceUnavailable	The *Server* does not have enough resources to add the group.
Bad_UserAccessDenied	The *Session* user does not have rights to create the group.

9.1.5.4 AddReaderGroup Method

This *Method* is used to add a new *ReaderGroup Object* to an instance of the *PubSubConnection*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddReaderGroup (
   [in]  ReaderGroupDataType    Configuration
   [out] NodeId                 GroupId
   );
```

Argument	Description
Configuration	Configuration parameters for the *ReaderGroup*. The parameters and the *ReaderGroupDataType* are defined in 6.2.7.
GroupId	The *NodeId* of the new *ReaderGroup Object*.

Method Result codes

ResultCode	Description
Bad_InvalidArgument	The *Server* is not able to apply the *GroupName*. The name may be too long or may contain invalid characters.
Bad_BrowseNameDuplicated	An *Object* with the name already exists in the connection.
Bad_ResourceUnavailable	The *Server* does not have enough resources to add the group.
Bad_UserAccessDenied	The *Session* user does not have rights to create the group.

9.1.5.5 RemoveGroup Method

This *Method* is used to remove a *PubSubGroup Object* from the connection.

A successful removal of the *PubSubGroup Object* removes all associated *DataSetWriter* or *DataSetReader Objects*. Before the *Objects* are removed, their state is set to Disabled_0.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
RemoveGroup (
   [in]  NodeId    GroupId
   );
```

Argument	Description
GroupId	*NodeId* of the group to remove from the connection

Method Result codes

ResultCode	Description
Bad_NodeIdUnknown	The *GroupId* is unknown.
Bad_UserAccessDenied	The *Session* user does not have rights to delete the group.

9.1.5.6 ConnectionTransportType

This *ObjectType* is the abstract base type for *Objects* representing transport protocol mapping specific settings for *PubSubConnections*. The *ConnectionTransportType* is formally defined in Table 113.

Table 113 – ConnectionTransportType definition

Attribute	Value				
BrowseName	ConnectionTransportType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType					

9.1.6 Group Model

9.1.6.1 Overview

Figure 41 depicts the *ObjectType* for the *PubSub* group model and its components and the relations to other parts of the model.

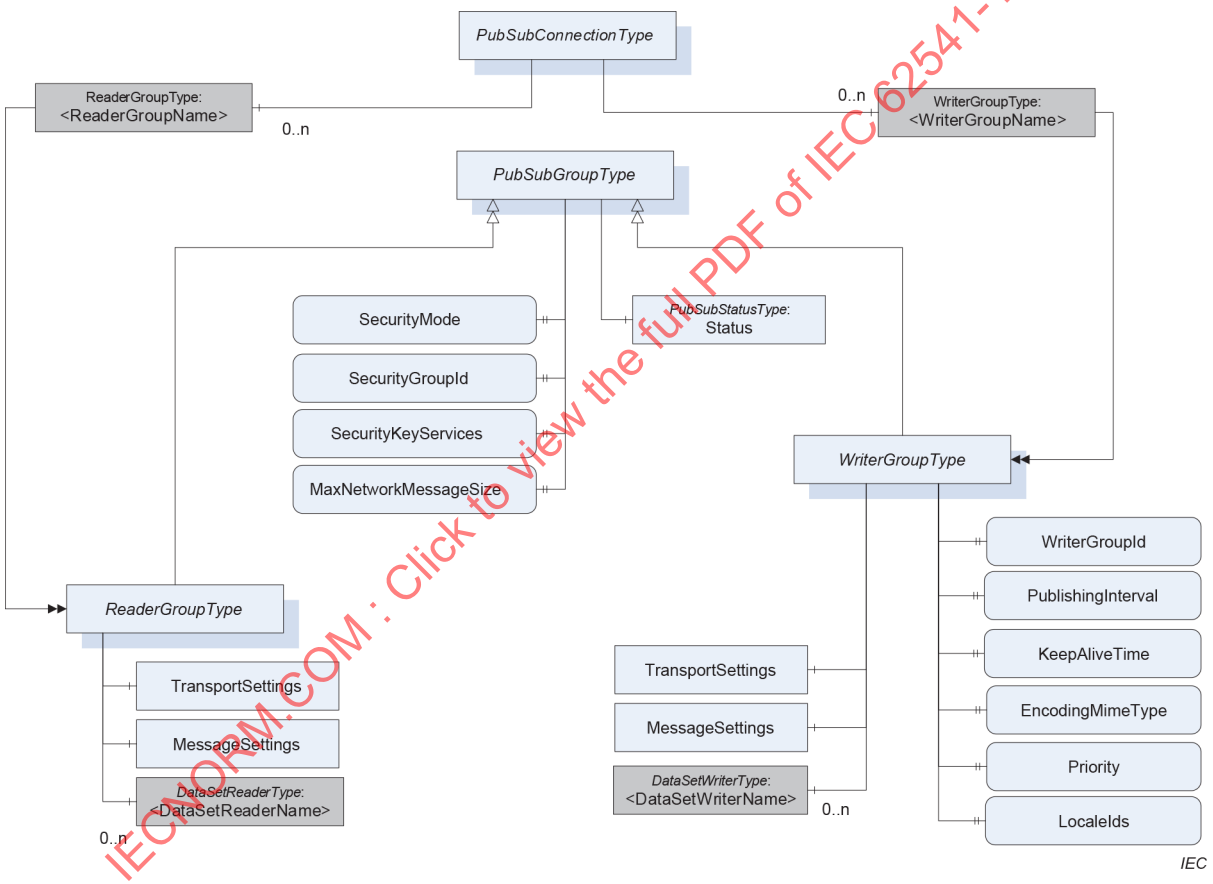


Figure 41 – PubSubGroupType overview

9.1.6.2 PubSubGroupType

This *ObjectType* is the abstract base type for *Objects* representing communication groupings for *PubSub* connections. The *PubSubGroupType* is formally defined in Table 114.

Table 114 – PubSubGroupType definition

Attribute	Value				
BrowseName	PubSubGroupType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5.					
HasProperty	Variable	SecurityMode	MessageSecurityMode	PropertyType	Mandatory
HasProperty	Variable	SecurityGroupId	String	PropertyType	Optional
HasProperty	Variable	SecurityKeyServices	EndpointDescription[]	PropertyType	Optional
HasProperty	Variable	MaxNetworkMessageSize	UInt32	PropertyType	Mandatory
HasProperty	Variable	GroupProperties	KeyValuePair[]	PropertyType	Mandatory
HasComponent	Object	Status		PubSubStatusType	Mandatory

The *SecurityMode* is defined in 6.2.4.2.

The *SecurityGroupId* is defined in 6.2.4.3. If the *SecurityMode* is not NONE_1, the *Property* shall provide the *SecurityGroupId*. The value of the *Property* is null or the *Property* is not present if the *SecurityMode* is NONE_1.

The *SecurityKeyServices* parameter is defined in 6.2.4.4. If the *SecurityMode* is not NONE_1, the *Property* shall provide the list of *Security Key Services* for the *SecurityGroupId*.

The *MaxNetworkMessageSize* is defined in 6.2.4.5.

The *GroupProperties* is defined in 6.2.4.6.

The *Status Object* provides the current operational status of the group. The *PubSubStatusType* is defined in 9.1.10. The state machine for the status and the relation to other *PubSub Objects* like *PubSubConnection*, *DataSetWriter* and *DataSetReader* are defined in 6.2.1.

9.1.6.3 WriterGroupType

Instances of *WriterGroupType* contain settings for a group of *DataSetWriters*. The *WriterGroupType* is formally defined in Table 115.

Table 115 – WriterGroupType definition

Attribute	Value				
BrowseName	WriterGroupType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubGroupType defined in 9.1.6.2					
HasProperty	Variable	WriterGroupId	UInt16	PropertyType	Mandatory
HasProperty	Variable	PublishingInterval	Duration	PropertyType	Mandatory
HasProperty	Variable	KeepAliveTime	Duration	PropertyType	Mandatory
HasProperty	Variable	Priority	Byte	PropertyType	Mandatory
HasProperty	Variable	LocaleIds	LocaleId[]	PropertyType	Mandatory
HasProperty	Variable	HeaderLayoutUri	String	PropertyType	Mandatory
HasComponent	Object	TransportSettings		WriterGroupTransportType	Optional
HasComponent	Object	MessageSettings		WriterGroupMessageType	Optional
HasDataSetWriter	Object	<DataSetWriterName>		DataSetWriterType	OptionalPlaceholder
HasComponent	Object	Diagnostics		PubSubDiagnosticsWriterGroupType	Optional
HasComponent	Method	AddDataSetWriter	Defined in 9.1.6.4.		Optional
HasComponent	Method	RemoveDataSetWriter	Defined in 9.1.6.5.		Optional

The *WriterGroupId* is defined in 6.2.5.1.

The *PublishingInterval* is defined in 6.2.5.2.

The *KeepAliveTime* is defined in 6.2.5.3.

The *Priority* is defined in 6.2.5.4.

The *LocaleIds* parameter is defined in 6.2.5.5.

The transport protocol mapping specific settings are provided in the optional *Object TransportSettings*. The *WriterGroupTransportType* is defined in 9.1.6.7. The *Object* shall be present if the transport protocol mapping requires specific settings.

The message mapping specific settings are provided in the optional *Object MessageSettings*. The *WriterGroupMessageType* is defined in 9.1.6.8. The *Object* shall be present if the message mapping defines specific parameters.

The configured *DataSetWriterType Objects* are added as components to the instance of the group. *DataSetWriterType Objects* may be configured with product-*specific* configuration tools or through OPC UA *Methods AddDataSetWriter* and *RemoveDataSetWriter*. The *DataSetWriterType* is defined in 9.1.7.1. The *ReferenceType HasDataSetWriter* is defined in 9.1.6.6.

The *Diagnostics Object* provides the current diagnostic information for a *WriterGroupType Object*. The *PubSubDiagnosticsWriterGroupType* is defined in 9.1.11.9.

9.1.6.4 AddDataSetWriter Method

This *Method* is used to add a new *DataSetWriterType Object* to an instance of the *WriterGroup*. A successful creation of the *DataSetWriter* shall also create a *Reference* from the related *PublishedDataSet Object* to the created *DataSetWriter*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddDataSetWriter (
   [in]  DataSetWriterDataType  Configuration
   [out] NodeId                 DataSetWriterNodeId
   );
```

Argument	Description
Configuration	Configuration parameters for the *DataSetWriter*. The parameters and the *DataSetWriterDataType* are defined in 6.2.3.
DataSetWriterNodeId	The *NodeId* of the new DataSetWriter Object.

Method Result codes

ResultCode	Description
Bad_InvalidArgument	The *Server* is not able to apply the name. The name may be too long or may contain invalid characters.
Bad_DataSetIdInvalid	The *DataSet* specified for the *DataSetWriter* creation is invalid.
Bad_BrowseNameDuplicated	An *Object* with the name already exists in the group.
Bad_ResourceUnavailable	The *Server* has not enough resources to add the *DataSetWriter*.
Bad_UserAccessDenied	The *Session* user does not have rights to create the *DataSetWriter*.

9.1.6.5 RemoveDataSetWriter Method

This *Method* is used to remove a *DataSetWriter Object* from the group. The state of the *DataSetWriter* is set to Disabled_0 before removing the *Object*. A successful removal of the *DataSetWriter* shall also delete the *Reference* from the related *PublishedDataSetType Object* to the removed *DataSetWriter*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
RemoveDataSetWriter (
   [in]  NodeId      DataSetWriterNodeId
   );
```

Argument	Description
DataSetWriterNodeId	*NodeId* of the DataSetWriter to remove from the group.

Method Result codes

ResultCode	Description
Bad_NodeIdUnknown	The *DataSetWriterNodeId* is unknown.
Bad_NodeIdInvalid	The *DataSetWriterNodeId* is not a *NodeId* of a *DataSetWriter*.
Bad_UserAccessDenied	The *Session* user is not allowed to delete a *DataSetWriter*.

9.1.6.6 HasDataSetWriter

The *HasDataSetWriter ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *HasComponent ReferenceType*.

The *SourceNode* of *References* of this type shall be an instance of the *WriterGroupType* defined in 9.1.6.3.

The *TargetNode* of this *ReferenceType* shall be an instance of the *DataSetWriterType* defined in 9.1.7.1.

The representation of the *HasDataSetWriter ReferenceType* in the *AddressSpace* is specified in Table 116.

Table 116 – HasDataSetWriter ReferenceType

Attributes	Value		
BrowseName	HasDataSetWriter		
InverseName	IsWriterInGroup		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
Subtype of HasComponent defined in IEC 62541-5.			

9.1.6.7 WriterGroupTransportType

This *ObjectType* is the abstract base type for *Objects* representing transport protocol mapping specific settings for *WriterGroups*. The *WriterGroupTransportType* is formally defined in Table 117.

Table 117 – WriterGroupTransportType definition

Attribute	Value				
BrowseName	WriterGroupTransportType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType					
HasSubtype	ObjectType	DatagramWriterGroupTransportType	Defined in 9.3.1.2.		
HasSubtype	ObjectType	BrokerWriterGroupTransportType	Defined in 9.3.2.2.		

9.1.6.8 WriterGroupMessageType

This *ObjectType* is the abstract base type for *Objects* representing message mapping specific settings for *WriterGroups*. The *WriterGroupMessageType* is formally defined in Table 118.

Table 118 – WriterGroupMessageType definition

Attribute	Value				
BrowseName	WriterGroupMessageType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType					
HasSubtype	ObjectType	UadpWriterGroupMessageType	Defined in 9.2.1.1.		
HasSubtype	ObjectType	JsonWriterGroupMessageType	Defined in 9.2.2.1.		

9.1.6.9 ReaderGroupType

This *ObjectType* is a concrete type for *Objects* representing *DataSetReader* groupings for *PubSub* connections. The *ReaderGroupType* is formally defined in Table 119.

Table 119 – ReaderGroupType definition

Attribute	Value				
BrowseName	ReaderGroupType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubGroupType defined in 9.1.6.2					
HasDataSetReader	Object	<DataSetReaderName>		DataSetReaderType	OptionalPlaceholder
HasComponent	Object	Diagnostics		PubSubDiagnostics ReaderGroupType	Optional
HasComponent	Object	TransportSettings		ReaderGroupTransportType	Optional
HasComponent	Object	MessageSettings		ReaderGroupMessageType	Optional
HasComponent	Method	AddDataSetReader	Defined in 9.1.6.10.		Optional
HasComponent	Method	RemoveDataSetReader	Defined in 9.1.6.11.		Optional

The configured *DataSetReaderType Objects* are added as components to the instance of the group. *DataSetReaderType Objects* may be configured with product-specific configuration tools or through OPC UA *Methods AddDataSetReader* and *RemoveDataSetReader*. The *DataSetReaderType* is defined in 9.1.8.1. The *ReferenceType HasDataSetReader* is defined in 9.1.6.12.

The *Diagnostics Object* provides the current diagnostic information for a *ReaderGroupType Object*. The *PubSubDiagnosticsReaderGroupType* is defined in 9.1.11.10.

The transport protocol mapping specific settings are provided in the optional *Object TransportSettings*. The *ReaderGroupTransportType* is defined in 9.1.6.13. The *Object* shall be present if the transport protocol mapping defines specific parameters.

The message mapping specific settings are provided in the optional *Object MessageSettings*. The *ReaderGroupMessageType* is defined in 9.1.6.14. The *Object* shall be present if the message mapping defines specific parameters.

9.1.6.10 AddDataSetReader Method

This *Method* is used to add a new *DataSetReaderType Object* to an instance of the *ReaderGroup*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddDataSetReader (
    [in]  DataSetReaderDataType  Configuration
    [out] NodeId                 DataSetReaderNodeId
    );
```

Argument	Description
Configuration	Configuration parameters for the *DataSetWriter*. The parameters and the *DataSetReaderDataType* are defined in 6.2.8.
DataSetReaderNodeId	The *NodeId* of the new *DataSetReader* Object.

Method Result codes

ResultCode	Description
Bad_InvalidArgument	The *Server* is not able to apply the name. The name may be too long or may contain invalid characters.
Bad_BrowseNameDuplicated	An *Object* with the name already exists in the group.
Bad_ResourceUnavailable	The *Server* does not have enough resources to add the *DataSetReader*.
Bad_UserAccessDenied	The *Session* user does not have rights to create the *DataSetReader*.

9.1.6.11 RemoveDataSetReader Method

This *Method* is used to remove a *DataSetReader Object* from the group. The state of the *DataSetReader* is set to Disabled_0 before the *Object* is removed.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
RemoveDataSetReader (
    [in]  NodeId  DataSetReaderNodeId
    );
```

Argument	Description
DataSetReaderNodeId	*NodeId* of the *DataSetReader* to remove from the group.

Method Result codes

ResultCode	Description
Bad_NodeIdUnknown	The *DataSetReaderNodeId* is unknown.
Bad_NodeIdInvalid	The *DataSetReaderNodeId* is not a NodeId of a *DataSetReader*.
Bad_UserAccessDenied	The *Session* user does not have rights to delete the *DataSetReader*.

9.1.6.12 HasDataSetReader

The *HasDataSetReader ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *HasComponent ReferenceType*.

The *SourceNode* of *References* of this type shall be an instance of the *ReaderGroupType* defined in 9.1.6.6.

The *TargetNode* of this *ReferenceType* shall be an instance of the *DataSetReaderType* defined in 9.1.8.1.

The representation of the *HasDataSetReader ReferenceType* in the *AddressSpace* is specified in Table 120.

Table 120 – HasDataSetReader ReferenceType

Attributes	Value		
BrowseName	HasDataSetReader		
InverseName	IsReaderInGroup		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
Subtype of HasComponent defined in IEC 62541-5.			

9.1.6.13 ReaderGroupTransportType

This *ObjectType* is the abstract base type for *Objects* representing transport protocol mapping specific settings for *ReaderGroups*. The *ReaderGroupTransportType* is formally defined in Table 121.

There is currently no transport protocol mapping specific setting defined.

Table 121 – ReaderGroupTransportType definition

Attribute	Value				
BrowseName	ReaderGroupTransportType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType					

9.1.6.14 ReaderGroupMessageType

This *ObjectType* is the abstract base type for *Objects* representing message mapping specific settings for *ReaderGroups*. The *ReaderGroupMessageType* is formally defined in Table 122.

There is currently no message mapping specific setting defined.

Table 122 – ReaderGroupMessageType Definition

Attribute	Value				
BrowseName	ReaderGroupMessageType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType					

9.1.7 DataSetWriter Model

9.1.7.1 Overview

Figure 42 depicts the *ObjectType* for the *PubSub DataSetWriter* model and its components and the relations to other parts of the model.

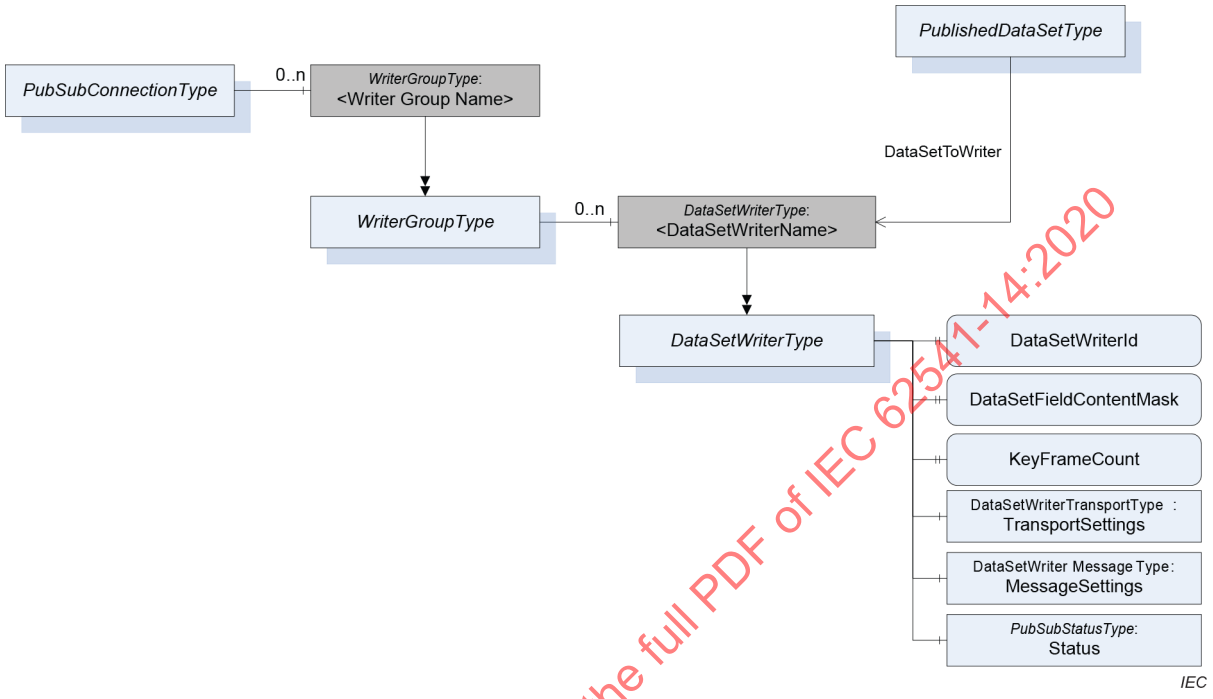


Figure 42 – DataSet Writer Model Overview

9.1.7.2 DataSetWriterType

An instance of this *ObjectType* represents the configuration for a *DataSetWriter*. The *DataSetWriterType* is formally defined Table 123.

Table 123 – DataSetWriterType definition

Attribute	Value				
BrowseName	DataSetWriterType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in IEC 62541-5					
HasProperty	Variable	DataSetWriterId	UInt16	PropertyType	Mandatory
HasProperty	Variable	DataSetFieldContentMask	DataSetField ContentMask	PropertyType	Mandatory
HasProperty	Variable	KeyFrameCount	UInt32	PropertyType	Optional
HasProperty	Variable	DataSetWriterProperties	KeyValuePair[]	PropertyType	Mandatory
HasComponent	Object	TransportSettings		DataSetWriterTransportType	Optional
HasComponent	Object	MessageSettings		DataSetWriterMessageType	Optional
HasComponent	Object	Status		PubSubStatusType	Mandatory
HasComponent	Object	Diagnostics		PubSubDiagnostics DataSetWriterType	Optional

The *DataSetWriterId* is defined in 6.2.3.1.

The *DataSetFieldContentMask* is defined in 6.2.3.2.

The *KeyFrameCount* is defined in 6.2.3.3. The *Property* shall be present for *PublishedDataSets* that provide cyclic updates of the *DataSet*.

The *DataSetWriterProperties* is defined in 6.2.3.4.

The transport protocol mapping specific settings are provided in the optional *Object TransportSettings*. The *DataSetWriterTransportType* is defined in 9.1.7.3. The *Object* shall be present if the transport protocol mapping defines specific parameters.

The message mapping specific settings are provided in the optional *Object MessageSettings*. The *DataSetWriterMessageType* is defined in 9.1.7.4. The *Object* shall be present if the message mapping defines specific parameters.

The *Status Object* provides the current operational status of the *DataSetWriter*. The *PubSubStatusType* is defined in 9.1.10. The state machine for the status and the relation to other *PubSub Objects* like *PubSubConnection* and *PubSubGroup* is defined in 6.2.1.

The *Diagnostics Object* provides the current diagnostic information for a *DataSetWriterType Object*. The *PubSubDiagnosticsDataSetWriterType* is defined in 9.1.11.11.

9.1.7.3 DataSetWriterTransportType

This *ObjectType* is the abstract base type for *Objects* defining protocol-specific transport settings of *DataSetMessages*. The *DataSetWriterTransportType* is formally defined Table 124.

Table 124 – DataSetWriterTransportType definition

Attribute	Value				
BrowseName	DataSetWriterTransportType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5					
HasSubtype	ObjectType	BrokerDataSetWriterTransportType	Defined in 9.3.2.3.		

9.1.7.4 DataSetWriterMessageType

This *ObjectType* is the abstract base type for *Objects* representing message mapping specific settings for *DataSetWriters*. The *DataSetWriterMessageType* is formally defined in Table 125.

Table 125 – DataSetWriterMessageType definition

Attribute	Value				
BrowseName	DataSetWriterMessageType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType					
HasSubtype	ObjectType	UadpDataSetWriterMessageType	Defined in 9.2.1.2.		
HasSubtype	ObjectType	JsonDataSetWriterMessageType	Defined in 9.2.2.2.		

9.1.8 DataSetReader Model

9.1.8.1 Overview

Figure 43 depicts the *ObjectType* for the *PubSub DataSetReader* model and its components and the relations to other parts of the model.

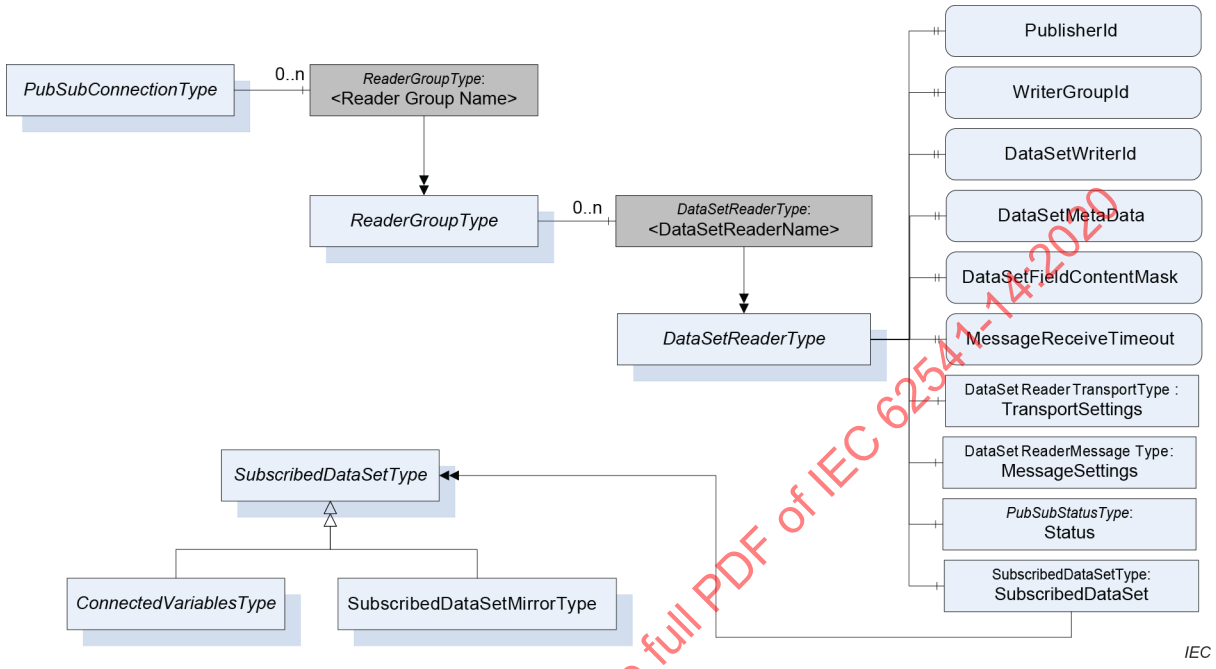


Figure 43 – DataSet Reader Model overview

9.1.8.2 DataSetReaderType

This *ObjectType* defines receiving behaviour of *DataSetMessages* and the decoding to *DataSets*. The *DataSetReaderType* is formally defined in Table 126.

The *SubscribedDataSetType* defined in 9.1.9.1 describes the processing of the received *DataSet* in a *Subscriber*.

Table 126 – DataSetReaderType definition

Attribute	Value				
BrowseName	DataSetReaderType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in IEC 62541-5					
HasProperty	Variable	PublisherId	BaseDataType	PropertyType	Mandatory
HasProperty	Variable	WriterGroupId	UInt16	PropertyType	Mandatory
HasProperty	Variable	DataSetWriterId	UInt16	PropertyType	Mandatory
HasProperty	Variable	DataSetMetaData	DataSetMetaDataType	PropertyType	Mandatory
HasProperty	Variable	DataSetFieldContentMask	DataSetFieldContentMask	PropertyType	Mandatory
HasProperty	Variable	MessageReceiveTimeout	Duration	PropertyType	Mandatory
HasProperty	Variable	KeyFrameCount	UInt32	PropertyType	Mandatory
HasProperty	Variable	HeaderLayoutUri	String	PropertyType	Mandatory
HasProperty	Variable	SecurityMode	MessageSecurityMode	PropertyType	Optional
HasProperty	Variable	SecurityGroupId	String	PropertyType	Optional
HasProperty	Variable	SecurityKeyServices	EndpointDescription[]	PropertyType	Optional
HasProperty	Variable	DataSetReaderProperties	KeyValuePair[]	PropertyType	Mandatory
HasComponent	Object	TransportSettings		DataSetReader TransportType	Optional
HasComponent	Object	MessageSettings		DataSetReader MessageType	Optional
HasComponent	Object	Status		PubSubStatusType	Mandatory
HasComponent	Object	Diagnostics		PubSubDiagnostics DataSetReaderType	Optional
HasComponent	Object	SubscribedDataSet		Subscribed DataSetType	Mandatory
HasComponent	Method	CreateTargetVariables	Defined in 9.1.8.5.		Optional
HasComponent	Method	CreateDataSetMirror	Defined in 9.1.8.6.		Optional

The *Properties PublisherId, WriterGroupId, DataSetWriterId* and *DataSetClassId* define filters for received *NetworkMessages*. If the value of the *Property* is set, it is used as filter and all messages that do not match the filter are dropped.

The *PublisherId* is defined in 6.2.8.1.

The *WriterGroupId* is defined in 6.2.8.2.

The *DataSetWriterId* is defined in 6.2.8.3.

The *DataSetMetaData* is defined in 6.2.8.4. If the *DataSetReader* receives an updated *DataSetMetaData*, the *DataSetReader* shall update the *Property DataSetMetaData*.

The *DataSetFieldContentMask* is defined in 6.2.8.5.

The *MessageReceiveTimeout* is defined in 6.2.8.6.

The *SecurityMode* is defined in 6.2.8.7. If present or if the value is not INVALID_0, it overwrites the settings on the group.

The *SecurityGroupId* is defined in 6.2.8.10.

The *SecurityKeyServices* is defined in 6.2.8.11.

The *DataSetReaderProperties* is defined in 6.2.8.12.

The transport protocol mapping specific settings are provided in the optional *Object TransportSettings*. The *DataSetWriterTransportType* is defined in 9.1.8.3. The *Object* shall be present if the transport protocol mapping defines specific parameters.

The message mapping specific settings are provided in the optional *Object MessageSettings*. The *DataSetWriterMessageType* is defined in 9.1.8.4. The *Object* shall be present if the message mapping defines specific parameters.

The *Status Object* provides the current operational state of the *DataSetReader*. The *PubSubStatusType* is defined in 9.1.10. The state machine for the status and the relation to other *PubSub Objects* like *PubSubConnection* and *PubSubGroup* are defined in 6.2.1.

The *Diagnostics Object* provides the current diagnostic information for a *DataSetReaderType Object*. The *PubSubDiagnosticsDataSetReaderType* is defined in 9.1.11.12.

The *SubscribedDataSet Object* contains the metadata for the subscribed *DataSet* and the information for the processing of *DataSetMessage*. The *SubscribedDataSetType* is defined in 9.1.9.1.

9.1.8.3 DataSetReaderTransportType

This *ObjectType* is the abstract base type for *Objects* defining the transport protocol-specific parameters for *DataSetReaders*. The *DataSetReaderTransportType* is formally defined in Table 127.

Table 127 – DataSetReaderTransportType definition

Attribute	Value				
BrowseName	DataSetReaderTransportType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5					
HasSubtype	ObjectType	BrokerDataSetReaderTransportType		Defined in 9.3.2.4.	

9.1.8.4 DataSetReaderMessageType

This *ObjectType* is the abstract base type for *Objects* representing message mapping specific settings for *DataSetReaders*. The *DataSetReaderMessageType* is formally defined in Table 128.

Table 128 – DataSetReaderMessageType definition

Attribute	Value				
BrowseName	DataSetReaderMessageType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType					
HasSubtype	ObjectType	UadpDataSetReaderMessageType	Defined in 9.2.1.3.		
HasSubtype	ObjectType	JsonDataSetReaderMessageType	Defined in 9.2.2.3.		

9.1.8.5 CreateTargetVariables Method

This *Method* is used to initially set the *SubscribedDataSet* to *TargetVariablesType* and to create the list of target *Variables* of a *SubscribedDataSetType*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
CreateTargetVariables (
    [in]  ConfigurationVersionDataType    ConfigurationVersion
    [in]  FieldTargetDataType[]           TargetVariablesToAdd
    [out] StatusCode[]                    AddResults
    );
```

Argument	Description
ConfigurationVersion	Configuration version of the *DataSet*. The configuration version provided through *CreateTargetVariables* shall match the current configuration version in *DataSetMetaData Property*. If it does not match, the result Bad_InvalidState shall be returned. The *ConfigurationVersionDataType* is defined in 6.2.2.1.5.
TargetVariablesToAdd	The list of target *Variables* to write received *DataSet* fields to. The *FieldTargetDataType* is defined in 6.2.9.2.3. The succeeded targets are added to the *TargetVariables Property*.
AddResults	The result codes for the *Variables* to connect.

Method Result codes

ResultCode	Description
Bad_NothingToDo	An empty list of *Variables* was provided.
Bad_InvalidState	The *DataSetReader* is not configured yet or the *ConfigurationVersion* does not match the version in the *Publisher*.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

Operation Result codes

ResultCode	Description
Bad_NodeIdInvalid	See IEC 62541-4 for the description of this result code.
Bad_NodeIdUnknown	See IEC 62541-4 for the description of this result code.
Bad_IndexRangeInvalid	See IEC 62541-4 for the description of this result code. This status code indicates either an invalid *ReceiverIndexRange* or an invalid *WriterIndexRange* or if the two settings result in a different size.
Bad_IndexRangeNoData	See IEC 62541-4 for the description of this result code. If the *ArrayDimensions* have a fixed length that cannot change and no data exists within the range of indexes specified, Bad_IndexRangeNoData is returned in *AddDataConnections*.
Bad_TooManyMonitoredItems	The *Server* has reached its maximum number of items for the *DataSetReader* object.
Bad_InvalidState	The TargetNodeId is already used by another connection.
Bad_TypeMismatch	The *Server* shall return a Bad_TypeMismatch error if the data type of the *DataSet* field is not the same type or subtype as the target *Variable DataType*. Based on the *DataType* hierarchy, subtypes of the *Variable DataType* shall be accepted by the *Server*. A *ByteString* is structurally the same as a one dimensional array of *Byte*. A *Server* shall accept a *ByteString* if an array of *Byte* is expected.

9.1.8.6 CreateDataSetMirror Method

This *Method* is used to set the *SubscribedDataSet* to *SubscribedDataSetMirrorType* used to represents the fields of the *DataSet* as *Variables* in the *Subscriber Address Space*. This *Method* creates an *Object* below the *SubscribedDataSet* and below this *Object* it creates a *Variable Node* for every field in the *DataSetMetaData*.

A *Variable* representing a field of the *DataSet* shall be created with the following rules.

- TypeDefinition is *BaseDataVariableType* or a subtype.
- The *Reference* from the parent *Node* to the *Variable* is of type *HasComponent*.
- The initial *AccessLevel* of the *Variables* is *CurrentRead*.
- The *RolePermissions* is derived from the parent *Node*.
- The other *Attribute* values are taken from the *FieldMetaData*.
- The *properties* in the *FieldMetaData* are created as *Properties* of the *Variable*.
- The *DataTypes* are created in the *Subscriber* from the *DataSetMetaData* if they do not exist. The *NamespaceUri* of the created *DataTypes* shall match the namespace contained in the *DataSetMetaData*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
CreateDataSetMirror (
   [in]  String                  ParentNodeName
   [in]  RolePermissionType[]    RolePermissions
   [out] NodeId                  ParentNodeId
   );
```

Argument	Description
ParentNodeName	This parameter defines the BrowseName and DisplayName of the parent *Node* for the *Variables* representing the fields of the subscribed *DataSet*.
RolePermissions	Value of the *RolePermissions* Attribute to be set on the parent Node. This value is also used as *RolePermissions* for all *Variables* of the *DataSet* mirror.
ParentNodeId	*NodeId* of the created parent *Node*.

Method Result codes

ResultCode	Description
Bad_InvalidState	The *DataSetReader* is not configured yet or the *ConfigurationVersion* does not match the version in the *Publisher*.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

9.1.9 Subscribed DataSet Model

9.1.9.1 SubscribedDataSetType

This *ObjectType* defines the metadata for the subscribed *DataSet* and the information for the processing of *DataSetMessages*. The *SubscribedDataSetType* is formally defined in Table 129.

Table 129 – SubscribedDataSetType definition

Attribute	Value				
BrowseName	SubscribedDataSetType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5					
HasSubtype	ObjectType	TargetVariablesType			
HasSubtype	ObjectType	SubscribedDataSetMirrorType			

9.1.9.2 TargetVariablesType

This *ObjectType* defines the metadata for the subscribed *DataSet* and the information for the processing of *DataSetMessages*. The *TargetVariablesType* is formally defined in Table 130.

Table 130 – TargetVariablesType definition

Attribute	Value				
BrowseName	TargetVariablesType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of SubscribedDataSetType defined in 9.1.9.1.					
HasProperty	Variable	TargetVariables	FieldTarget DataType[]	PropertyType	Mandatory
HasComponent	Method	AddTargetVariables	Defined in 9.1.9.3.		Optional
HasComponent	Method	RemoveTargetVariables	Defined in 9.1.9.4.		Optional

The *TargetVariables* is defined in 6.2.9.2.

9.1.9.3 AddTargetVariables Method

This *Method* is used to add target *Variables* to an existing list of target *Variables* of a *TargetVariablesType Object*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
AddTargetVariables (
    [in]  ConfigurationVersionDataType      ConfigurationVersion
    [in]  FieldTargetDataType[]             TargetVariablesToAdd
    [out] StatusCode[]                      AddResults
    );
```

Argument	Description
ConfigurationVersion	Configuration version of the *DataSet*. The configuration version provided through *AddDataConnections* shall match the current configuration version in *DataSetMetaData Property*. If it does not match, the result Bad_InvalidState shall be returned. The *ConfigurationVersionDataType* is defined in 6.2.2.1.5.
TargetVariablesToAdd	The list of target *Variables* to write received *DataSet* fields to. The *FieldTargetDataType* is defined in 6.2.9.2.3. The succeeded connections are added to the *TargetVariables Property*.
AddResults	The result codes for the *Variables* to connect.

Method Result codes

ResultCode	Description
Bad_NothingToDo	An empty list of *Variables* was provided.
Bad_InvalidState	The *DataSetReader* is not configured yet or the *ConfigurationVersion* does not match the version in the *Publisher*.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

Operation Result codes

ResultCode	Description
Bad_NodeIdInvalid	See IEC 62541-4 for the description of this result code.
Bad_NodeIdUnknown	See IEC 62541-4 for the description of this result code.
Bad_IndexRangeInvalid	See IEC 62541-4 for the description of this result code. This status code indicates either an invalid ReceiverIndexRange or an invalid WriterIndexRange or if the two settings result in a different size.
Bad_IndexRangeNoData	See IEC 62541-4 for the description of this result code. If the *ArrayDimensions* have a fixed length that cannot change and no data exists within the range of indexes specified, Bad_IndexRangeNoData is returned in *AddDataConnections*.
Bad_TooManyMonitoredItems	The *Server* has reached its maximum number of items for the DataSetReader object.
Bad_InvalidState	The TargetNodeId is already used by another target *Variable*.
Bad_TypeMismatch	The *Server* shall return a Bad_TypeMismatch error if the data type of the *DataSet* field is not the same type or subtype as the target *Variable DataType*. Based on the *DataType* hierarchy, subtypes of the *Variable DataType* shall be accepted by the *Server*. A *ByteString* is structurally the same as a one dimensional array of *Byte*. A *Server* shall accept a *ByteString* if an array of *Byte* is expected.

9.1.9.4 RemoveTargetVariables Method

This *Method* is used to remove entries from the list of target *Variables* of a *TargetVariablesType Object*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
RemoveTargetVariables (
    [in]  ConfigurationVersionDataType      ConfigurationVersion
    [in]  UInt32[]                          TargetsToRemove
    [out] StatusCode[]                      RemoveResults
    );
```

Argument	Description
ConfigurationVersion	Configuration version of the *DataSet*. The configuration version provided through *RemoveDataConnections* shall match the current configuration version in *DataSetMetaData Property*. If it does not match, the result Bad_InvalidState shall be returned. The *ConfigurationVersionDataType* is defined in 6.2.2.1.5.
TargetsToRemove	Array of indices of connections to remove from the list of target Variables.
RemoveResults	The result codes for the connections to remove.

Method Result codes

ResultCode	Description
Bad_NothingToDo	An empty list of *Variables* was provided.
Bad_InvalidState	The *DataSetReader* is not configured yet or the *ConfigurationVersion* does not match the version in the *DataSetMetaData*.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

Operation Result codes

ResultCode	Description
Bad_InvalidArgument	The provided index is invalid.

9.1.9.5 SubscribedDataSetMirrorType

This *ObjectType* defines the information for the processing of *DataSetMessages* as mirror Variables. For each field of the *DataSet* a mirror *Variable* is created in the *Subscriber AddressSpace*. The *SubscribedDataSetMirrorType* is formally defined in Table 131.

Table 131 – SubscribedDataSetMirrorType definition

Attribute	Value				
BrowseName	SubscribedDataSetMirrorType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of SubscribedDataSetType defined in 9.1.9.1.					

An *Object* of this type shall contain an *Object* with the *ParentNodeName* passed to the *Method CreateDataSetMirror* used to set the *SubscribedDataSet* into the mirror mode.

9.1.10 PubSub Status Object

9.1.10.1 PubSubStatusType

This *ObjectType* is used to indicate and change the status of a *PubSub Object* like *PubSubConnection, DataSetWriter* or *DataSetReader*. The *PubSubStatusType* is formally defined in Table 132.

Table 132 – PubSubStatusType definition

Attribute	Value				
BrowseName	PubSubStatusType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5.					
HasComponent	Variable	State	PubSubState	BaseDataVariableType	Mandatory
HasComponent	Method	Enable	Defined in 9.1.10.2.		Optional
HasComponent	Method	Disable	Defined in 9.1.10.3.		Optional

The *State Variable* provides the current operational state of the *PubSub Object*. The default value is *Disabled_0*. The *PubSubState Enumeration* and the related state machine are defined in 6.2.1.

The *State* may be changed with product-specific configuration tools or with the *Methods Enable* and *Disable*.

9.1.10.2 Enable Method

This *Method* is used to enable a configured *PubSub Object*. The related state machine and the transitions triggered by a successful call to this *Method* are defined in 6.2.1.

The *Server* shall reject *Enable Method* calls if the current *State* is not *Disabled_0*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
Enable ();
```

Method Result codes

ResultCode	Description
Bad_InvalidState	The state of the *Object* is not disabled.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

9.1.10.3 Disable Method

This *Method* is used to disable a *PubSub Object*. The related state machine and the transitions triggered by a successful call to this *Method* are defined in 6.2.1.

The *Server* shall reject *Disable Method* calls if the current *State* is *Disabled_0*.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
Disable ();
```

Method Result codes

ResultCode	Description
Bad_InvalidState	The state of the *Object* is not operational.
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

9.1.10.4 Status Object

PubSub ObjectTypes that require a status *Object* add a component with the *BrowseName* Status. It is formally defined in Table 133.

Table 133 – Status Object definition

Attribute	Value				
BrowseName	Status				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
TypeDefinition	ObjectType	PubSubStatusType			

9.1.11 PubSub Diagnostics Objects

9.1.11.1 General

The following types are used to expose diagnostics information in the *PubSub* information model. Each level of the *PubSub* hierarchy shall contain its own diagnostics element in a standardized format. An overview over the proposed diagnostics architecture is given in Figure 44.

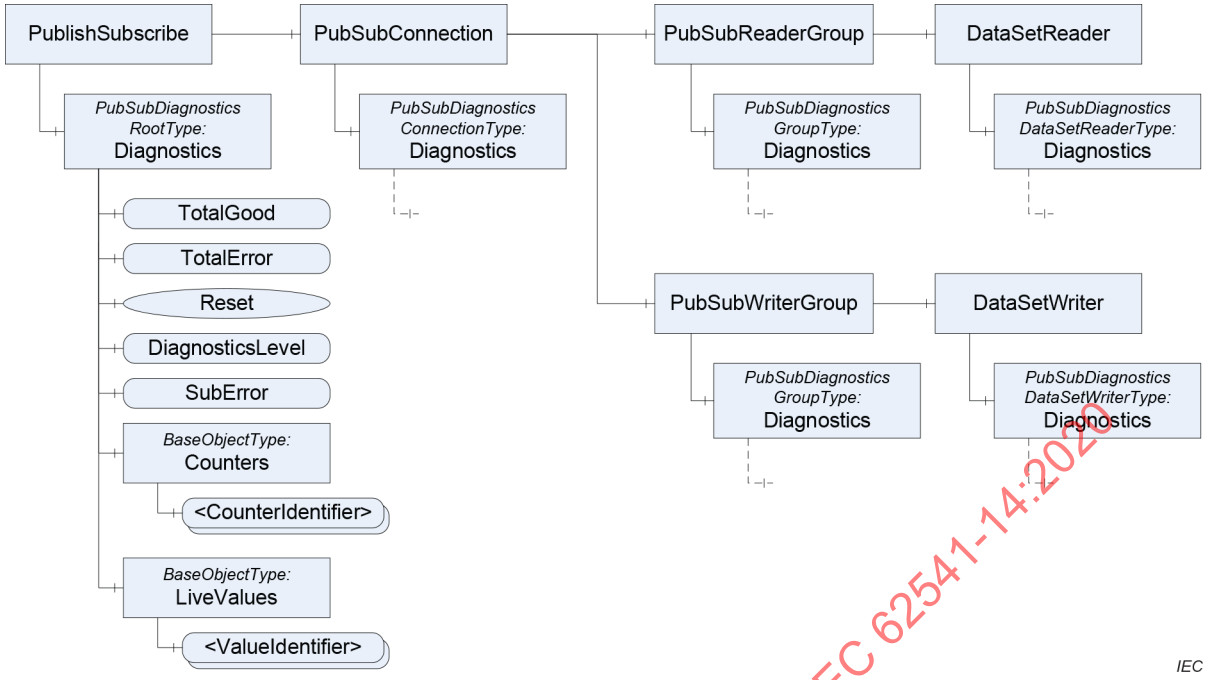


Figure 44 – PubSub Diagnostics overview

Figure 45 shows the structure of a *Variable* which holds a diagnostics counter with defined *Properties*. The *PubSubDiagnosticsCounterType* is formally defined in 9.1.11.5.

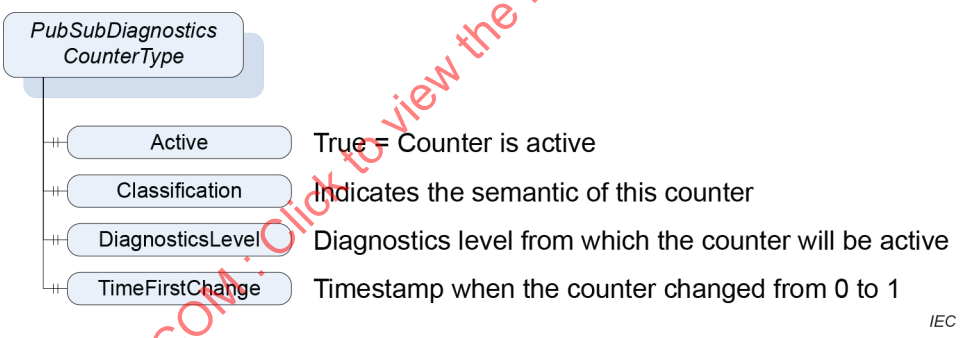


Figure 45 – PubSubDiagnosticsCounterType

9.1.11.2 PubSubDiagnosticsType

The *PubSubDiagnosticsType* is the base type for the diagnostics objects and is formally defined in Table 134.

Table 134 – PubSubDiagnosticsType

Attribute	Value				
BrowseName	PubSubDiagnosticsType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5.					
HasComponent	Variable	DiagnosticsLevel	DiagnosticsLevel	BaseDataVariableType	Mandatory
HasComponent	Variable	TotalInformation	UInt32	PubSubDiagnosticsCounterType	Mandatory
HasComponent	Variable	TotalError	UInt32	PubSubDiagnosticsCounterType	Mandatory
HasComponent	Method	Reset	Defined in 9.1.11.3.		Mandatory
HasComponent	Variable	SubError	Boolean	BaseDataVariableType	Mandatory
HasComponent	Object	Counters		BaseObjectType	Mandatory
HasComponent	Object	LiveValues		BaseObjectType	Mandatory

The *DiagnosticsLevel Variable* configures the current diagnostics level used for the *Object*. The *DiagnosticsLevel DataType* is defined in 9.1.11.4.

The *TotalInformation Variable* provides the sum of all counters in the *Object* diagnostics counters with classification *Information_0*.

The *TotalError Variable* provides the sum of all counters in the *Object* diagnostics counters with classification *Error_1*.

The *SubError Variable* indicates if any statistics *Object* of the next *PubSub* layer *Objects* shows a value > 0 in *TotalError*.

The *Object Counters* contains all diagnostics counters for the diagnostics *Object*. The counters use the *VariableType PubSubDiagnosticsCounterType* defined in 9.1.11.5. The counter Variables of the *PubSubDiagnosticsType* are defined in Table 135.

Table 135 – Counters for PubSubDiagnosticsType

BrowseName	Modelling Rule	Diagnostics Level	Class	Description
StateError	Mandatory	Basic_0	Error_1	PubSubState state machine defined in 6.2.1 changed to Error_3 state
StateOperationalByMethod	Mandatory	Basic_0	Information_0	State changed to Operational_2 state triggered by *Enable Method* call.
StateOperationalByParent	Mandatory	Basic_0	Information_0	State changed to Operational_2 state triggered by an operational parent
StateOperationalFromError	Mandatory	Basic_0	Information_0	State changed from Error_3 to Operational_2.
StatePausedByParent	Mandatory	Basic_0	Information_0	State changed to Paused_1 state triggered by a paused or disabled parent.
StateDisabledByMethod	Mandatory	Basic_0	Information_0	State changed to Disabled_0 state triggered by *Disable Method* call.

The *Object LiveValues* contains all live values of the diagnostics *Object*. If not further specified, the live values *Variables* use the *VariableType BaseDataVariableType*.

The nodes in the *Objects Counters* and *LiveValues* may be activated/deactivated by the parameter *DiagnosticsLevel* in the *PubSubDiagnosticsType*.

The value of a node in the *Object Counters* shall be set to 0 whenever the counter changes from inactive to active.

The *Server* should dynamically remove inactive nodes from the *Address Space* in order to avoid confusion of the user by long lists of counters where only a few of them might be active. In case inactive nodes cannot be removed from the *Address Space*, the *Server* shall set the *StatusCode* of the *Variable Value* to *Bad_OutOfService.*

9.1.11.3 Reset Method

This *Method* is used to set all counters in the *Object* diagnostics counters to the initial value.

The *Client* shall be authorized to modify the configuration for the *PubSub* functionality when invoking this *Method* on the *Server*.

Signature

```
Reset ();
```

Method Result codes

ResultCode	Description
Bad_UserAccessDenied	The *Session* user is not allowed to configure the *Object*.

9.1.11.4 DiagnosticsLevel

PubSub diagnostics are intended to assure users about the correct operation of a *PubSub* system and to help in the discovery of potential faults. Depending on the situation, not all diagnostic *Objects* might be needed, and on the other hand providing them requires resources. As a result, diagnostic objects are assigned to different diagnostic levels. Only diagnostic *Objects* belonging to the currently set diagnostic level or a more severe level shall be provided. This mechanism provides the user with the ability to select a suitable diagnostic configuration depending on the application.

The *DiagnosticsLevel* is an enumeration that specifies the possible diagnostics levels. The possible enumeration values are described in Table 136.

Table 136 – DiagnosticsLevel Values

Value	Description
Basic_0	Diagnostic objects from this level cannot be disabled, and thus objects from this level are the minimum diagnostic feature set that can be expected on any device that supports *PubSub* diagnostics at all.
Advanced_1	Diagnostic objects related to exceptional behaviour are contained in the Advanced_1 diagnostic level.
Info_2	The Info_2 diagnostic level contains high-level diagnostic objects related to the normal operation of a *PubSub* system.
Log_3	Diagnostic objects for the detailed logging of the operation of a *PubSub* system are contained in the Log_3 diagnostic level.
Debug_4	Diagnostic objects with debug information specific to a given implementation of *PubSub* are contained in the Debug_4 diagnostic level. As this level is intended for implementation-specific diagnostics, no such objects are specified by this document.

9.1.11.5 PubSubDiagnosticsCounterType

The *PubSubDiagnosticsCounterType* is formally defined in Table 137.

Table 137 – PubSubDiagnosticsCounterType

Attribute	Value				
BrowseName	PubSubDiagnosticsCounterType				
IsAbstract	False				
ValueRank	-1 (-1 = 'Scalar')				
DataType	UInt32				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseDataVariableType defined in IEC 62541-5.					
HasProperty	Variable	Active	Boolean	PropertyType	Mandatory
HasProperty	Variable	Classification	PubSubDiagnostics Counter Classification	PropertyType	Mandatory
HasProperty	Variable	DiagnosticsLevel	DiagnosticsLevel	PropertyType	Mandatory
HasProperty	Variable	TimeFirstChange	DateTime	PropertyType	Optional

The *Value* shall be reset to 0 when the *Method Clear* of the parent *PubSubDiagnosticsType Object* is called.

The *Value* shall be incremented by 1 for each corresponding event.

The *Value* shall not be incremented anymore when the maximum is reached (0xFFFFFFFF).

If the maximum is reached and a new event occurs, the *SourceTimestamp* of the *Value* shall be updated, even if the *Value* does not change. The *Property Active* indicates if the counter is active.

The *Property Classification* indicates whether this counter counts errors or other events according to *PubSubDiagnosticsCounterClassification* defined in 9.1.11.6.

The *Property DiagnosticsLevel* indicates the diagnostics level the counter belongs to. The *DiagnosticsLevel* is defined in 9.1.11.4.

The *Property TimeFirstChange* contains the *Server* time when the counter value changed from 0 to 1. If the counter value is 0 the *Value* is null.

9.1.11.6 PubSubDiagnosticsCounterClassification

The *PubSubDiagnosticsCounterClassification* is an enumeration that specifies the possible diagnostics counter classifications. The possible enumeration values are described in Table 138.

Table 138 – PubSubDiagnosticsCounterClassification Values

Value	Description
Information_0	The semantic of this diagnostics counter indicates expected events, which are not considered as errors.
Error_1	The semantic of this diagnostics counter indicates errors.

9.1.11.7 PubSubDiagnosticsRootType

The *PubSubDiagnosticsRootType* defines the diagnostic information for the *PublishSubscribe Object* and is formally defined in Table 139.

Table 139 – PubSubDiagnosticsRootType

Attribute	Value				
BrowseName	PubSubDiagnosticsRootType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubDiagnosticsType defined in 9.1.11.2.					
HasComponent	Object	LiveValues		BaseObjectType	Mandatory

The *Object LiveValues* contains all live values of the diagnostics *Object*. If not further specified, the live values *Variables* use the *VariableType BaseDataVariableType*. The live values *Variables* of the *PubSubDiagnosticsRootType* are defined in Table 140.

Table 140 – LiveValues for PubSubDiagnosticsRootType

BrowseName	ModellingRule	Diagnostics Level	DataType	Description
ConfiguredDataSetWriters	Mandatory	Basic_0	UInt16	Number of configured *DataSetWriters* on this *Server*
ConfiguredDataSetReaders	Mandatory	Basic_0	UInt16	Number of configured *DataSetReaders* on this *Server*
OperationalDataSetWriters	Mandatory	Basic_0	UInt16	Number of *DataSetWriters* with state Operational
OperationalDataSetReaders	Mandatory	Basic_0	UInt16	Number of *DataSetReaders* with state Operational

9.1.11.8 PubSubDiagnosticsConnectionType

The *PubSubDiagnosticsConnectionType* defines the diagnostic information for a *PubSubConnectionType Object* and is formally defined in Table 141.

Table 141 – PubSubDiagnosticsConnectionType

Attribute	Value				
BrowseName	PubSubDiagnosticsConnectionType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubDiagnosticsType defined in 9.1.11.2.					
HasComponent	Object	LiveValues		BaseObjectType	Mandatory

The *Object LiveValues* contains all live values of the diagnostics *Object*. If not further specified, the live values *Variables* use the *VariableType BaseDataVariableType*. The live values *Variables* of the *PubSubDiagnosticsConnectionType* are defined in Table 142.

Table 142 – LiveValues for PubSubDiagnosticsConnectionType

BrowseName	ModellingRule	Diagnostics Level	DataType	Description
ResolvedAddress	Mandatory	Basic_0	String	Resolved address of the connection (e.g. IP Address)

9.1.11.9 PubSubDiagnosticsWriterGroupType

The *PubSubDiagnosticsWriterGroupType* defines the diagnostic information for a *WriterGroupType Object* and is formally defined in Table 143.

Table 143 – PubSubDiagnosticsWriterGroupType

Attribute	Value				
BrowseName	PubSubDiagnosticsWriterGroupType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubDiagnosticsType defined in 9.1.11.2.					
HasComponent	Object	Counters		BaseObjectType	Mandatory
HasComponent	Object	LiveValues		BaseObjectType	Mandatory

The *Object Counters* contains all diagnostics counters for the diagnostics *Object*. The counters use the *VariableType PubSubDiagnosticsCounterType* defined in 9.1.11.5. The counter *Variables* of the *PubSubDiagnosticsWriterGroupType* are defined in Table 144.

Table 144 – Counters for PubSubDiagnosticsWriterGroupType

BrowseName	Modelling Rule	Diagnostics Level	Class	Description
Inherited counters from *PubSubDiagnosticsType*				
SentNetworkMessages	Mandatory	Basic_0	Information_0	Sent *NetworkMessages*
FailedTransmissions	Mandatory	Basic_0	Error_1	Error on *NetworkMessage* transmission
EncryptionErrors	Optional	Advanced_1	Error_1	Error on signing or encrypting *NetworkMessage*

The *Object LiveValues* contains all live values of the diagnostics *Object*. If not further specified, the live values *Variables* use the *VariableType BaseDataVariableType*. The live values *Variables* of the *PubSubDiagnosticsWriterGroupType* are defined in Table 145.

Table 145 – LiveValues for PubSubDiagnosticsWriterGroupType

BrowseName	Modelling Rule	Diagnostics Level	DataType	Description
ConfiguredDataSetWriters	Mandatory	Basic_0	UInt16	Number of configured DataSetWriters in this group
OperationalDataSetWriters	Mandatory	Basic_0	UInt16	Number of DataSetWriters with state Operational
SecurityTokenID	Optional	Info_2	UInt32	Currently used SecurityTokenID
TimeToNextTokenID	Optional	Info_2	Duration	Time until the next key change is expected

9.1.11.10 PubSubDiagnosticsReaderGroupType

The *PubSubDiagnosticsReaderGroupType* defines the diagnostic information for a *ReaderGroupType Object* and is formally defined in Table 146.

Table 146 – PubSubDiagnosticsReaderGroupType

Attribute	Value				
BrowseName	PubSubDiagnosticsReaderGroupType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubDiagnosticsType defined in 9.1.11.2.					
HasComponent	Object	Counters		BaseObjectType	Mandatory
HasComponent	Object	LiveValues		BaseObjectType	Mandatory

The *Object Counters* contains all diagnostics counters for the diagnostics *Object*. The counters use the *VariableType PubSubDiagnosticsCounterType* defined in 9.1.11.5. The counter Variables of the *PubSubDiagnosticsReaderGroupType* are defined in Table 147.

Table 147 – Counters for PubSubDiagnosticsReaderGroupType

BrowseName	Modelling Rule	Diagnostics Level	Class	Description
Inherited counters from *PubSubDiagnosticsType*				
ReceivedNetworkMessages	Mandatory	Basic_0	Information_0	Received and processed *NetworkMessages*
ReceivedInvalidNetwork Messages	Optional	Advanced_1	Error_1	Invalid format of *NetworkMessage* Header
DecryptionErrors	Optional	Advanced_1	Error_1	Decryption or signature check errors

The *Object LiveValues* contains all live values of the diagnostics *Object*. If not further specified, the live values *Variables* use the *VariableType BaseDataVariableType*. The live values *Variables* of the *PubSubDiagnosticsReaderGroupType* are defined in Table 148.

Table 148 – LiveValues for PubSubDiagnosticsReaderGroupType

BrowseName	Modelling Rule	Diagnostics Level	DataType	Description
ConfiguredDataSetReaders	Mandatory	Basic_0	UInt16	Number of configured DataSetReaders in this group
OperationalDataSetReaders	Mandatory	Basic_0	UInt16	Number of DataSetReaders with state Operational

9.1.11.11 PubSubDiagnosticsDataSetWriterType

The *PubSubDiagnosticsDataSetWriterType* defines the diagnostic information for a *PubSubDataSetWriterType Object* and is formally defined in Table 149.

Table 149 – PubSubDiagnosticsDataSetWriterType

Attribute	Value				
BrowseName	PubSubDiagnosticsDataSetWriterType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubDiagnosticsType defined in 9.1.11.2.					
HasComponent	Object	Counters		BaseObjectType	Mandatory
HasComponent	Object	LiveValues		BaseObjectType	Mandatory

The *Object Counters* contains all diagnostics counters for the diagnostics *Object*. The counters use the *VariableType PubSubDiagnosticsCounterType* defined in 9.1.11.5. The counter Variables of the *PubSubDiagnosticsDataSetWriterType* are defined in Table 150.

Table 150 – Counters for PubSubDiagnosticsDataSetWriterType

BrowseName	Modelling Rule	Diagnostics Level	Class	Description
Inherited counters from *PubSubDiagnosticsType*				
FailedDataSetMessages	Mandatory	Basic_0	Error_1	Number of failed *DataSetMessages*

The *Object LiveValues* contains all live values of the diagnostics *Object*. If not further specified, the live values *Variables* use the *VariableType BaseDataVariableType*. The live values *Variables* of the *PubSubDiagnosticsDataSetWriterType* are defined in Table 151.

Table 151 – LiveValues for PubSubDiagnosticsDataSetWriterType

BrowseName	Modelling Rule	Diagnostics Level	DataType	Description
MessageSequenceNumber	Optional	Info_2	UInt16	Sequence number of last *DataSetMessage*
StatusCode	Optional	Info_2	StatusCode	Status of last *DataSetMessage*
MajorVersion	Optional	Info_2	UInt32	*MajorVersion* used for *DataSet*
MinorVersion	Optional	Info_2	UInt32	*MinorVersion* used for *DataSet*

9.1.11.12 PubSubDiagnosticsDataSetReaderType

The *PubSubDiagnosticsDataSetReaderType* defines the diagnostic information for a *PubSubDataSetReaderType Object* and is formally defined in Table 152.

Table 152 – PubSubDiagnosticsDataSetReaderType

Attribute	Value				
BrowseName	PubSubDiagnosticsDataSetReaderType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubDiagnosticsType defined in 9.1.11.2.					
HasComponent	Object	Counters		BaseObjectType	Mandatory
HasComponent	Object	LiveValues		BaseObjectType	Mandatory

The *Object Counters* contains all diagnostics counters for the diagnostics *Object*. The counters use the *VariableType PubSubDiagnosticsCounterType* defined in 9.1.11.5. The counter Variables of the *PubSubDiagnosticsDataSetReaderType* are defined in Table 153.

Table 153 – Counters for PubSubDiagnosticsDataSetReaderType

BrowseName	Modelling Rule	Diagnostics Level	Class	Description
Inherited counters from *PubSubDiagnosticsType*				
FailedDataSetMessages	Mandatory	Basic_0	Error_1	e.g. because of unknown *MajorVersion*
DecryptionErrors	Optional	Advanced_1	Error_1	

The *Object LiveValues* contains all live values of the diagnostics *Object*. If not further specified, the live values *Variables* use the *VariableType BaseDataVariableType*. The live values *Variables* of the *PubSubDiagnosticsDataSetReaderType* are defined in Table 154.

Table 154 – LiveValues for PubSubDiagnosticsDataSetReaderType

BrowseName	Modelling Rule	Diagnostics Level	DataType	Description
MessageSequenceNumber	Optional	Info_2	UInt16	SequenceNumber of last *DataSetMessage*
StatusCode	Optional	Info_2	StatusCode	Status of last *DataSetMessage*
MajorVersion	Optional	Info_2	UInt32	*MajorVersion* of available *DataSetMetaData*
MinorVersion	Optional	Info_2	UInt32	*MinorVersion* of available *DataSetMetaData*
SecurityTokenID	Optional	Info_2	UInt32	Currently used SecurityTokenID
TimeToNextTokenID	Optional	Info_2	Duration	Time until the next key change is expected

9.1.12 PubSub Status Events

9.1.12.1 PubSubStatusEventType

This *EventType* is a base type for events which indicate an error or status change associated with a *PubSubConnectionType*, *PubSubGroupType*, *DataSetWriterType* or *DataSetReaderType Object*. The *PubSubStatusEventType* is formally defined in Table 155.

Table 155 – PubSubStatusEventType definition

Attribute	Value				
BrowseName	PubSubStatusEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of SystemEventType defined in IEC 62541-5.					
HasProperty	Variable	ConnectionId	NodeId	PropertyType	Mandatory
HasProperty	Variable	GroupId	NodeId	PropertyType	Mandatory
HasProperty	Variable	State	PubSubState	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *SystemEventType*. Their semantic is defined in IEC 62541-5.

The *SourceNode* is the *NodeId* of the *PubSubConnectionType*, *PubSubGroupType*, *DataSetWriterType* or *DataSetReaderType Object* associated with the *Event*.

The *SourceName* is the *BrowseName* of the *SourceNode*.

The *ConnectionId Property* is the *NodeId* of the *PubSubConnectionType Object* associated with the source of the status *Event*.

The *GroupId Property* is the *NodeId* of the *PubSubGroupType Object* associated with the source of the status *Event*. The *GroupId* is Null if a *PubSubConnection* is the source of the *Event*.

The *State Property* is the current state of the *PubSubStatus Object* associated with the source of the status *Event*.

9.1.12.2 PubSubTransportLimitsExceedEventType

This *EventType* indicates that a *NetworkMessage* could not be published because it exceeds the limits of transport. The *PubSubTransportLimitsExceedEventType* is formally defined in Table 156.

Table 156 – PubSubTransportLimitsExceedEventType definition

Attribute	Value				
BrowseName	PubSubTransportLimitsExceedEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubStatusEventType defined in 9.1.12.2.					
HasProperty	Variable	Actual	UInt32	PropertyType	Mandatory
HasProperty	Variable	Maximum	UInt32	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *PubSubStatusEventType*.

The *Actual Property* has the size in bytes of the actual *NetworkMessage*.

The *Maximum Property* has the maximum size of *NetworkMessages* in bytes allowed by the transport.

9.1.12.3 PubSubCommunicationFailureEventType

This *EventType* indicates that a *NetworkMessage* could not be published because of a communication failure. The *PubSubCommunicationFailureEventType* is formally defined in Table 157.

Table 157 – PubSubCommunicationFailureEventType definition

Attribute	Value				
BrowseName	PubSubCommunicationFailureEventType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of PubSubStatusEventType defined in 9.1.12.2.					
HasProperty	Variable	Error	StatusCode	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *PubSubStatusEventType*.

The *Message Event* field inherited from *BaseEventType* has a localized description of the error.

The *Error Property* has the *StatusCode* associated with the error.

9.2 Message Mapping Configuration Model

9.2.1 UADP Message Mapping

9.2.1.1 UadpWriterGroupMessageType

This *ObjectType* represents UADP message mapping specific parameters for a *WriterGroup*. The *UadpWriterGroupMessageType* is formally defined in Table 158.

Table 158 – UadpWriterGroupMessageType definition

Attribute	Value				
BrowseName	UadpWriterGroupMessageType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of WriterGroupMessageType defined in 9.1.6.8.					
HasProperty	Variable	GroupVersion	VersionTime	PropertyType	Mandatory
HasProperty	Variable	DataSetOrdering	DataSetOrderingType	PropertyType	Mandatory
HasProperty	Variable	NetworkMessage ContentMask	UadpNetworkMessageContentMask	PropertyType	Mandatory
HasProperty	Variable	SamplingOffset	Duration	PropertyType	Optional
HasProperty	Variable	PublishingOffset	Duration	PropertyType	Mandatory

The *GroupVersion* is defined in 6.3.1.1.2.

The *DataSetOrdering* is defined in 6.3.1.1.3.

The *NetworkMessageContentMask* is defined in 6.3.1.1.4.

The *SamplingOffset* is defined in 6.3.1.1.5.

The *PublishingOffset* is defined in 6.3.1.1.6. The *ValueRank* of the *PublishingOffset* shall be −3 if the *Publisher* supports scheduling of multiple *NetworkMessages* per *PublishingInterval*. If only a single offset can be configured, the *ValueRank* shall be −1. Therefore, the *Value* of the *PublishingOffset* can be a scalar value or a one-dimensional array value. The default *Value* is scalar value.

9.2.1.2 UadpDataSetWriterMessageType

This *ObjectType* represents UADP message mapping specific parameters for a *DataSetWriter*. The *UadpDataSetWriterMessageType* is formally defined in Table 159.

Table 159 – UadpDataSetWriterMessageType definition

Attribute	Value				
BrowseName	UadpDataSetWriterMessageType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of DataSetWriterMessageType defined in 9.1.7.4.					
HasProperty	Variable	DataSetMessageContentMask	UadpDataSetMessage ContentMask	PropertyType	Mandatory
HasProperty	Variable	ConfiguredSize	UInt16	PropertyType	Mandatory
HasProperty	Variable	NetworkMessageNumber	UInt16	PropertyType	Mandatory
HasProperty	Variable	DataSetOffset	UInt16	PropertyType	Mandatory

The *DataSetMessageContentMask* is defined in 6.3.1.2.2.

The *ConfiguredSize* is defined in 6.3.1.2.2.

The *NetworkMessage* is defined in 6.3.1.2.4.

The *DataSetOffset* is defined in 6.3.1.2.5.

9.2.1.3 UadpDataSetReaderMessageType

This *ObjectType* represents UADP message mapping specific parameters for a *DataSetReader*. The *UadpDataSetWriterMessageType* is formally defined in Table 160.

Table 160 – UadpDataSetReaderMessageType definition

Attribute	Value				
BrowseName	UadpDataSetReaderMessageType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of DataSetReaderMessageType defined in 9.1.8.4.					
HasProperty	Variable	GroupVersion	VersionTime	PropertyType	Mandatory
HasProperty	Variable	NetworkMessageNumber	UInt16	PropertyType	Mandatory
HasProperty	Variable	DataSetOffset	UInt16	PropertyType	Mandatory
HasProperty	Variable	DataSetClassId	Guid	PropertyType	Mandatory
HasProperty	Variable	NetworkMessageContentMask	UadpNetworkMessage ContentMask	PropertyType	Mandatory
HasProperty	Variable	DataSetMessageContentMask	UapdDataSetMessage ContentMask	PropertyType	Mandatory
HasProperty	Variable	PublishingInterval	Duration	PropertyType	Mandatory
HasProperty	Variable	ReceiveOffset	Duration	PropertyType	Mandatory
HasProperty	Variable	ProcessingOffset	Duration	PropertyType	Mandatory

The *GroupVersion* is defined in 6.3.1.3.1.

The *NetworkMessageNumber* is defined in 6.3.1.3.2.

The *DataSetOffset* is defined in 6.3.1.3.3.

The *DataSetClassId* is defined in 6.3.1.3.4. The initial value is null.

The *NetworkMessageContentMask* is defined in 6.3.1.3.5.

The *DataSetMessageContentMask* is defined in 6.3.1.3.6.

The *PublishingInterval* is defined in 6.3.1.3.7.

The *ReceiveOffset* is defined in 6.3.1.3.8.

The *ProcessingOffset* is defined in 6.3.1.3.9.

9.2.2 JSON Message Mapping

9.2.2.1 JsonWriterGroupMessageType

This *ObjectType* represents JSON message mapping specific parameters for a *WriterGroup*. The *JsonWriterGroupMessageType* is formally defined in Table 161.

Table 161 – JsonWriterGroupMessageType Definition

Attribute	Value				
BrowseName	JsonWriterGroupMessageType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of WriterGroupMessageType defined in 9.1.6.8.					
HasProperty	Variable	NetworkMessage ContentMask	JsonNetworkMessage ContentMask	PropertyType	Mandatory

The *NetworkMessageContentMask* is defined in 6.3.2.3.1.

9.2.2.2 JsonDataSetWriterMessageType

This *ObjectType* represents UADP message mapping specific parameters for a *DataSetWriter*. The *JsonDataSetWriterMessageType* is formally defined in Table 162.

Table 162 – JsonDataSetWriterMessageType definition

Attribute	Value				
BrowseName	JsonDataSetWriterMessageType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of DataSetWriterMessageType defined in 9.1.7.4.					
HasProperty	Variable	DataSetMessageContentMask	JsonDataSetMessage ContentMask	PropertyType	Mandatory

The *DataSetMessageContentMask* is defined in 6.3.2.2.1.

9.2.2.3 JsonDataSetReaderMessageType

This *ObjectType* represents UADP message mapping specific parameters for a *DataSetReader*. The *JsonDataSetReaderMessageType* is formally defined in Table 163.

Table 163 – JsonDataSetReaderMessageType definition

Attribute	Value				
BrowseName	JsonDataSetReaderMessageType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of DataSetReaderMessageType defined in 9.1.8.4.					
HasProperty	Variable	NetworkMessageContentMask	JsonNetworkMessage ContentMask	PropertyType	Mandatory
HasProperty	Variable	DataSetMessageContentMask	JsonDataSetMessage ContentMask	PropertyType	Mandatory

The *NetworkMessageContentMask* is defined in 6.3.2.3.1.

The *DataSetMessageContentMask* is defined in 6.3.2.3.2.

9.3 Transport Protocol Mapping Configuration Model

9.3.1 Datagram Transport Protocol Mapping

9.3.1.1 DatagramConnectionTransportType

This *ObjectType* represents datagram transport protocol mapping specific parameters for a *PubSubConnection*. The *DatagramConnectionTransportType* is formally defined in Table 164.

Table 164 – DatagramConnectionTransportType definition

Attribute	Value				
BrowseName	DatagramConnectionTransportType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of ConnectionTransportType defined in 9.1.5.6.					
HasComponent	Object	DiscoveryAddress		NetworkAddressType	Mandatory

The *DiscoveryAddress* is defined in 6.4.1.1.1.

9.3.1.2 DatagramWriterGroupTransportType

This *ObjectType* represents datagram transport protocol mapping specific parameters for a *WriterGroup*. The *DatagramWriterGroupTransportType* is formally defined in Table 165.

Table 165 – DatagramWriterGroupTransportType definition

Attribute	Value				
BrowseName	DatagramWriterGroupTransportType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of WriterGroupTransportType defined in 9.1.6.7.					
HasProperty	Variable	MessageRepeatCount	Byte	PropertyType	Optional
HasProperty	Variable	MessageRepeatDelay	Duration	PropertyType	Optional

The *MessageRepeatCount* is defined in 6.4.1.2.1.

The *MessageRepeatDelay* is defined in 6.4.1.2.2.

9.3.1.3 DatagramDataSetWriterTransportType

There is no datagram-specific transport protocol mapping parameter defined for the *DataSetWriter*.

9.3.1.4 DatagramDataSetReaderTransportType

There is no datagram-specific transport protocol mapping parameter defined for the *DataSetReader*.

9.3.2 Broker Transport Protocol Mapping

9.3.2.1 BrokerConnectionTransportType

This *ObjectType* represents broker transport protocol mapping specific parameters for a *PubSubConnection*. The *BrokerConnectionTransportType* is formally defined in Table 166.

Table 166 – BrokerConnectionTransportType definition

Attribute	Value				
BrowseName	BrokerConnectionTransportType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of ConnectionTransportType defined in 9.1.5.6.					
HasProperty	Variable	ResourceUri	String	PropertyType	Mandatory
HasProperty	Variable	AuthenticationProfileUri	String	PropertyType	Mandatory

The *ResourceUri* is defined in 6.4.2.1.1.

The *AuthenticationProfileUri* is defined in 6.4.2.1.2.

9.3.2.2 BrokerWriterGroupTransportType

This *ObjectType* represents broker transport protocol mapping specific parameters for a *WriterGroup*. The *BrokerWriterGroupTransportType* is formally defined in Table 167.

Table 167 – BrokerWriterGroupTransportType definition

Attribute	Value				
BrowseName	BrokerWriterGroupTransportType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of WriterGroupTransportType defined in 9.1.6.7.					
HasProperty	Variable	QueueName	String	PropertyType	Mandatory
HasProperty	Variable	ResourceUri	String	PropertyType	Mandatory
HasProperty	Variable	AuthenticationProfileUri	String	PropertyType	Mandatory
HasProperty	Variable	RequestedDeliveryGuarantee	BrokerTransportQuality OfService	PropertyType	Mandatory

The *QueueName* is defined in 6.4.2.2.1.

The *ResourceUri* is defined in 6.4.2.2.2.

The *AuthenticationProfileUri* is defined in 6.4.2.2.3.

The *RequestedDeliveryGuarantee* is defined in 6.4.2.2.4.

9.3.2.3 BrokerDataSetWriterTransportType

This *ObjectType* represents broker transport protocol mapping specific parameters for a *DataSetWriter*. The *BrokerDataSetWriterTransportType* is formally defined in Table 168.

Table 168 – BrokerDataSetWriterTransportType definition

Attribute	Value				
BrowseName	BrokerDataSetWriterTransportType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of DataSetWriterTransportType defined in 9.1.7.3.					
HasProperty	Variable	QueueName	String	PropertyType	Mandatory
HasProperty	Variable	MetaDataQueueName	String	PropertyType	Mandatory
HasProperty	Variable	ResourceUri	String	PropertyType	Mandatory
HasProperty	Variable	AuthenticationProfileUri	String	PropertyType	Mandatory
HasProperty	Variable	RequestedDeliveryGuarantee	BrokerTransportQualityOfService	PropertyType	Mandatory
HasProperty	Variable	MetaDataUpdateTime	Duration	PropertyType	Mandatory

The *QueueName* is defined in 6.4.2.3.1.

The *ResourceUri* is defined in 6.4.2.3.2.

The *AuthenticationProfileUri* is defined in 6.4.2.3.3.

The *RequestedDeliveryGuarantee* is defined in 6.4.2.3.4.

The *MetaDataQueueName* is defined in 6.4.2.3.5.

The *MetaDataUpdateTime* is defined in 6.4.2.3.6.

This type extends the list of well-known extension field names defined in Table 107 with the names defined in Table 169.

Table 169 – Broker Writer well-known extension field names

Name	Type	Description
QueueName	String	The *Broker* queue destination for Data messages.
MetaDataQueueName	String	The *Broker* queue destination for metadata messages.

9.3.2.4 BrokerDataSetReaderTransportType

This *ObjectType* represents datagram transport protocol mapping specific parameters for a *DataSetReader*. The *BrokerDataSetReaderTransportType* is formally defined in Table 170.

Table 170 – BrokerDataSetReaderTransportType definition

Attribute	Value				
BrowseName	BrokerDataSetReaderTransportType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of DataSetReaderTransportType defined in 9.1.8.3.					
HasProperty	Variable	QueueName	String	PropertyType	Mandatory
HasProperty	Variable	ResourceUri	String	PropertyType	Mandatory
HasProperty	Variable	AuthenticationProfileUri	String	PropertyType	Mandatory
HasProperty	Variable	RequestedDeliveryGuarantee	BrokerTransport QualityOfService	PropertyType	Mandatory
HasProperty	Variable	MetaDataQueueName	String	PropertyType	Mandatory

The *QueueName* is defined in 6.4.2.4.1.

The *ResourceUri* is defined in 6.4.2.4.2.

The *AuthenticationProfileUri* is defined in 6.4.2.4.3.

The *RequestedDeliveryGuarantee* is defined in 6.4.2.4.4.

The *MetaDataQueueName* is defined in 6.4.2.4.5.

Annex A
(normative)

Common types

A.1 DataType Schema Header structures

A.1.1 DataTypeSchemaHeader

This *Structure DataType* is the abstract base type used to provide OPC UA *DataType* definitions for an OPC UA Binary encoded byte blob used outside an OPC UA *Server AddressSpace*.

The *DataTypeSchemaHeader* is formally defined in Table A.1.

Table A.1 – DataTypeSchemaHeader structure

Name	Type	Description
DataTypeSchemaHeader	Structure	
namespaces	String[]	Defines an array of namespace URIs. The index into the array is referred to as *NamespaceIndex*. The *NamespaceIndex* is used in *NodeIds* and *QualifiedNames*, rather than the longer namespace URI. NamespaceIndex 0 is reserved for the OPC UA namespace and it is not included in this array. The array contains the namespaces used in the data that follows the *DataTypeSchemaHeader*. The index used in *NodeId* and *QualifiedNames* identify an element in this list. The first entry in this array maps to NamespaceIndex 1.
structureDataTypes	StructureDescription[]	Description of *Structure* and *Union DataTypes* used in the data that follows the *DataTypeSchemaHeader*. This includes nested *Structures*. DataType NodeIds for *Structure DataTypes* used in the data refer to entries in this array. The *StructureDescription DataType* is defined in A.1.3.
enumDataTypes	EnumDescription[]	Description of *Enumeration* or *OptionSet DataTypes* used in the data that follows the *DataTypeSchemaHeader*. DataType NodeIds for *Enumeration* or *OptionSet DataTypes* used in the data refer to entries in this array. The *EnumDescription DataType* is defined in A.1.4.
simpleDataTypes	SimpleTypeDescription[]	Description of *DataTypes* derived from built-in DataTypes. This excludes *OptionSet DataTypes*.

The *DataTypeSchemaHeader Structure* representation in the *AddressSpace* is defined in Table A.2.

Table A.2 – DataTypeSchemaHeader definition

Attributes	Value		
BrowseName	DataTypeSchemaHeader		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**IsAbstract**
Subtype of Structure defined in IEC 62541-5.			
HasSubtype	DataType	UABinaryFileDataType	False

A.1.2 DataTypeDescription

This *Structure DataType* is the abstract base type for all *DataType* descriptions containing the *DataType NodeId* and the definition for custom *DataTypes* like *Structures and Enumerations*. The *DataTypeDescription* is formally defined in Table A.3.

Table A.3 – DataTypeDescription structure

Name	Type	Description
DataTypeDescription	Structure	
dataTypeId	NodeId	The *NodeId* of the *DataType*.
name	QualifiedName	A unique name for the data type.

The *DataTypeDescription Structure* representation in the AddressSpace is defined in Table A.4.

Table A.4 – DataTypeDescription definition

Attributes	Value		
BrowseName	DataTypeDescription		
IsAbstract	True		
References	**NodeClass**	**BrowseName**	**IsAbstract**
Subtype of Structure defined in IEC 62541-5.			
HasSubtype	DataType	StructureDescription	FALSE
HasSubtype	DataType	EnumDescription	FALSE

A.1.3 StructureDescription

This *Structure DataType* provides the concrete *DataTypeDescription* for *Structure DataTypes*. It is a subtype of the *DataTypeDescription DataType*. The *StructureDescription* is formally defined in Table A.5.

Table A.5 – StructureDescription structure

Name	Type	Description
StructureDescription	Structure	
structureDefinition	StructureDefinition	The definition of the structure *DataType*. The *StructureDefinition DataType* is defined in IEC 62541-3.

Its representation in the AddressSpace is defined in Table A.6.

Table A.6 – StructureDescription definition

Attributes	Value
BrowseName	StructureDescription
IsAbstract	False
Subtype of DataTypeDescription defined in 6.2.2.1.5.	

A.1.4 EnumDescription

This *Structure DataType* provides the concrete *DataTypeDescription* for *Enumeration DataTypes*. It is a subtype of the *DataTypeDescription DataType*. The *EnumDescription* is formally defined in Table A.7.

Table A.7 – EnumDescription Structure

Name	Type	Description
EnumDescription	Structure	
enumDefinition	EnumDefinition	The definition of the enumeration *DataType*. The *EnumDefinition DataType* is defined in IEC 62541-3.
builtInType	Byte	The *builtInType* indicates if the *DataType* is an *Enumeration* or an *OptionSet*. If the *builtInType* is *Int32*, the *DataType* is an *Enumeration*. If the *builtInType* is one of the *UInteger DataTypes* or *ExtensionObject*, the *DataType* is an *OptionSet*.

Its representation in the AddressSpace is defined in Table A.8.

Table A.8 – EnumDescription definition

Attributes	Value
BrowseName	EnumDescription
IsAbstract	False
Subtype of DataTypeDescription defined in 6.2.2.1.5.	

A.1.5 SimpleTypeDescription

This *Structure DataType* provides the information for *DataTypes* derived from built-in *DataTypes*. It is a subtype of *Structure*. The *SimpleTypeDescription* is formally defined in Table A.9.

Table A.9 – SimpleTypeDescription structure

Name	Type	Description
SimpleTypeDescription	Structure	
baseDataType	NodeId	The base *DataType* of the simple *DataType*.
builtInType	Byte	The *builtInType* used for the encoding of the simple *DataType*.

A.2 UABinaryFileDataType

This *Structure DataType* defines the base layout of an OPC UA *Binary* encoded file. The content of the file is the *UABinaryFileDataType* encoded as *ExtensionObject*.

The file-specific metadata is provided by the *DataTypeSchemaHeader* which is the base type for the *UABinaryFileDataType Structure*.

If the file is provided through a *FileType Object*, the *MimeType Property* of the *Object* shall have the value application/opcua+uabinary.

If the file is stored on disc, the file extension shall be uabinary.

The *UABinaryFileDataType* is formally defined in Table A.10.

Table A.10 – UABinaryFileDataType structure

Name	Type	Description
UABinaryFileDataType	Structure	
schemaLocation	String	Reference to a file that contains the *DataTypeSchemaHeader* for the content of the file represented by an instance of this structure. The *schemaLocation* is either a fully qualified URL or a URN which is a relative path to the file location. If the *schemaLocation* is provided, the *DataType* descriptions can be skipped but the *namespaces* used shall match the *namespaces* in the schema file.
fileHeader	KeyValuePair[]	The file-specific header.
body	BaseDataType	The body of the file. The *DataTypes* used in the body are described through the *structureDataTypes*, *enumDataTypes* and *simpleDataTypes* fields of the *DataTypeSchemaHeader Structure* which is the base type for the *UABinaryFileDataType*. *DataTypes* defined by OPC UA can be omitted.

Its representation in the *UABinaryFileDataType* is defined in Table A.11.

Table A.11 – UABinaryFileDataType definition

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

A.3 NetworkAddress Model

A.3.1 NetworkAddressType

An instance of a subtype of this abstract *ObjectType* represents network address information. The *NetworkAddressType* is formally defined in Table A.12.

Table A.12 – NetworkAddressType definition

Attribute	Value				
BrowseName	NetworkAddressType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of BaseObjectType defined in IEC 62541-5.					
HasComponent	Variable	NetworkInterface	String	SelectionListType	Mandatory
HasSubtype	ObjectType	NetworkAddressUrlType	Defined in A.3.2.		

The *NetworkInterface Variable* allows the selection of the network interface used for the communication relation. The network interface can be listed by name, by IP address or a combination of name and IP address. The *SelectionValues Property* of the *SelectionListType* shall contain the list of available network interfaces as application-specific strings. The Value of the Variable contains the selected network interface as *String*. The *SelectionListType* is defined in IEC 62541-5. The *Object* may allow providing additional *Strings* not defined in the *SelectionValues*. In this case the *NotRestrictToList Property* of the *SelectionListType* is set to true.

A.3.2 NetworkAddressUrlType

An instance of this *ObjectType* represents network address information in the form of a URL *String*. The *NetworkAddressUrlType* is formally defined in Table A.13.

Table A.13 – NetworkAddressUrlType definition

Attribute	Value				
BrowseName	NetworkAddressUrlType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of NetworkAddressType defined in A.3.1.					
HasComponent	Variable	Url	String	BaseDataVariableType	Mandatory

The *URL Variable* contains the address string for the communication middleware or the communication relation. The syntax of the URL is defined by the transport protocol.

Annex B
(informative)

Client Server vs. Publish Subscribe

B.1 Overview

OPC UA *Applications* represent software or devices that provide information to other OPC UA *Applications* or consume information from other OPC UA *Applications*.

Annex B contrasts the *Subscription* functionality available in the *Client Server* communication model with the data distribution mechanism of *PubSub*. See IEC TR 62541-1 for an overview of the complete functionality available with the *Client Server* model.

B.2 Client Server Subscriptions

In the *Client Server* communication model the application exposing information consisting of physical and software objects is the OPC UA *Server* and the application operating upon this information is the OPC UA *Client*.

The information provided by an OPC UA *Server* is organized in the *Server Address Space*. *Services* like *Read*, *Write* and *Browse* are available with a request/response pattern used by OPC UA *Clients* to access information provided by an OPC UA *Server*.

Every *Client* creates individual *Sessions*, *Subscriptions* and *MonitoredItems* which are not shared with other *Clients*. In other words, the data that is published only goes to the *Client* that created the *Subscription*.

Sessions are used to manage the communication relationship between *Client* and *Server*. *MonitoredItems* represent the settings used to subscribe to *Events* and *Variable Value* data changes from the OPC UA *Server Address Space*. *MonitoredItems* are grouped in *Subscriptions*.

The entities used by OPC UA *Clients* to subscribe to information from an OPC UA *Server* are illustrated in Figure B.1.

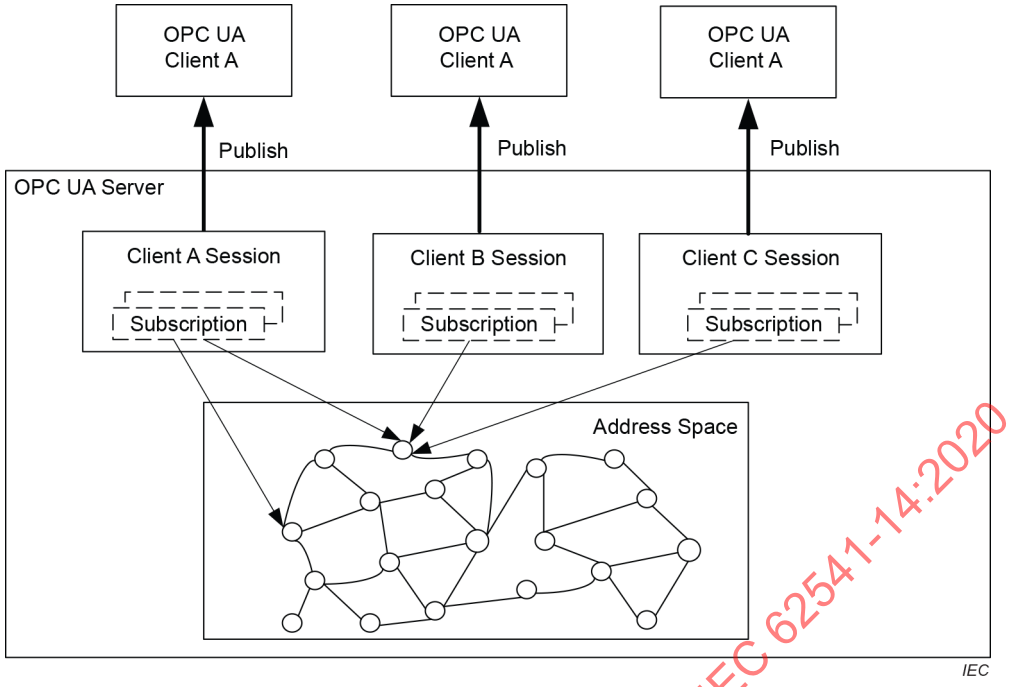


Figure B.1 – Subscriptions in OPC UA Client Server Model

In this model the *Client* is the active entity. It chooses what *Nodes* of the *Server AddressSpace* and what *Services* to use. *Subscriptions* are created or deleted on the fly. The published data only goes to the *Client* that created a *Subscription*.

The *Client Server Subscription* model provides reliable delivery using buffering, acknowledgements, and retransmissions. This requires resources in the *Server* for each connected *Client*.

Resource-constrained *Servers* limit the number of parallel *Client* connections, *Subscriptions,* and *MonitoredItems*. Similar limitations can also occur in the *Client*. *Clients* that continuously need data from a larger number of *Servers* also consume significant resources.

B.3 Publish-Subscribe

With *PubSub*, OPC UA *Applications* do not directly exchange requests and responses. Instead, *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.

Figure B.2 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.

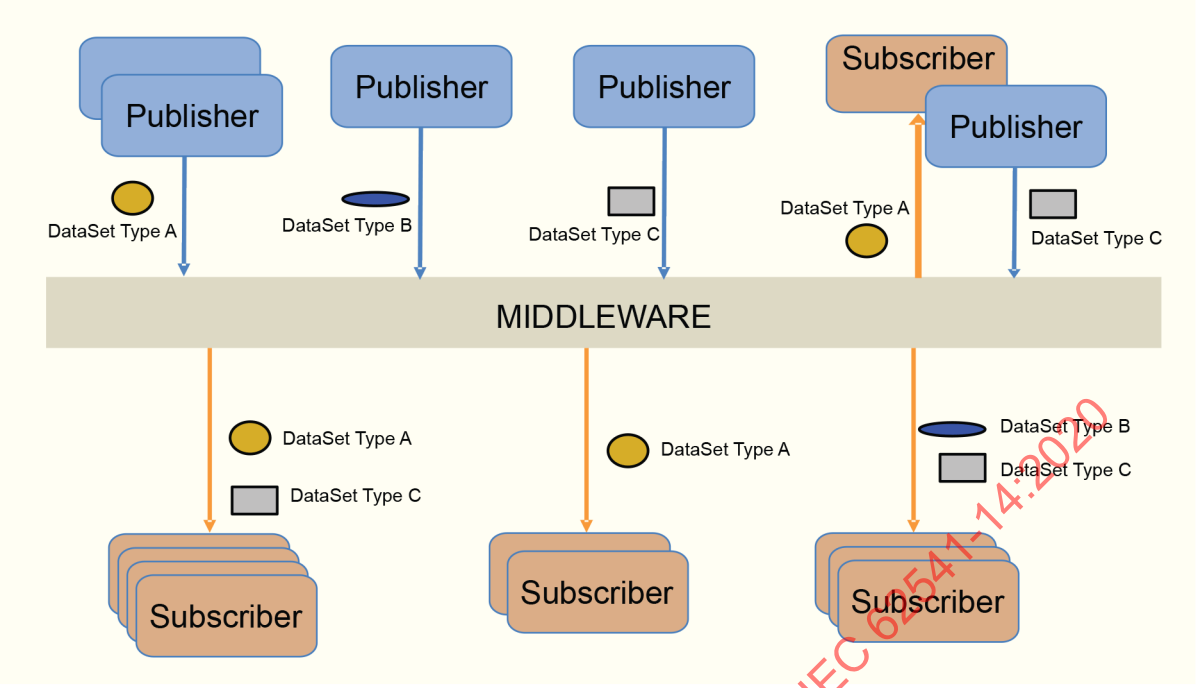


Figure B.2 – Publish Subscribe Model Overview

PubSub is used to communicate messages between different system components without these components having to know each other's identity.

A *Publisher* is pre-configured with what data to send. There is no connection establishment between *Publisher* and *Subscriber*.

The identity of the *Subscribers* and the forwarding of published data to the *Subscribers* is the responsibility of the *Message Oriented Middleware*. The *Publisher* does not know or even care if there is one or many *Subscribers*. Effort and resource requirements for the *Publisher* are predictable and do not depend on the number of *Subscribers*.

B.4 Synergy of models

PubSub and *Client Server* are both based on the OPC UA *Information Model*. *PubSub* therefore can easily be integrated into OPC UA *Servers* and OPC UA *Clients*. Quite typically, a *Publisher* will be an OPC UA *Server* (the owner of information) and a *Subscriber* is often an OPC UA *Client*. Above all, the *PubSub Information Model* for configuration (see 6.2.2) promotes the configuration of *Publishers* and *Subscribers* using the OPC UA *Client Server* model.

Nevertheless, the *PubSub* communication does not require such a role dependency. In other words, 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.

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

ARCHITECTURE UNIFIÉE OPC –

Partie 14: PubSub

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 62541-14 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

Le texte de cette Norme internationale est issu des documents suivants:

FDIS	Rapport de vote
65E/720/FDIS	65E/736/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette Norme internationale.

Ce document a été rédigé selon les Directives ISO/IEC, Partie 2.

Dans l'ensemble du présent document et dans les autres parties de la série IEC 62541, certaines conventions de document sont utilisées:

Le format *italique* est utilisé pour mettre en évidence un terme défini ou une définition qui apparaît à l'Article 3 dans l'une des parties de la série.

Le format *italique* est également utilisé pour mettre en évidence le nom d'un paramètre d'entrée ou de sortie de service, ou le nom d'une structure ou d'un élément de structure habituellement défini dans les tableaux.

Par ailleurs, les *termes* et les *noms en italique* sont, à quelques exceptions près, écrits en camel-case (pratique qui consiste à joindre, sans espace, les éléments des mots ou expressions composés, la première lettre de chaque élément étant en majuscule). Par exemple, le terme défini est *AddressSpace* et non Espace d'adressage. Cela permet de mieux comprendre qu'il existe une définition unique pour *AddressSpace*, et non deux définitions distinctes pour Espace et pour Adressage.

Une liste de toutes les parties de la série IEC 62541, publiées sous le titre général *Architecture unifiée OPC*, peut être consultée sur le site web de l'IEC.

Le comité a décidé que le contenu de ce document ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "http://webstore.iec.ch" dans les données relatives au document recherché. A cette date, le document sera

- reconduit,
- supprimé,
- remplacé par une édition révisée, ou
- amendé.

IMPORTANT – Le logo *"colour inside"* qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

ARCHITECTURE UNIFIÉE OPC –

Partie 14: PubSub

1 Domaine d'application

La présente partie de l'IEC 62541 définit le modèle de communication *PubSub* de l'architecture unifiée OPC (OPC UA). Elle définit un modèle publication/abonnement OPC UA qui vient compléter le modèle client/serveur défini par les *Services* dans l'IEC 62541-4. Une présentation des deux modèles et de leurs utilisations est donnée dans l'IEC TR 62541-1.

PubSub permet de distribuer des données et des événements provenant d'une source d'informations OPC UA aux observateurs d'intérêt à l'intérieur d'un réseau de dispositifs ainsi que dans les systèmes Cloud informatiques et d'analyse.

Le présent document se compose:

- d'une présentation générale des concepts *PubSub*;
- d'une définition des paramètres de configuration *PubSub*;
- d'un mapping des concepts et des paramètres de configuration *PubSub* avec les messages et les protocoles de transport; et
- d'un modèle de configuration PubSub.

Il n'est pas nécessaire que l'ensemble des *Applications* OPC UA mettent en œuvre tous les mappings avec les messages et les protocoles de transport. L'IEC 62541-7 définit le *Profil* qui dicte les mappings qu'il est nécessaire de mettre en œuvre afin d'être conforme à un *Profil* particulier.

2 Références normatives

Les documents suivants cités dans le texte constituent, pour tout ou partie de leur contenu, des exigences du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC TR 62541-1, *OPC Unified Architecture – Part 1: Overview and Concepts* (disponible en anglais seulement)

IEC TR 62541-2, *OPC Unified Architecture – Part 2: Security Model* (disponible en anglais seulement)

IEC 62541-3, *Architecture unifiée OPC – Partie 3: Modèle d'espace d'adressage*

IEC 62541-4, *Architecture unifiée OPC – Partie 4: Services*

IEC 62541-5, *Architecture unifiée OPC – Partie 5: Modèle d'Information*

IEC 62541-6, *Architecture unifiée OPC – Partie 6: Mappings*

IEC 62541-7, *Architecture unifiée OPC – Partie 7: Profils*

IEC 62541-8, *Architecture unifiée OPC – Partie 8: Accès aux données*

IEC 62541-12, *Architecture unifiée OPC – Partie 12: Services globaux et de découverte*

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

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

IETF RFC 7159, *The JavaScript Object Notation (JSON) Data Interchange Format* http://www.ietf.org/rfc/rfc7159.txt

3 Termes, définitions et termes abrégés

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans l'IEC TR 62541-1, l'IEC 62541-3 et l'IEC 62541-4 ainsi que les suivants s'appliquent.

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse http://www.electropedia.org/
- ISO Online browsing platform: disponible à l'adresse http://www.iso.org/obp

3.1.1
DataSetClass
modèle déclarant le contenu d'un *DataSet*

Note 1 à l'article: Une *DataSetClass* est utilisée pour saisir les *DataSets* à utiliser dans différents *Éditeurs* et pour filtrer dans les *Abonnés*.

3.1.2
DataSetMetaData
données qui décrivent le contenu et la sémantique d'un *DataSet*

3.1.3
DataSetReader
entité qui reçoit des DataSetMessages à partir d'un Intergiciel Orienté Message

Note 1 à l'article: Il s'agit du composant qui extrait un *DataSetMessage* d'un *NetworkMessage* envoyé par l'*Intergiciel Orienté Message* et qui décode le *DataSetMessage* en *DataSet* afin de le traiter ultérieurement dans l'*Abonné*.

3.1.4
DataSetWriter
entité qui crée des *DataSetMessages* à partir de *DataSets* et qui les publie par l'intermédiaire d'un *Intergiciel Orienté Message*

Note 1 à l'article: Un *DataSetWriter* code un *DataSet* en un *DataSetMessage* et inclut ce dernier dans un *NetworkMessage* afin de le publier par l'intermédiaire d'un *Intergiciel Orienté Message*.

3.1.5
PublishedDataSet
configuration des données d'application à publier en tant que *DataSet*

Note 1 à l'article: Un *PublishedDataSet* peut être une liste de *Variables* surveillées ou une sélection d'*Événements*.

3.1.6
SecurityGroup
ensemble de paramètres de sécurité et de clés de sécurité utilisés pour accéder aux messages provenant d'un *Éditeur*

Note 1 à l'article: Un *SecurityGroup* est une abstraction qui représente les paramètres de sécurité et les clés de sécurité pouvant être utilisés pour accéder aux messages provenant d'un *Éditeur*. Il est identifié par un identificateur unique appelé *SecurityGroupId*. Cet identificateur est unique dans le *Service de Clés de Sécurité*.

3.1.7
SubscribedDataSet
configuration de distribution des *DataSets* reçus

Note 1 à l'article: Un *SubscribedDataSet* peut être un mapping de champs *DataSet* avec les *Variables* de l'*AddressSpace* de l'*Abonné*.

3.2 Termes abrégés

AMQP Advanced Message Queuing Protocol (Protocole avancé de mise en file d'attente de messages)

AS authorization service (service d'autorisation)

CTL certificate trust list (liste de certificats de confiance)

IHM interface homme-machine

IGMP Internet Group Management Protocol (Protocole de gestion des groupes Internet)

MIME Multipurpose Internet Mail Extensions (Extensions multifonctions du courrier Internet)

MQTT Message Queuing Telemetry Transport (Transport par télémesure des messages en file d'attente)

MTU maximum transmission unit (taille maximale de l'unité de transfert)

PCP priority code point (code de priorité)

QoS quality of service (qualité de service)

SKS security key service (service de clés de sécurité)

UA Unified Architecture (Architecture unifiée)

UADP UA Datagram Protocol (Protocole datagramme UA)

UDP User Datagram Protocol (Protocole datagramme d'utilisateur)

URI Uniform Resource Identifier (Identificateur uniforme de ressources)

URL Uniform Resource Locator (Localisateur uniforme de ressources)

VID VLAN identifier (identificateur du réseau local virtuel)

4 Vue d'ensemble

4.1 Domaines d'application

Dans *PubSub*, les *Applications* OPC UA dont les rôles sont *Éditeurs* et *Abonnés* sont découplées. Le nombre d'*Abonnés* recevant des données d'un *Éditeur* n'a aucun impact sur l'*Éditeur*. *PubSub* est ainsi adapté aux applications où l'indépendance d'emplacement et/ou l'évolutivité sont exigées.

Les exemples suivants représentent des utilisations de *PubSub*:

- communication configurable entre homologues entre les contrôleurs et entre les contrôleurs et les IHM. Les homologues ne sont pas directement connectés et n'ont même pas besoin de connaître l'existence de chacun. L'échange de données exige souvent un intervalle de temps fixe; il peut s'agir d'une liaison point à point ou d'une distribution de données à plusieurs récepteurs;

- flux de travaux asynchrones. Par exemple, une application de traitement des commandes peut placer une commande dans une file d'attente de messages ou un ESB (*Enterprise Service Bus*, bus de services d'entreprises). La commande peut ensuite être traitée par un ou plusieurs employés;
- connexion à plusieurs systèmes. Par exemple, les capteurs ou les actionneurs peuvent écrire des journaux dans un système de surveillance, une IHM, une application d'archivage pour interrogation ultérieure, etc.;
- les *Serveurs* OPC UA qui représentent des services ou des dispositifs peuvent transférer des données aux applications hébergées dans le Cloud. Par exemple, les serveurs d'arrière-plan, les outils d'analyse des données volumineuses pour l'optimisation du système et la maintenance prédictive.

4.2 Couches d'abstraction

PubSub est conçu pour être flexible et n'est pas lié à un système de messagerie particulier. L'ensemble des composants et des activités est décrit de manière abstraite dans l'Article 5 et ne représente pas une spécification à mettre en œuvre. Les paramètres de communication concrets sont spécifiés à l'Article 6. Les mappings de protocole de transport concrets et les mappings de message concrets sont spécifiés à l'Article 7.

Défini avec ces couches d'abstraction, *PubSub* peut être utilisé pour transporter différents types d'informations par l'intermédiaire de réseaux comportant des caractéristiques différentes, comme représenté dans les deux exemples suivants:

- un modèle *PubSub* avec transport UDP et messages à codage binaire peut être parfaitement adapté aux environnements de production pour le transfert fréquent de volumes de données réduits. Il permet également d'échanger des données dans des configurations un à un et un à plusieurs;
- l'utilisation de protocoles de messagerie normalisés établis (par exemple AMQP ou MQTT) avec le codage de données JSON prend en charge le chemin d'intégration Cloud et permet de traiter rapidement les informations dans des systèmes modernes d'analyse des séquences et des lots.

4.3 Découplage à l'aide d'un intergiciel

Dans *PubSub*, les *Applications* OPC UA participantes peuvent endosser les rôles d'*Éditeur* et d'*Abonné*. Les *Éditeurs* sont les sources des données, alors que les *Abonnés* utilisent ces données. La communication dans *PubSub* repose sur les messages. Les *Éditeurs* envoient des messages à un *Intergiciel Orienté Message*, sans connaître les *Abonnés* éventuels pouvant exister. De même, les *Abonnés* manifestent de l'intérêt pour certains types de données et traitent les messages contenant ces données sans connaître les *Éditeurs*.

Un *Intergiciel Orienté Message* est une infrastructure logicielle ou matérielle qui prend en charge l'envoi et la réception de messages entre les systèmes distribués. La mise en œuvre de cette distribution dépend de l'*Intergiciel Orienté Message*.

La Figure 1 indique que les *Éditeurs* et les *Abonnés* n'interagissent qu'avec l'*Intergiciel Orienté Message*, qui offre un mécanisme de transfert des données vers un ou plusieurs récepteurs.

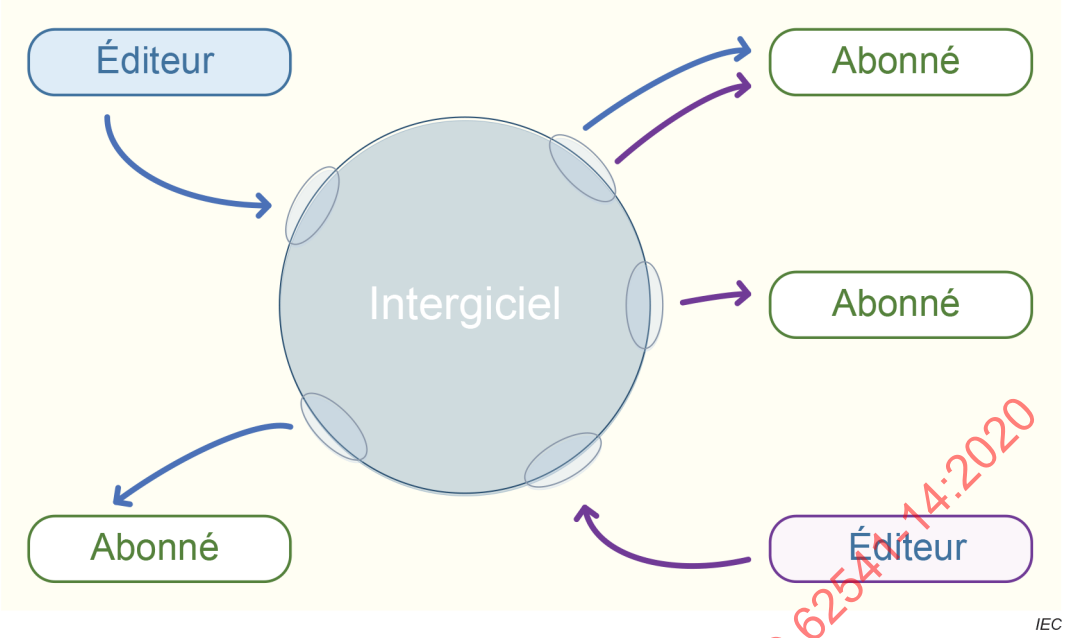


Figure 1 – Vue d'ensemble du modèle Publication/Abonnement

Pour couvrir un grand nombre de cas d'utilisation, le modèle *PubSub* OPC UA prend en charge deux variantes d'*Intergiciel Orienté Message* très différentes:

- une forme sans courtier, où l'*Intergiciel Orienté Message* est l'infrastructure réseau capable d'acheminer les messages fondés sur le datagramme. Les *Abonnés* et les *Éditeurs* utilisent des protocoles de datagramme comme UDP;
- une forme reposant sur un courtier, où le composant principal de l'*Intergiciel Orienté Message* est un *Courtier* de messages. Les *Abonnés* et les *Éditeurs* utilisent des protocoles de messagerie normalisés comme AMQP ou MQTT pour communiquer avec le *Courtier*. Tous les messages sont publiés dans des files d'attente spécifiques (par exemple, sujets, nœuds) présentées par le *Courtier*, et les *Abonnés* peuvent se mettre à l'écoute pour détecter ces files d'attente. Le *Courtier* peut traduire les messages depuis le protocole de messagerie formel de l'*Éditeur* vers le protocole de messagerie formel de l'*Abonné*.

4.4 Synergie des modèles

Les modèles *PubSub* et *Client/Serveur* reposent tous deux sur le *Modèle d'Information* OPC UA. Ainsi, *PubSub* peut facilement être intégré dans les *Serveurs* OPC UA et les *Clients* OPC UA. En règle générale, un *Éditeur* est un *Serveur* OPC UA (le propriétaire des informations) et un *Abonné* est un *Client* OPC UA. Le *Modèle d'Information PubSub* pour la configuration (voir 6.2.2) promeut avant tout la configuration des *Éditeurs* et des *Abonnés* à l'aide du modèle *Client/Serveur* OPC UA.

La communication *PubSub* n'exige toutefois pas une telle dépendance de rôle. Les *Clients* OPC UA peuvent ainsi être des *Éditeurs* et les *Serveurs* OPC UA peuvent être des *Abonnés*. En réalité, il n'est pas nécessaire que les *Éditeurs* ou *Abonnés* soient un *Serveur* OPC UA ou un *Client* OPC UA pour prendre part aux communications *PubSub*.

Plus de détails sur la façon dont les *Abonnements* dans le modèle de communication *Client Serveur* se comparent avec *PubSub*, sont décrits dans l'Annexe B.

5 Concepts PubSub

5.1 Généralités

L'Article 5 décrit les concepts *PubSub* OPC UA généraux.

Le *DataSet* constitue la charge utile des messages fournis par l'*Éditeur* et utilisés par l'*Abonné*. Le *DataSet* est décrit en 5.2. Le mapping avec les messages est décrit en 5.3. Les entités participantes telles que l'*Éditeur* et l'*Abonné* sont décrites en 5.4.

Les paramètres de communication abstraits sont décrits à l'Article 6.

Le mapping de ce modèle avec les messages et les protocoles de transport concrets est défini à l'Article 7.

Le *Modèle d'Information* OPC UA pour la configuration *PubSub* décrit à l'Article 9 spécifie les *Objets* normalisés contenus dans un *AddressSpace* OPC UA utilisés pour créer, modifier et présenter une configuration *PubSub* OPC UA.

La Figure 2 représente les entités *Éditeur* et *Abonné*. Elle représente le flux des messages entre un *Éditeur* et un ou plusieurs *Abonnés*. Le modèle de communication *PubSub* prend en charge plusieurs autres scénarios; par exemple, un *Éditeur* peut envoyer un *DataSet* à plusieurs *Intergiciels Orientés Message* et un *Abonné* peut recevoir des messages de plusieurs *Éditeurs*.

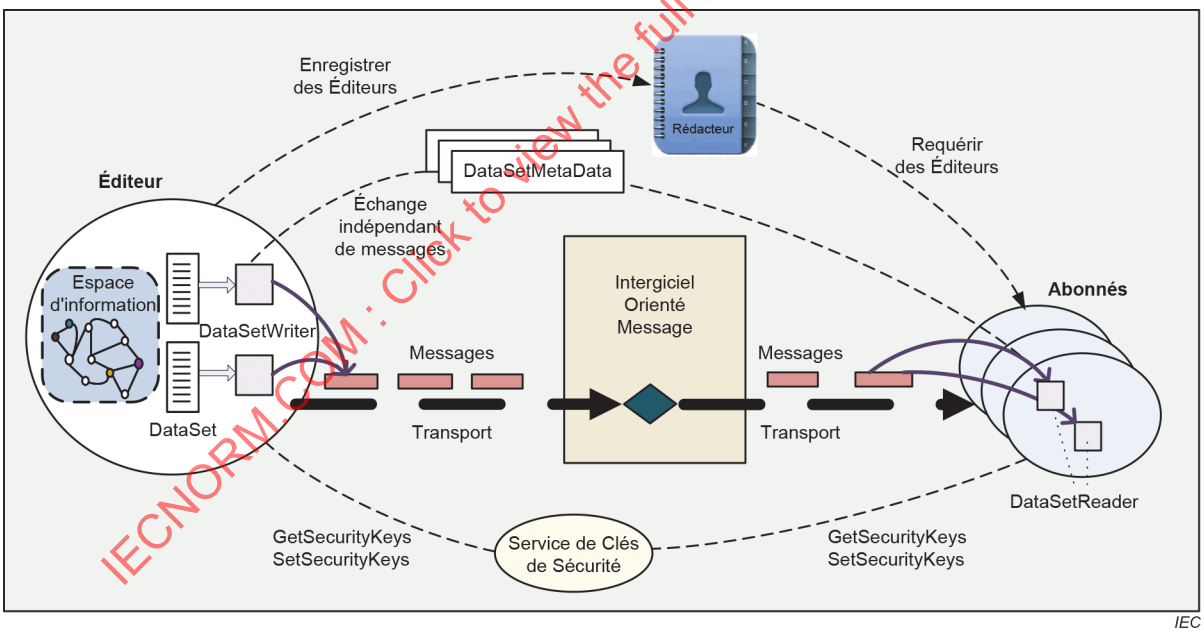


Figure 2 – Entités Éditeur et Abonné

Les *Éditeurs* et les *Abonnés* sont faiblement couplés. Dans la plupart des cas, ils ne se connaissent même pas. Leur principale relation repose sur la compréhension partagée de types de données spécifiques (*DataSets*), les caractéristiques de publication des messages qui contiennent ces données, et l'*Intergiciel Orienté Message*.

Les "messages" de la Figure 2 représentent des *NetworkMessages.* Chaque *NetworkMessage* inclut des informations d'en-tête (données d'identification et de sécurité, par exemple) et un ou plusieurs *DataSetMessages* (la charge utile). Les *DataSetMessages* peuvent être signés et chiffrés conformément à la sécurité de messages configurée. Un *Serveur de Clés de Sécurité* est chargé de distribuer les clés de sécurité nécessaires à la sécurité des messages.

Chaque *DataSetMessage* est créé à partir d'un *DataSet*. Un composant d'*Éditeur* appelé *DataSetWriter* génère une séquence continue de *DataSetMessages*. La syntaxe et la sémantique des *DataSets* sont décrites par les *DataSetMetaData*. Le choix des informations pour un *DataSet* dans l'*Éditeur* et les paramètres d'acquisition des données sont appelés *PublishedDataSet*. *DataSet*, *DataSetMetaData* et *PublishedDataSet* sont décrits en 5.2.

NOTE Le répertoire PubSub est une entité facultative qui permet aux *Éditeurs* de notifier leurs *PublishedDataSets* et leurs paramètres de communication. Cette fonction de répertoire est planifiée pour une édition ultérieure de l'IEC 62541-14.

5.2 DataSet

5.2.1 Généralités

Un *DataSet* peut être vu comme une liste de paires composées d'un nom et d'une valeur représentant un *Événement* ou comme une liste de *Valeurs de Variables*.

Un *DataSet* peut être créé à partir d'un *Événement* ou d'un échantillon de *Valeurs de Variables*. La configuration de ce collecteur de données d'application est appelée *PublishedDataSet*. Les champs *DataSet* peuvent être définis pour représenter n'importe quelle information. Ils peuvent par exemple représenter des *Variables* internes à l'*Éditeur*, des *Événements* issus de l'*Éditeur* ou collectés par ce dernier, des données de réseau, ou des données issues de sous-dispositifs.

Les *DataSetMetaData* décrites en 5.2.3 définissent la structure et le contenu d'un *DataSet*.

Pour la publication, un *DataSet* est codé en *DataSetMessage*. Un ou plusieurs *DataSetMessages* sont combinés pour constituer la charge utile d'un *NetworkMessage*.

La Figure 3 représente l'utilisation des *DataSets* à des fins de publication.

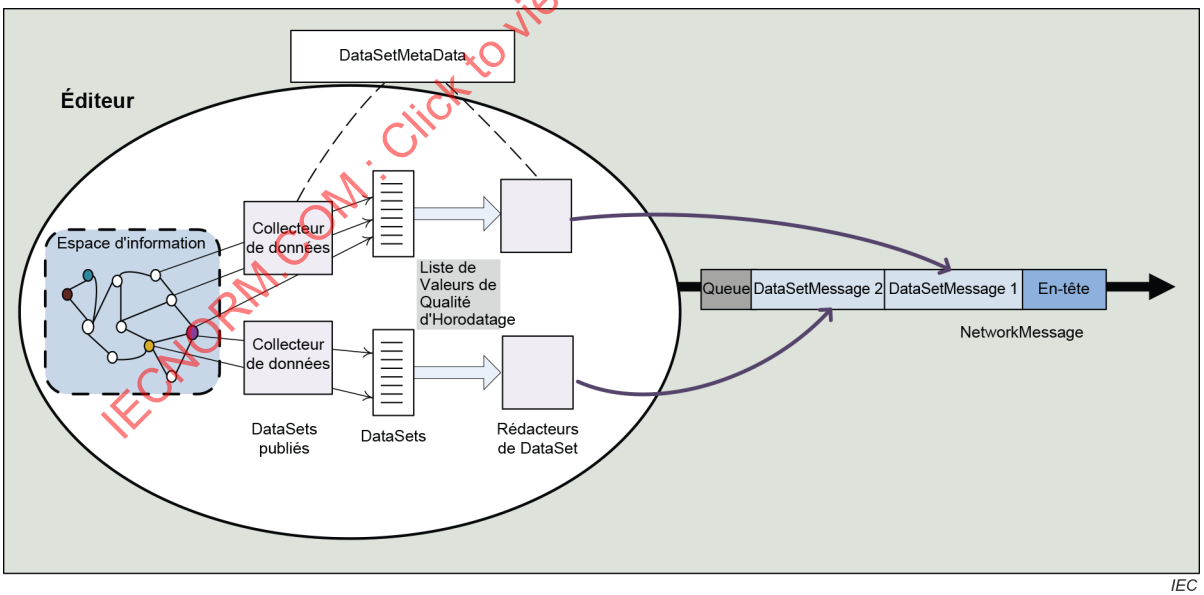


Figure 3 – DataSet dans le processus de publication

Un *PublishedDataSet* est similaire à un *Événement MonitoredItem* ou à une liste de données *MonitoredItems* dans le modèle d'*Abonnement Client/Serveur*. De la même manière qu'un *Événement MonitoredItem*, un *PublishedDataSet* peut choisir une liste de champs d'*Événement*. De la même manière que les données *MonitoredItems*, le *PublishedDataSet* peut contenir une liste de *Variables*.

Un *DataSet* ne définit pas le mécanisme permettant de le coder, de le sécuriser et de le transporter. Un *DataSetWriter* gère la création d'un *DataSetMessage* pour un *DataSet*. Le *DataSetWriter* contient les paramètres nécessaires au codage et au transport d'un *DataSetMessage*. La plupart de ces paramètres dépendent de l'*Intergiciel Orienté Message* choisi.

La configuration des *DataSets* et la méthode d'obtention des données en vue de leur publication peuvent être configurées en utilisant le modèle de configuration *PubSub* défini en 8.2 ou les outils de configuration propres au fournisseur.

5.2.2 DataSetClass

Les *DataSets* peuvent être individuels pour un *Éditeur*, ou ils peuvent être dérivés d'une *DataSetClass*. Cette *DataSetClass* agit comme un modèle déclarant le contenu d'un *DataSet*. La *DataSetClass* est identifiée par un identificateur globalement unique, le *DataSetClassId* (voir 6.2.2.2).

Les *DataSetMetaData* sont identiques pour tous les *PublishedDataSets* dont la configuration repose sur cette *DataSetClass*. Le *DataSetClassId* doit figurer dans le champ correspondant des *DataSetMetaData*.

Lorsque tous les *DataSetMessages* d'un *NetworkMessage* sont créés à partir de *DataSets* qui sont des instances de la même *DataSetClass*, le *DataSetClassId* de cette classe peut être fourni dans l'en-tête du *NetworkMessage*.

5.2.3 DataSetMetaData

Les *DataSetMetaData* décrivent le contenu et la sémantique d'un *DataSet*. La description de la structure inclut les attributs de *DataSet* globaux (par exemple nom et version) et un ensemble de champs avec leur nom et leur type de données. L'ordre des champs des *DataSetMetaData* doit correspondre à l'ordre des valeurs des *DataSetMessages* publiés.

Le *DataSetMetaDataType* est défini en 6.2.2.1.2.

Exemple de description (simplifiée, en pseudo-code):

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

Les *Abonnés* utilisent les *DataSetMetaData* pour décoder les valeurs d'un *DataSetMessage* en *DataSet*. Les *Abonnés* peuvent utiliser le nom et le type de données pour traiter ou afficher les données publiées ultérieurement.

Chaque *DataSetMessage* contient également la version des *DataSetMetaData* à laquelle il est conforme. Cela permet aux *Abonnés* de vérifier s'ils possèdent les *DataSetMetaData* correspondantes. Le *ConfigurationVersionDataType* associé est défini en 6.2.2.1.5.

Les *DataSetMetaData* peuvent être spécifiques à un *PublishedDataSet* unique ou identiques pour tous les *PublishedDataSets* dont la configuration repose sur une *DataSetClass* (voir 5.2.2).

Les *Abonnés* ont plusieurs options pour obtenir les *DataSetMetaData* initiales:

- l'*Abonné* est un *Client* OPC UA capable d'obtenir les informations de configuration nécessaires auprès du modèle de configuration *PubSub* (voir 9.1.4.2.1) fourni par l'*Éditeur* ou auprès d'un serveur de configuration;
- l'*Abonné* prend en charge les *Méthodes* de configuration OPC UA définies dans le modèle de configuration *PubSub*;

- l'*Abonné* reçoit les *DataSetMetaData* sous la forme d'un *NetworkMessage* de la part de l'*Éditeur*. Une option permettant à l'*Abonné* de demander ce *NetworkMessage* auprès de l'*Éditeur* peut être exigée;
- l'*Abonné* est configuré en utilisant les méthodes de configuration propres au produit.

Il existe plusieurs options pour échanger les *DataSetMetaData* entre l'*Éditeur* et l'*Abonné* si la configuration est modifiée.

- Les *DataSetMetaData* sont envoyées sous la forme d'un *NetworkMessage* de l'*Éditeur* à l'*Abonné* avant d'envoyer les *DataSetMessages* dont le contenu a été modifié. Il convient que l'*Intergiciel Orienté Message* assure une livraison fiable du message. Le mapping pour l'*Intergiciel Orienté Message* définit une méthode permettant à l'*Abonné* de demander les *DataSetMetaData*. L'*Abonné* bascule à l'état d'erreur s'il n'a pas reçu les nouvelles *DataSetMetaData* qui correspondent à la *ConfigurationVersion* du *DataSetMessage* reçu.
- L'*Abonné* est mis à jour automatiquement à l'aide des *Méthodes* de configuration OPC UA définies dans le modèle de configuration *PubSub* lors de la mise à jour du *DataSet* dans l'*Éditeur*.
- L'*Abonné* est un *Client* OPC UA capable d'obtenir la mise à jour de l'*Éditeur* ou d'un serveur de configuration par le biais des informations présentées par le modèle de configuration *PubSub*.
- L'*Abonné* est mis à jour à l'aide des méthodes de configuration propres au produit lorsque le *DataSet* de l'*Éditeur* a été modifié.

5.3 Messages

5.3.1 Généralités

Le terme "message" est utilisé à différentes fins dans le domaine de la messagerie. Il se réfère parfois uniquement à la charge utile (les données d'application) et parfois au paquet réseau qui inclut également les données de protocole, de sécurité ou de codage. Pour éviter toute confusion, le présent document définit formellement le terme *DataSetMessage* comme les données d'application (la charge utile) fournies par l'*Éditeur* et le terme *NetworkMessage* comme le message transféré et envoyé par un *Intergiciel Orienté Message* spécifique. Les *DataSetMessages* sont imbriqués dans les *NetworkMessages*. La Figure 4 représente la relation de ces types de messages.

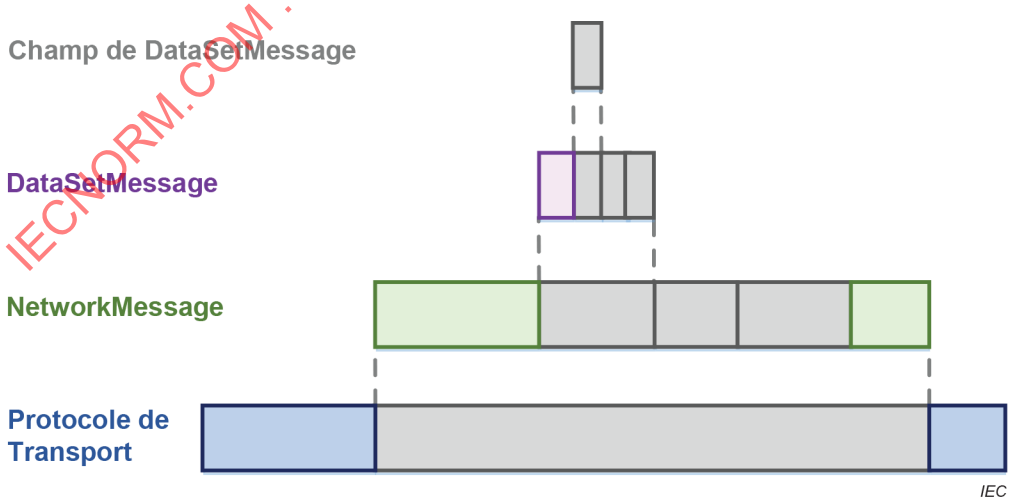


Figure 4 – Couches de messages PubSub OPC UA

Les en-têtes et les définitions propres au protocole de transport sont donnés en 7.3.

Une définition abstraite de *DataSetMessage* et de *NetworkMessage* est donnée en 5.3.2 à 5.3.4. La structure concrète dépend du mapping avec les messages; elle est décrite en 7.2.

5.3.2 Champ DataSetMessage

Un champ *DataSetMessage* est la représentation d'un champ *DataSet* dans un *DataSetMessage*.

Un champ *DataSet* contient la valeur réelle ainsi que des informations supplémentaires sur la valeur, telles que son statut et son horodatage.

Un champ *DataSet* peut être représenté sous la forme d'une *DataValue*, d'une *Variante* ou de *RawData* dans le champ *DataSetMessage*. La représentation dépend du *DataSetFieldContentMask* défini en 6.2.3.2.

La représentation sous la forme d'une *DataValue* est utilisée si la valeur, le statut et l'horodatage sont inclus dans le *DataSetMessage*.

La représentation sous la forme d'une *Variante* est utilisée s'il convient d'inclure la valeur ou le statut "bad" dans le *DataSetMessage*.

La représentation sous la forme de *RawData* est le format le plus efficace qui est utilisé si un statut et un horodatage communs par DataSet sont suffisants.

5.3.3 DataSetMessage

Un *DataSetMessage* est créé à partir d'un *DataSet*. Il se compose d'un en-tête et des champs codés du *DataSet*.

Selon le *DataSetMessageContentMask* configuré, un *DataSetMessage* peut exister sous différentes formes et présenter différentes caractéristiques. Les *DataSetMessages* ne contiennent aucune information sur l'acquisition de données ou la source d'informations dans l'*Éditeur*.

Les informations supplémentaires suivantes sont incluses dans l'en-tête:

DataSetWriterId	Identifie le *DataSetWriter* et indirectement le *PublishedDataSet.*
Numéro de séquence	Nombre incrémenté pour chaque *DataSetMessage*. Peut être utilisé pour vérifier l'ordre des messages et pour détecter des messages manquants.
Horodatage	Horodatage qui indique le moment où les données de ce *DataSetMessage* ont été obtenues.
Version	Informations de version sur la configuration des *DataSetMetaData.*
Statut	Informations de statut sur les données contenues dans ce *DataSetMessage.*
Entretien	Lorsqu'aucun *DataSetMessage* n'est envoyé pendant une période de temps configurée, un *DataSetMessage* d'entretien est envoyé pour signaler aux *Abonnés* que l'*Éditeur* est toujours actif.

Certains codages établissent une distinction entre les *DataSetMessages* de type image clé et les *DataSetMessages* de type image delta. Un *DataSetMessages* de type image clé inclut des valeurs pour tous les champs du *DataSet.* Un *DataSetMessage* de type image delta contient uniquement le sous-ensemble qui a été modifié depuis le *DataSetMessage* précédent.

Un *DataSetMessage* de type image clé est envoyé après un nombre configuré de *DataSetMessages*.

5.3.4 NetworkMessage

Le *NetworkMessage* est un conteneur pour *DataSetMessages* et inclut les informations partagées entre *DataSetMessages*. Ces informations comprennent les éléments suivants:

PublisherId	Identifie l'*Éditeur*.
Données de sécurité	Uniquement disponibles pour les codages prenant en charge la sécurité de messages. Les informations pertinentes sont spécifiées dans le mapping avec les messages.
Champs promus	Champs choisis dans le *DataSet* également envoyés dans l'en-tête.
Charge utile	Un ou plusieurs *DataSetMessages.*

La charge utile, qui se compose des *DataSetMessages*, est chiffrée conformément à la sécurité de messages configurée. Les champs individuels d'un *DataSetMessage* peuvent être marqués comme étant des "champs promus". Ces champs, destinés au filtrage ou au routage, ne sont jamais chiffrés. La méthode et l'emplacement d'insertion des champs promus dépendent du format du *NetworkMessage* et du protocole utilisé. L'en-tête du *NetworkMessage* n'est pas chiffré pour permettre un filtrage efficace.

5.3.5 Sécurité de messages

La sécurité de messages dans *PubSub* concerne l'intégrité et la confidentialité de la charge utile des messages publiés. Les concepts de base de la sécurité OPC UA sont définis dans l'IEC TR 62541-2. Le niveau de sécurité peut être:

- pas de sécurité;
- signature sans chiffrement;
- signature et chiffrement.

La sécurité de messages est une sécurité de bout en bout (de l'*Éditeur* à l'*Abonné*) qui exige de connaître les clés cryptographiques nécessaires à la signature et au chiffrement côté *Éditeur* ainsi qu'à la validation de la signature et au déchiffrement côté *Abonné*.

Les clés utilisées pour la sécurité des messages sont gérées dans le contexte d'un *SecurityGroup.* Les concepts de base d'un *SecurityGroup* sont décrits en 5.3.7.

Le présent document définit un cadre de distribution général pour les clés cryptographiques. Ce cadre est présenté en 5.4.3.

L'ensemble des paramètres pertinents pour la sécurité de messages est décrit en 6.2.4. Ces paramètres ne dépendent d'aucune sécurité de transport de niveau *Courtier*.

La sécurité de messages pour *PubSub* ne dépend pas du mapping avec les protocoles de transport et est entièrement définie par OPC UA.

5.3.6 Sécurité de transport

La sécurité de transport est spécifique au mapping avec les protocoles de transport.

Lors de l'utilisation d'un intergiciel reposant sur un courtier (voir 5.4.4.2.2), la confidentialité et l'intégrité peuvent être assurées avec la sécurité de transport entre les *Éditeurs* et le *Courtier*, et entre les *Abonnés* et le *Courtier*. La sécurité de niveau *Courtier* exige également que l'ensemble des *Éditeurs* et des *Abonnés* possède des authentifiants leur accordant un accès à une ressource de *Courtier*.

La sécurité de transport peut être une sécurité pas à pas et présenter des risques d'attaques de l'homme du milieu. Elle exige également de faire confiance au *Courtier*, étant donné que

ce dernier peut lire les messages. La combinaison de la sécurité de transport à la sécurité de messages réduit ce risque.

5.3.7 SecurityGroup

Un *SecurityGroup* est une abstraction qui représente les paramètres de sécurité des messages et les clés de sécurité pour un sous-ensemble de *NetworkMessages* échangés entre les *Éditeurs* et les *Abonnés*. Les clés de sécurité permettent de chiffrer et de déchiffrer les *NetworkMessages* et de générer et vérifier les signatures sur un *NetworkMessage*.

Un *Service de Clés de Sécurité* (SKS, Security Key Service) gère les *SecurityGroups* et maintient un mapping entre les *Rôles* et leurs *Autorisations* d'accès pour un *SecurityGroup*. Ce mapping définit si un *Éditeur* ou un *Abonné* détient un accès aux clés de sécurité d'un *SecurityGroup*. Ce SKS est décrit en détail en 5.4.3.

Il est identifié par un identificateur unique appelé *SecurityGroupId*. Cet identificateur est unique au sein du SKS. Pour ses *PublishedDataSets,* un *Éditeur* a besoin de connaître le *SecurityGroupId*. Pour les *Abonnés,* le *SecurityGroupId* est distribué sous la forme de métadonnées avec les *DataSetMetaData*. Les métadonnées d'un *SecurityGroupId* incluent l'*EndpointDescription* du SKS en question. Les Éditeurs et les Abonnés utilisent l'*EndpointDescription* pour accéder au SKS et le *SecurityGroupId* pour obtenir les clés de sécurité d'un *SecurityGroup*.

5.4 Entités

5.4.1 Éditeur

5.4.1.1 Généralités

L'*Éditeur* est l'entité *PubSub* qui envoie des *NetworkMessages* à un *Intergiciel Orienté Message*. Il représente une certaine source d'informations, comme un dispositif de commande, un processus de fabrication, une station météorologique ou une place boursière.

En règle générale, un *Éditeur* est aussi un *Serveur* OPC UA. Pour les concepts *PubSub* abstraits, il s'agit toutefois d'une entité arbitraire et il convient de ne pas prendre pour hypothèse qu'il s'agit d'un individu ou même d'un nœud de réseau spécifique (une adresse IP ou MAC) ou d'une application spécifique. La Figure 5 représente un *Éditeur* avec les processus de collecte des données, de codage et d'envoi des messages.

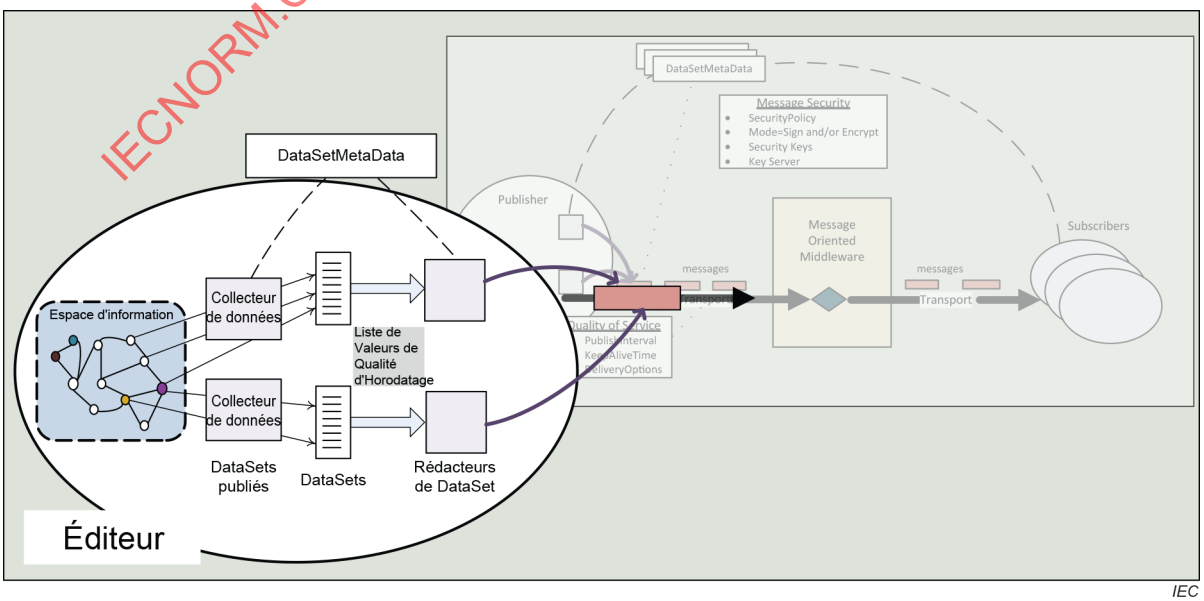


Figure 5 – Détails d'un Éditeur

Un seul *Éditeur* peut prendre en charge plusieurs *PublishedDataSets* et plusieurs *DataSetWriters* dans un ou plusieurs *Intergiciels Orientés Message*. Un *DataSetWriter* est un composant logique d'un *Éditeur*. Pour plus d'informations sur le processus d'écriture de *DataSets*, voir 5.4.1.2.

Si l'*Éditeur* est un *Serveur* OPC UA, il peut présenter la configuration de l'*Éditeur* dans son *AddressSpace*. Ces informations peuvent être créées au moyen d'outils de configuration propres au produit ou à l'aide des *Méthodes* définies par OPC UA. Le *Modèle d'Information* OPC UA pour la configuration *PubSub* est spécifié à l'Article 9.

5.4.1.2 Envoi de messages

La Figure 6 représente le processus interne à un *Éditeur* lors de la création et de l'envoi de messages ainsi que les paramètres exigés pour ce processus. Il convient de voir les composants tels que la collecte de *DataSets* ou le *DataSetWriter* comme étant abstraits. Ils peuvent ne pas exister dans chaque *Éditeur* sous la forme d'entités indépendantes. Il est essentiel que des processus similaires existent pour générer les messages *PubSub* OPC UA.

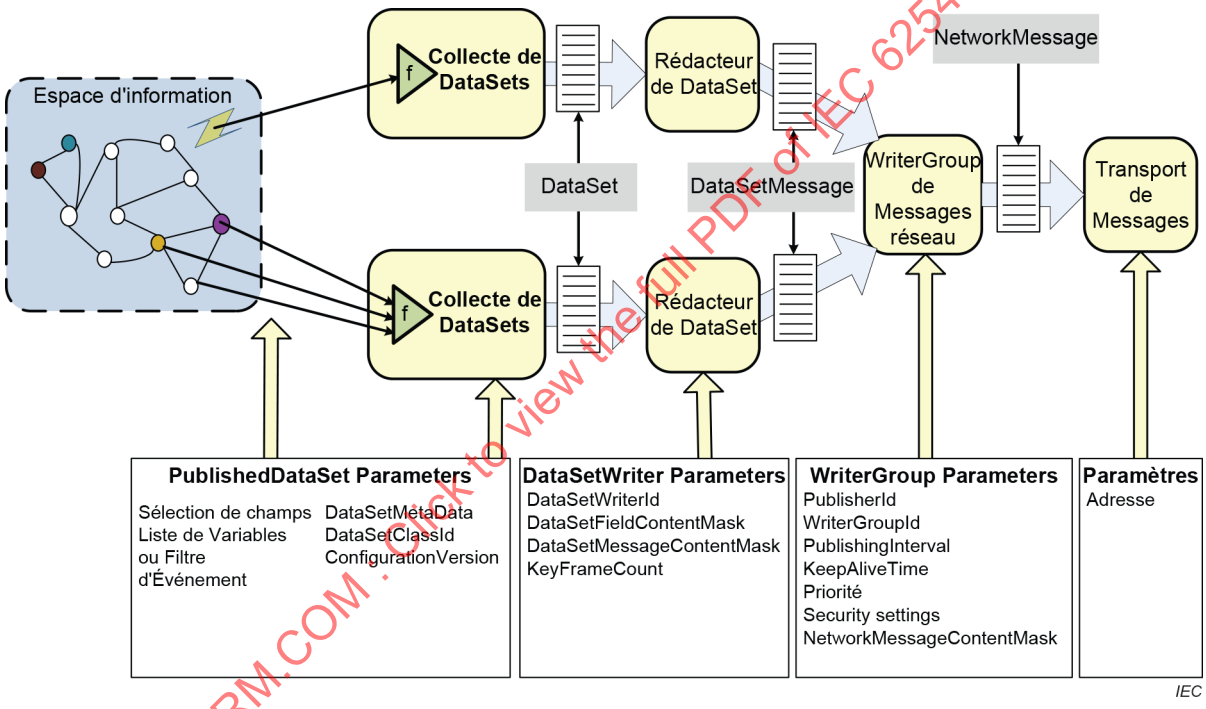


Figure 6 – Séquence d'envoi de messages d'Éditeur

Le processus d'envoi est guidé par différents paramètres au cours de différentes étapes logiques. Les paramètres définissent par exemple quand et comment déclencher la séquence d'envoi ainsi que le codage et la sécurité des messages. Les paramètres de communication PubSub sont définis à l'Article 6.

La première étape consiste à collecter les données (*DataSet*) à publier. La configuration de cette collecte est appelée *PublishedDataSet*. Le *PublishedDataSet* définit également les *DataSetMetaData*. La collecte est une expression générique qui s'applique à différentes options, comme la surveillance de *Variables* dans un *AddressSpace de Serveur* OPC UA, le traitement d'*Événements* OPC UA, ou la lecture de données issues de paquets réseau. Finalement, le processus de collecte génère des valeurs pour les champs individuels d'un *DataSet*.

Au cours de l'étape suivante, un *DataSetWriter* utilise le *DataSet* pour créer un *DataSetMessage*. Les *DataSetMessages* créés par les *DataSetWriters* dans un *WriterGroup*

peuvent être insérés dans un *NetworkMessage* unique. La création d'un *DataSetMessage* est guidée par les paramètres suivants:

- le *DataSetFieldContentMask* (voir 6.2.3.2) contrôle les attributs d'une valeur qui doivent être codés;
- le *DataSetMessageContentMask* (voir 6.3.1.2.2) contrôle les champs d'en-tête qui doivent être codés;
- le *KeyFrameCount* indique si un *DataSetMessage* de type image clé ou image delta doit être créé.

Le *DataSetMessage* généré est transféré à l'étape suivante avec le *DataSetWriterId* (voir 6.2.3.1), le *DataSetClassId* (voir 6.2.2.2), la *ConfigurationVersion* des *DataSetMetaData* (voir 6.2.2.1.5), et une liste de valeurs qui correspondent aux champs propagés configurés.

L'étape suivante consiste à créer le *NetworkMessage*. Le processus utilise les données fournies à l'étape précédente ainsi que le *PublisherId* (voir 6.2.6.1) défini dans le *WriterGroup*. La structure de ce message est propre au protocole. Si le *SecurityMode* (voir 6.2.4.2) exige une sécurité de messages, le *SecurityGroupId* (voir 6.2.4.3) est utilisé pour extraire la *SecurityPolicy* et les clés de sécurité du SKS (voir 5.4.3). Ces informations sont utilisées pour chiffrer et/ou signer le *NetworkMessage* comme exigé par le *SecurityMode*.

L'étape finale consiste à livrer le *NetworkMessage* à l'*Intergiciel Orienté Message* par l'intermédiaire de l'*Adresse* configurée.

5.4.2 Abonné

5.4.2.1 Généralités

Les *Abonnés* sont les utilisateurs des *NetworkMessages* depuis l'*Intergiciel Orienté Message*. Il peut s'agir de *Clients* OPC UA, de *Serveurs* OPC UA ou d'applications qui ne sont ni des applications *Client*, ni des applications *Serveur*, mais qui comprennent uniquement la structure des messages *PubSub* OPC UA. La Figure 7 représente un *Abonné* avec les processus de filtrage, de décodage et de distribution des *NetworkMessages*.

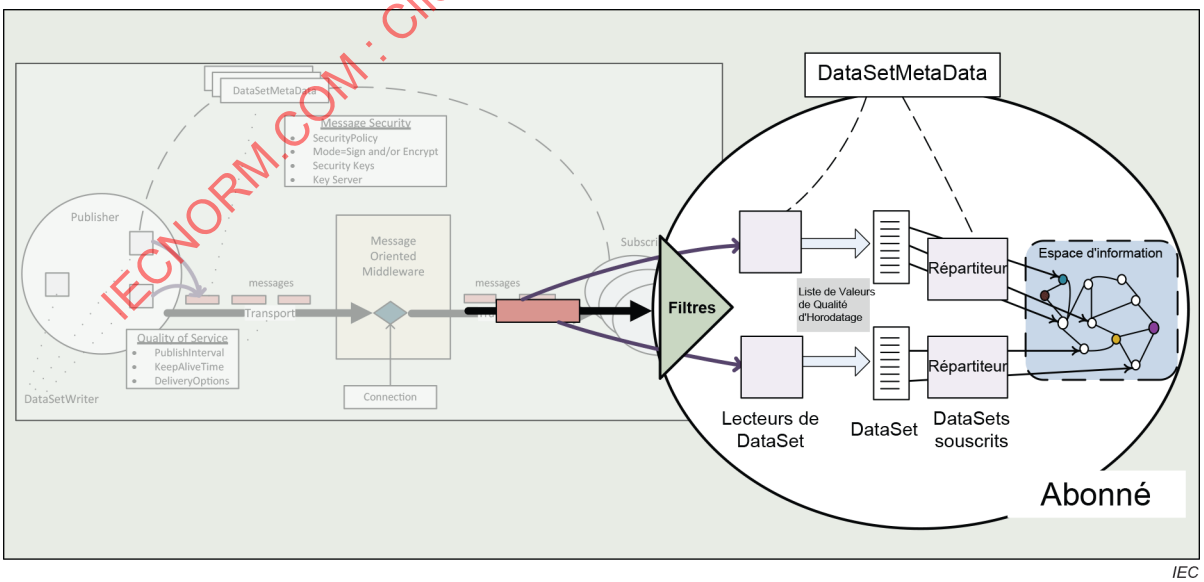


Figure 7 – Détails d'un Abonné

Afin de déterminer les *DataSetMessages* pour lesquels s'abonner et l'*Intergiciel Orienté Message* sur lequel s'abonner, l'*Abonné* a besoin d'être configuré et/ou d'utiliser des mécanismes de découverte.

Les *Abonnés* doivent se préparer à recevoir des messages qu'ils ne comprennent pas ou qui ne sont pas pertinents. Chaque *NetworkMessage* fournit des données non chiffrées dans l'en-tête du *NetworkMessage* afin de prendre en charge l'identification et le filtrage des *Éditeurs*, *DataSetMessages*, *DataSetClasses* pertinentes, ou de tout autre contenu de message pertinent (voir 5.3).

Si un *NetworkMessage* est signé ou signé et chiffré, l'*Abonné* a besoin des clés de sécurité appropriées (voir 5.3.5) pour vérifier la signature et déchiffrer les *DataSetMessages* pertinents.

Lorsqu'un *DataSetMessage* a été choisi comme étant pertinent, il est transféré au *DataSetReader* correspondant pour être décodé en *DataSet*. Pour plus d'informations sur ce processus de lecture de *DataSets*, voir 5.4.2.2. Le *DataSet* généré est ensuite traité ou distribué dans l'*Abonné*.

Si l'*Abonné* est un *Serveur* OPC UA, il peut présenter la configuration du lecteur dans son *AddressSpace*. Ces informations peuvent être créées au moyen d'outils de configuration propres au produit ou à l'aide du modèle de configuration défini par OPC UA. Le *Modèle d'Information* OPC UA pour la configuration *PubSub* est spécifié à l'Article 9.

5.4.2.2 Réception de messages

La Figure 8 représente le processus interne à un *Abonné* lors de la réception, du décodage et de l'interprétation de messages ainsi que le modèle de paramètre exigé pour ce processus. Concernant l'*Éditeur*, il convient de voir les composants comme abstraits.

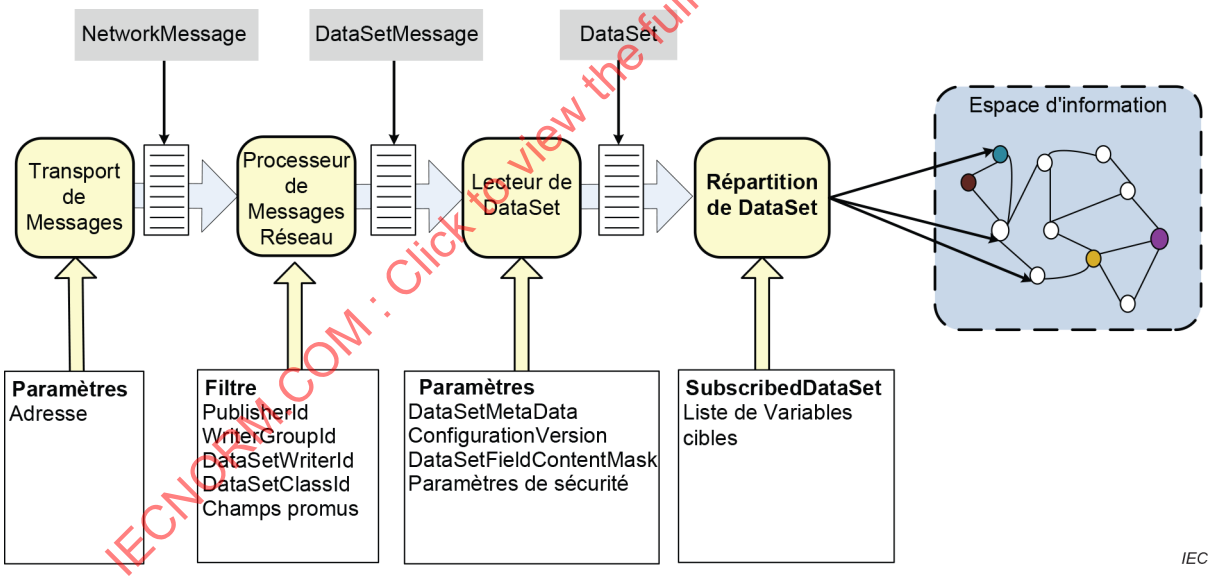


Figure 8 – Séquence de réception de messages d'Abonné

L'*Abonné* a besoin de choisir l'*Intergiciel Orienté Message* exigé et établir une connexion avec ce dernier à l'aide de l'*Adresse* fournie. Cette connexion peut être une simple adresse de multidiffusion si le protocole UDP OPC UA est utilisé, ou une connexion à un *Courtier* de messages si le protocole MQTT ou AMQP est utilisé. Après son abonnement, l'*Abonné* bascule en mode écoute. La séquence démarre lorsqu'un *NetworkMessage* est reçu. L'*Abonné* peut avoir des filtres configurés (tels qu'un *PublisherId*, un *DataSetWriterId* ou un *DataSetClassId*) de sorte qu'il puisse abandonner tous les messages qui ne correspondent pas au filtre.

Lorsqu'un *NetworkMessage* a été accepté, il est déchiffré et décodé. Les paramètres de sécurité sont identiques à ceux de l'*Éditeur*.

Chaque *DataSetMessage* pertinent est transmis à un *DataSetReader*. Les *DataSetMetaData* sont ici utilisées pour décoder le contenu du *DataSetMessage* en *DataSet*. Les *DataSetMetaData* fournissent notamment la syntaxe de champ complète, y compris le nom, le type de données et d'autres *Propriétés* pertinentes telles que les unités techniques. Les informations de version qui figurent dans le *DataSetMessage* et dans les *DataSetMetaData* permettent à l'*Abonné* de détecter les modifications apportées à la version. Si une modification majeure est appliquée, il est nécessaire que l'*Abonné* obtienne des *DataSetMetaData* actualisées.

Tout autre traitement est propre à l'application. Par exemple, une étape de distribution supplémentaire peut mettre en correspondance les valeurs reçues avec les *Nœuds* contenus dans l'*AddressSpace* OPC UA des *Abonnés*. La configuration d'une telle distribution est appelée *SubscribedDataSet*.

5.4.3 Service de clés de sécurité

5.4.3.1 Généralités

Un *Service de Clés de Sécurité* (SKS, Security Key Service) fournit les clés pour la sécurité de messages qui peuvent être utilisées par l'*Éditeur* pour signer et chiffrer les *NetworkMessages*, et par l'*Abonné* pour vérifier la signature des *NetworkMessages* et pour les déchiffrer.

Le SKS est chargé de gérer les clés utilisées pour publier ou utiliser les *NetworkMessages PubSub*. Des clés distinctes sont associées à chaque *SecurityGroupId* dans le système. La *Méthode GetSecurityKeys* présentée par le SKS doit être appelée pour recevoir les informations de clé nécessaires pour un *SecurityGroupId*. *GetSecurityKeys* peut renvoyer plusieurs clés. Dans ce cas, la clé suivante peut être utilisée lorsque la clé actuelle est obsolète, sans faire appel à *GetSecurityKeys* pour chaque clé nécessaire. Le *PubSubKeyServiceType* défini en 8.2 spécifie la *Méthode GetSecurityKeys*.

La *Méthode GetSecurityKeys* peut être mise en œuvre par un *Éditeur* ou par un SKS central. Dans les deux cas, les *NodeIds* notoires de l'*Objet PublishSubscribe* et la *Méthode GetSecurityKeys* correspondante sont utilisés pour appeler la *Méthode GetSecurityKeys*. L'*Objet PublishSubscribe* est défini en 8.4.

La *Méthode SetSecurityKeys* est généralement utilisée par un SKS central pour pousser les clés de sécurité d'un *SecurityGroup* dans un *Éditeur* ou un *Abonné*. La *Méthode* est présentée par les *Éditeurs* ou les *Abonnés* qui ne possèdent pas la fonctionnalité *Client* OPC UA. Elle fait partie du *PublishSubscribeType* défini en 9.1.3.2.

5.4.3.2 Gestion de SecurityGroup

Le SKS est l'entité qui connaît les *SecurityGroups* et qui maintient un mapping entre les *Rôles* et les *SecurityGroups*. Le modèle d'*Autorisation d'Utilisateur* connexe est défini dans l'IEC 62541-3. Ce modèle définit le mapping des identités avec les *Rôles* ainsi que le mécanisme de définition des *Autorisations* pour les *Rôles* sur un *Nœud*. Les *Autorisations* appliquées à un *Objet SecurityGroup* sont utilisées pour déterminer si un *Rôle* a accès aux clés pour le *SecurityGroup*.

La Figure 9 représente un exemple de configuration d'un *SecurityGroup* ainsi que la configuration des *Éditeurs* et *Abonnés* impactés.

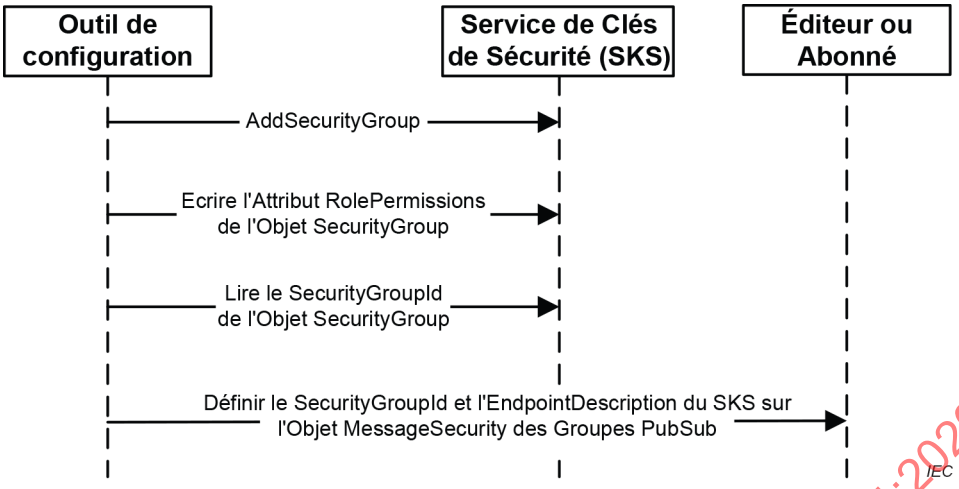


Figure 9 – Séquence de gestion de SecurityGroup

Pour sécuriser les *NetworkMessages*, les clés fournies dans le contexte d'un SecurityGroup doivent être utilisées. Un *SecurityGroup* est créé sur un SKS au moyen de la *Méthode AddSecurityGroup*.

Pour limiter l'accès au *SecurityGroup* et donc aux clés de sécurité, les *Autorisations* doivent être définies sur l'*Objet SecurityGroup*. Cela exige de gérer les *Rôles* et les *Autorisations* dans le SKS.

Pour définir la relation du *SecurityGroup* sur les *Éditeurs* et les *Abonnés*, le *SecurityGroupId* et les *EndpointDescriptions* du SKS sont configurés dans un groupe *PubSub*.

5.4.3.3 Établissement d'une liaison de sécurité pour l'acquisition de clé

L'*Éditeur* ou l'*Abonné* utilise les clés fournies par un SKS pour sécuriser les messages échangés par le biais de l'*Intergiciel Orienté Message*. L'établissement d'une liaison de sécurité permettant de tirer les clés d'un SKS est représenté à la Figure 10. L'établissement d'une liaison de sécurité permettant de pousser les clés d'un SKS vers les *Éditeurs* et les *Abonnés* est décrit à la Figure 11.

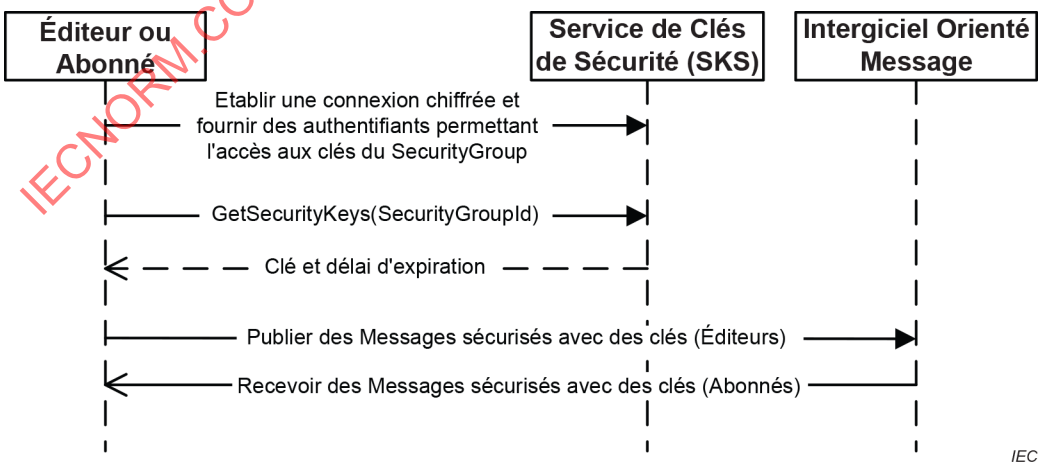


Figure 10 – Établissement d'une liaison de sécurité utilisée pour tirer les clés d'un SKS

Pour tirer les clés, l'*Éditeur* ou l'*Abonné* crée une connexion chiffrée et fournit les authentifiants qui lui permettent d'accéder au *SecurityGroup*. Il transmet ensuite l'identificateur du *SecurityGroup* à la *Méthode GetSecurityKeys* qui vérifie l'*identité* et renvoie

les clés utilisées pour sécuriser les messages du *PubSubGroup*. La *Méthode GetSecurityKeys* est définie en 8.4.

L'accès à la *Méthode GetSecurityKeys* peut utiliser les appels du *Service SessionlessInvoke*. Ces appels utilisent généralement un *Jeton d'Accès* récupéré à partir d'un *Service d'Autorisation*. Les deux concepts sont définis dans l'IEC 62541-4.

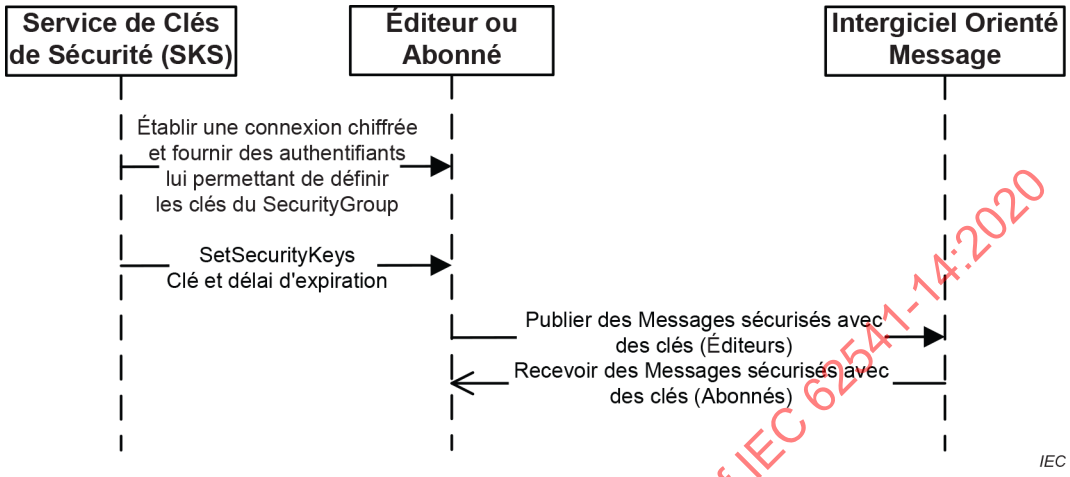


Figure 11 – Établissement d'une liaison de sécurité utilisée pour pousser les clés vers les Éditeurs et les Abonnés

Pour pousser les clés, le SKS crée une connexion chiffrée à un *Éditeur* ou un *Abonné* et fournit les authentifiants qui lui permettent de fournir les clés d'un *SecurityGroup*. Il transmet ensuite l'identificateur du *SecurityGroup* et les clés utilisées pour sécuriser les messages du *SecurityGroup* à la *Méthode SetSecurityKeys*. La *Méthode SetSecurityKeys* est définie en 9.1.3.3.

5.4.3.4 Services d'Autorisation et Service de Clés de Sécurité

L'accès au SKS peut être géré par un *Service d'Autorisation*, comme représenté à la Figure 12.

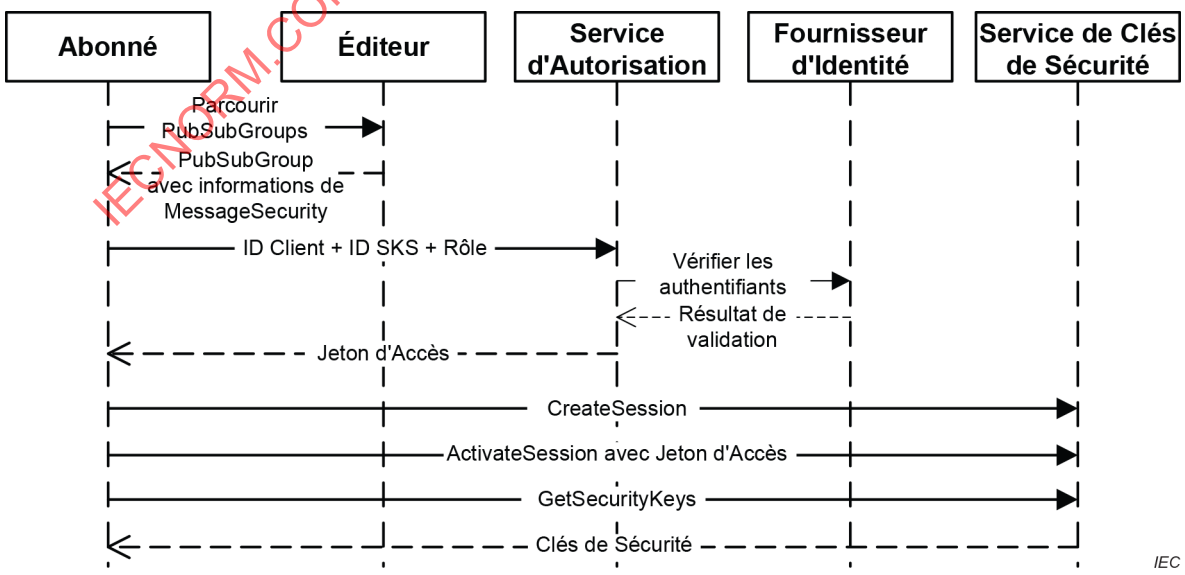


Figure 12 – Établissement d'une liaison de sécurité avec un Service de Clés de Sécurité

Le SKS est un *Serveur* qui présente une *Méthode* appelée *GetSecurityKeys*. Le *Jeton d'Accès* est utilisé pour déterminer si l'application appelante est autorisée à accéder aux clés. L'une des méthodes possibles consiste à vérifier les *Autorisations* attribuées à l'*Objet SecurityGroup* identifié par les arguments de la *Méthode GetSecurityKeys*. Les *Éditeurs* et les *Abonnés* peuvent demander des clés si le *Jeton d'Accès* qu'ils fournissent est mis en correspondance avec les *Rôles* auxquels l'*Autorisation* de *Parcourir* l'*Objet SecurityGroup* a été accordée.

5.4.4 Intergiciel Orienté Message

5.4.4.1 Généralités

L'*Intergiciel Orienté Message* utilisé dans le présent document est une infrastructure prenant en charge l'envoi et la réception de *NetworkMessages* entre les applications distribuées. OPC UA ne définit pas d'*Intergiciel Orienté Message*, mais utilise plutôt des protocoles qui permettent la connexion, l'envoi et la réception de données. Les mappings avec les protocoles de transport pour *PubSub* sont décrits en 7.3.

Afin de couvrir un grand nombre de cas d'utilisation, le présent document décrit deux types d'*Intergiciels Orientés Message* généraux. Les deux types d'intergiciels, sans courtier et reposant sur un courtier, sont décrits en 5.4.4.2 et 5.4.4.3.

5.4.4.2 Intergiciel sans courtier

5.4.4.2.1 Généralités

Cette option permet à *PubSub* OPC UA de se reposer sur l'infrastructure réseau pour distribuer des *NetworkMessages* à un ou plusieurs récepteurs. Les dispositifs réseau tels que les routeurs de réseau, les commutateurs ou les ponts sont généralement utilisés dans ce but.

La Figure 13 représente un exemple de réseau commuté et l'utilisation du protocole UDP avec des messages en monodiffusion et en multidiffusion.

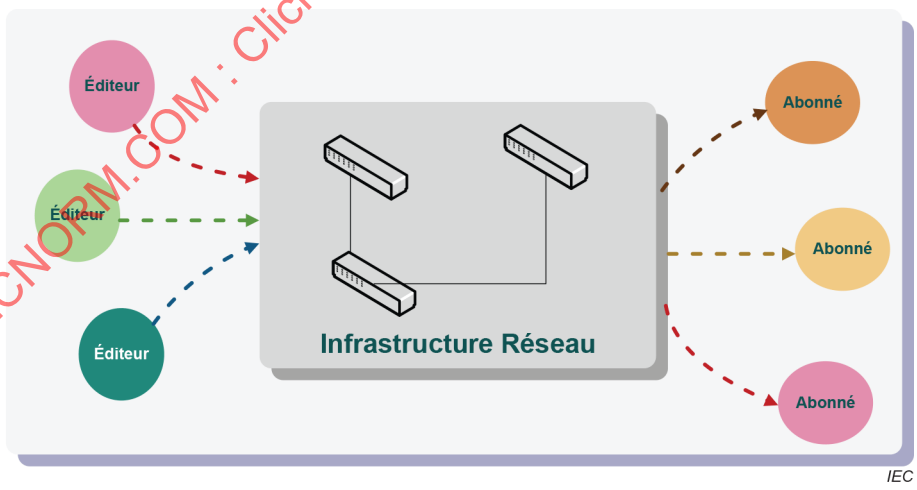


Figure 13 – Utilisation d'une infrastructure réseau avec PubSub

Les avantages de ce modèle sont les suivants:

- il exige uniquement des équipements de réseau normalisés et aucun composant logiciel supplémentaire, comme un *Courtier*;
- il est admis par hypothèse que la distribution des messages est directe, sans intermédiaire logiciel, permettant ainsi de réduire la latence et la surcharge;

- le protocole UDP prend en charge plusieurs abonnés au moyen de l'adressage en multidiffusion.

5.4.4.2.2 Modèle sans courtier avec le protocole UDP OPC UA

La Figure 14 représente les applications, les entités et les messages impliqués dans la communication entre homologues utilisant le protocole UDP, qui n'exige pas de *Courtier*.

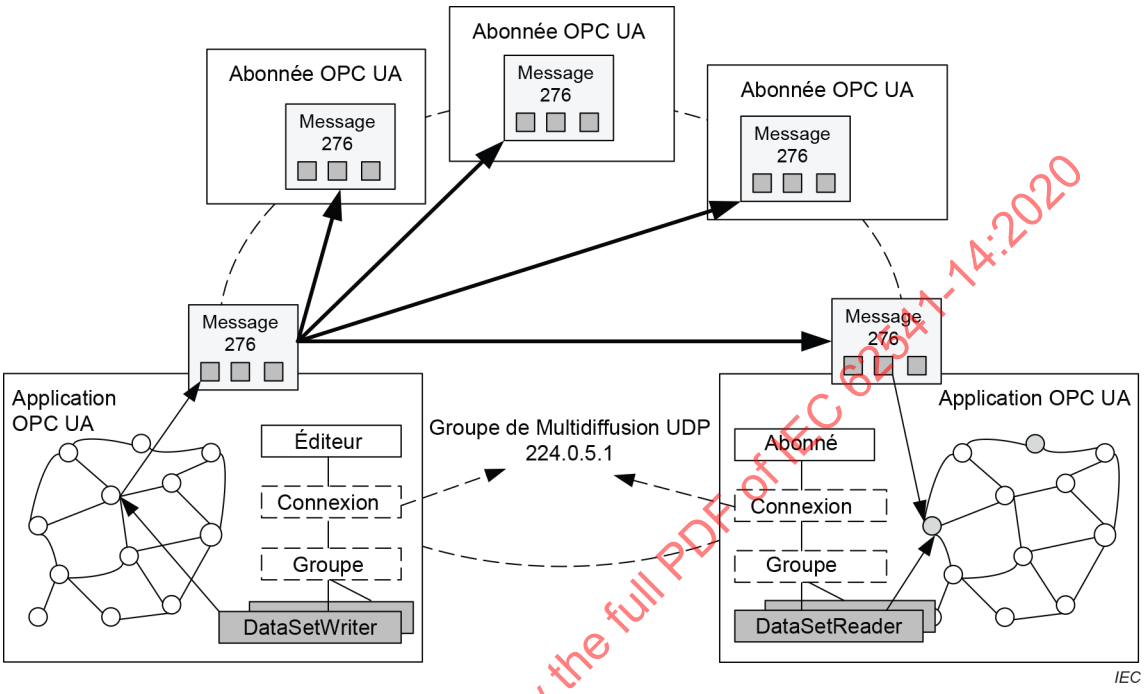


Figure 14 – Vue d'ensemble de la Multidiffusion UDP

L'*Objet PublishSubscribe* contient un *Objet* de connexion pour chaque adresse, comme une adresse IP de multidiffusion. La connexion peut avoir un ou plusieurs groupes comportant des *DataSetWriters*. Un groupe peut publier des *DataSets* à l'intervalle de publication défini.

Dans chaque intervalle de publication, un *DataSet* est collecté pour un *PublishedDataSet* qui peut être une liste d'éléments de données échantillonnés dans l'*Espace d'Adressage* OPC UA de l'*Éditeur*. Pour chaque *DataSet*, un *DataSetMessage* est créé. Les *DataSetMessages* sont envoyés dans un *NetworkMessage* à l'adresse IP de multidiffusion.

Les *Applications* OPC UA telles que les applications d'IHM utilisent les valeurs du *DataSetMessage* qui les intéressent.

Une *Application* OPC UA qui met en correspondance les champs de données des *DataSetMessages* UADP avec les *Variables* internes peut être configurée par le biais de l'*Objet DataSetReader* et du distributeur dans l'*Abonné*. La configuration d'un *DataSetReader* définit la méthode de décodage du DataSetMessage en *DataSet*. Le *SubscribedDataSet* définit quel champ du *DataSet* est associé par mapping avec quelle *Variable* de l'*Application* OPC UA.

Le protocole UDP OPC UA n'offre aucune garantie de ponctualité, de distribution, de mise en séquence ou de protection des doublons. Les numéros de séquence contenus dans les *DataSetMessages* apportent une solution pour la mise en séquence et la détection des doublons. Au moyen de l'option permettant d'envoyer le *DataSet* complet dans chaque *DataSetMessage* ou de l'option permettant de répéter des *NetworkMessages*, la fiabilité est améliorée.

Les autres mapping avec les protocoles de transport utilisés avec le modèle sans courtier peuvent offrir une garantie de ponctualité, de distribution, de mise en séquence ou de protection des doublons.

5.4.4.3 Intergiciel reposant sur un courtier

5.4.4.3.1 Généralités

Cette option prend pour hypothèse un *Courtier* de messagerie au centre, comme représenté à la Figure 15. Aucune application ne communique directement avec les autres applications. L'ensemble des communications est transmis par l'intermédiaire du *Courtier*. Le *Courtier* achemine les *NetworkMessages* vers les applications correspondantes en se basant sur les critères commerciaux ("nom de la file d'attente", "clé de routage", "sujet", etc.) plutôt que sur la topologie physique (adresses IP, noms d'hôte).

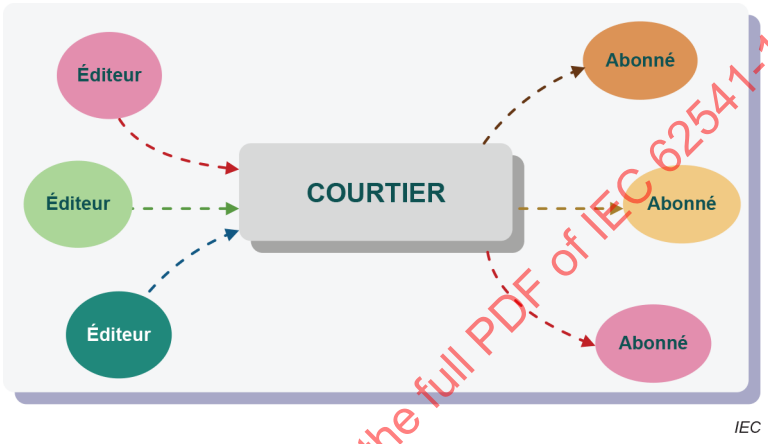


Figure 15 – Utilisation d'un courtier avec PubSub

Les avantages de ce modèle (qui dépendent en partie du *Courtier* utilisé et de sa configuration) sont les suivants:

- l'*Éditeur* et l'*Abonné* n'ont pas à être directement adressables. Ils peuvent se situer n'importe où, à condition qu'ils aient accès au *Courtier*;
- le déploiement peut s'effectuer sur une très grande liste d'*Abonnés*, sur plusieurs réseaux ou même sur des *Courtiers* chaînés ou évolutifs;
- les durées de vie de l'*Éditeur* et de l'*Abonné* n'ont pas à se chevaucher. L'application de l'*Éditeur* peut pousser les *NetworkMessage* vers le *Courtier* et s'arrêter. Les *NetworkMessages* seront disponibles ultérieurement pour l'application de l'*Abonné*;
- l'*Éditeur* et l'*Abonné* peuvent utiliser des protocoles de messagerie différents pour communiquer avec le *Courtier*.

De plus, le modèle du *Courtier* résiste dans une certaine mesure aux défaillances d'application. Ainsi, si l'application est boguée et sujette aux défaillances, les *NetworkMessages* qui figurent déjà dans le *Courtier* sont conservés même si l'application échoue.

5.4.4.3.2 Modèle reposant sur un courtier

La Figure 16 représente les applications, les entités et les messages impliqués dans les scénarios de communication types utilisant un *Courtier*. Ce modèle exige l'utilisation de protocoles de messagerie qu'un *Courtier* comprend, comme le protocole AMQP défini dans l'ISO/IEC 19464:2014 ou le protocole MQTT défini dans l'ISO/IEC 20922:2016. Dans ce modèle, l'*Intergiciel Orienté Message* est un *Courtier* qui relaie les *NetworkMessages* des *Éditeurs* vers les *Abonnés*. Le *Courtier* peut également être en mesure de placer les messages en file d'attente et d'envoyer le même message à plusieurs *Abonnés*.

Noter que la fonctionnalité du *Courtier* ne relève pas du domaine d'application du présent document. En ce qui concerne les protocoles de messagerie, le *Courtier* est un serveur de messagerie (l'*Éditeur* OPC UA et l'*Abonné* OPC UA sont des clients de messagerie). Les protocoles de messagerie définissent la méthode de connexion à un serveur de messagerie et les champs d'un message qui ont un impact sur la fonctionnalité du *Courtier*.

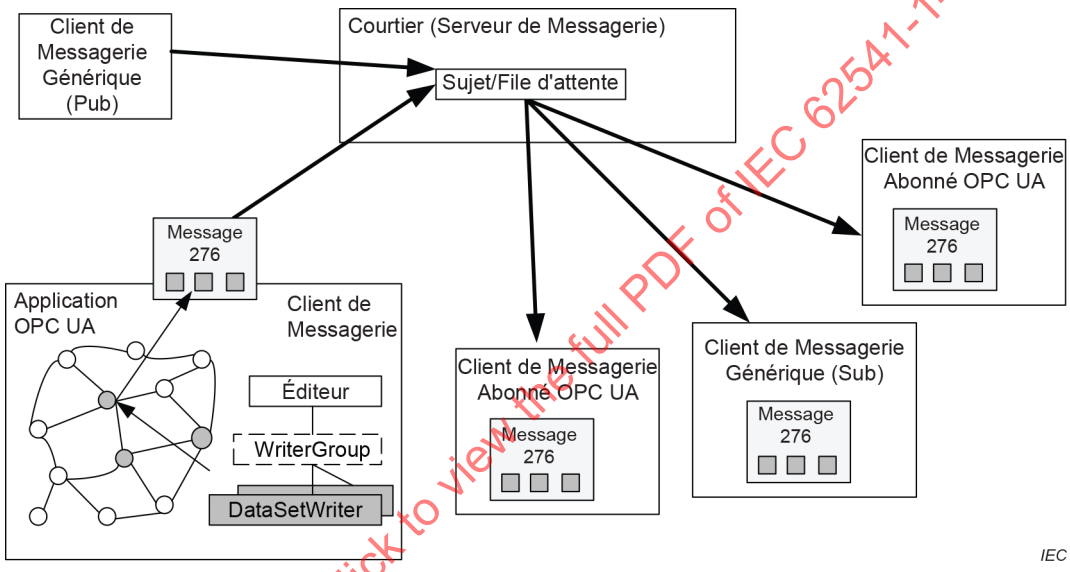


Figure 16 – Vue d'ensemble du Courtier

Un *Éditeur* OPC UA qui publie des données peut être configuré par l'intermédiaire du modèle de configuration *PubSub*. Il comprend un *Objet* de connexion par *Courtier*. Le *Courtier* est configuré par le biais d'une URL dans la connexion. La connexion peut avoir un ou plusieurs groupes qui identifient des files d'attente ou des sujets spécifiques. Chaque groupe peut contenir un ou plusieurs *DataSetWriters* qui formatent un *DataSet* comme exigé pour le protocole de messagerie. Un *DataSet* peut être collecté à partir d'une liste de champs d'*Événement* et/ou de *Variables* choisies. Cette configuration est appelée *PublishedDataSet*.

Chaque *DataSet* est envoyé sous la forme d'un *DataSetMessage* distinct sérialisé dans un format qui dépend du *DataSetWriter*. L'un des formats de *DataSetMessage* est le mapping avec les messages JSON qui représente le *DataSet* dans un format que les *Abonnés* peuvent comprendre sans connaître OPC UA. Le mapping avec les messages UADP est un autre format de *DataSetMessage*.

La confidentialité des messages et l'intégrité avec le modèle de communication reposant sur un *Courtier* peuvent être assurées à deux niveaux:

- sécurité de transport entre les *Éditeurs* ou *Abonnés* et le *Courtier*, ou
- sécurité de messages de bout en bout entre l'*Éditeur* et l'*Abonné*.

La sécurité de niveau *Courtier* exige que l'ensemble des *Éditeurs* et des *Abonnés* possède des authentifiants leur accordant un accès à la file d'attente ou au sujet nécessaire. De plus,

l'ensemble des communications avec le *Courtier* utilise la sécurité de niveau transport pour garantir la confidentialité. Les paramètres de sécurité sont spécifiés dans la connexion et le groupe.

La sécurité de messages fournie par l'*Éditeur* n'est définie que pour le mapping avec les messages UADP.

6 Paramètres de communication PubSub

6.1 Vue d'ensemble

PubSub définit différents paramètres de configuration pour les différents composants *PubSub*. Ces paramètres définissent le comportement de l'*Éditeur* et de l'*Abonné*. Ils sont regroupés par composant et sont divisés en différentes catégories: "communs", "mapping avec les messages" et "mapping avec les protocoles de transport".

Les paramètres communs sont définis en 6.2. Les paramètres des différents mapping avec les messages sont définis en 6.3. Les paramètres des différents mapping avec les protocoles de transport sont définis en 6.4. Les types communs qui sont uniquement utilisés avec *PubSub* sont définis dans l'Annexe A.

L'application des paramètres de communication pour les mapping concrèts avec les messages et les protocoles de transport est définie à l'Article 7.

La configuration de ces paramètres peut être réalisée en utilisant le *Modèle d'Information* OPC UA pour la configuration *PubSub* défini à l'Article 9 ou au moyen de mécanismes propres au fournisseur. Les regroupements de paramètres contenus dans l'Article 6 définissent les paramètres ainsi que les *Structures* utilisées pour représenter les paramètres des regroupements. Ces *Structures* sont utilisées dans le modèle de configuration *PubSub* décrit à l'Article 9. Elles peuvent également être utilisées pour la configuration hors ligne ou pour les mécanismes de configuration propres au fournisseur.

La Figure 17 représente les différents composants et leur relation. Le *WriterGroup*, le *DataSetWriter* et le *PublishedDataSet* définissent l'acquisition de données pour les *DataSets*, la génération de messages et l'envoi côté *Éditeur*. Il est nécessaire de connaître ces paramètres côté *Abonné* pour configurer les *DataSetReaders* et pour filtrer et traiter les *DataSetMessages*.

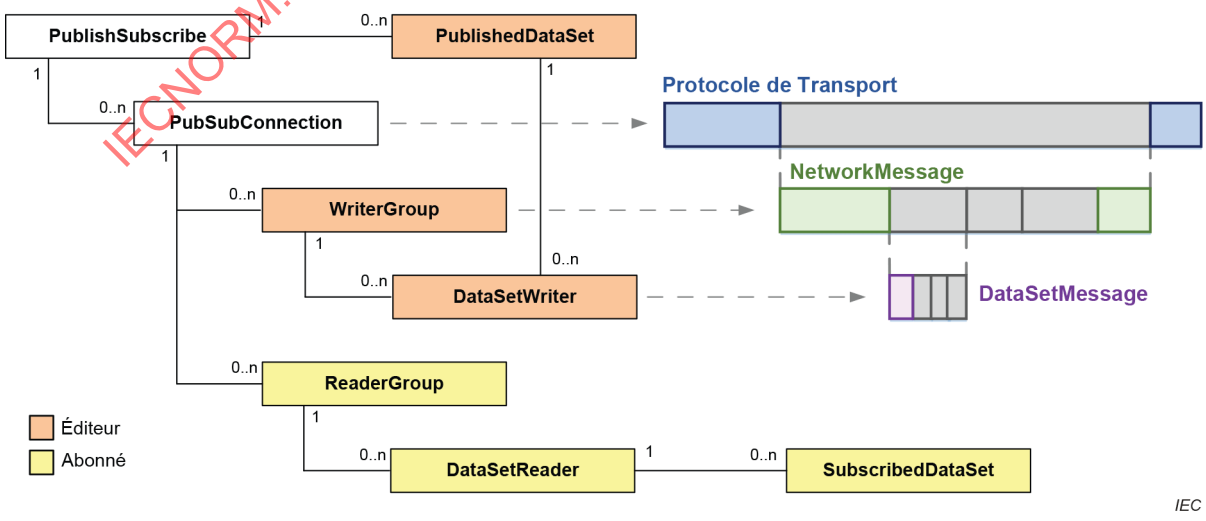


Figure 17 – Vue d'ensemble des composants PubSub

La Figure 17 représente les composants suivants:

- le *PublishedDataSet* contient les *DataSetMetaData* qui décrivent le contenu des *DataSets* générés par le *PublishedDataSet*, ainsi que les paramètres d'acquisition de données correspondants;
- les paramètres du *DataSetWriter* sont nécessaires à la création des *DataSetMessages*. Chaque *DataSetWriter* est lié à un *PublishedDataSet* unique. Un *PublishedDataSet* peut avoir plusieurs *DataSetWriters*;
- les paramètres du *WriterGroup* sont nécessaires pour créer un *NetworkMessage*. Chaque groupe de rédacteurs peut avoir un ou plusieurs *DataSetWriters*. Certains de ces paramètres sont utilisés pour créer les *DataSetMessages*. Ils sont regroupés ici, car ils sont identiques pour tous les *DataSetMessages* contenus dans un *NetworkMessage* unique;
- les paramètres de *PubSubConnection* représentent les paramètres nécessaires pour le protocole de transport. Une connexion peut avoir un certain nombre de groupes de rédacteurs et de groupes de lecteurs;
- le *ReaderGroup* est utilisé pour regrouper une liste de *DataSetReaders* et contient quelques paramètres partagés entre ces derniers. Il n'est pas symétrique à un *WriterGroup* ni rattaché à un *NetworkMessage* particulier. Les paramètres de filtre associés au *NetworkMessage* figurent dans les *DataSetReaders*;
- les paramètres du *DataSetReader* représentent les paramètres nécessaires au filtrage des *NetworkMessages* et *DataSetMessages* reçus, ainsi qu'au décodage des *DataSetMessages* pertinents;
- les paramètres du *SubscribedDataSet* définissent le traitement du *DataSet* décodé dans l'*Abonné* pour un *DataSetReader*;
- *PublishSubscribe* est l'objet de gestion globale des regroupements *PubSub*. Il contient une liste de *PublishedDataSets* ainsi qu'une liste de *PubSubConnections*.

Les différents regroupements de paramètres spécifiques au mapping PubSub sont représentés à la Figure 18.

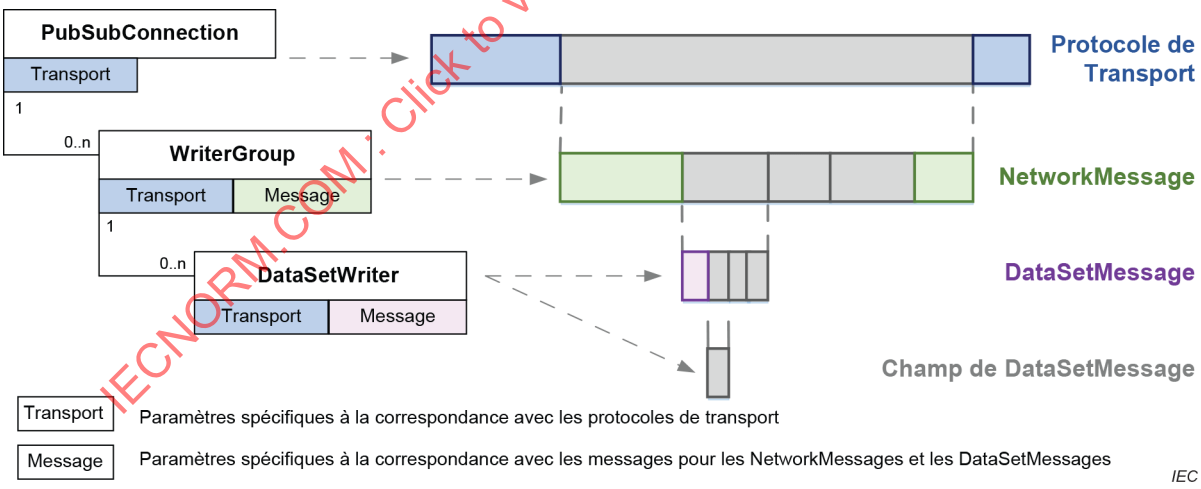


Figure 18 – Vue d'ensemble des paramètres propres au mapping PubSub

Les paramètres spécifiques au mapping avec les protocoles de transport peuvent être définis pour la *PubSubConnection*, le *WriterGroup* ou le *DataSetWriter*.

Les paramètres spécifiques au mapping avec les messages sont définis pour les *NetworkMessages* dans le*WriterGroup* et pour les *DataSetMessages* dans le *DataSetWriter*.

6.2 Paramètres de configuration communs

6.2.1 Diagramme d'états PubSubState

Le *PubSubState* est utilisé pour présenter et contrôler le fonctionnement d'un composant *PubSub*. Il s'agit d'une énumération des états possibles. Les valeurs d'énumération sont décrites dans le Tableau 1.

Tableau 1 – Valeurs de PubSubState

Valeur	Description
Disabled_0	Le composant *PubSub* est configuré, mais actuellement désactivé.
Paused_1	Le composant *PubSub* est activé, mais actuellement mis en pause par un composant parent. Le composant parent est *Disabled_0* ou *Paused_1*.
Operational_2	Le composant *PubSub* est opérationnel.
Error_3	Le composant *PubSub* est à l'état d'erreur.

La Figure 19 décrit les composants *PubSub* qui ont un état *PubSub* ainsi que leur relation parent-enfant. Les changements d'état des enfants se fondent sur les changements d'état du parent. Le composant *PublishSubscribe* se trouve à la racine de la hiérarchie.

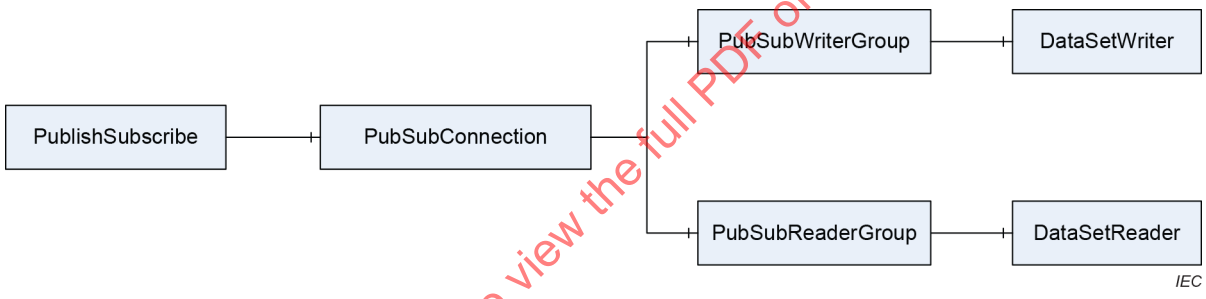


Figure 19 – Dépendances d'état des composants PubSub

La Figure 20 décrit le diagramme d'états formel et les transitions possibles.

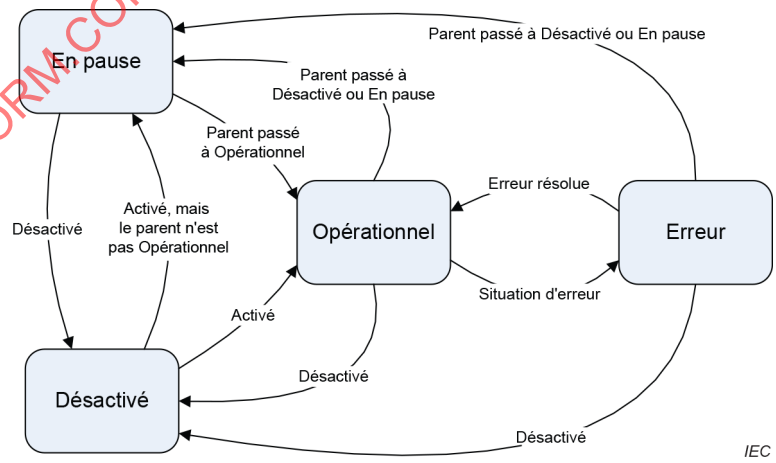


Figure 20 – Diagramme d'états PubSubState

Le Tableau 2 définit de manière formelle les transitions du diagramme d'états.

Tableau 2 – Diagramme d'états PubSubState

État source	État cible	Description du déclencheur
Disabled_0	Paused_1	Le composant a été activé avec succès, mais le composant parent est à l'état Disabled_0 ou Paused_1.
Disabled_0	Operational_2	Le composant a été activé avec succès.
Paused_1	Disabled_0	Le composant a été désactivé avec succès.
Paused_1	Operational_2	L'état du composant parent est passé à Operational_2.
Operational_2	Disabled_0	Le composant a été désactivé avec succès.
Operational_2	Paused_1	L'état du composant parent est passé à Disabled_0 ou Paused_1.
Operational_2	Error_3	Le composant *PubSub* connexe est dans une situation d'erreur en attente.
Error_3	Disabled_0	Le composant a été désactivé avec succès.
Error_3	Paused_1	L'état du composant parent est passé à Disabled_0 ou Paused_1.
Error_3	Operational_2	La situation d'erreur a été résolue pour le composant *PubSub* connexe.

6.2.2 Paramètres de PublishedDataSet

6.2.2.1 DataSetMetaData

6.2.2.1.1 Généralités

Les *DataSetMetaData* décrivent le contenu et la sémantique d'un *DataSet*. L'ordre des champs des *DataSetMetaData* doit correspondre à l'ordre des champs du *DataSet* lorsqu'ils sont inclus dans les *DataSetMessages* publiés. Le *DataSetMetaDataType* est défini en 6.2.2.1.2.

6.2.2.1.2 DataSetMetaDataType

Ce *DataType Structure* est un sous-type de *DataTypeSchemaHeader* et est utilisé pour fournir les métadonnées d'un *DataSet*. Le *DataSetMetaDataType* est défini de manière formelle dans le Tableau 3.

Le *DataTypeSchemaHeader* fournit les définitions de *DataType* OPC UA utilisées dans les *DataSetMetaData*. Le *DataTypeSchemaHeader* est défini en A.1.1.

Tableau 3 – Structure de DataSetMetaDataType

Nom	Type	Description
DataSetMetaDataType	Structure	
name	Chaîne	Nom du *DataSet*.
description	LocalizedText	Description du *DataSet*. La valeur par défaut est un *LocalizedText* nul.
fields	FieldMetaData	Métadonnées des champs du *DataSet*. Le DataType FieldMetaData est défini en 6.2.2.1.3.
dataSetClassId	Guid	Ce champ contient l'identificateur globalement unique de la classe de *DataSet* si le *DataSet* repose sur une *DataSetClass*. Dans ce cas, ce champ doit correspondre au *DataSetClassId* de la configuration concrète de *DataSet*. Si les *DataSets* ne sont pas créés à partir d'une classe, ce champ est nul.
configurationVersion	Configuration VersionDataType	Version de la configuration actuelle du *DataSet*.

Sa représentation dans l'AddressSpace est définie dans le Tableau 4.

Tableau 4 – Définition de DataSetMetaDataType

Attributs	Valeur
BrowseName	DataSetMetaDataType
IsAbstract	False
Sous-type de DataTypeSchemaHeader défini en A.1.1.	

6.2.2.1.3 FieldMetaData

Ce *DataType Structure* est utilisé pour fournir les métadonnées d'un champ dans un *DataSet*. Le *FieldMetaData* est défini de manière formelle dans le Tableau 5.

Tableau 5 – Structure de FieldMetaData

Nom	Type	Description
FieldMetaData	Structure	
name	Chaîne	Nom du champ. Le nom doit être unique dans le *DataSet*.
description	LocalizedText	Description du champ. La valeur par défaut doit être un *LocalizedText* nul.
fieldFlags	DataSetFieldFlags	Fanions du champ.
builtInType	Octet	Type de données intégré du champ. Les valeurs de types intégrés possibles sont définies dans l'IEC 62541-6. Tous les types de données sont transférés dans les *DataSetMessages* sous la forme de l'un des types de données intégrés. Dans la plupart des cas, l'identificateur du *NodeId* de *DataType* correspond au type intégré. Il est essentiel de prendre en compte également les cas particuliers suivants: (1) Les types abstraits possèdent toujours le type intégré de *Variante*, car ils peuvent générer différents types concrets dans un *DataSetMessage*. Le champ *dataType* peut fournir des restrictions supplémentaires, si le type abstrait est *Nombre*, par exemple. Les types abstraits ne doivent pas être utilisés si le champ est représenté sous la forme de *RawData* définies par le *DataSetFieldContentMask*, défini en 6.2.3.1. (2) Les *DataTypes Énumération* sont codés en *Int32*. Les chaînes d'*Énumération* sont définies par le biais d'un *DataType* référencé par le champ *dataType*. (3) Les *DataTypes Structure* et *Union* sont codés en *ExtensionObject*. Les règles de codage sont définies par le biais d'un *DataType* référencé par le champ *dataType*. (4) Les *DataTypes* dérivés de types intégrés possèdent le *BuiltInType* du DataType de base correspondant. Le sous-type concret est défini par le champ *dataType*. (5) Les *DataTypes OptionSet* sont codés soit dans l'un des *DataTypes UInteger* concrets, soit en instance d'un *OptionSetType* dans un *ExtensionObject*.
dataType	NodeId	Le *NodeId* du *DataType* de ce champ. Si le *DataType* est *Énumération* ou *OptionSet*, la sémantique du *DataType Énumération* est fournie par le champ *enumDataTypes* des *DataSetMetaData*. Si le *DataType* est *Structure* ou *Union*, la description du codage et du décodage du *DataType Structure* est fournie par le champ *structureDataTypes* des *DataSetMetaData*.

Nom	Type	Description
valueRank	Int32	Indique si le *dataType* est une matrice et indique le nombre de dimensions de cette matrice. Il peut prendre les valeurs suivantes: n > 1: Le *dataType* est une matrice ayant le nombre spécifié de dimensions. OneDimension (1): Le *dataType* est une matrice unidimensionnelle. OneOrMoreDimensions (0): Le *dataType* est une matrice à une ou plusieurs dimensions. Scalaire (−1): Le *dataType* n'est pas une matrice. Indifférente (−2): Le *dataType* peut être scalaire ou une matrice ayant n'importe quel nombre de dimensions. ScalarOrOneDimension (−3): Le *dataType* peut être scalaire ou une matrice unidimensionnelle. NOTE Tous les *DataTypes* sont considérés comme des valeurs scalaires, même s'ils ont une sémantique de type matrice comme *ByteString* et *String*.
arrayDimensions	UInt32	Ce champ spécifie la longueur maximale prise en charge de chaque dimension. Si la valeur maximale n'est pas connue, la valeur doit être 0. Le nombre d'éléments doit être égal à la valeur du *champ valueRank*. Ce champ doit être nul si *valueRank* ≤ 0. Le nombre maximal d'éléments d'une matrice transférés sur le réseau est 2 147 483 647 (max Int32). Il s'agit du nombre total d'éléments dans l'ensemble des dimensions fondé sur les règles de codage UA Binaire pour les matrices.
maxStringLength	UInt32	Si le champ *dataType* est une *Chaîne* ou *ByteString*, il spécifie alors la longueur maximale prise en charge. Si la valeur maximale n'est pas connue, la valeur doit être 0. Si le champ dataType n'est pas défini sur *Chaîne* ou *ByteString*, la valeur doit être 0. Si le *valueRank* est supérieur à 0, ce champ s'applique à chaque élément de la matrice.
dataSetFieldId	Guid	Identificateur unique du champ dans le *DataSet*. L'identificateur est généré lorsque le champ est ajouté à la liste. Un changement de position du champ dans la liste ne doit pas modifier l'identificateur.
properties	KeyValuePair	Liste des valeurs de *Propriétés* qui définissent la sémantique supplémentaire du champ. Si au moins une valeur de *Propriété* est modifiée, l'attribut *MajorVersion* du champ *ConfigurationVersion* doit être mis à jour. Si la *Propriété* est *EngineeringUnits*, l'unité de la *Valeur* de *Champ* doit correspondre à l'unité des *FieldMetaData*. Le DataType *KeyValuePair* est défini dans l'IEC 62541-5. Pour ce champ, la clé contenue dans la structure *KeyValuePair* est le *BrowseName* de la *Propriété* et la valeur contenue dans la structure *KeyValuePair* est la *Valeur* de la *Propriété*.

6.2.2.1.4 DataSetFieldFlags

Ce *DataType* définit les fanions des champs de DataSet.

Les *DataSetFieldFlags* sont définis de manière formelle dans le Tableau 6.

Tableau 6 – Valeurs de DataSetFieldFlags

Valeur	N° de bit	Description
PromotedField	0	Ce fanion indique si le champ est promu dans les *NetworkMessages* ou dans l'en-tête de protocole de transport. La définition de ce fanion augmente la taille des *NetworkMessages*, car les informations issues du corps du *DataSetMessage* sont également promues dans l'en-tête. Selon la sécurité utilisée, l'en-tête qui comprend le champ peut être déchiffré. Les champs promus sont toujours inclus dans l'en-tête, même si la charge utile du *DataSetMessage* est une image delta et que le champ *DataSet* n'est pas inclus dans l'image delta. Dans ce cas, la dernière valeur envoyée est transférée dans l'en-tête. L'ordre des champs des *DataSetMetaData* promus dans l'en-tête doit correspondre à l'ordre des champs dans l'en-tête, sauf si l'en-tête contient des noms de champs.

La représentation de *DataSetFieldFlags* dans l'*AddressSpace* est définie dans le Tableau 7.

Tableau 7 – Définition de DataSetFieldFlags

Attributs	Valeur		
BrowseName	DataSetFieldFlags		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**DataType**
Sous-type de UInt16 défini dans l'IEC 62541-5.			
HasProperty	Variable	OptionSetValues	LocalizedText [ ]

6.2.2.1.5 ConfigurationVersionDataType

Ce *DataType Structure* est utilisé pour indiquer les modifications de configuration appliquées dans les informations publiées pour un *DataSet*. Le *ConfigurationVersionDataType* est défini de manière formelle dans le Tableau 8.

Tableau 8 – Structure de ConfigurationVersionDataType

Nom	Type	Description
ConfigurationVersionDataType	Structure	
majorVersion	VersionTime	L'attribut *MajorVersion* représente l'heure de la dernière modification majeure apportée au contenu du *DataSet*. Le *DataType VersionTime* est défini dans l'IEC 62541-4. Pour assurer l'interopérabilité, l'*Abonné* doit utiliser les *DataSetMetaData* pour le décodage avec une *MajorVersion* qui correspond à la valeur de *MajorVersion* des *DataSetMessages* envoyés par l'*Éditeur*. La suppression de champs du contenu du *DataSet*, la réorganisation des champs, l'ajout de champs entre d'autres champs ou une modification du DataType dans les champs doit entraîner la mise à jour de l'attribut *MajorVersion*. Si au moins une valeur de *Propriété* d'un champ *DataSetMetaData* est modifiée, l'attribut *MajorVersion* doit être mis à jour. Dans certains cas, les configurations plus anciennes d'un *Éditeur* peuvent être chargées et modifiées au moyen d'outils de configuration propres au produit. Ainsi, l'attribut *MajorVersion* doit être mis à jour si l'outil de configuration n'est pas capable de vérifier si la modification ne fait qu'étendre la configuration et ne modifie pas le contenu existant. Les critères supplémentaires de modification de *MajorVersion* ou *MinorVersion* sont définis dans le présent document.
minorVersion	VersionTime	L'attribut *MinorVersion* représente l'heure de la dernière modification. Il ne doit être mis à jour que si des champs sont ajoutés à la fin du contenu du *DataSet*. Si la version de *MajorVersion* est mise à jour, l'attribut *MinorVersion* est mis à jour sur la même valeur que *MajorVersion*.

6.2.2.2 DataSetClassId

Les *DataSetMetaData* peuvent être spécifiques à un seul *Éditeur* et à un unique ensemble d'informations ou universelles, par exemple définies par un organisme de normalisation ou par un opérateur de centrale en tant que *DataSetClass*. Les *DataSets* conformes à cette *DataSetClass* sont identifiés par un *DataSetClassId*.

Il s'agit de l'identificateur globalement unique (*Guid*) d'une *DataSetClass*. Il est inclus dans les *DataSetMetaData*. Le *NetworkMessageContentMask* contrôle la disponibilité du *DataSetClassId* dans le *NetworkMessage*.

6.2.2.3 ExtensionFields

Le paramètre *ExtensionFields* permet de configurer les champs dont les valeurs sont à inclure dans le *DataSet* lorsque l'*AddressSpace* existant de l'*Éditeur* ne fournit pas les informations nécessaires. Les *ExtensionFields* sont représentés sous la forme d'une matrice de *Structures KeyValuePair*.

6.2.2.4 PublishedDataSetDataType

Ce *DataType Structure* représente les paramètres du *PublishedDataSet*. Le *PublishedDataSetDataType* est défini de manière formelle dans le Tableau 9.

Tableau 9 – Structure de PublishedDataSetDataType

Nom	Type	Description
PublishedDataSetDataType	Structure	
name	Chaîne	Nom du *PublishedDataSet*. Le nom du *PublishedDataSet* doit être unique dans l'*Éditeur*.
dataSetFolder	Chaîne	Chemin facultatif du dossier de *DataSet* utilisé pour regrouper les *PublishedDataSets* dans lequel chaque entrée de la matrice *Chaîne* représente un niveau dans une hiérarchie de dossiers de *DataSet*. Si le regroupement n'est pas nécessaire, le paramètre est une matrice *Chaîne* nulle.
dataSetMetaData	DataSetMetaData	Défini en 6.2.2.1.
extensionFields	KeyValuePair	Défini en 6.2.2.3.
dataSetSource	PublishedDataSetSourceDataType	Défini en 6.2.2.5.

6.2.2.5 PublishedDataSetSourceDataType

La *Structure PublishedDataSetSourceDataType* est un type de base abstrait sans champs qui définit la source du *PublishedDataSet*. Sa représentation dans l'*AddressSpace* est définie dans le Tableau 10.

Tableau 10 – Définition de PublishedDataSetSourceDataType

Attributs	Valeur			
BrowseName	PublishedDataSetSourceDataType			
IsAbstract	True			
Références	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Sous-type de Structure défini dans l'IEC 62541-5:				
HasSubtype	DataType	PublishedDataItemsDataType	FALSE	Défini en 6.2.2.6.2.
HasSubtype	DataType	PublishedEventsDataType	FALSE	Défini en 6.2.2.7.4.

6.2.2.6 Éléments de données publiés

6.2.2.6.1 PublishedData

Le paramètre *PublishedData* définit le contenu d'un *DataSet* créé à partir des *Valeurs de Variables* et donc le contenu du *DataSetMessage* envoyé par un *DataSetWriter*. Les sources des champs *DataSet* sont définies par une matrice de *PublishedVariableDataType*.

L'indice figurant dans la matrice joue un rôle important pour les *Éditeurs* et pour les outils de configuration. Il permet de référencer la *Valeur* des *DataSetMessages* reçus par les *Abonnés*. L'indice peut évoluer en fonction des modifications de configuration effectuées. Les modifications sont indiquées par l'attribut *ConfigurationVersion* du *DataSet* et les applications qui utilisent l'indice doivent toujours vérifier la *ConfigurationVersion* avant d'utiliser l'indice.

Si une entrée des *PublishedData* fait référence à l'un des ExtensionFields, la *substituteValue* doit contenir le QualifiedName de l'entrée d'ExtensionFields. Tous les autres champs de cet élément de matrice *PublishedVariableDataType* doivent être nuls.

Le *DataType PublishedVariableDataType* représente les informations de configuration d'une Variable. Le *PublishedVariableDataType* est défini de manière formelle dans le Tableau 11.

Tableau 11 – Structure de PublishedVariableDataType

Nom	Type	Description
PublishedVariableDataType	Structure	
publishedVariable	NodeId	*NodeId* de la *Variable* publiée. Certains protocoles de transport exigent de connaître, du côté du récepteur du message, le DataType, le ValueRank et les ArrayDimensions pour pouvoir décoder le contenu du message. Ces informations sont fournies par les *DataSetMetaData* fournies pour le *DataSet*.
attributeId	IntegerId	Identificateur de l'*Attribut* à publier, par exemple l'*Attribut Valeur*. Il doit s'agir d'un identificateur d'*Attribut* valide. Les *Attributs* sont définis dans l'IEC 62541-3. Le *DataType IntegerId* est défini dans l'IEC 62541-4. Les *IntegerIds* des *Attributs* sont définis dans l'IEC 62541-6.
samplingIntervalHint	Durée	Fréquence d'acquisition des nouvelles valeurs recommandée pour l'évaluation des modifications ou de la bande morte. Il convient qu'un *Éditeur* utilise cette valeur comme une indication pour régler la fréquence d'échantillonnage interne. La valeur 0 indique qu'il convient que le *Serveur* utilise la fréquence pratique la plus rapide. La valeur −1 indique que l'intervalle d'échantillonnage par défaut défini par le *PublishingInterval* du *WriterGroup* est demandé. Tout nombre négatif est interprété comme −1.
deadbandType	UInt32	Une valeur qui définit le type et le comportement de *Deadband*. Valeur — Description None_0 — Il convient de n'appliquer aucun calcul de *Deadband*. Absolute_1 — AbsoluteDeadband (Ce type est spécifié dans l'IEC 62541-4) Percent_2 — PercentDeadband (Ce type est spécifié dans l'IEC 62541-8)
deadbandValue	Double	Valeur de la bande morte pour le *DeadbandType* correspondant. La signification de cette valeur dépend du *DeadbandType*.
indexRange	NumericRange	Ce paramètre est utilisé pour identifier un élément simple d'une matrice ou une plage simple d'indices pour les matrices. Le type *NumericRange* et la logique d'*IndexRange* sont définis dans l'IEC 62541-4.
substituteValue	BaseDataType	Valeur contenue dans le *DataSet* si le *StatusCode* de la *DataValue* est "Bad". Dans ce cas, le *StatusCode* est réglé sur Uncertain_SubstituteValue. Cette Valeur doit correspondre au *DataType* de la *PublishedVariable*, car les *DataSetWriters* peuvent dépendre d'une *Valeur* valide avec le *DataType* approprié qui correspond à la *ConfigurationVersion*. Si la valeur de *SubstituteValue* est nulle, le *StatusCode* de la *DataValue* est traité. Le traitement de *SubstituteValue* est défini en 6.2.10.
metaDataProperties	QualifiedName [ ]	Ce paramètre spécifie une matrice de *Propriétés* à inclure dans les *FieldMetaData* créées pour cette *Variable*. Il doit être utilisé pour remplir l'élément *properties* du champ généré dans les *DataSetMetaData*.

6.2.2.6.2 PublishedDataItemsDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres spécifiques aux *PublishedDataItems*. Il s'agit d'un sous-type du *PublishedDataSetSourceDataType* défini en 6.2.2.5.

Le *PublishedDataItemsDataType* est défini de manière formelle dans le Tableau 12.

Tableau 12 – Structure de PublishedDataItemsDataType

Nom	Type	Description
PublishedDataItemsDataType	Structure	
publishedData	PublishedVariableDataType	Défini en 6.2.2.6.1.

6.2.2.7 Événements publiés

6.2.2.7.1 EventNotifier

Le paramètre *EventNotifier* définit le *NodeId* de l'*Objet* dans l'arbre de notification d'événements du *Serveur* OPC UA à partir duquel les *Événements* sont collectés.

6.2.2.7.2 SelectedFields

Le paramètre *SelectedFields* définit le choix des champs d'*Événement* contenus dans le *DataSet* généré pour un *Événement* et envoyés par le *DataSetWriter*. Le *DataType SimpleAttributeOperand* est défini dans l'IEC 62541-4. Le *DataType* du champ d'*Événement* choisi dans l'*EventType* définit le *DataType* du champ *DataSet*. Les champs d'*Événement* peuvent être nuls ou la valeur du champ peut être un *StatusCode*. Le codage des *DataSetMessages* fondés sur un *Événement* doit être capable de traiter ces cas. Les *ExtensionFields* définis pour l'instance du *PublishedEventsType* peuvent être inclus dans les *SelectedFields* en définissant le *NodeId* du *PublishedEventsType* sur typeId dans le *SimpleAttributeOperand* et en spécifiant le *BrowseName* du champ d'extension dans le *browsePath* du *SimpleAttributeOperand*.

L'indice figurant dans la liste des entrées des *SelectedFields* joue un rôle important pour les *Abonnés*. Il permet de référencer le champ d'*Événement* des *DataSetMessages* reçus par les *Abonnés*. L'indice peut évoluer en fonction des modifications de configuration effectuées. Les modifications sont indiquées par l'attribut *ConfigurationVersion* et les applications qui utilisent l'indice doivent toujours vérifier la *ConfigurationVersion* avant d'utiliser l'indice. Si une modification apportée aux *SelectedFields* ajoute des champs supplémentaires, l'attribut *MinorVersion* de la *ConfigurationVersion* doit être mis à jour. Si une modification apportée aux *SelectedFields* retire des champs, l'attribut *MajorVersion* de la *ConfigurationVersion* doit être mis à jour. Le *ConfigurationVersionDataType* et les règles de définition de la version sont définis en 6.2.2.1.5.

6.2.2.7.3 Filtre

Le paramètre *Filtre* définit le filtre appliqué aux *Événements*. Il permet de réduire les *DataSets* générés à partir d'*Événements* au moyen d'un filtre. Le *ContentFilter DataType* est défini en IEC 62541-4.

6.2.2.7.4 PublishedEventsDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres spécifiques aux *PublishedEvents*. Il s'agit d'un sous-type du *PublishedDataSetSourceDataType* défini en 6.2.2.5.

Le *PublishedEventsDataType* est défini de manière formelle dans le Tableau 13.

Tableau 13 – Structure de PublishedEventsDataType

Nom	Type	Description
PublishedEventsDataType	Structure	
eventNotifier	NodeId	Défini en 6.2.2.7.1.
selectedFields	SimpleAttributeOperand	Défini en 6.2.2.7.2.
filter	ContentFilter	Défini en 6.2.2.7.3.

6.2.3 Paramètres de DataSetWriter

6.2.3.1 DataSetWriterId

Le *DataSetWriterId* de *DataType UInt16* définit l'identificateur unique du *DataSetWriter* pour un *PublishedDataSet*. Il permet de choisir des *DataSetMessages* pour un *PublishedDataSet* côté *Abonné*.

Il doit être unique dans l'ensemble des *DataSetWriters* pour un *PublisherId*.

Toutes les valeurs, autres que 0, sont des *DataSetWriterIds* valides. La valeur 0 est définie comme une valeur nulle.

6.2.3.2 DataSetFieldContentMask

Un champ *DataSet* se compose d'une valeur et de métadonnées connexes. Dans la plupart des cas, la valeur est accompagnée d'un statut et d'un horodatage.

Ce *DataType* définit les fanions permettant d'inclure des informations relatives au champ *DataSet*, telles que le statut et l'horodatage, en plus de la valeur contenue dans le *DataSetMessage*.

Le *DataSetFieldContentMask* est défini de manière formelle dans le Tableau 14.

Le traitement du statut "bad" pour les différentes représentations de champ est défini à la Figure 21 et dans le Tableau 16.

Tableau 14 – Valeurs de DataSetFieldContentMask

Valeur	N° de bit	Description
Les champs *DataSet* peuvent être représentés sous la forme de RawData, de Variante ou de DataValue, comme décrit en 5.3.2. Si aucun fanion n'est défini, les champs sont représentés sous la forme d'une *Variante*. Si le fanion *RawData* est défini, les champs sont représentés sous forme de *RawData* et tous les autres bits sont ignorés. Si un bit de 0 à 4 est défini, les champs sont représentés sous la forme d'une *DataValue*.		
StatusCode	0	Le champ *StatusCode* de structure *DataValue* est inclus dans les *DataSetMessages*. Si ce fanion est défini, les champs sont représentés sous la forme d'une *DataValue*.
SourceTimestamp	1	Le champ *SourceTimestamp* de structure *DataValue* est inclus dans les *DataSetMessages*. Si ce fanion est défini, les champs sont représentés sous la forme d'une *DataValue*.
ServerTimestamp	2	Le champ *ServerTimestamp* de structure *DataValue* est inclus dans les *DataSetMessages*. Si ce fanion est défini, les champs sont représentés sous la forme d'une *DataValue*.
SourcePicoSeconds	3	Le champ *SourcePicoSeconds* de structure *DataValue* est inclus dans les *DataSetMessages*. Si ce fanion est défini, les champs sont représentés sous la forme d'une *DataValue*. Ce fanion est ignoré si le fanion *SourceTimestamp* n'est pas défini.
ServerPicoSeconds	4	Le champ *ServerPicoSeconds* de structure *DataValue* est inclus dans les *DataSetMessages*. Si ce fanion est défini, les champs sont représentés sous la forme d'une *DataValue*. Ce fanion est ignoré si le fanion *ServerTimestamp* n'est pas défini.
RawData	5	Si ce fanion est défini, les valeurs du *DataSet* sont codées en *Structure* et tous les autres fanions associés au champ doivent être ignorés. La représentation des *RawData* est traitée comme le *DataType Structure* où les champs *DataSet* sont traités comme des champs *Structure* et les champs dont le *DataType* est *Structure* sont traités comme des structures imbriquées. L'ensemble des restrictions de codage des *DataTypes Structure* s'applique également au *Codage du Champ RawData*. Les champs ne doivent pas avoir de DataType abstrait ou doivent avoir un ValueRank fixe. Les dimensions des champs doivent être définies si le *DataType* est *Chaîne* ou *ByteString* ou s'il s'agit d'une matrice. Les champs *Structure* comportant ces champs sont inclus. Le fanion doit être ignoré et les champs doivent être représentés sous la forme d'une *Variante* s'ils ne satisfont pas à ces exigences.

La représentation du *DataSetFieldContentMask* dans l'AddressSpace est définie dans le Tableau 15.

Tableau 15 – Définition de DataSetFieldContentMask

Attributs	Valeur		
BrowseName	DataSetFieldContentMask		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**DataType**
Sous-type de UInt32 défini dans l'IEC 62541-5.			
HasProperty	Variable	OptionSetValues	LocalizedText [ ]

Le *DataSetFieldContentMask* définit les différentes options qui ont un impact sur le flux d'informations entre l'*Éditeur* et l'*Abonné* en cas de Valeur de Statut "Bad" ou d'autres situations d'erreur. La Figure 21 représente les paramètres et le flux d'informations depuis le champ *DataSet* pour la création du *DataSetMessage* côté *Éditeur* jusqu'au champ DataSet décodé côté Abonné. Le *DataSetFieldContentMask* contrôle la représentation des champs DataSet d'un *DataSetMessage*.

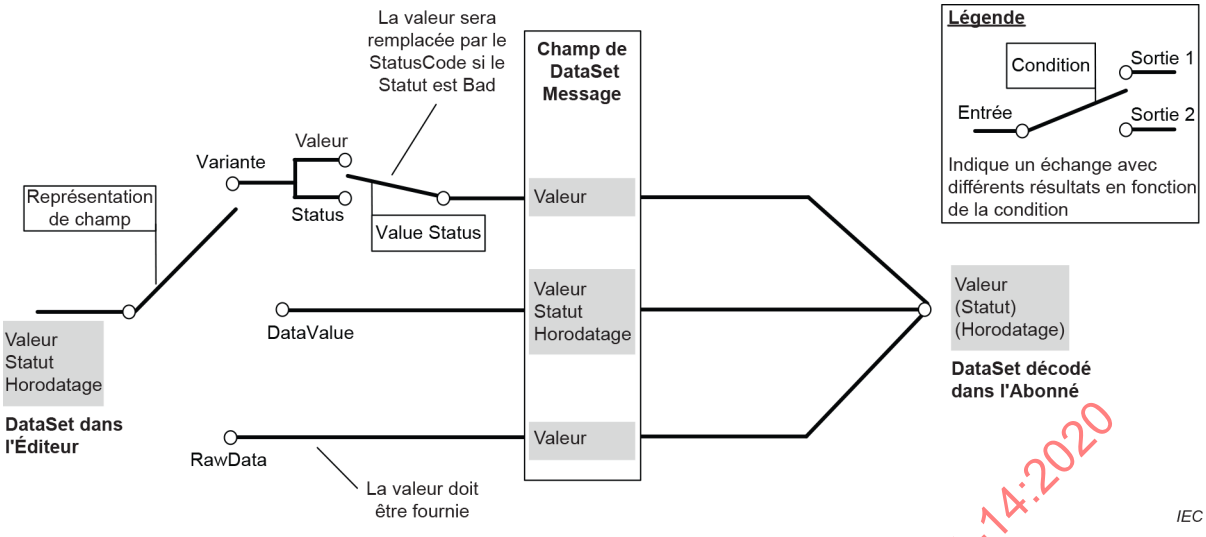


Figure 21 – Dépendance du flux d'informations PubSub à la représentation des champs

La représentation des champs *DataSet* dans un *DataSetMessage* côté *Éditeur* et le décodage des champs *DataSet* côté *Abonné* sont définis dans le Tableau 16. La représentation côté *Éditeur* dépend de la représentation des champs définie dans le *DataSetFieldContentMask*.

Tableau 16 – Options de représentation des champs d'un DataSetMessage

Éditeur de DataSet		Champ	DataSetMessage		Abonné de DataSet	
Valeur	**Statut**[d]		**Valeur**	**Statut**[d]	**Valeur**	**Statut**[d]
Valeur 1	Good_*	Variante	Valeur 1	N/A [a]	Valeur 1	N/A [a]
Valeur 1	Uncertain_*		Valeur 1		Valeur 1	
Nulle	Bad_*		Bad_* [a]		Nulle	Bad_*
Valeur 1	Good_*	DataValue	Valeur 1	Good_*	Valeur 1	Good_*
Valeur 1	Uncertain_*		Valeur 1	Uncertain_*	Valeur 1	Uncertain_*
Nulle	Bad_*		Nulle	Bad_*	Nulle	Bad_*
Valeur 1	Good_*	RawData	Valeur 1	N/A	Valeur 1	N/A
Valeur 1	Uncertain_*		Valeur 1 [b]		Valeur 1	
Nulle	Bad_*		DefaultValue [c]		DefaultValue	

[a] Le statut "bad" est transféré en lieu et place d'une valeur. Le statut "uncertain" n'est pas transféré pour un champ. Si le champ de statut est inclus dans l'en-tête du *DataSetMessage*, le statut est défini sur "uncertain" si l'un des champs a le statut "uncertain".

[b] Si le statut le plus défavorable d'un ou de plusieurs champs est "uncertain", le statut du *DataSetMessage* doit être défini sur *Uncertain*.

[c] Si le statut le plus défavorable d'un ou de plusieurs champs est "bad", le statut du *DataSetMessage* doit être défini sur *Bad*.

[d] Si aucun *StatusCode* spécifique n'est utilisé, le regroupement dans la sévérité *Good*, *Uncertain* ou *Bad* est utilisé. Dans ce cas, le *Statut* généré correspond au *Statut* d'entrée.

6.2.3.3 KeyFrameCount

Le *KeyFrameCount* de *DataType UInt32* représente le multiplicateur du *PublishingInterval* qui définit le nombre maximal de fois où le *PublishingInterval* expire avant l'envoi d'un message de type image clé, comportant des valeurs pour toutes les *Variables* publiées. Les *DataSetMessages* de type image delta contiennent uniquement les valeurs modifiées. Si aucune modification n'a été apportée, le *DataSetMessage* de type image delta ne doit pas être envoyé. Si le *KeyFrameCount* est défini sur 1, chaque message contient une image clé.

Pour les *PublishedDataSets* tels que les *PublishedDataItems* qui fournissent des mises à jour cycliques du *DataSet*, la valeur doit être supérieure ou égale à 1. Pour les *PublishedDataSets* non cycliques, tels que les *PublishedEvents*, qui fournissent des *DataSets* fondés sur un événement, la valeur doit être 0.

6.2.3.4 DataSetWriterProperties

Le paramètre *DataSetWriterProperties* est une matrice de *DataType KeyValuePair* qui spécifie les propriétés supplémentaires du *DataSetWriter* configuré. Le *DataType KeyValuePair*, défini dans l'IEC 62541-5, se compose d'un *QualifiedName* et d'une valeur de *BaseDataType*.

Le mapping du nom et de la valeur avec la fonctionnalité concrète peut être défini par les mappings avec les protocoles de transport, les éditions ultérieures de l'IEC 62541-14 ou les extensions propres au fournisseur.

6.2.3.5 Structure de DataSetWriter

6.2.3.5.1 DataSetWriterDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres *DataSetWriter*. Le *DataSetWriterDataType* est défini de manière formelle dans le Tableau 17.

Tableau 17 – Structure de DataSetWriterDataType

Nom	Type	Description
DataSetWriterDataType	Structure	
name	Chaîne	Nom du *DataSetWriter*.
enabled	Booléen	État activé du *DataSetWriter*.
dataSetWriterId	UInt16	Défini en 6.2.3.1.
dataSetFieldContentMask	DataSetFieldContentMask	Défini en 6.2.3.2.
keyFrameCount	UInt32	Défini en 6.2.3.3.
dataSetName	Chaîne	Nom du *PublishedDataSet* correspondant.
dataSetWriterProperties	KeyValuePair	Défini en 6.2.3.4.
transportSettings	DataSetWriterTransportDataType	Paramètres du *DataSetWriter* spécifiques aumapping avec le transport. Le type de base abstrait est défini en 6.2.3.5.2. Les sous-types concrets sont définis dans les paragraphes relatifs aux paramètres spécifiques au mapping avec le transport.
messageSettings	DataSetWriterMessageDataType	Paramètres du *DataSetWriter* spécifiques au mapping de *DataSetMessage*. Le type de base abstrait est défini en 6.2.3.5.3. Les sous-types concrets sont définis dans les paragraphes relatifs aux paramètres spécifiques au mapping avec les messages.

6.2.3.5.2 DataSetWriterTransportDataType

Ce *DataType Structure* est un type de base abstrait qui s'applique aux paramètres du *DataSetWriter* spécifiques au mapping avec le transport. Le *DataType* abstrait ne définit pas de champs.

La représentation de la *Structure* du *DataSetWriterTransportDataType* dans l'*AddressSpace* est définie dans le Tableau 18.

Tableau 18 – Définition de DataSetWriterTransportDataType

Attributs	Valeur			
BrowseName	DataSetWriterTransportDataType			
IsAbstract	True			
Références	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Sous-type de Structure défini dans l'IEC 62541-5.				
HasSubtype	DataType	BrokerDataSetWriterTransportDataType	FALSE	Défini en 6.4.2.3.7.

6.2.3.5.3 DataSetWriterMessageDataType

Ce *DataType Structure* est un type de base abstrait qui s'applique aux paramètres du *DataSetWriter* spécifiques au mapping avec les messages. Le *DataType* abstrait ne définit pas de champs.

La représentation de la *Structure DataSetWriterMessageDataType* dans l'*AddressSpace* est définie dans le Tableau 19.

Tableau 19 – Structure de DataSetWriterMessageDataType

Attributs	Valeur			
BrowseName	DataSetWriterMessageDataType			
IsAbstract	True			
Références	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Sous-type de Structure défini dans l'IEC 62541-5.				
HasSubtype	DataType	UadpDataSetWriterMessageDataType	FALSE	Défini en 6.3.1.2.6.
HasSubtype	DataType	JsonDataSetWriterMessageDataType	FALSE	Défini en 6.3.2.2.2.

6.2.4 Paramètres de PubSubGroup partagés

6.2.4.1 Généralités

Les paramètres sont partagés entre le *WriterGroup* et le *ReaderGroup*.

Les paramètres sont liés à la sécurité des *NetworkMessages PubSub*. Voir 5.4.3 pour obtenir une présentation de la sécurité PubSub et l'Article 8 pour obtenir la définition du Service de Clés de Sécurité PubSub.

6.2.4.2 SecurityMode

Le *SecurityMode* indique le niveau de sécurité appliqué aux *NetworkMessages* publiés par un *WriterGroup* ou reçus par un *ReaderGroup*. Le *DataType MessageSecurityMode* est défini dans l'IEC 62541-4.

6.2.4.3 SecurityGroupId

Le *SecurityGroupId* de *DataType Chaîne* représente l'identificateur d'un *SecurityGroup* dans le *Serveur de Clés de Sécurité*. Il est unique au sein d'un SKS.

Ce paramètre est nul si le *SecurityMode* est défini sur NONE_1.

Si le *SecurityMode* n'est pas défini sur NONE_1, le *SecurityGroupId* identifie le *SecurityGroup*. Le *SecurityGroup définit la SecurityPolicy* et les clés de sécurité utilisées pour

la sécurité du *NetworkMessage*. Le *PubSubGroup* définit le *SecurityMode* des *NetworkMessages* envoyés par le groupe.

6.2.4.4 SecurityKeyServices

SecurityKeyServices est une matrice de *DataType EndpointDescription* qui définit un ou plusieurs *Serveurs de Clés de Sécurité* (SKS) qui gèrent les clés de sécurité du *SecurityGroup* affecté au *PubSubGroup*. Le *DataType EndpointDescription* est défini dans l'IEC 62541-4.

Ce paramètre est nul si le *SecurityMode* est défini sur NONE_1.

Chaque élément de la matrice est un *Point d'extrémité* pour un SKS qui peut fournir les clés de sécurité du *SecurityGroupId*. Il existe plusieurs *Points d'extrémité,* car un SKS peut prendre en charge plusieurs profils de transport et/ou peut avoir plusieurs instances redondantes. Les *UserTokenPolicies* de chaque *Endpoint* spécifient les authentifiants d'utilisateur exigés. L'IEC 62541-4 décrit les *UserTokenPolicies* en détail.

6.2.4.5 MaxNetworkMessageSize

La *MaxNetworkMessageSize* de *DataType UInt32* indique la taille maximale en octets des *NetworkMessages* créés par le *WriterGroup*. Elle fait référence à la taille du *NetworkMessage* complet, y compris le remplissage et la signature, sans en-tête supplémentaire ajouté par le mapping avec les protocoles de transport. Si la taille d'un *NetworkMessage* dépasse la *MaxNetworkMessageSize*, le comportement dépend du mapping avec les messages.

Les mappings avec les protocoles de transport définies en 7.3 peuvent définir des restrictions pour la valeur maximale de ce paramètre.

NOTE 1 Il convient de configurer la valeur du paramètre *MaxNetworkMessageSize* de façon à garantir que la taille des *NetworkMessages* et des en-têtes supplémentaires ajoutés par le protocole de transport reste inférieure ou égale à la taille maximale de l'unité de transfert (MTU) du protocole de transport.

6.2.4.6 GroupProperties

Le paramètre *GroupProperties* est une matrice de *DataType KeyValuePair* qui spécifie les propriétés supplémentaires du groupe configuré. Le *DataType KeyValuePair*, défini dans l'IEC 62541-5, se compose d'un *QualifiedName* et d'une valeur de *BaseDataType*.

Le mapping du nom et de la valeur avec la fonctionnalité concrète peut être défini par les mapping savec les protocoles de transport, les éditions ultérieures de l'IEC 62541-14 ou les extensions propres au fournisseur.

6.2.4.7 Structure de PubSubGroup

Ce *DataType Structure* est un type de base abstrait qui s'applique aux *PubSubGroups*. Le *PubSubGroupDataType* est défini de manière formelle dans le Tableau 20.

Tableau 20 – Structure de PubSubGroupDataType

Nom	Type	Description
PubSubGroupDataType	Structure	
name	Chaîne	Nom du *PubSubGroup*.
enabled	Booléen	État activé du *PubSubGroup*.
securityMode	MessageSecurityMode	Défini en 6.2.4.2.
securityGroupId	Chaîne	Défini en 6.2.4.3.
securityKeyServices	EndpointDescription	Défini en 6.2.4.4.
maxNetworkMessageSize	UInt32	Défini en 6.2.4.5.
groupProperties	KeyValuePair	Défini en 6.2.4.6.

La représentation de la *Structure du PubSubGroupDataType* dans l'*AddressSpace* est définie dans le Tableau 21.

Tableau 21 – Définition de PubSubGroupDataType

Attributs	Valeur			
BrowseName	PubSubGroupDataType			
IsAbstract	True			
Références	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Sous-type de Structure défini dans l'IEC 62541-5.				
HasSubtype	DataType	WriterGroupDataType	FALSE	Défini en 6.2.5.7.1.
HasSubtype	DataType	ReaderGroupDataType	FALSE	Défini en 6.2.7.2.1.

6.2.5 Paramètres de WriterGroup

6.2.5.1 WriterGroupId

Le *WriterGroupId* de *DataType UInt16* est un identificateur du *WriterGroup* et il doit être unique dans l'ensemble des *WriterGroups* pour un *PublisherId.* Toutes les valeurs, autres que 0, sont valides. La valeur 0 est définie comme une valeur nulle.

6.2.5.2 PublishingInterval

Le *PublishingInterval* de *DataType Durée* définit l'intervalle en millisecondes pour la publication de *NetworkMessages* et des *DataSetMessages* imbriqués créés par les *DataSetWriters* associés.

Dans le cas de *DataSets* fondés sur un *Événement*, d'aucun ou à plusieurs *DataSetMessages* peuvent être générés pour un *PublishedDataSet* dans un *PublishingInterval*. Tous les *Événements* qui se produisent entre deux *PublishingIntervals* doivent être mis en mémoire tampon jusqu'à l'envoi du *NetworkMessage* suivant. Si le nombre d'Événements dépasse la capacité de la mémoire tampon du DataSetWriter, un *Événement* de type *EventQueueOverflowEventType* est inséré dans la mémoire tampon.

Le *DataType Durée* est un sous-type de *Double* et permet de configurer des intervalles inférieurs à une milliseconde.

6.2.5.3 KeepAliveTime

Le *KeepAliveTime* de *DataType Durée* définit la durée en millisecondes au-delà de laquelle l'*Éditeur* envoie un *DataSetMessage* d'entretien si aucun *DataSetMessage* n'a été envoyé au

cours de cette période par un *DataSetWriter*. La valeur minimale doit être égale au *PublishingInterval*.

6.2.5.4 Priorité

La *Priorité* de *DataType Octet* définit la priorité relative du *WriterGroup* par rapport à tous les autres *WriterGroups* dans l'ensemble des *PubSubConnections* de l'*Éditeur*.

S'il est nécessaire de traiter plusieurs *WriterGroups*, le numéro de priorité définit l'ordre de traitement. La priorité la plus élevée est traitée en premier.

La priorité la plus faible est zéro et la plus élevée est 255.

6.2.5.5 LocaleIds

Les *LocaleIds* de DataType *LocaleId* définissent une liste d'identificateurs de paramètres d'emplacement, classés par ordre de priorité, pour les chaînes localisées de l'ensemble des *DataSetWriters* contenus dans le *WriterGroup*. Le premier *LocaleId* dans la liste a la priorité la plus haute.

Si l'*Éditeur* envoie une *Chaîne* localisée, il doit envoyer la traduction dont la priorité est la plus haute qu'il puisse. S'il n'a de traduction pour aucun des paramètres d'emplacement identifiés dans cette liste, il doit alors envoyer la valeur de *Chaîne* qu'il possède et inclure le *LocaleId* avec la *Chaîne*. Si aucun identificateur de paramètre d'emplacement n'est configuré, l'*Éditeur* doit utiliser un identificateur qu'il possède. Voir l'IEC 62541-3 pour plus d'informations sur *LocaleId*.

6.2.5.6 HeaderLayoutUri

Le *HeaderLayoutUri*, de *DataType Chaîne*, définit le choix d'une configuration bien définie pour un sous-ensemble des paramètres de communication PubSub. Le sous-ensemble affecté est défini par la présentation de l'en-tête.

Une *Chaîne* nulle correspond à l'absence de choix de présentation.

Si une présentation est choisie, tous les paramètres affectés doivent être définis sur les valeurs spécifiées pour la présentation.

6.2.5.7 Structures de WriterGroup

6.2.5.7.1 WriterGroupDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres de configuration pour les *WriterGroups*. Il s'agit d'un sous-type de *PubSubGroupDataType* défini en 6.2.4.7.

Le *WriterGroupDataType* est défini de manière formelle dans le Tableau 22.

Tableau 22 – Structure de WriterGroupDataType

Nom	Type	Description
WriterGroupDataType	Structure	
writerGroupId	UInt16	Défini en 6.2.5.1.
publishingInterval	Durée	Défini en 6.2.5.2.
keepAliveTime	Durée	Défini en 6.2.5.3.
priority	Octet	Défini en 6.2.5.4.
localeIds	LocaleId	Défini en 6.2.5.5.
headerLayoutUri	Chaîne	Défini en 6.2.5.6.
transportSettings	WriterGroupTransportDataType	Paramètres du *WriterGroup* spécifiques au mapping avec le transport. Le type de base abstrait est défini en 6.2.5.7.2. Les sous-types concrets sont définis dans les paragraphes relatifs aux paramètres spécifiques au mapping avec le transport. Si aucun sous-type concret n'est défini pour le mapping avec le transport, le champ doit être nul.
messageSettings	WriterGroupMessageDataType	Paramètres du *WriterGroup* spécifiques au mapping avec les *NetworkMessages*. Le type de base abstrait est défini en 6.2.5.7.3. Les sous-types concrets sont définis dans les paragraphes relatifs aux paramètres spécifiques au mapping avec les messages. Si aucun sous-type concret n'est défini pour le mapping avec le transport, le champ doit être nul.
dataSetWriters	DataSetWriterDataType	DataSetWriters contenus dans le *WriterGroup*. Les paramètres du *DataSetWriter* sont définis en 6.2.3.

La représentation de la *Structure WriterGroupDataType* dans l'*AddressSpace* est définie dans le Tableau 23.

Tableau 23 – Définition de WriterGroupDataType

Attributs	Valeur		
BrowseName	WriterGroupDataType		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
Sous-type de PubSubGroupDataType défini en 6.2.4.7.			

6.2.5.7.2 WriterGroupTransportDataType

Ce *DataType Structure* est un type de base abstrait qui s'applique aux paramètres du *WriterGroup* spécifiques au mapping avec le transport. Le *DataType* abstrait ne définit pas de champs.

La représentation de la *Structure WriterGroupTransportDataType* dans l'*AddressSpace* est définie dans le Tableau 24.

Tableau 24 – Définition de WriterGroupTransportDataType

Attributs	Valeur			
BrowseName	WriterGroupTransportDataType			
IsAbstract	True			
Références	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Sous-type de Structure défini dans l'IEC 62541-5.				
HasSubtype	DataType	DatagramWriterGroupTransportDataType	FALSE	Défini en 6.4.1.2.3.
HasSubtype	DataType	BrokerWriterGroupTransportDataType	FALSE	Défini en 6.4.2.2.6.

6.2.5.7.3 WriterGroupMessageDataType

Ce *DataType Structure* est un type de base abstrait qui s'applique aux paramètres du *WriterGroup* spécifiques au mapping avec les messages. Le *DataType* abstrait ne définit pas de champs.

La représentation de la *Structure WriterGroupMessageDataType* dans l'*AddressSpace* est définie dans le Tableau 25.

Tableau 25 – Structure de WriterGroupMessageDataType

Attributs	Valeur			
BrowseName	WriterGroupMessageDataType			
IsAbstract	True			
Références	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Sous-type de Structure défini dans l'IEC 62541-5.				
HasSubtype	DataType	UadpWriterGroupMessageDataType	FALSE	Défini en 6.3.1.1.7.
HasSubtype	DataType	JsonWriterGroupMessageDataType	FALSE	Défini en 6.3.2.1.2.

6.2.6 Paramètres de PubSubConnection

6.2.6.1 PublisherId

Le *PublisherId* est l'identificateur unique d'un *Éditeur* au sein d'un *Intergiciel Orienté Message*. Il peut être inclus dans un *NetworkMessage* envoyé à des fins d'identification ou de filtrage. En général, la valeur du *PublisherId* est partagée entre les *PubSubConnections*; toutefois, l'affectation du *PublisherId* est spécifique au fournisseur.

Le paramètre *PublisherId* est uniquement pertinent pour les fonctionnalités de l'*Éditeur* au sein d'une *PubSubConnection*. La configuration du filtre côté *Abonné* est contenue dans les paramètres *DataSetReader*.

Les *DataTypes* valides sont *UInteger* et *Chaîne*.

6.2.6.2 TransportProfileUri

Le paramètre *TransportProfileUri* de *DataType Chaîne* indique les mappings avec les protocoles de transport et les messages utilisées.

Les valeurs possibles pour le paramètre *TransportProfileUri* sont définies en tant qu'URI des protocoles de transport définis comme des *Facettes* de transport *PubSub* dans l'IEC 62541-7.

6.2.6.3 Adresse

Le paramètre *Addresse* comporte les informations d'adresse réseau pour l'intergiciel de communication. Les différents DataTypes Structure utilisés pour représenter l'adresse sont définis en 6.2.6.5.3.

6.2.6.4 ConnectionProperties

Le paramètre *ConnectionProperties* est une matrice de *DataType KeyValuePair* qui spécifie les propriétés supplémentaires de la connexion configurée. Le type *KeyValuePair* est défini dans l'IEC 62541-5 et est constitué d'un *QualifiedName* et d'une valeur de *BaseDataType*.

Le mapping de l'espace de nom, du nom et de la valeur avec la fonctionnalité concrète peut être définie par les mappings avec les protocoles de transport, les éditions ultérieures de l'IEC 62541-14 ou les extensions propres au fournisseur.

6.2.6.5 Structure de PubSubConnection

6.2.6.5.1 PubSubConnectionDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres de configuration pour les *PubSubConnections*. Le *PubSubConnectionDataType* est défini de manière formelle dans le Tableau 26.

Tableau 26 – Structure de PubSubConnectionDataType

Nom	Type	Description
PubSubConnectionDataType	Structure	
name	Chaîne	Nom de la *PubSubConnection*.
enabled	Booléen	État activé de la *PubSubConnection*.
publisherId	BaseDataType	Défini en 6.2.6.1.
transportProfileUri	Chaîne	Défini en 6.2.6.2.
address	NetworkAddressDataType	Défini en 6.2.6.3. Le *NetworkAddressDataType* est défini en 6.2.6.5.3.
connectionProperties	KeyValuePair	Défini en 6.2.6.4.
transportSettings	ConnectionTransportDataType	Paramètres de la *PubSubConnection* spécifiques au mapping avec le transport. Le type de base abstrait est défini en 6.2.6.5.2. Les sous-types concrets sont définis dans les paragraphes relatifs aux paramètres spécifiques au mapping avec le transport.
writerGroups	WriterGroupDataType	*WriterGroups* contenus dans la *PubSubConnection*. Le *WriterGroup* est défini en 6.2.5.
readerGroups	ReaderGroupDataType	*ReaderGroups* contenus dans la *PubSubConnection*. Le *ReaderGroup* est défini en 6.2.7.

6.2.6.5.2 ConnectionTransportDataType

Ce *DataType Structure* est un type de base abstrait qui s'applique aux paramètres de la *PubSubConnection* spécifiques au mapping avec le transport. Le *DataType* abstrait ne définit pas de champs.

La représentation de la *Structure ConnectionTransportDataType* dans l'*AddressSpace* est définie dans le Tableau 27.

Tableau 27 – Définition de ConnectionTransportDataType

Attributs	Valeur		
BrowseName	ConnectionTransportDataType		
IsAbstract	True		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
Sous-type de Structure défini dans l'IEC 62541-5.			

6.2.6.5.3 NetworkAddressDataType

Les sous-types de ce *DataType Structure* abstrait sont utilisés pour représenter les informations d'adresse réseau. Le *NetworkAddressDataType* est défini de manière formelle dans le Tableau 28.

Tableau 28 – Structure de NetworkAddressDataType

Nom	Type	Description
NetworkAddressDataType	Structure	
networkInterface	Chaîne	Nom de l'interface réseau utilisée pour la relation de communication.

La représentation de la *Structure NetworkAddressDataType* dans l'*AddressSpace* est définie dans le Tableau 29.

Tableau 29 – Définition de NetworkAddressDataType

Attributs	Valeur			
BrowseName	NetworkAddressDataType			
IsAbstract	True			
Références	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Sous-type de Structure défini dans l'IEC 62541-5.				
HasSubtype	DataType	NetworkAddressUrlDataType	False	Défini en 6.2.6.5.4.

6.2.6.5.4 NetworkAddressUrlDataType

Ce *DataType Structure* est utilisé pour représenter les informations d'adresse réseau sous la forme d'une *Chaîne* URL. Le *NetworkAddressUrlDataType* est défini de manière formelle dans le Tableau 30.

Tableau 30 – Structure de NetworkAddressUrlDataType

Nom	Type	Description
NetworkAddressUrlDataType	Structure	
url	Chaîne	Chaîne d'adresse pour la relation de communication sous la forme d'une *Chaîne* URL.

La représentation de la *Structure NetworkAddressUrlDataType* dans l'*AddressSpace* est définie dans le Tableau 31.

Tableau 31 – Définition de NetworkAddressUrlDataType

Attributs	Valeur		
BrowseName	NetworkAddressUrlDataType		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
Sous-type de NetworkAddressDataType défini en 6.2.6.5.3.			

6.2.7 Paramètres de ReaderGroup

6.2.7.1 Généralités

Le *ReaderGroup* n'ajoute pas de paramètres aux paramètres PubSubGroup partagés.

Le *ReaderGroup* est utilisé pour regrouper une liste de *DataSetReaders*. Il n'est pas symétrique à un *WriterGroup* ni rattaché à un *NetworkMessage* particulier. Les paramètres de filtre associés au *NetworkMessage* figurent dans les *DataSetReaders*.

6.2.7.2 Structures de ReaderGroup

6.2.7.2.1 ReaderGroupDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres de configuration pour les *ReaderGroups*. Le *ReaderGroupDataType* est défini de manière formelle dans le Tableau 32.

Tableau 32 – Structure de ReaderGroupDataType

Nom	Type	Description
ReaderGroupDataType	Structure	
transportSettings	ReaderGroupTransportDataType	Paramètres du *ReaderGroup* spécifiques au mapping avec le transport. Le type de base abstrait est défini en 6.2.7.2.2. Les sous-types concrets sont définis dans les paragraphes relatifs aux paramètres spécifiques au mapping avec le transport.
messageSettings	ReaderGroupMessageDataType	Paramètres du *ReaderGroup* spécifiques au mapping avec les *NetworkMessages*. Le type de base abstrait est défini en 6.2.7.2.3. Les sous-types concrets sont définis dans les paragraphes relatifs aux paramètres spécifiques au mapping avec les messages.
dataSetReaders	DataSetReaderDataType	DataSetReaders contenus dans le ReaderGroup. Le DataSetReader est défini en 6.2.8.

La représentation de la *Structure ReaderGroupDataType* dans l'*AddressSpace* est définie dans le Tableau 33.

Tableau 33 – Définition de ReaderGroupDataType

Attributs	Valeur		
BrowseName	ReaderGroupDataType		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
Sous-type de PubSubGroupDataType défini en 6.2.4.7.			

6.2.7.2.2 ReaderGroupTransportDataType

Ce *DataType Structure* est un type de base abstrait qui s'applique aux paramètres du *ReaderGroup* spécifiques au mapping avec le transport. Le *DataType* abstrait ne définit pas de champs.

La représentation de la *Structure ReaderGroupTransportDataType* dans l'*AddressSpace* est définie dans le Tableau 34.

Tableau 34 – Définition de ReaderGroupTransportDataType

Attributs	Valeur		
BrowseName	ReaderGroupTransportDataType		
IsAbstract	True		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
Sous-type de Structure défini dans l'IEC 62541-5.			

6.2.7.2.3 ReaderGroupMessageDataType

Ce *DataType Structure* est un type de base abstrait qui s'applique aux paramètres du *ReaderGroup* spécifiques au mapping avec les messages. Le *DataType* abstrait ne définit pas de champs.

La représentation de la *Structure ReaderGroupMessageDataType* dans l'*AddressSpace* est définie dans le Tableau 35.

Tableau 35 – Structure de ReaderGroupMessageDataType

Attributs	Valeur		
BrowseName	ReaderGroupMessageDataType		
IsAbstract	True		
Références	**NodeClass**	**BrowseName**	**IsAbstract**
Sous-type de Structure défini dans l'IEC 62541-5.			

6.2.8 Paramètres de DataSetReader

6.2.8.1 PublisherId

Le paramètre *PublisherId* définit l'*Éditeur* de sorte à en recevoir les *NetworkMessages*.

Si la valeur est nulle, le paramètre doit être ignoré et l'ensemble des *NetworkMessages* reçus traversent le filtre *PublisherId*.

Les *DataTypes* valides sont *UInteger* et *Chaîne*.

6.2.8.2 WriterGroupId

Le paramètre *WriterGroupId* de *DataType UInt16* définit l'identificateur du *WriterGroup* correspondant.

La valeur par défaut 0 est définie comme valeur nulle; elle signifie que ce paramètre doit être ignoré.

6.2.8.3 DataSetWriterId

Le paramètre *DataSetWriterId* de *DataType UInt16* définit le *DataSet* choisi dans l'*Éditeur* pour le DataSetReader.

Si la valeur est 0 (nulle), le paramètre doit être ignoré et l'ensemble des *DataSetMessages* reçus traversent le filtre *DataSetWriterId*.

6.2.8.4 DataSetMetaData

Le paramètre *DataSetMetaData* fournit les informations nécessaires pour décoder les *DataSetMessages* en provenance de l'*Éditeur*. Si le *DataSetMetaData* est modifié dans l'*Éditeur* et que la *MajorVersion* a été modifiée, il est nécessaire que le *DataSetReader* mette à jour les *DataSetMetaData* pour continuer à fonctionner. Si la mise à jour ne peut être récupérée pendant le *MessageReceiveTimeout*, l'*État* du *DataSetReader* doit passer à Error_3. Le *PublishedDataSet* associé est défini en 6.2.2. Le *DataSetMetaDataType* est défini en 6.2.2.1.2. Les options de récupération de la mise à jour des *DataSetMetaData* sont décrites en 5.2.3.

6.2.8.5 DataSetFieldContentMask

Le paramètre *DataSetFieldContentMask* de *DataType DataSetFieldContentMask* indique les champs d'une *DataValue* inclus dans les *DataSetMessages*.

Le DataType *DataSetFieldContentMask* est défini en 6.2.3.2.

6.2.8.6 MessageReceiveTimeout

Le paramètre *MessageReceiveTimeout* est le délai maximal acceptable entre deux *DataSetMessages*. Lorsqu'aucun *DataSetMessage* n'est reçu pendant cette période, l'*État* du *DataSetReader* doit être redéfini sur Error_3 jusqu'à la réception du *DataSetMessage* suivant. Les *DataSetMessages* peuvent être des messages de données ou d'entretien.

Le *MessageReceiveTimeout* est rattaché aux paramètres *PublishingInterval*, *KeepAliveTime* et *KeyFrameCount* côté *Éditeur*.

6.2.8.7 KeyFrameCount

Le *KeyFrameCount* de *DataType UInt32* représente le multiplicateur du *PublishingInterval* qui définit le nombre maximal de fois où le *PublishingInterval* expire avant la réception d'un message de type image clé, comportant des valeurs pour tous les champs.

Pour les *DataSets* qui fournissent des mises à jour cycliques, la valeur doit être supérieure ou égale à 1. Pour les *DataSets* non cycliques, tels que les *PublishedEvents*, qui fournissent des *DataSets* fondés sur un événement, la valeur doit être 0.

6.2.8.8 HeaderLayoutUri

Le *HeaderLayoutUri*, de *DataType Chaîne*, définit le choix d'une configuration bien définie pour un sous-ensemble des paramètres de communication PubSub. Le sous-ensemble affecté est défini par la présentation de l'en-tête.

Une *Chaîne* nulle correspond à l'absence de choix de présentation.

Si une présentation est choisie, tous les paramètres affectés doivent être définis sur les valeurs spécifiées pour la présentation.

6.2.8.9 SecurityMode

Ce paramètre est défini en 6.2.4.2.

Ce paramètre écrase le paramètre correspondant sur le *ReaderGroup* si la valeur n'est pas INVALID_0.

6.2.8.10 SecurityGroupId

Ce paramètre est défini en 6.2.4.3.

Ce paramètre doit être nul si le *SecurityMode* est INVALID_0.

6.2.8.11 SecurityKeyServices

Ce paramètre est défini en 6.2.4.4.

Ce paramètre doit être nul si le *SecurityMode* est INVALID_0.

6.2.8.12 DataSetReaderProperties

Le paramètre *DataSetReaderProperties* est une matrice de *DataType KeyValuePair* qui spécifie les propriétés supplémentaires du *DataSetReader* configuré. Le *DataType KeyValuePair*, défini dans l'IEC 62541-5, se compose d'un *QualifiedName* et d'une valeur de *BaseDataType*.

Le mapping du nom et de la valeur avec la fonctionnalité concrète peut être définie par les mapping avec les protocoles de transport, les éditions ultérieures de l'IEC 62541-14 ou les extensions propres au fournisseur.

6.2.8.13 Structure de DataSetReader

6.2.8.13.1 DataSetReaderDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres *DataSetReader*. Le *DataSetReaderDataType* est défini de manière formelle dans le Tableau 36.

Tableau 36 – Structure de DataSetReaderDataType

Nom	Type	Description
DataSetReaderDataType	Structure	
name	Chaîne	Nom du DataSetReader. Le nom doit être unique dans l'ensemble d'un *ReaderGroup*. Il est recommandé d'utiliser un nom lisible par l'homme.
enabled	Booléen	État activé du DataSetReader.
publisherId	BaseDataType	Défini en 6.2.8.1.
writerGroupId	UInt16	Défini en 6.2.8.2.
dataSetWriterId	UInt16	Défini en 6.2.8.3.
dataSetMetaData	DataSetMetaDataType	Défini en 6.2.8.4.
dataSetFieldContentMask	DataSetFieldContentMask	Défini en 6.2.8.5.
messageReceiveTimeout	Durée	Défini en 6.2.8.6.
keyFrameCount	UInt32	Défini en 6.2.8.9.
headerLayoutUri	Chaîne	Défini en 6.2.8.9.
securityMode	MessageSecurityMode	Défini en 6.2.8.9.
securityGroupId	Chaîne	Défini en 6.2.8.10.
securityKeyServices	EndpointDescription	Défini en 6.2.8.11.
dataSetReaderProperties	KeyValuePair	Défini en 6.2.8.12.
transportSettings	DataSetReaderTransportDataType	Paramètres du DataSetReader spécifiques au transport. Le type de base abstrait est défini en 6.2.8.13.2. Les sous-types concrets sont définis dans les paragraphes relatifs aux paramètres spécifiques au mapping avec le transport. Si aucun sous-type concret n'est défini pour le mapping avec les messages, le champ doit être nul.
messageSettings	DataSetReaderMessageDataType	Paramètres du DataSetReader spécifiques au mapping avec les DataSetMessages. Le type de base abstrait est défini en 6.2.8.13.3. Les sous-types concrets sont définis dans les paragraphes relatifs aux paramètres spécifiques au mapping avec les messages. Si aucun sous-type concret n'est défini pour le mapping avec les messages, le champ doit être nul.
subscribedDataSet	SubscribedDataSetDataType	Paramètres spécifiques au SubscribedDataSet. Le type de base abstrait et les sous-types concrets sont définis en 6.2.9.

6.2.8.13.2 DataSetReaderTransportDataType

Ce *DataType Structure* est un type de base abstrait qui s'applique aux paramètres du *DataSetReader* spécifiques à la correspondance de transport. Le *DataSetReaderTransportDataType* est défini de manière formelle dans le Tableau 37.

Tableau 37 – Structure de DataSetReaderTransportDataType

Nom	Type	Description
DataSetReaderTransportDataType	Structure	

La représentation de la *Structure DataSetReaderTransportDataType* dans l'*AddressSpace* est définie dans le Tableau 38.

Tableau 38 – Définition de DataSetReaderTransportDataType

Attributs	Valeur			
BrowseName	DataSetReaderTransportDataType			
IsAbstract	True			
Références	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Sous-type de Structure défini dans l'IEC 62541-5.				
HasSubtype	DataType	BrokerDataSetReaderTransportDataType	FALSE	Défini en 6.4.2.4.6.

6.2.8.13.3 DataSetReaderMessageDataType

Ce *DataType Structure* est un type de base abstrait qui s'applique aux paramètres du *DataSetReader* spécifiques au mapping avec les messages. Le *DataSetReaderMessageDataType* est défini de manière formelle dans le Tableau 39.

Tableau 39 – Structure de DataSetReaderMessageDataType

Nom	Type	Description
DataSetReaderMessageDataType	Structure	

La représentation de la *Structure DataSetReaderMessageDataType* dans l'*AddressSpace* est définie dans le Tableau 40.

Tableau 40 – Définition de DataSetReaderMessageDataType

Attributs	Valeur			
BrowseName	DataSetReaderMessageDataType			
IsAbstract	True			
Références	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Sous-type de Structure défini dans l'IEC 62541-5.				
HasSubtype	DataType	UadpDataSetReaderMessageDataType	FALSE	Défini en 6.3.1.3.10.
HasSubtype	DataType	JsonDataSetReaderMessageDataType	FALSE	Défini en 6.3.2.3.3.

6.2.9 Paramètres de SubscribedDataSet

6.2.9.1 SubscribedDataSetDataType

Ce *DataType Structure* est un type de base abstrait qui s'applique aux paramètres du *SubscribedDataSet*. Le *SubscribedDataSetDataType* est défini de manière formelle dans le Tableau 41.

Tableau 41 – Structure de SubscribedDataSetDataType

Nom	Type	Description
SubscribedDataSetDataType	Structure	

La représentation de la *Structure SubscribedDataSetDataType* dans l'*AddressSpace* est définie dans le Tableau 42.

Tableau 42 – Définition de SubscribedDataSetDataType

Attributs	Valeur			
BrowseName	SubscribedDataSetDataType			
IsAbstract	True			
Références	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Sous-type de Structure défini dans l'IEC 62541-5.				
HasSubtype	DataType	TargetVariablesDataType	FALSE	Défini en 6.2.9.2.2.
HasSubtype	DataType	SubscribedDataSetMirrorDataType	FALSE	Défini en 6.2.9.3.3.

6.2.9.2 TargetVariables

6.2.9.2.1 Généralités

L'option *SubscribedDataSet* des *TargetVariables* définit une liste de mappings de *Variables* entre les champs *DataSet* reçus et les *Variables* cibles de l'*AddressSpace de l'Abonné*. Le *FieldTargetDataType* est défini en 6.2.9.2.3. Les *Variables* cibles ne doivent être utilisées qu'une seule fois au sein d'une même liste de *TargetVariables*.

6.2.9.2.2 TargetVariablesDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres spécifiques aux *TargetVariables*. Il s'agit d'un sous-type du *SubscribedDataSetDataType* défini en 6.2.9.1.

Le *TargetVariablesDataType* est défini de manière formelle dans le Tableau 43.

Tableau 43 – Structure de TargetVariablesDataType

Nom	Type	Description
TargetVariablesDataType	Structure	
targetVariables	FieldTargetDataType	Défini en 6.2.9.2.1.

6.2.9.2.3 FieldTargetDataType

Ce *DataType* est utilisé pour fournir les métadonnées de la relation entre un champ d'un *DataSetMessage* et la *Variable* cible d'un *DataSetReader*. Le *FieldTargetDataType* est défini de manière formelle dans le Tableau 44.

Tableau 44 – Structure de FieldTargetDataType

Nom	Type	Description
FieldTargetDataType	Structure	
dataSetFieldId	Guid	Identificateur unique du champ dans le *DataSet*. Les champs et leurs identificateurs uniques sont définis dans la *Structure DataSetMetaData*.
receiverIndexRange	NumericRange	Plage d'indices utilisée pour extraire les parties d'une matrice à partir des données reçues. Elle est utilisée pour identifier un élément simple d'une matrice ou une plage simple d'indices pour les matrices du champ *DataSet* reçu. Si une plage d'éléments est spécifiée, les valeurs sont retournées sous forme de composé. Le premier élément est identifié par l'indice 0 (zéro). Le type *NumericRange* est défini dans l'IEC 62541-4. Ce paramètre est nul si l'Attribut spécifié n'est pas une matrice. Cependant, si l'Attribut spécifié est une matrice et que ce paramètre est nul, la matrice complète est utilisée. La taille de la matrice de données obtenue pour cette *NumericRange* doit correspondre à la taille de la matrice de données obtenue pour le paramètre *writeIndexRange* de la *NumericRange*.
targetNodeId	NodeId	*NodeId* de la *Variable* sur laquelle la valeur du champ *DataSetMessage* reçu est écrite.
attributeId	IntegerId	Identificateur de l'*Attribut* à écrire, par exemple l'*Attribut Valeur*. Il doit s'agir d'un *AttributeId* valide. Les Attributs sont définis dans l'IEC 62541-3. Le *DataType IntegerId* est défini dans l'IEC 62541-4. Les *IntegerIds* des *Attributs* sont définis dans l'IEC 62541-6.
writeIndexRange	NumericRange	Plage d'indices utilisée pour écrire les données reçues sur le nœud cible. Elle est utilisée pour identifier un élément simple d'une matrice ou une plage simple d'indices pour les matrices de l'opération d'écriture sur le *Nœud* cible. Si une plage d'éléments est spécifiée, les valeurs sont écrites sous forme de composé. Le premier élément est identifié par l'indice 0 (zéro). Le type *NumericRange* est défini dans l'IEC 62541-4. Ce paramètre est nul si l'*Attribut* spécifié n'est pas une matrice. Cependant, si l'*Attribut* spécifié est une matrice et que ce paramètre est nul, la matrice complète est utilisée.
overrideValueHandling	OverrideValueHandling	La valeur est utilisée pour définir le comportement de gestion des valeurs de remplacement si l'État du *DataSetReader* n'est pas Operational_2 ou si le champ correspondant du *DataSet* comporte un *StatusCode Bad*. La gestion de l'*OverrideValue* dans différents scénarios est définie en 6.2.10. Le *DataType* d'énumération *OverrideValueHandling* est défini en 6.2.9.2.4.
overrideValue	Variante	Cette valeur est utilisée si l'*OverrideValueHandling* est définie sur *OverrideValue_2* et que l'État du *DataSetReader* n'est pas Operational_2 ou que le champ correspondant du *DataSet* comporte un *StatusCode Bad*. La gestion de l'*OverrideValue* dans différents scénarios est définie en 6.2.10. Cette Valeur doit correspondre au *DataType* du *Nœud* cible.

6.2.9.2.4 OverrideValueHandling

L'*OverrideValueHandling* est une énumération qui spécifie les options possibles pour la gestion des valeurs de Remplacement. Les valeurs d'énumération possibles sont décrites dans le Tableau 45.

Tableau 45 – Valeurs d'OverrideValueHandling

Valeur	Description
Disabled_0	La gestion des valeurs de remplacement est désactivée.
LastUsableValue_1	En cas d'erreur, la dernière valeur utilisable est utilisée. En l'absence de dernière valeur utilisable disponible, la valeur par défaut du type de données est utilisée.
OverrideValue_2	En cas d'erreur, la valeur de remplacement configurée est utilisée.

6.2.9.3 SubscribedDataSetMirror

6.2.9.3.1 ParentNodeName

Ce paramètre de *DataType Chaîne* définit le *BrowseName* et le *DisplayName* du *Nœud* parent des *Variables* qui représentent les champs du *DataSet* souscrit.

6.2.9.3.2 RolePermissions

Ce paramètre de *DataType RolePermissionType* définit la valeur de l'Attribut *RolePermissions* à configurer sur le Nœud parent. Cette valeur est également utilisée comme *RolePermissions* pour toutes les *Variables* du miroir *DataSet*.

6.2.9.3.3 SubscribedDataSetMirrorDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres spécifiques au *SubscribedDataSetMirror*. Il s'agit d'un sous-type du *SubscribedDataSetDataType* défini en 6.2.9.1.

Le *SubscribedDataSetMirrorDataType* est défini de manière formelle dans le Tableau 46.

Tableau 46 – Structure de SubscribedDataSetMirrorDataType

Nom	Type	Description
SubscribedDataSetMirrorDataType	Structure	
parentNodeName	Chaîne	Défini en 6.2.9.3.1.
rolePermissions	RolePermissionType	Défini en 6.2.9.3.2.

6.2.10 Flux d'informations et gestion des statuts

Le modèle de configuration définit différents paramètres qui influencent le flux d'informations de l'*Éditeur* vers l'*Abonné* en cas de Valeur de Statut "Bad" ou d'autres situations d'erreur. La Figure 22 représente les paramètres et le flux d'informations à l'intérieur d'un *Éditeur* et à l'intérieur d'un *Abonné*.

Les paramètres et le comportement adapté au codage d'un *DataSetMessage* côté *Éditeur* et au décodage du *DataSetMessage* côté *Abonné* sont définis en 6.2.3.1 avec le *DataSetFieldContentMask*.

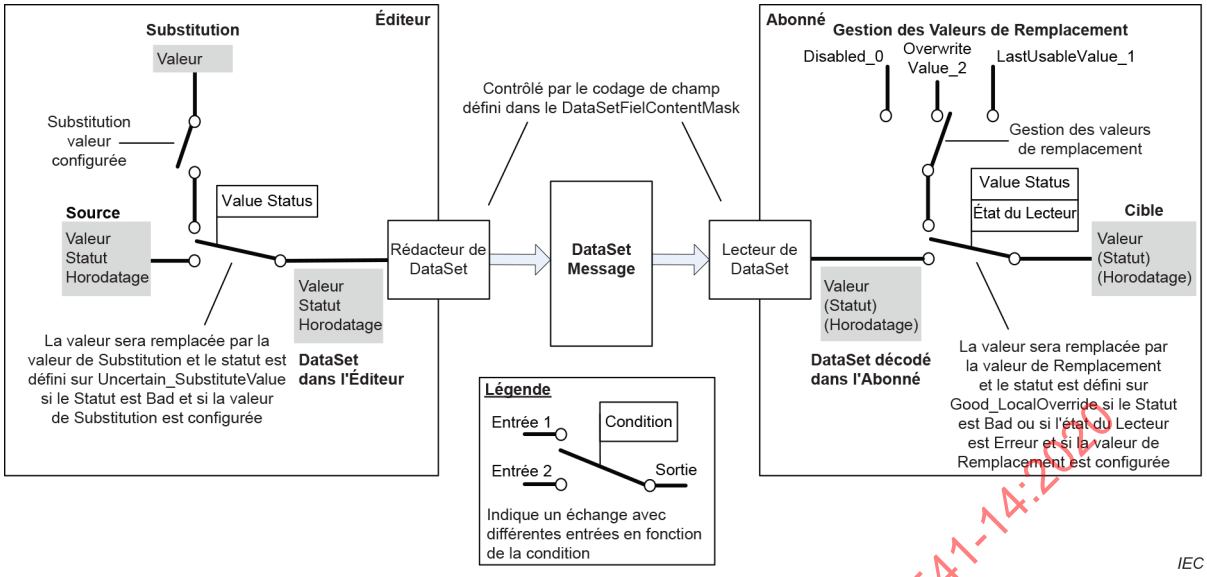


Figure 22 – Flux d'informations PubSub

Le mapping de la valeur et du statut sources avec le *DataSet* dans l'*Éditeur* dépend de la valeur de substitution. Les dépendances sont définies dans le Tableau 47.

Tableau 47 – Mapping de la source vers l'entrée de message

Source		Valeur de substitution	Côté Éditeur du DataSet	
Valeur	**Statut**[a]		**Valeur**	**Statut**[a]
Valeur 1	Good_*	Valeur 2	Valeur 1	Good_*
Valeur 1	Uncertain_*		Valeur 1	Uncertain_*
Nulle	Bad_*		Valeur 2	Uncertain_SubstituteValue
Valeur 1	Good_*	Nulle	Valeur 1	Good_*
Valeur 1	Uncertain_*		Valeur 1	Uncertain_*
Nulle	Bad_*		Nulle	Bad_*

[a] Si aucun *StatusCode* spécifique n'est utilisé, le regroupement dans la sévérité "Good", "Uncertain" ou "Bad" est utilisé. Dans ce cas, le Statut obtenu correspond au Statut d'entrée.

Le mapping du *DataSet* décodé côté *Abonné* avec la valeur et le statut de la *Variable* cible dépend de la valeur de remplacement. Les dépendances sont définies dans le Tableau 48.

Tableau 48 – Mapping de la sortie de message vers la cible

Abonné de DataSet décodé		Énumération de gestion des valeurs de remplacement	Valeur de remplacement	État du lecteur	Cible	
Valeur	Statut[a]				Valeur	Statut[a]
Valeur 1	Good_*	OverrideValue_2	Valeur 2	Operational_2	Valeur 1	Good_*
Valeur 1	Uncertain_*				Valeur 1	Uncertain_*
Nulle	Bad_*				Valeur 2	Good_LocalOverride
Valeur 1	Good_*	LastUsableValue_1	Nulle		Valeur 1	Good_*
Valeur 1	Uncertain_*				Valeur 1	Uncertain_*
Nulle	Bad_*				LastValue[b]	Uncertain_LastUsableValue
Valeur 1	Good_*	Disabled_0	Nulle		Valeur 1	Good_*
Valeur 1	Uncertain_*				Valeur 1	Uncertain_*
Nulle	Bad_*				Nulle	Bad_*
Aucun message reçu. Les valeurs cibles sont mises à jour après un changement d'état du lecteur.		OverrideValue_2	Valeur 2	Diabled_0 Paused_1	Valeur 2	Good_LocalOverride
		LastUsableValue_1	Nulle		LastValue[b]	Uncertain_LastUsableValue
		Disabled_0	Nulle		Nulle	Bad_OutOfService
		OverrideValue_2	Valeur 2	Error_3	Valeur 2	Good_LocalOverride
		LastUsableValue_1	Nulle		LastValue[b]	Uncertain_LastUsableValue
		Disabled_0	Nulle		Nulle	Bad_NoCommunication

[a] Si aucun *StatusCode* spécifique n'est utilisé, le regroupement dans la sévérité "Good", "Uncertain" ou "Bad" est utilisé. Dans ce cas, le Statut obtenu correspond au Statut d'entrée.

[b] La dernière valeur correspond à la dernière valeur reçue ou à la valeur par défaut du type de données si aucune valeur n'a été reçue auparavant.

6.2.11 PubSubConfigurationDataType

Ce *DataType Structure* est utilisé pour représenter la configuration *PubSub* d'une *Application* OPC UA. Le *PubSubConfigurationDataType* est défini de manière formelle dans le Tableau 49.

Tableau 49 – Structure de PubSubConfigurationDataType

Nom	Type	Description
PubSubConfigurationDataType	Structure	
publishedDataSets	PublishedDataSetDataType	*PublishedDataSets* contenus dans la configuration. Le *PublishedDataSet* est défini en 6.2.2.
connections	PubSubConnectionDataType	*PubSubConnections* contenues dans la configuration. La *PubSubConnection* est définie en 6.2.6. La connexion inclut les *WriterGroups* et les *ReaderGroups*.
enabled	Booléen	État activé de la configuration *PubSub*.

Si la configuration *PubSub* est stockée dans un fichier, le *UABinaryFileDataType* et les définitions associées de A.2 doivent être utilisés pour coder le contenu du fichier. Les valeurs de la structure *UABinaryFileDataType* sont décrites dans le Tableau 50.

Tableau 50 – Contenu de fichier PubSubConfiguration

Champ	Type	Valeur
namespaces	Chaîne[]	nulle Les *DataTypes* utilisés pour la configuration sont définis dans l'espace de nom OPC UA.
structureDataTypes	StructureDescription[]	nulle Les *DataTypes* utilisés pour la configuration sont définis par OPC UA.
enumDataTypes	EnumDescription[]	nulle Les *DataTypes* utilisés pour la configuration sont définis par OPC UA.
simpleDataTypes	SimpleTypeDescription[]	nulle Les *DataTypes* utilisés pour la configuration sont définis par OPC UA.
schemaLocation	Chaîne	nulle
fileHeader	KeyValuePair[]	nulle
body	BaseDataType	*Structure PubSubConfigurationDataType* Configuration *PubSub* représentée par le *PubSubConfigurationDataType*.

6.3 Paramètres de configuration de mapping de message

6.3.1 Mapping de message UADP

6.3.1.1 Rédacteur de NetworkMessage UADP

6.3.1.1.1 Relation des paramètres de temps

Le *PublishingInterval*, le *SamplingOffset*, le *PublishingOffset* et l'*Horodatage* de l'en-tête du *NetworkMessage* doivent utiliser la même base temporelle.

Si un réseau sous-jacent fournit une horloge globale synchronisée, cette horloge doit être utilisée comme base temporelle de l'*Éditeur* et de l'*Abonné*.

Le début d'un *PublishingInterval* doit être un multiple du *PublishingInterval* associé au début de la base temporelle. L'heure de début de référence du *PublishsingInterval* peut être calculée en utilisant la formule suivante:

Début de l'exécution périodique =

heure actuelle + PublishingInterval − (heure actuelle / PublishingInterval)

L'heure actuelle est le nombre de nanosecondes écoulées depuis le début de l'époque utilisée par l'horloge de référence.

Le *PublishingInterval* est la durée en nanosecondes.

Le début de l'exécution périodique est le nombre de nanosecondes écoulées depuis le début de l'époque correspondant au prochain début possible d'un *PublishingInterval*.

La Figure 23 représente un exemple de choix pour le début possible d'un *PublishingInterval*.

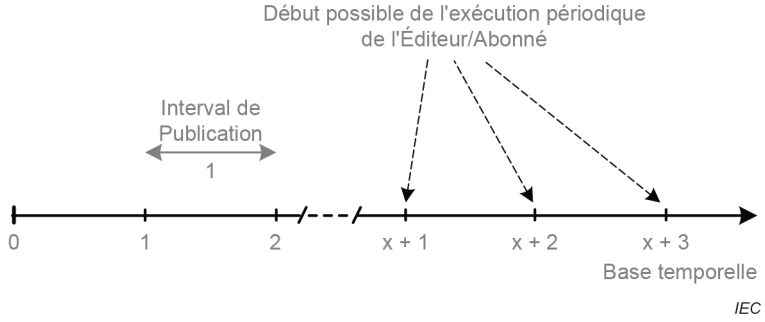


Figure 23 – Début de l'exécution périodique de l'éditeur

Les différents décalages de temps au sein d'un cycle *PublishingInterval* côtés *Éditeur* et *Abonné* sont représentés à la Figure 24. Le *SamplingOffset* et le *PublishingOffset* sont définis comme des paramètres du *WriterGroup* UADP. Le *ReceiveOffset* et le *ProcessingOffset* sont définis comme des paramètres du *DataSetReader* UADP en 6.3.1.3.

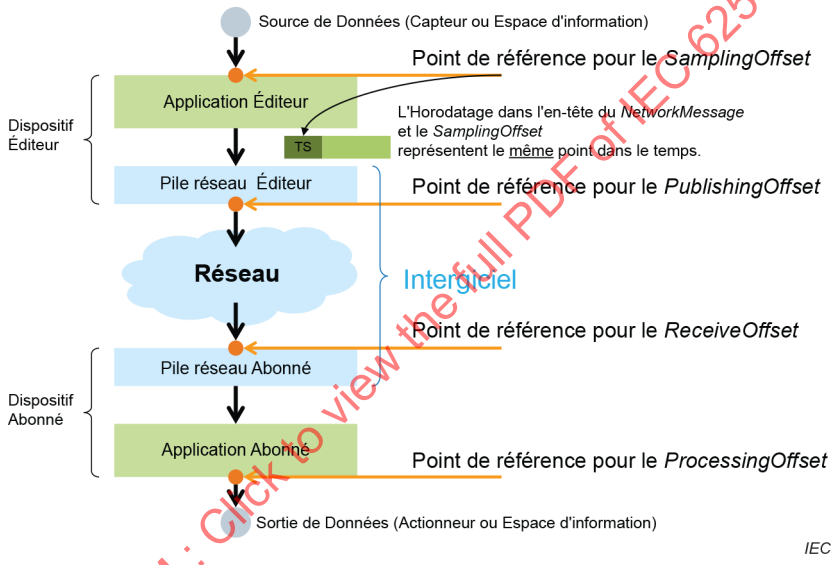


Figure 24 – Décalages de temps au sein d'un PublishingInterval

6.3.1.1.2 GroupVersion

La *GroupVersion* de *DataType VersionTime* reflète l'heure de la dernière modification de présentation du contenu des *NetworkMessages* publiés par le *WriterGroup*. Le *DataType VersionTime* est défini dans l'IEC 62541-4. La *GroupVersion* varie lorsque l'un des paramètres suivants est modifié:

- NetworkMessageContentMask de ce WriterGroup;
- Décalage d'un DataSetWriter du WriterGroup;
- MinorVersion du DataSet d'un DataSetWriter du WriterGroup;
- DataSetFieldContentMask d'un DataSetWriter du WriterGroup;
- DataSetMessageContentMask d'un DataSetWriter de ce WriterGroup;
- DataSetWriterId d'un DataSetWriter du WriterGroup.

La *GroupVersion* est valide pour tous les *NetworkMessages* obtenus à partir de ce *WriterGroup*.

6.3.1.1.3 DataSetOrdering

Le *DataSetOrdering* définit le classement des *DataSetMessages* dans les *NetworkMessages*. Les valeurs possibles pour le *DataSetOrdering* sont décrites dans le Tableau 51. La valeur par défaut est Undefined_0.

Le *DataSetOrderingType* est une énumération qui spécifie les options possibles pour le classement des *DataSetMessages* dans les *NetworkMessages*. Les valeurs d'énumération possibles sont décrites dans le Tableau 51.

Tableau 51 – Valeurs de DataSetOrderingType

Valeur	Description
Undefined_0	Le classement des *DataSetMessages* n'est pas spécifié.
AscendingWriterId_1	Les *DataSetMessages* sont classés par ordre croissant d'après la valeur des *DataSetWriterIds* correspondants.
AscendingWriterIdSingle_2	Les *DataSetMessages* sont classés par ordre croissant d'après la valeur des *DataSetWriterIds* correspondants; un seul *DataSetMessage* est envoyé par *NetworkMessage*.

Si le *DataSetOrdering* est *Undefined_0*, tout classement entre les DataSets et leur distribution au sein des *NetworkMessages* est admis. Le classement et la distribution peuvent même varier entre chaque *PublishingInterval*. Si le *DataSetOrdering* est défini sur *AscendingWriterId_1*, l'*Éditeur* doit remplir chaque *NetworkMessage* avec les *DataSets* dans l'ordre croissant des *DataSetWriterIds* correspondants, tant que les tailles de *DataSet* cumulées ne dépassent pas la *MaxNetworkMessageSize*. Les différentes options sont représentées à la Figure 25.

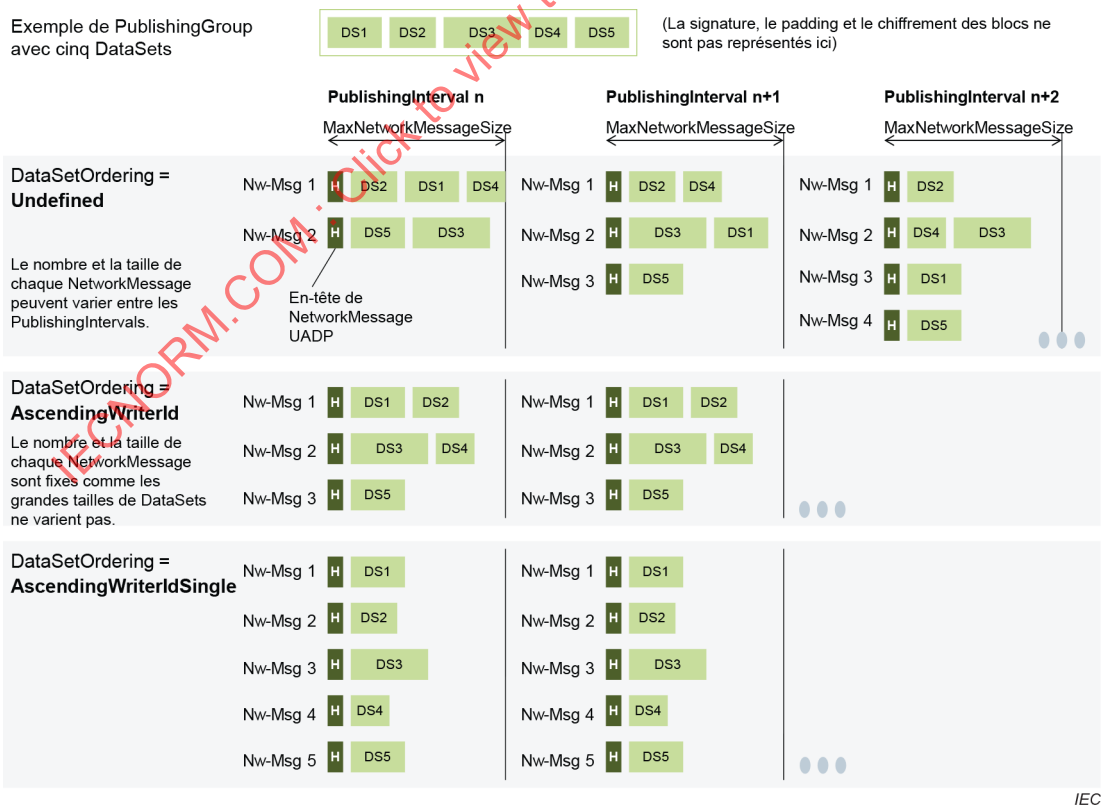


Figure 25 – DataSetOrdering et MaxNetworkMessageSize

6.3.1.1.4 NetworkMessageContentMask

Le paramètre *NetworkMessageContentMask* définit les champs d'en-tête facultatifs à inclure dans les *NetworkMessages* générés par le *WriterGroup*. Le *DataType* pour le mapping de *NetworkMessage* UADP est *UadpNetworkMessageContentMask*.

Le *DataType UadpNetworkMessageContentMask* est défini de manière formelle dans le Tableau 52.

Tableau 52 – Valeurs de UadpNetworkMessageContentMask

Valeur	N° de bit	Description
PublisherId	0	Le *PublisherId* est inclus dans les *NetworkMessages*.
GroupHeader	1	Le GroupHeader est inclus dans les *NetworkMessages*.
WriterGroupId	2	Le champ *WriterGroupId* est inclus dans le *GroupHeader*. Le fanion est uniquement valide si Bit 1 est configuré.
GroupVersion	3	Le champ *GroupVersion* est inclus dans le *GroupHeader*. Le fanion est uniquement valide si Bit 1 est configuré.
NetworkMessageNumber	4	Le champ *NetworkMessageNumber* est inclus dans le *GroupHeader*. Le champ est exigé si plus d'un NetworkMessage est nécessaire pour transférer tous les DataSets du groupe. Le fanion est uniquement valide si Bit 1 est configuré.
SequenceNumber	5	Le champ *SequenceNumber* est inclus dans le *GroupHeader*. Le fanion est uniquement valide si Bit 1 est configuré.
PayloadHeader	6	Le *PayloadHeader* est inclus dans les *NetworkMessages*.
Horodatage	7	L'horodatage de l'expéditeur est inclus dans les *NetworkMessages*.
PicoSeconds	8	La partie *PicoSeconds* de l'horodatage de l'expéditeur est incluse dans les *NetworkMessages*.
DataSetClassId	9	Le *DataSetClassId* est inclus dans les *NetworkMessages*.
PromotedFields	10	Les *PromotedFields* sont inclus dans les *NetworkMessages*.

La représentation de *UadpNetworkMessageContentMask* dans l'*AddressSpace* est définie dans le Tableau 53.

Tableau 53 – Définition de UadpNetworkMessageContentMask

Attributs	Valeur		
BrowseName	UadpNetworkMessageContentMask		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**DataType**
Sous-type de UInt32 défini dans l'IEC 62541-5.			
HasProperty	Variable	OptionSetValues	LocalizedText [ ]

6.3.1.1.5 SamplingOffset

Le *SamplingOffset* de *DataType Durée* définit le temps (en millisecondes) du décalage de création du *NetworkMessage* dans le cycle *PublishingInterval*.

Toute valeur négative indique que le paramètre facultatif n'est pas configuré. Dans ce cas, l'*Éditeur* doit calculer le temps avant le *PublishingOffset* nécessaire pour créer le *NetworkMessage* à temps pour l'envoi au *PublishingOffset*.

Le *DataType Durée* est un sous-type de *Double* et permet de configurer des intervalles inférieurs à une milliseconde.

6.3.1.1.6 PublishingOffset

Le *PublishingOffset* est une matrice de *DataType Durée* qui définit le temps (en millisecondes) du décalage dans le cycle *PublishingInterval* d'envoi du *NetworkMessage* au réseau.

Le *DataType Durée* est un sous-type de *Double* et permet de configurer des intervalles inférieurs à une milliseconde.

La Figure 26 représente les différentes variations des paramètres *PublishingOffset* qui affectent l'envoi de plusieurs *NetworkMessages*.

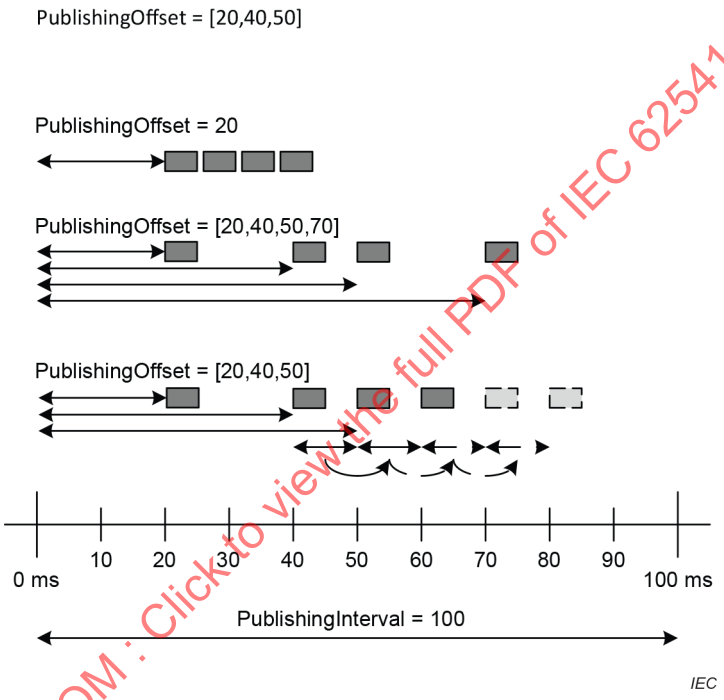


Figure 26 – Options PublishingOffset pour plusieurs *NetworkMessages*

Si tous les *DataSets* d'un groupe sont transférés avec un unique *NetworkMessage*, la valeur scalaire de la première valeur de la matrice définit le décalage pour l'envoi du *NetworkMessage* par rapport au début du cycle *PublishingInterval*. Si les *DataSets* d'un groupe sont envoyés une série de *NetworkMessages*, les valeurs de la matrice définissent les décalages d'envoi des *NetworkMessages* par rapport au début du cycle *PublishingInterval*. Si une valeur scalaire est configurée, le premier *NetworkMessage* est envoyé au moment du décalage et les *NetworkMessages* suivants sont envoyés immédiatement les uns après les autres. Si le nombre de *NetworkMessages* disponibles pour l'envoi est plus important que les valeurs de décalage de la matrice, le décalage pour les *NetworkMessages* restants est extrapolé à partir des deux dernières valeurs de décalage de la matrice.

Le *PublishingInterval*, le *SamplingOffset*, le *PublishingOffset* et l'*Horodatage* de l'en-tête du *NetworkMessage* doivent utiliser la même base temporelle.

6.3.1.1.7 Structure de UadpWriterGroupMessageDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres WriterGroup spécifiques mapping de *NetworkMessage* UADP. Il s'agit d'un sous-type de *WriterGroupMessageDataType* défini en 6.2.5.7.3.

Le *UadpWriterGroupMessageDataType* est défini de manière formelle dans le Tableau 54.

Tableau 54 – Structure de UadpWriterGroupMessageDataType

Nom	Type	Description
UadpWriterGroupMessageDataType	Structure	
groupVersion	UInt32	Défini en 6.3.1.1.2.
dataSetOrdering	DataSetOrderingType	Défini en 6.3.1.1.3.
networkMessageContentMask	UadpNetworkMessageContentMask	Défini en 6.3.1.1.4.
samplingOffset	Durée	Défini en 6.3.1.1.5.
publishingOffset	Durée	Défini en 6.3.1.1.6.

6.3.1.2 Rédacteur de DataSetMessage UADP

6.3.1.2.1 Généralités

La configuration des *DataSetWriters* dans un *WriterGroup* peut entraîner une présentation fixe du *NetworkMessage* où tous les *DataSets* ont une position statique entre les *NetworkMessages*.

Dans ce cas, les paramètres *NetworkMessageNumber* et *DataSetOffset* fournissent des informations concernant la position statique du *DataSetMessage* dans un *NetworkMessage* sur lesquelles les *Abonnés* peuvent s'appuyer. Si la valeur de l'un des deux paramètres est 0, la position n'est pas garantie comme étant statique.

NOTE Un *Éditeur* peut uniquement fournir des valeurs valides pour les paramètres *NetworkMessageNumber* et *DataSetOffset* si le mapping de message permet de conserver une valeur constante pour ces *Propriétés*, sauf lorsque la configuration du *WriterGroup* est modifiée.

6.3.1.2.2 DataSetMessageContentMask

Le *DataSetMessageContentMask* définit les fanions pour le contenu de l'en-tête du *DataSetMessage*. Les fanions spécifiques au mapping de message UADP sont définis par le *DataType UadpDataSetMessageContentMask*.

Le *DataType UadpDataSetMessageContentMask* est défini de manière formelle dans le Tableau 55.

Tableau 55 – Valeurs de UadpDataSetMessageContentMask

Valeur	N° de bit	Description
Horodatage	0	Si ce fanion est défini, un horodatage doit être inclus dans l'en-tête du *DataSetMessage*.
PicoSeconds	1	Si ce fanion est défini, un champ d'horodatage *PicoSeconds* doit être inclus dans l'en-tête du *DataSetMessage*. Ce fanion est ignoré si le fanion *HeaderTimestamp* n'est pas défini.
Statut	2	Si ce fanion est défini, le statut du *DataSetMessage* est inclus dans l'en-tête du *DataSetMessage*. Les règles de création du statut du *DataSetMessage* sont définies dans le Tableau 16.
MajorVersion	3	Si ce fanion est défini, la *ConfigurationVersion.MajorVersion* est incluse dans l'en-tête du *DataSetMessage*.
MinorVersion	4	Si ce fanion est défini, la *ConfigurationVersion.MinorVersion* est incluse dans l'en-tête du *DataSetMessage*.
SequenceNumber	5	Si ce fanion est défini, le DataSetMessageSequenceNumber est inclus dans l'en-tête du DataSetMessage.

La représentation de *UadpDataSetMessageContentMask* dans l'*AddressSpace* est définie dans le Tableau 56.

Tableau 56 – Définition de UadpDataSetMessageContentMask

Attributs	**Valeur**		
BrowseName	UadpDataSetMessageContentMask		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**DataType**
Sous-type de UInt32 défini dans l'IEC 62541-5.			
HasProperty	Variable	OptionSetValues	LocalizedText [ ]

6.3.1.2.3 ConfiguredSize

Le paramètre *ConfiguredSize* de *DataType UInt16* définit la taille fixe (en octets) qu'utilise un *DataSetMessage* à l'intérieur d'un *NetworkMessage*. La valeur par défaut est 0, ce qui indique une longueur dynamique. Si un *DataSetMessage* est plus léger (par exemple en raison des valeurs actuelles codées), le *DataSetMessage* est rempli d'octets de valeur zéro. S'il est plus lourd, l'*Éditeur* doit paramétrer le bit 0 du *DataSetFlags1* sur "False" afin d'indiquer que le *DataSetMessage* n'est pas valide.

NOTE Le paramètre *ConfiguredSize* peut être utilisé pour différentes raisons. L'une d'entre elles est la réservation d'espace dans un *NetworkMessage* en paramétrant la *ConfiguredSize* sur une valeur plus élevée que celle réellement exigée par le *DataSet* affecté. Les modifications (extensions, par exemple) du *DataSet* ne modifient donc pas la bande passante exigée sur le réseau, ce qui réduit le risque d'effets collatéraux. Une autre raison est de conserver un comportement réseau prévisible, y compris en utilisant des *DataTypes* à champ volatile comme *Chaîne* ou *ByteString*.

6.3.1.2.4 NetworkMessageNumber

Le paramètre *NetworkMessageNumber* de *DataType UInt16* est un paramètre en lecture seule défini par l'*Éditeur* en cas de présentation fixe du *NetworkMessage*. La valeur par défaut est 0 et indique que la position du *DataSetMessage* dans un *NetworkMessage* n'est pas fixe.

Si la présentation du *NetworkMessage* est fixe et que tous les *DataSetMessages* d'un *WriterGroup* s'ajustent dans un unique *NetworkMessage*, la valeur de *NetworkMessageNumber* doit être 1. Si les *DataSetMessages* d'un *WriterGroup* sont distribués ou regroupés sur plusieurs *NetworkMessages*, le premier *NetworkMessage* d'un *PublishingInterval* doit être généré avec la valeur 1; les *NetworkMessages* suivants doivent être générés avec des *NetworkMessageNumbers* incrémentiels. Afin d'éviter qu'il ne repasse à 1, le nombre de *NetworkMessages* générés à partir d'un *WriterGroup* dans un *PublishingInterval* est limité à 65 535.

6.3.1.2.5 DataSetOffset

Le paramètre *DataSetOffset* de *DataType UInt16* est un paramètre en lecture seule défini par l'*Éditeur* qui spécifie le décalage (en octets) à l'intérieur d'un *NetworkMessage* sur lequel se trouve le *DataSetMessage* par rapport au début du *NetworkMessage*. La valeur par défaut 0 indique que la position du *DataSetMessage* dans un *NetworkMessage* n'est pas fixe.

6.3.1.2.6 Structure de UadpDataSetWriterMessageDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres *DataSetWriter* spécifiques au mapping de DataSetMessage UADP. Il s'agit d'un sous-type du *DataSetWriterMessageDataType* défini en 6.2.3.5.3.

Le *UadpDataSetWriterMessageDataType* est défini de manière formelle dans le Tableau 57.

Tableau 57 – Structure de UadpDataSetWriterMessageDataType

Nom	Type	Description
UadpDataSetWriterMessageDataType	Structure	
dataSetMessageContentMask	UadpDataSetMessageContentMask	Défini en 6.3.1.2.2.
configuredSize	UInt16	Défini en 6.3.1.2.3.
networkMessageNumber	UInt16	Défini en 6.3.1.2.4.
dataSetOffset	UInt16	Défini en 6.3.1.2.5.

6.3.1.3 Lecteur de DataSetMessage UADP

6.3.1.3.1 GroupVersion

Le paramètre *GroupVersion* de *DataType VersionTime* définit la valeur attendue dans le champ *GroupVersion* de l'en-tête du *NetworkMessage*. La valeur par défaut 0 est définie comme valeur nulle; elle signifie que ce paramètre doit être ignoré.

6.3.1.3.2 NetworkMessageNumber

Le paramètre *NetworkMessageNumber* de *DataType UInt16* est le nombre de *NetworkMessages* à l'intérieur d'un *PublishingInterval* dans lequel le *DataSetMessage* est publié. La valeur par défaut 0 est définie comme valeur nulle; elle signifie que ce paramètre doit être ignoré.

6.3.1.3.3 DataSetOffset

Le paramètre *DataSetOffset* de *DataType UInt16* définit le décalage pour le *DataSetMessage* à l'intérieur du *NetworkMessage* correspondant. La valeur par défaut 0 est définie comme valeur nulle; elle signifie que ce paramètre doit être ignoré.

6.3.1.3.4 DataSetClassId

Le paramètre *DataSetClassId* de *DataType Guid* définit un filtre associé à la classe de *DataSet*. Si la valeur est nulle, le filtre *DataSetClassId* n'est pas appliqué.

6.3.1.3.5 Network Message ContentMask

Le *NetworkMessageContentMask* de *DataType UadpNetworkMessageContentMask* indique les champs d'en-tête facultatifs inclus dans les *NetworkMessages* reçus. Le *DataType UadpNetworkMessageContentMask* est défini en 6.3.1.1.4.

6.3.1.3.6 DataSetMessage ContentMask

Le *DataSetMessageContentMask* de *DataType UadpDataSetMessageContentMask* indique les champs d'en-tête facultatifs inclus dans les *DataSetMessages*.

Le *DataType UadpDataSetMessageContentMask* est défini en 6.3.1.2.2.

6.3.1.3.7 PublishingInterval

Le *PublishingInterval* de *DataType Durée* indique le rythme auquel l'*Éditeur* envoie des *NetworkMessages* associés au *DataSet*. L'heure de début de l'exécution périodique de l'*Abonné* doit être calculée conformément au 6.3.1.1.1.

6.3.1.3.8 ReceiveOffset

Le *ReceiveOffset* de *DataType Durée* définit le temps (en millisecondes) du décalage dans le cycle *PublishingInterval* pour l'heure de réception attendue du *NetworkMessage* pour le *DataSet* en provenance du réseau.

6.3.1.3.9 ProcessingOffset

Le *ProcessingOffset* de *DataType Durée* définit le temps (en millisecondes) du décalage dans le cycle *PublishingInterval* lorsque le DataSet reçu a besoin d'être traité par l'application dans l'*Abonné*.

Les différents décalages de temps au sein d'un cycle *PublishingInterval* côtés *Éditeur* et *Abonné* sont représentés à la Figure 24.

6.3.1.3.10 UadpDataSetReaderMessageDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres *DataSetReader* spécifiques au mapping de message UADP. Il s'agit d'un sous-type du *DataSetReaderMessageDataType* défini en 6.2.8.13.3.

Le *UadpDataSetReaderMessageDataType* est défini de manière formelle dans le Tableau 58.

Tableau 58 – Structure de UadpDataSetReaderMessageDataType

Nom	Type	Description
UadpDataSetReaderMessageDataType	Structure	
groupVersion	VersionTime	Défini en 6.3.1.3.1.
networkMessageNumber	UInt16	Défini en 6.3.1.3.2.
dataSetOffset	UInt16	Défini en 6.3.1.3.3.
dataSetClassId	Guid	Défini en 6.3.1.3.4.
networkMessageContentMask	UadpNetworkMessageContentMask	Défini en 6.3.1.3.5.
dataSetMessageContentMask	UadpDataSetMessageContentMask	Défini en 6.3.1.3.6.
publishingInterval	Durée	Défini en 6.3.1.3.7.
receiveOffset	Durée	Défini en 6.3.1.3.8.
processingOffset	Durée	Défini en 6.3.1.3.9.

6.3.2 Mapping de message JSON

6.3.2.1 Rédacteur de NetworkMessage JSON

6.3.2.1.1 NetworkMessageContentMask

Le paramètre *NetworkMessageContentMask* définit les champs d'en-tête facultatifs à inclure dans les *NetworkMessages* générés par le *WriterGroup*. Le *DataType* pour le mapping de *NetworkMessage* JSON est *JsonNetworkMessageContentMask*.

Le *DataType JsonNetworkMessageContentMask* est défini de manière formelle dans le Tableau 59.

Tableau 59 – Valeurs de JsonNetworkMessageContentMask

Valeur	N° de bit	Description
NetworkMessageHeader	0	L'en-tête du *NetworkMessage* JSON est inclus dans les *NetworkMessages*. Si ce bit est "False", les bits 2 à 4 doivent être 0.
DataSetMessageHeader	1	L'en-tête du *DataSetMessage* JSON est inclus dans chaque *DataSetMessage*. Si ce bit est "False", le *DataSetMessageContentMask* pour les *DataSetWriters* est ignoré (voir 6.3.2.2.1).
SingleDataSetMessage	2	Chaque *NetworkMessage* JSON comporte un seul *DataSetMessage*.
PublisherId	3	Le *PublisherId* est inclus dans les *NetworkMessages*.
DataSetClassId	4	Le *DataSetClassId* est inclus dans les *NetworkMessages*.
ReplyTo	5	*ReplyTo* est inclus dans les *NetworkMessages*.

La représentation de *JsonNetworkMessageContentMask* dans l'*AddressSpace* est définie dans le Tableau 60.

Tableau 60 – Définition de JsonNetworkMessageContentMask

Attributs	Valeur		
BrowseName	JsonNetworkMessageContentMask		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**DataType**
Sous-type de UInt32 défini dans l'IEC 62541-5.			
HasProperty	Variable	OptionSetValues	LocalizedText [ ]

6.3.2.1.2 Structure de JsonWriterGroupMessageDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres *WriterGroup* spécifiques au mapping de *NetworkMessage* JSON. Il s'agit d'un sous-type de *WriterGroupMessageDataType* défini en 6.2.5.7.3.

Le *JsonWriterGroupMessageDataType* est défini de manière formelle dans le Tableau 61.

Tableau 61 – Structure de JsonWriterGroupMessageDataType

Nom	Type	Description
JsonWriterGroupMessageDataType	Structure	
networkMessageContentMask	JsonNetworkMessageContentMask	Défini en 6.3.2.1.1.

6.3.2.2 Rédacteur de DataSetMessage JSON

6.3.2.2.1 DataSetMessageContentMask

Le *DataSetMessageContentMask* définit les fanions pour le contenu de l'en-tête du *DataSetMessage*. Les fanions spécifiques au mapping de message JSON sont définis par le *DataType JsonDataSetMessageContentMask*.

Le *DataType JsonDataSetMessageContentMask* est défini de manière formelle dans le Tableau 62.

Tableau 62 – Valeurs de JsonDataSetMessageContentMask

Valeur	N° de bit	Description
DataSetWriterId	0	Si ce fanion est défini, un *DataSetWriterId* doit être inclus dans l'en-tête du *DataSetMessage*.
MetaDataVersion	1	Si ce fanion est défini, la *ConfigurationVersion* est incluse dans l'en-tête du *DataSetMessage*.
SequenceNumber	2	Si ce fanion est défini, le *DataSetMessageSequenceNumber* est inclus dans l'en-tête du *DataSetMessage*.
Horodatage	3	Si ce fanion est défini, un horodatage doit être inclus dans l'en-tête du *DataSetMessage*.
Statut	4	Si ce fanion est défini, le statut global est inclus dans l'en-tête du *DataSetMessage*.

La représentation de *JsonDataSetMessageContentMask* dans l'*AddressSpace* est définie dans le Tableau 63.

Tableau 63 – Définition de JsonDataSetMessageContentMask

Attributs	**Valeur**		
BrowseName	JsonDataSetMessageContentMask		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**DataType**
Sous-type de UInt32 défini dans l'IEC 62541-5.			
HasProperty	Variable	OptionSetValues	LocalizedText [ ]

6.3.2.2.2 Structure de JsonDataSetWriterMessageDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres *DataSetWriter* spécifiques au mapping de *DataSetMessage* JSON. Il s'agit d'un sous-type du *DataSetWriterMessageDataType* défini en 6.2.3.5.3.

Le *JsonDataSetWriterMessageDataType* est défini de manière formelle dans le Tableau 64.

Tableau 64 – Structure de JsonDataSetWriterMessageDataType

Nom	Type	Description
JsonDataSetWriterMessageDataType	Structure	
dataSetMessageContentMask	JsonDataSetMessageContentMask	Défini en 6.3.2.2.1.

6.3.2.3 Lecteur de DataSetMessage JSON

6.3.2.3.1 NetworkMessageContentMask

Le *NetworkMessageContentMask* de *DataType JsonNetworkMessageContentMask* indique les champs d'en-tête facultatifs inclus dans les *NetworkMessages* reçus. Le *DataType JsonNetworkMessageContentMask* est défini en 6.3.2.1.1.

6.3.2.3.2 DataSetMessageContentMask

Le *DataSetMessageContentMask* de *DataType JsonDataSetMessageContentMask* indique les champs d'en-tête facultatifs inclus dans les *DataSetMessages*.

Le *DataType JsonDataSetMessageContentMask* est défini en 6.3.2.2.1.

6.3.2.3.3 Structure de JsonDataSetReaderMessageDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres *DataSetReader* spécifiques au mapping de *DataSetMessage* JSON. Il s'agit d'un sous-type du *DataSetReaderMessageDataType* défini en 6.2.8.13.3.

Le *JsonDataSetReaderMessageDataType* est défini de manière formelle dans le Tableau 65.

Tableau 65 – Structure de JsonDataSetReaderMessageDataType

Nom	Type	Description
JsonDataSetWriterMessageDataType	Structure	
networkMessageContentMask	JsonNetworkMessageContentMask	Défini en 6.3.2.3.1.
dataSetMessageContentMask	JsonDataSetMessageContentMask	Défini en 6.3.2.3.2.

6.4 Paramètres de configuration du mapping avec les protocoles de transport

6.4.1 Protocole de transport de datagramme

6.4.1.1 PubSubConnection de datagramme

6.4.1.1.1 DiscoveryAddress

Le paramètre *DiscoveryAddress* comporte les informations d'adresse réseau utilisées pour la demande de découverte et les messages de réponse. Les différents *DataTypes Structure* utilisés pour représenter l'Adresse sont définis en 6.2.6.5.3.

6.4.1.1.2 Structure de DatagramConnectionTransportDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres de configuration pour les paramètres des *PubSubConnections* spécifiques au protocole de transport de datagramme. Il s'agit d'un sous-type du *ConnectionTransportDataType* défini en 6.2.6.5.2.

Le *DatagramConnectionTransportDataType* est défini de manière formelle dans le Tableau 66.

Tableau 66 – Structure de DatagramConnectionTransportDataType

Nom	Type	Description
DatagramConnectionTransportDataType	Structure	
discoveryAddress	NetworkAddressDataType	Défini en 6.4.1.1.1. Le *NetworkAddressDataType* est défini en 6.2.6.5.3.

6.4.1.2 WriterGroup de datagramme

6.4.1.2.1 MessageRepeatCount

Le *MessageRepeatCount* de *DataType Octet* définit le nombre de répétitions de chaque *NetworkMessage*. La valeur par défaut est 0 et désactive la répétition.

6.4.1.2.2 MessageRepeatDelay

Le *MessageRepeatDelay* de *DataType Durée* définit le temps (en millisecondes) entre les répétitions du *NetworkMessage*. Un paramètre doit être ignoré si le paramètre *MessageRepeatCount* est défini sur 0.

6.4.1.2.3 Structure de DatagramWriterGroupTransportDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres de mapping de transport spécifiques au datagramme pour les *WriterGroups*. Il s'agit d'un sous-type du *WriterGroupTransportDataType* défini en 6.2.5.7.2.

Le *DatagramWriterGroupTransportDataType* est défini de manière formelle dans le Tableau 67.

Tableau 67 – Structure de DatagramWriterGroupTransportDataType

Nom	Type	Description
DatagramWriterGroupTransportDataType	Structure	
messageRepeatCount	Octet	Défini en 6.4.1.2.1.
messageRepeatDelay	Durée	Défini en 6.4.1.2.2.

6.4.1.3 Paramètres DataSetWriter de datagramme

Il n'existe aucun paramètre de mapping de transport spécifique au datagramme défini pour le *DataSetWriter*.

6.4.1.4 DataSetReader de datagramme

Il n'existe aucun paramètre de mapping de transport spécifique au datagramme défini pour le *DataSetReader*.

6.4.2 Protocole de transport de courtier

6.4.2.1 PubSubConnection de courtier

6.4.2.1.1 ResourceUri

Le paramètre *ResourceUri* de *DataType Chaîne* permet la mise en œuvre du transport afin de rechercher une clé configurée à partir de l'instance de *KeyCredentialConfigurationType* correspondante définie dans l'IEC 62541-12 à utiliser pour l'accès d'authentification au courtier au niveau de la connexion ou aux files d'attente configurées sous la connexion.

Si la valeur est nulle, l'hypothèse retenue par défaut doit être qu'il n'y a aucune authentification ou que l'authentification est anonyme, sauf si les paramètres d'authentification sont fournis sur un *WriterGroup* subordonné ou un *DataSetWriter* pour authentifier l'accès aux files d'attente individuelles.

6.4.2.1.2 AuthenticationProfileUri

Le paramètre *AuthenticationProfileUri* de *DataType Chaîne* permet de choisir le protocole d'authentification utilisé par la mise en œuvre du transport. Celui-ci est mis en correspondance vers la *Propriété ProfileUri* dans l'instance de *KeyCredentialConfigurationType* choisie par le biais des *Chaînes ResourceUri* et *AuthenticationProfileUri*.

Ce paramètre est facultatif. Si plusieurs *ProfileUri* décrivant le protocole à utiliser pour l'authentification sont configurés et que cette valeur est nulle, le transport en choisit un. Si le transport ne peut pas trouver de mécanisme d'authentification adapté dans la matrice de *ProfileUri*, le transport définit l'*État* de la *PubSubConnection* sur *Error_3*.

6.4.2.1.3 Structure de BrokerConnectionTransportDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres de mapping de transport spécifiques au Courtier pour la *PubSubConnection*. Il s'agit d'un sous-type du *ConnectionTransportDataType* défini en 6.2.6.5.2.

Le *BrokerConnectionTransportDataType* est défini de manière formelle dans le Tableau 68.

Tableau 68 – Structure de BrokerConnectionTransportDataType

Nom	Type	Description
BrokerConnectionTransportDataType	Structure	
resourceUri	Chaîne	Défini en 6.4.2.1.1.
authenticationProfileUri	Chaîne	Défini en 6.4.2.1.2.

6.4.2.2 WriterGroup de courtier

6.4.2.2.1 QueueName

Le paramètre *QueueName* de *DataType Chaîne* spécifie la file d'attente dans le *Courtier* qui reçoit les *NetworkMessages* envoyés par l'*Éditeur*. Il peut s'agir du nom d'une file d'attente ou d'une rubrique définie dans le *Courtier*.

6.4.2.2.2 ResourceUri

La propriété *ResourceUri* de *DataType Chaîne* permet la mise en œuvre du transport afin de rechercher une clé configurée à partir de l'instance de *KeyCredentialConfigurationType* correspondante définie dans l'IEC 62541-12 à utiliser pour l'accès d'authentification à la file d'attente spécifiée.

Si cette *Chaîne* n'est pas nulle, elle remplace le *ResourceUri* des paramètres d'authentification de la *PubSubConnection*.

6.4.2.2.3 AuthenticationProfileUri

Le paramètre *AuthenticationProfileUri* de *DataType Chaîne* permet de choisir le protocole d'authentification utilisé par la mise en œuvre du transport pour l'accès d'authentification à la file d'attente spécifiée.

Si cette *Chaîne* n'est pas nulle, elle remplace l'*AuthenticationProfileUri* des paramètres de transport *PubSubConnection* définis en 6.4.2.1.2.

6.4.2.2.4 RequestedDeliveryGuarantee

Le paramètre *RequestedDeliveryGuarantee* de *DataType BrokerTransportQualityOfService* spécifie la garantie de livraison qui doit s'appliquer à l'ensemble des *NetworkMessages* publiés par le *WriterGroup*, sauf spécification contraire des paramètres de transport du *DataSetWriter*. Le *DataType BrokerTransportQualityOfService* est défini en 6.4.2.2.5.

La valeur *NotSpecified_0* n'est pas admise pour le *WriterGroup*. Si la garantie de livraison choisie ne peut être appliquée, le *WriterGroup* doit définir l'état sur *Error_3*.

6.4.2.2.5 Énumération BrokerTransportQualityOfService

Le *DataType* d'Énumération *UadpDataSetMessageContentMask* est défini de manière formelle dans le Tableau 69.

Le mapping de la qualité de service avec la mise en œuvre spécifique au transport de courtier est défini en 7.3.4.5 pour AMQP et en 7.3.5.5 pour MQTT.

Tableau 69 – Valeurs de BrokerTransportQualityOfService

Valeur	Description
NotSpecified_0	La valeur n'est pas spécifiée et la valeur de l'objet parent doit être utilisée.
BestEffort_1	Le transport doit user du meilleur effort pour livrer un message. Le cas le plus défavorable porte sur la possibilité d'une perte ou d'une duplication de données.
AtLeastOnce_2	Le transport garantit que le message doit être livré au moins une fois; toutefois, une duplication reste possible. Les lecteurs doivent dédupliquer à partir de l'identificateur du message ou du numéro de séquence.
AtMostOnce_3	Le transport garantit que le message doit être envoyé une fois; toutefois, il n'est pas renvoyé en cas de perte.
ExactlyOnce_4	L'établissement d'une liaison de sécurité de transport garantit que le message doit être livré au courtier exactement une fois (ni plus, ni moins).

6.4.2.2.6 Structure de BrokerWriterGroupTransportDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres de mapping de transport spécifiques au Courtier pour les *WriterGroups*. Il s'agit d'un sous-type du *WriterGroupTransportDataType* défini en 6.2.5.7.2.

Le *BrokerWriterGroupTransportDataType* est défini de manière formelle dans le Tableau 70.

Tableau 70 – Structure de BrokerWriterGroupTransportDataType

Nom	Type	Description
BrokerWriterGroupTransportDataType	Structure	
queueName	Chaîne	Défini en 6.4.2.2.1.
resourceUri	Chaîne	Défini en 6.4.2.2.2.
authenticationProfileUri	Chaîne	Défini en 6.4.2.2.3.
requestedDeliveryGuarantee	BrokerTransportQualityOfService	Défini en 6.4.2.2.4.

6.4.2.3 DataSetWriter de courtier

6.4.2.3.1 QueueName

Le paramètre *QueueName* de *DataType Chaîne* spécifie la file d'attente dans le *Courtier* qui reçoit les *NetworkMessages* envoyés par l'*Éditeur* pour le DataSetWriter. Il peut s'agir du nom d'une file d'attente ou d'une rubrique définie dans le *Courtier*. Ce paramètre est uniquement valide si les *NetworkMessages* du *WriterGroup* comportent un seul *DataSetMessage*.

Si cette *Chaîne* n'est pas nulle, elle remplace le *QueueName* des paramètres de transport du *WriterGroup*.

6.4.2.3.2 ResourceUri

La propriété *ResourceUri* de *DataType Chaîne* permet la mise en œuvre du transport afin de rechercher une clé configurée à partir de l'instance de *KeyCredentialConfigurationType* correspondante définie dans l'IEC 62541-12 à utiliser pour l'accès d'authentification à la file d'attente spécifiée.

Si cette *Chaîne* n'est pas nulle, elle remplace le *ResourceUri* des paramètres d'authentification du *WriterGroup*.

6.4.2.3.3 AuthenticationProfileUri

Le paramètre *AuthenticationProfileUri* de *DataType Chaîne* permet de choisir le protocole d'authentification utilisé par la mise en œuvre du transport pour l'accès d'authentification à la file d'attente spécifiée.

Si cette *Chaîne* n'est pas nulle, elle remplace l'*AuthenticationProfileUri* des paramètres de transport du *WriterGroup*.

6.4.2.3.4 RequestedDeliveryGuarantee

Le paramètre *RequestedDeliveryGuarantee* de *DataType BrokerTransportQualityOfService* spécifie la garantie de livraison qui doit s'appliquer à l'ensemble des messages publiés par le *DataSetWriter*. Le *DataType BrokerTransportQualityOfService* est défini en 6.4.2.2.5.

Si la valeur n'est pas *NotSpecified*_0, elle remplace la *RequestedDeliveryGuarantee* des paramètres de transport du *WriterGroup*.

Si la garantie de livraison choisie ne peut être appliquée, le *DataSetWriter* doit définir l'état sur *Error_3*.

6.4.2.3.5 MetaDataQueueName

Pour les mapping de message comme UADP, il est nécessaire que l'*Abonné* accède aux *DataSetMetaData* pour traiter les *DataSetMessages* reçus. L'éditeur peut fournir les *DataSetMetaData* au moyen d'une file d'attente dédiée.

Le paramètre *MetaDataQueueName* de *DataType Chaîne* spécifie la file d'attente du *Courtier* qui reçoit les messages avec les *DataSetMetaData* envoyées par l'*Éditeur* pour ce *DataSetWriter*. Il peut s'agir du nom d'une file d'attente ou d'une rubrique définie dans le *Courtier*.

6.4.2.3.6 MetaDataUpdateTime

Spécifie l'intervalle (en millisecondes) avec le Type de Données Durée auquel l'éditeur doit envoyer les DataSetMetaData au MetaDataQueueName. Une valeur de 0 ou toute valeur négative doit être interprétée comme un intervalle infini.

Le transport de courtier doit publier tous les messages avec un délai d'expiration supérieur ou égal à cette valeur.

Si le temps de mise à jour est infini, un transport de courtier doit tenter de négocier la conservation du message, si possible. Dans ce cas, les *DataSetMetaData* sont uniquement envoyées si la *ConfigurationVersion* des *DataSetMetaData* correspondantes est modifiée et les *DataSetWriters* doivent tenter de négocier les garanties de livraison *AtLeastOnce*_2 ou *ExactlyOnce*_4 avec le courtier pour toutes *DataSetMetaData* envoyées afin de garantir la disponibilité des métadonnées aux lecteurs.

Les paramètres *DataSetWriterProperties* s'appliquent également aux *DataSetMetaData* envoyées à la file d'attente nommée par le paramètre *MetaDataQueueName*.

6.4.2.3.7 Structure de BrokerDataSetWriterTransportDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres de mapping de transport spécifiques au Courtier pour les *DataSetWriters*. Il s'agit d'un sous-type du *DataSetWriterTransportDataType* défini en 6.2.3.5.2.

Le *BrokerDataSetWriterTransportDataType* est défini de manière formelle dans le Tableau 71.

Tableau 71 – Structure de BrokerDataSetWriterTransportDataType

Nom	Type	Description
BrokerDataSetWriterTransportDataType	Structure	
queueName	Chaîne	Défini en 6.4.2.3.1.
resourceUri	Chaîne	Défini en 6.4.2.3.2.
authenticationProfileUri	Chaîne	Défini en 6.4.2.3.3.
requestedDeliveryGuarantee	BrokerTransportQualityOfService	Défini en 6.4.2.3.4.
metaDataQueueName	Chaîne	Défini en 6.4.2.3.5.
metaDataUpdateTime	Durée	Défini en 6.4.2.3.6.

6.4.2.4 DataSetReader de courtier

6.4.2.4.1 QueueName

Le paramètre *QueueName* de *DataType Chaîne* spécifie la file d'attente dans le *Courtier* où le *DataSetReader* peut recevoir les *NetworkMessages* avec le DataSet d'intérêt envoyé par l'*Éditeur*. Il peut s'agir du nom d'une file d'attente ou d'une rubrique définie dans le *Courtier*. Ce paramètre est uniquement valide si les *NetworkMessages* du *WriterGroup* comportent un seul *DataSetMessage*.

6.4.2.4.2 ResourceUri

La propriété *ResourceUri* de *DataType Chaîne* permet la mise en œuvre du transport afin de rechercher une clé configurée à partir de l'instance de *KeyCredentialConfigurationType* correspondante définie dans l'IEC 62541-12 à utiliser pour l'accès d'authentification à la file d'attente spécifiée.

Si cette *Chaîne* n'est pas nulle, elle remplace le *ResourceUri* des paramètres d'authentification de la *PubSubConnection*.

6.4.2.4.3 AuthenticationProfileUri

Le paramètre *AuthenticationProfileUri* de *DataType Chaîne* permet de choisir le protocole d'authentification utilisé par la mise en œuvre du transport pour l'accès d'authentification à la file d'attente spécifiée.

Si cette *Chaîne* n'est pas nulle, elle remplace l'*AuthenticationProfileUri* des paramètres de transport *PubSubConnection* définis en 6.4.2.1.2.

6.4.2.4.4 RequestedDeliveryGuarantee

Le paramètre *RequestedDeliveryGuarantee* de *DataType BrokerTransportQualityOfService* spécifie la garantie de livraison que le *DataSetReader* négocie avec le courtier pour tous les messages reçus. Le *DataType BrokerTransportQualityOfService* est défini en 6.4.2.2.5.

La valeur *NotSpecified_*0 n'est pas admise pour le *DataSetReader*. Si la garantie de livraison choisie ne peut être appliquée, le *DataSetReader* doit définir l'état sur *Error_3*.

6.4.2.4.5 MetaDataQueueName

Le paramètre *MetaDataQueueName* de *DataType Chaîne* spécifie la file d'attente du *Courtier* qui fournit les messages avec les *DataSetMetaData* envoyées par l'*Éditeur* pour le *DataSet* d'intérêt. Il peut s'agir du nom d'une file d'attente ou d'une rubrique définie dans le *Courtier*.

6.4.2.4.6 Structure de BrokerDataSetReaderTransportDataType

Ce *DataType Structure* est utilisé pour représenter les paramètres de mapping de transport spécifiques au Courtier pour les *DataSetWriters*. Il s'agit d'un sous-type du *DataSetReaderTransportDataType* défini en 6.2.8.13.2.

Le *BrokerDataSetReaderTransportDataType* est défini de manière formelle dans le Tableau 72.

Tableau 72 – Structure de BrokerDataSetReaderTransportDataType

Nom	Type	Description
BrokerDataSetReaderTransportDataType	Structure	
queueName	Chaîne	Défini en 6.4.2.4.1.
resourceUri	Chaîne	Défini en 6.4.2.4.2.
authenticationProfileUri	Chaîne	Défini en 6.4.2.4.3.
requestedDeliveryGuarantee	BrokerTransportQualityOfService	Défini en 6.4.2.4.4.
metaDataQueueName	Chaîne	Défini en 6.4.2.4.5.

7 Mappings PubSub

7.1 Généralités

L'Article 7 spécifie le mapping entre les concepts *PubSub* décrits à l'Article 5 et les paramètres de configuration *PubSub* définis à l'Article 6 pour les mapping de message concrèts et les mapping de protocole de transport pouvant être utilisés pour leur mise en œuvre.

Les mapping de *DataSetMessage*, les mapping de *NetworkMessage* et les mapping de protocole de transport sont combinés pour créer les profils de transport définis dans l'IEC 62541-7. Toutes les applications *PubSub* doivent mettre en œuvre au moins un profil de transport.

7.2 Mappings de message

7.2.1 Généralités

Les mapping de message spécifient une structure et un codage spécifiques pour les *NetworkMessages*. Une telle structure représente la charge utile des mapping avec les protocoles de transport tels UDP, MQTT ou AMQP.

Différents mapping sont définis pour différents cas d'utilisation.

7.2.2 Mapping de message UADP

7.2.2.1 Généralités

Le mapping de message UADP utilise un codage UA Binaire optimisé et assure la sécurité des messages pour PubSub OPC UA. Les mappings de protocole disponibles sont définis en 7.3.

Le mapping de message UADP définit différents champs d'en-tête facultatifs, les variations des paramètres de champ, ainsi que différents types de messages et codages de données.

Un *Éditeur* doit prendre en charge toutes les variations admises par la configuration. L'ensemble de fonctionnalités exigé est défini par les profils de l'IEC 62541-7.

Un *Abonné* doit être capable de traiter tous les *NetworkMessages* possibles et d'ignorer les informations qui ne l'intéressent pas. L'*Abonné* peut ne pas prendre en charge toutes les politiques de sécurité. Les fonctionnalités associées au traitement de différents codages de *DataSet* sont définies dans l'IEC 62541-7.

7.2.2.2 NetworkMessage

7.2.2.2.1 Généralités

L'en-tête du *NetworkMessage* UADP et les autres parties du *NetworkMessage* sont représentés à la Figure 27.

Lors de l'utilisation de la sécurité, la charge utile et le champ *Padding* sont chiffrés et, après cela, l'intégralité du *NetworkMessage* est signée lorsque la signature et le chiffrement sont actifs. Le *NetworkMessage* doit être signé sans être chiffré lorsque seule la signature est active.

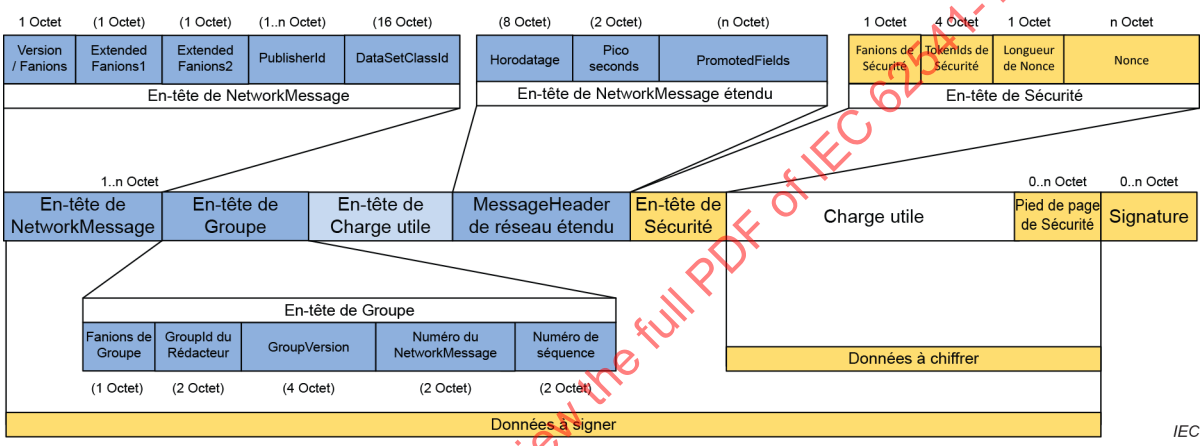


Figure 27 – NetworkMessage UADP

7.2.2.2.2 Présentation du NetworkMessage

Le codage du *NetworkMessage* UADP est spécifié dans le Tableau 73.

Le paramètre *NetworkMessageContentMask* de l'*Éditeur* contrôle les fanions des champs *UADPFlags* et *ExtendedFlags1*. Le paramètre *SecurityMode* de l'*Éditeur* contrôle le fanion de sécurité activé d'*ExtendedFlags1*. Le paramètre des fanions ne doit pas varier tant que la configuration de l'*Éditeur* ne varie pas.

Tableau 73 – NetworkMessage UADP

Nom	Type	Description
UADPVersion	Bit[0-3]	Plage de bits 0-3: Version du NetworkMessage UADP. La *UADPVersion* pour cette version de la spécification est 1.
UADPFlags	Bit[4-7]	Bit 4: *PublisherId* activé Si le PublisherId est activé, le type de PublisherId est indiqué dans le champ ExtendedFlags1. Bit 5: *GroupHeader* activé Bit 6: *PayloadHeader* activé Bit 7: *ExtendedFlags1* activé Le bit doit être "False", si ExtendedFlags1 est 0.

Nom	Type	Description
ExtendedFlags1	Octet	L'*ExtendedFlags1* doit être omis si le bit 7 des *UADPFlags* est "False". Si le champ est omis, l'*Abonné* doit gérer les bits associés comme "False". Plage de bits 0-2: Type de *PublisherId* 000 Le *PublisherId* est de *DataType Octet*. Il s'agit de la valeur par défaut si *ExtendedFlags1* est omis 001 Le *PublisherId* est de *DataType UInt16* 010 Le *PublisherId* est de *DataType UInt32* 011 Le *PublisherId* est de *DataType UInt64* 100 Le *PublisherId* est de *DataType Chaîne* 101 Réservé 11x Réservé 111 Réservé Bit 3: *DataSetClassId* activé Bit 4: Sécurité activée Si le *SecurityMode* est SIGN_1 ou SIGNANDENCRYPT_2, ce fanion est défini, la sécurité de messages est activée et le *SecurityHeader* est contenu dans l'en-tête du NetworkMessage. Si ce fanion n'est pas défini, le *SecurityHeader* est omis. Bit 5: *Horodatage* activé Bit 6: PicoSeconds activées Bit 7: *ExtendedFlags2* activé Le bit doit être "False" si ExtendedFlags2 est 0.
ExtendedFlags2	Octet	L'*ExtendedFlags2* doit être omis si le bit 7 de l'*ExtendedFlags1* est "False". Si le champ est omis, l'*Abonné* doit gérer les bits associés comme "False". Bit 0: Message-bloc défini en 7.2.2.2.4. Bit 1: *PromotedFields* activés Les champs promus peuvent uniquement être envoyés si le *NetworkMessage* comporte un seul *DataSetMessage*. Plage de bits 2-4: Type de *NetworkMessage* UADP 000 *NetworkMessage* avec charge utile de *DataSetMessage* définie en 7.2.2.2.4. Si le champ *ExtendedFlags2* n'est pas renseigné, il s'agit dutype de *NetworkMessage* par défaut. 001 *NetworkMessage* avec charge utile de demande de découverte définie en 7.2.2.3.4. 010 *NetworkMessage* avec charge utile de réponse de découverte définie en 7.2.2.4.2. 011 Réservé 1xx Réservé Bit 5: Réservé Bit 6: Réservé Bit 7: Réservé aux champs de fanion étendus ultérieurs
PublisherId	Octet[*]	Le *PublisherId* doit être omis si le bit 4 des *UADPFlags* est "False". Identificateur de l'*Éditeur* qui envoie les données. Les *DataTypes* valides sont *UInteger* et *Chaîne*. Le DataType est indiqué par les bits 0-2 de l'*ExtendedFlags1*. Un *Abonné* peut ignorer les *NetworkMessages* des *Éditeurs* desquels il n'attend aucun *NetworkMessage*.
DataSetClassId	Guid	*DataSetClassId* associé aux *DataSets* dans le NetworkMessage. Tous les DataSetMessages dans le NetworkMessage doivent avoir le même DataSetClassId. Le *DataSetClassId* doit être omis si le bit 3 de l'*ExtendedFlags1* est "False".

Nom	Type	Description
GroupHeader		L'en-tête de groupe doit être omis si le bit 5 des *UADPFlags* est "False".
GroupFlags	Octet	Bit 0: *WriterGroupId* activé Bit 1: *GroupVersion* activée Bit 2: *NetworkMessageNumber* activé Bit 3: *SequenceNumber* activé Bits 4-6: Réservé Bit 7: Réservé aux champs de fanion étendus ultérieurs
WriterGroupId	UInt16	Identificateur unique pour le *WriterGroup* dans l'Éditeur. Un *Abonné* peut ignorer les *NetworkMessages* des *WriterGroups* desquels il n'attend aucun *NetworkMessage*. Ce champ doit être omis si le bit 0 des *GroupFlags* est "False".
GroupVersion	VersionTime	Version de l'en-tête et configuration de la présentation de la charge utile des *NetworkMessages* envoyés au groupe. Ce champ doit être omis si le bit 1 des *GroupFlags* est "False".
NetworkMessage Number	UInt16	Numéro unique d'un *NetworkMessage* dans la combinaison *PublisherId-WriterGroupId* au sein d'un *PublishingInterval*. Ce numéro est nécessaire si les *DataSetMessages* d'un groupe sont décomposés en plusieurs *NetworkMessages* au sein d'un *PublishingInterval*. La valeur 0 n'est pas valide. Ce champ doit être omis si le bit 2 des *GroupFlags* est "False".
SequenceNumber	UInt16	Numéro de séquence du *NetworkMessage*. Ce champ doit être omis si le bit 3 des *GroupFlags* est "False".
PayloadHeader	Octet [*]	L'en-tête de la charge utile dépend des fanions de Type *NetworkMessage* UADP définis dans la plage de bits 2-4 d'*ExtendedFlags2*. La valeur par défaut est *DataSetMessage* si le champ *ExtendedFlags2* n'est pas activé. Le PayloadHeader doit être omis si le bit 6 des *UADPFlags* est "False". Le *PayloadHeader* n'est pas contenu dans la charge utile; toutefois, il est contenu dans l'en-tête du *NetworkMessage* non chiffré, puisqu'il comporte des informations nécessaires pour filtrer les *DataSetMessages* côté *Abonné*.
Horodatage	DateTime	Moment où le NetworkMessage a été créé. L'*Horodatage* doit être omis si le bit 5 d'*ExtendedFlags1* est "False". Le *PublishingInterval*, le *SamplingOffset*, le *PublishingOffset*, l'*Horodatage* et les *PicoSeconds* de l'en-tête du *NetworkMessage* doivent utiliser la même base temporelle.
PicoSeconds	UInt16	Spécifie le nombre d'intervalles de 10 picosecondes ($1,0 \times 10^{-11}$ seconde) qui doit être ajouté à l'*Horodatage*. Les *PicoSeconds* doivent être omises si le bit 6 d'*ExtendedFlags1* est "False".
PromotedFields		Les *PromotedFields* doivent être omis si le bit 1 de l'*ExtendedFlags2* est "False". Si les *PromotedFields* sont fournis, le nombre de *DataSetMessages* dans le *Network Message* doit être un.
Taille	UInt16	Taille totale en *Octets* des *Champs* contenus dans les *PromotedFields*.
Champs	BaseDataType[ ]	Matrice des champs promus. La taille, l'ordre et les *DataTypes* des champs dépendent des paramètres des *FieldMetaData* des *DataSetMetaData* associées au *DataSetMessage* contenu dans le *NetworkMessage*.

Nom	Type	Description
SecurityHeader		L'en-tête de sécurité doit être omis si le bit 4 de l'*ExtendedFlags1* est "False".
SecurityFlags	Octet	Bit 0: *NetworkMessage* signé Bit 1: *NetworkMessage* chiffré Bit 2: SecurityFooter activé Bit 3: Réinitialisation forcée de la clé Ce bit est défini si toutes les clés sont invalidées. Il est défini jusqu'à ce qu'une nouvelle clé soit utilisée. L'éditeur a besoin de fournir aux abonnés un délai raisonnable pour demander de nouvelles clés. Le délai minimal est de cinq fois le *KeepAliveTime* configuré pour le groupe PubSub correspondant. Ce fanion est généralement défini si toutes les clés sont invalidées pour exclure les Abonnés, qui n'ont plus accès aux clés. Plage de bits 4-7: Réservé
SecurityTokenId	IntegerId	Identificateur du jeton de sécurité qui identifie la clé de sécurité d'un *SecurityGroup*. La relation avec le *SecurityGroup* s'effectue au moyen des *DataSetWriterIds* contenus dans le *NetworkMessage*.
NonceLength	Octet	Longueur du nonce utilisé pour initialiser l'algorithme de chiffrement.
MessageNonce	Octet [NonceLength]	Numéro utilisé exactement une fois pour une clé de sécurité donnée. Pour une clé de sécurité donnée, un nonce unique doit être généré pour chaque *NetworkMessage*. Les règles de construction du *MessageNonce* sont définies pour la Sécurité de Messages UADP en 7.2.2.2.3.
SecurityFooterSize	UInt16	Taille du *SecurityFooter*. La taille du pied de page de sécurité doit être omise si le bit 2 des *UADPFlags* est "False".
Payload	Octet [*]	La charge utile dépend des fanions de Type *NetworkMessage* UADP définis dans la plage de bits 2-4 d'*ExtendedFlags2*.
SecurityFooter	Octet [*]	Le pied de page de sécurité facultatif doit être omis si le bit 2 des *UADPFlags* est "False". Le contenu du pied de page de sécurité est défini par la *SecurityPolicy*.
Signature	Octet [*]	Signature du *NetworkMessage*.

7.2.2.2.3 Sécurité de messages UADP

7.2.2.2.3.1 Généralités

L'algorithme et la longueur du nonce utilisés pour la sécurité du *NetworkMessage* UADP dépendent de la *SecurityPolicy* choisie. Ils sont définis par le *SymmetricPubSubEncryptionAlgorithm* et la *SymmetricPubSubNonceLength*.

Les clés utilisées pour chiffrer et signer les messages sont renvoyées par la méthode *GetSecurityKeys* (voir 8.4). Cette *Méthode* renvoie une séquence de données aléatoires avec une longueur qui dépend du *SecurityPolicyUri*, lequel est également renvoyé par la *Méthode*. La présentation des données aléatoires est définie dans le Tableau 74.

Tableau 74 – Présentation des données clés pour la sécurité des messages UADP

Nom	Type	Description
SigningKey	Octet [Longueur de Clé du SymmetricSignatureAlgorithm]	Partie signature de la clé pour les données de clé renvoyées par la méthode *GetSecurityKeys*. Le SymmetricSignatureAlgorithm est défini dans la SecurityPolicy.
EncryptingKey	Octet [KeyLength du SymmetricEncryptionAlgorithm]	Partie chiffrement de la clé pour les données de clé renvoyées par la méthode *GetSecurityKeys*. Le SymmetricEncryptionAlgorithm est défini dans la SecurityPolicy.
KeyNonce	Octet [SymmetricPubSubNonceLength]	Partie nonce des données de clé renvoyées par la méthode *GetSecurityKeys*.

7.2.2.2.3.2 AES-CTR

La présentation du MessageNonce pour le mode AES-CTR est définie dans le Tableau 75.

Tableau 75 – Présentation du MessageNonce pour le mode AES-CTR

Nom	Type	Description
Aléatoire	Octet [4]	Partie aléatoire du MessageNonce. Il n'est pas nécessaire qu'il s'agisse d'un numéro aléatoire par chiffrement; il peut s'agir d'un numéro pseudo-aléatoire.
SequenceNumber	UInt32	Numéro de séquence à croissance monolithique stricte affecté par l'éditeur à chaque *NetworkMessage* envoyé pour une combinaison *SecurityTokenId-PublisherId*. Le numéro de séquence est réinitialisé à 1 après la mise à jour de la clé et du *SecurityTokenId* dans l'*Éditeur*. Il convient qu'un récepteur ignore les *NetworkMessages* plus anciens que la dernière séquence traitée s'il ne gère pas le reclassement des *NetworkMessages*. Les récepteurs ont besoin de connaître les limites des numéros de séquence (passage de 4 294 967 295 à 0). Pour déterminer si un *NetworkMessage* reçu est plus récent que le dernier *NetworkMessage* traité, la formule suivante doit être utilisée: (4 294 967 295 + numéro de séquence reçu – dernier numéro de séquence traité) modulo 4 294 967 296. Les résultats inférieurs à 1 073 741 824 indiquent que le *NetworkMessage* reçu est plus récent que le dernier *NetworkMessage* traité, et le *NetworkMessage* reçu est traité. Les résultats supérieurs à 3 221 225 472 indiquent que le message reçu est plus ancien (ou aussi ancien) que le dernier *NetworkMessage* traité et il convient d'ignorer le *NetworkMessage* reçu si le reclassement des *NetworkMessages* n'est pas nécessaire. Les autres résultats ne sont pas valides et les *NetworkMessages* doivent être ignorés. Il convient de déterminer la durée de vie de la clé de sorte que la nouvelle clé soit utilisée avant que le *SequenceNumber* ne repasse à 0. Les abonnés doivent réinitialiser les enregistrements qu'ils conservent pour leurs numéros de séquence lorsqu'ils ne reçoivent pas de messages pendant une durée correspondant à deux fois le délai d'entretien afin de traiter les Éditeurs qui sont hors service et n'ont pas pu poursuivre à partir du dernier *SequenceNumber* utilisé.

Le chiffrement et le déchiffrement des messages en mode AES-CTR utilisent une clé secrète et un bloc de compteur. La clé secrète est l'*EncryptingKey* des données de clé définies dans le Tableau 74. La présentation et le contenu du bloc de compteur sont définis dans le Tableau 76.

Tableau 76 – Présentation du bloc de compteur pour la sécurité des messages UADP

Nom	Type	Description
KeyNonce	Octet [4]	Partie KeyNonce des données de clé renvoyées par la méthode *GetSecurityKeys*.
MessageNonce	Octet [8]	Les 8 premiers octets du *Nonce* dans le *SecurityHeader* du *NetworkMessage*. Pour le mode AES-CTR, la longueur du *Nonce SecurityHeader* doit être de 8 octets.
BlockCounter	Octet [4]	Compteur pour chaque bloc chiffré du NetworkMessage. Le compteur est un entier gros-boutiste de 32 bits (opposé du codage normal pour les valeurs UInt32 dans OPC UA. Cette convention provient du RFC AES-CTR). Le compteur commence par 0 au niveau du premier bloc. Le compteur est incrémenté de 1 pour chaque bloc.

Le mode AES-CTR prend le bloc de compteur et le chiffre à l'aide de la clé de chiffrement. Le flux de la clé chiffrée est ensuite logiquement soumis à la fonction XOR avec les données à chiffrer ou déchiffrer. Le processus est répété pour chaque bloc en texte brut. Aucun remplissage n'est ajouté à la fin du texte brut. Le mode AES-CTR ne modifie pas la taille des données de texte brut et peut être directement appliqué à un tampon mémoire contenant le message.

La signature est calculée sur l'intégralité du *NetworkMessage*, y compris toutes données chiffrées. L'algorithme de signature est spécifié par le *SecurityPolicyUri* dans l'IEC 62541-7.

Lorsqu'un abonné reçoit un *NetworkMessage*, il doit au préalable vérifier la signature. Si la vérification échoue, il abandonne le *NetworkMessage*.

Une autre *SecurityPolicy* peut spécifier différentes longueurs de clé ou algorithmes de cryptographie.

7.2.2.2.4 NetworkMessage-bloc UADP

S'il est essentiel que la charge utile d'un *NetworkMessage* comme un *DataSetMessage* ou un message de réponse de découverte soit décomposée entre plusieurs *NetworkMessages*, les blocs sont envoyés avec l'en-tête de la charge utile définie dans le Tableau 77 et la charge utile définie dans le Tableau 78. Un bloc *NetworkMessage* peut uniquement contenir une charge utile regroupée sur un *DataSetMessage*.

Tableau 77 – En-tête de la charge utile d'un NetworkMessage-bloc

Nom	Type	Description
DataSetWriterId	UInt16	DataSetWriterId contenu dans le *NetworkMessage*. Le *DataSetWriterId* identifie le *PublishedDataSet* et le *DataSetWriter* responsable de l'envoi de Messages pour le *DataSet*. Un *Abonné* peut ignorer les *DataSetMessages* des *DataSetWriters* desquels il n'attend aucun *DataSetMessages*. Le *DataSetWriterId* doit être défini sur 0 pour les messages de réponse de découverte.

Tableau 78 – Champs de la charge utile d'un NetworkMessage-bloc

Nom	Type	Description
MessageSequenceNumber	UInt16	Numéro de séquence de la charge utile défini pour le type de *NetworkMessage* tel que *DataSetMessageSequenceNumber* dans un *DataSetMessage*. Les *NetworkMessages* peuvent être reçus non classés. Dans ce cas, un bloc de la charge utile suivante peut être reçu avant de recevoir le dernier bloc de la charge utile précédente. Si le numéro de séquence suivant est reçu par un *Abonné* pouvant gérer seulement une charge utile, les blocs de la charge utile précédente sont ignorés s'ils n'ont pas été reçus entièrement.
ChunkOffset	UInt32	Position du décalage d'octet du bloc dans la charge utile complète du *NetworkMessage*. Le dernier bloc est détecté si l'addition *ChunkOffset* + taille du bloc actuel équivaut à la *TotalSize*. Tous les blocs, à l'exception du dernier, doivent avoir la même taille. La taille de tous les blocs autres que le dernier peut être utilisée pour calculer le nombre de blocs attendus. La charge utile réassemblée du *NetworkMessage* peut être traitée après réception de tous les blocs. Selon le mapping avec les protocoles de transport, les blocs peuvent être reçus non classés et le dernier bloc peut être reçu avant tous les autres.
TotalSize	UInt32	Taille totale de la charge utile du *NetworkMessage* en octets.
ChunkData	ByteString	Les morceaux du *DataSetMessage* original sont copiés dans le bloc jusqu'à ce que la taille maximale admise pour un seul *NetworkMessage* soit atteinte, moins l'espace pour la signature. Les données copiées dans le bloc suivant commencent par l'octet suivant le dernier octet copié dans le bloc actuel. Un *DataSetMessage* est entièrement reçu lorsque tous les blocs sont reçus et que le *DataSetMessage* peut être complètement traité.

7.2.2.3 DataSetMessage

7.2.2.3.1 Généralités

L'en-tête de la charge utile du *DataSet* UADP et les autres parties du *NetworkMessage* sont représentés à la Figure 28.

Différents types de *DataSetMessages* peuvent être combinés dans un *NetworkMessage*.

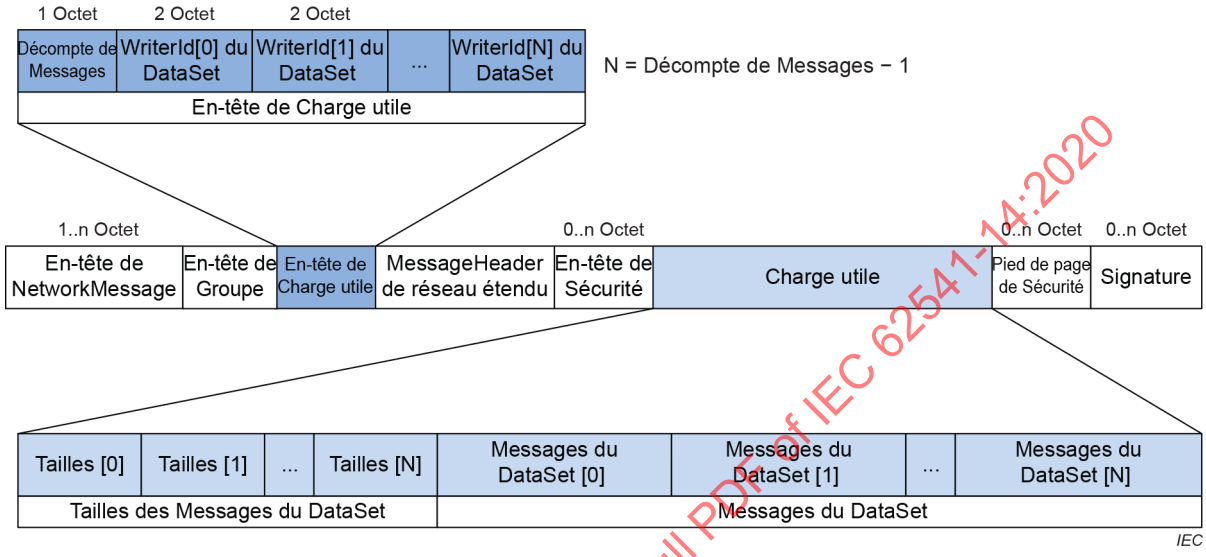


Figure 28 – Charge utile d'un DataSet UADP

7.2.2.3.2 En-tête de la charge utile d'un DataSet

Le codage de la charge utile d'un *DataSet* UADP est spécifié dans le Tableau 79. L'en-tête de la charge utile est non chiffré. Cet en-tête doit être omis si le bit 6 des *UADPFlags* est "False".

Tableau 79 – En-tête de la charge utile d'un DataSet UADP

Nom	Type	Description
Décompte	Octet	Nombre de *DataSetMessages* contenus dans le *NetworkMessage*. Le *NetworkMessage* doit contenir au moins un *DataSetMessage* si le type de *NetworkMessage* est la charge utile du *DataSetMessage*.
DataSetWriterIds	UInt16 [Décompte]	Liste des *DataSetWriterIds* contenus dans le *NetworkMessage*. La taille de la liste est définie par le *Décompte*. Le *DataSetWriterId* identifie le *PublishedDataSet* et le *DataSetWriter* responsable de l'envoi de Messages pour le *DataSet*. Un *Abonné* peut ignorer les *DataSetMessages* des *DataSetWriters* desquels il n'attend aucun *DataSetMessages*.

7.2.2.3.3 Charge utile d'un DataSet

La charge utile d'un *DataSet* est définie dans le Tableau 80. La charge utile est chiffrée.

Tableau 80 – Charge utile d'un DataSet UADP

Nom	Type	Description
Tailles	UInt16 [Décompte]	Liste des tailles (en octets) du *DataSetMessage*. La taille de la liste est définie par le *Décompte* dans l'en-tête de la charge utile du *DataSet*. Si la taille de la charge utile dépasse 65 535, le *DataSetMessage* doit être affecté à des *NetworkMessages* distincts. Si un seul *DataSetMessage* dépasse la taille de la charge utile, il doit être décomposé en *Blocs de NetworkMessages*. Ce champ doit être omis si le décompte est un ou si le bit 6 des *UADPFlags* est "False".
DataSetMessages	DataSetMessage [Décompte]	*DataSetMessages* contenus dans le *NetworkMessage*. La taille de la liste est définie par le *Décompte* dans l'en-tête de la charge utile du *DataSet*. Le type de codage utilisé pour les *DataSetMessages* est défini par le *DataSetWriter*. Les codages du *DataSetMessage* sont définis en 7.2.2.3.4.

7.2.2.3.4 En-tête de DataSetMessage

La structure de l'en-tête d'un *DataSetMessage* et la relation avec les autres parties d'un *NetworkMessage* sont représentées à la Figure 29.

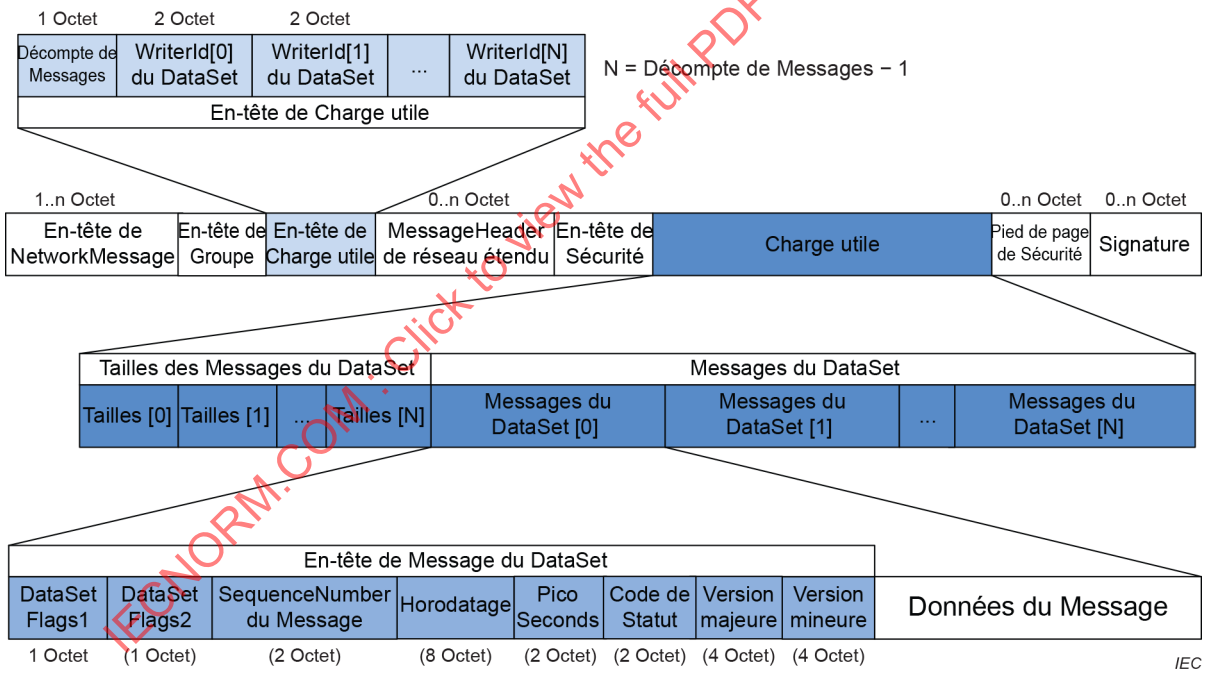


Figure 29 – Structure de l'en-tête d'un DataSetMessage

Le codage de la structure de l'en-tête d'un *DataSetMessage* est spécifié dans le Tableau 81.

Les paramètres *DataSetFieldContentMask* et *DataSetMessageContentMask* du *DataSetWriter* contrôlent les fanions des champs *DataSetFlags1* et *DataSetFlags2*. Le paramètre des fanions ne doit pas varier tant que la configuration du *DataSetWriter* ne varie pas.

Tableau 81 – Structure de l'en-tête d'un DataSetMessage

Nom	Type	Description
DataSetFlags1	Octet	Bit 0: Le DataSetMessage est valide. Si ce bit est défini sur "False", le reste du DataSetMessage est jugé non valide et ne doit pas être traité par l'*Abonné*. Plage de bits 1-2: Codage du champ 00 Les champs du *DataSet* sont codés en Variante Une *Variante* peut contenir un *StatusCode* à la place du *DataType* attendu si le statut du champ est "Bad". Une *Variante* peut contenir une DataValue avec la valeur et le statusCode si le statut du champ est "Uncertain". 01 Codage du champ RawData Le codage du champ RawData est défini en 7.2.2.3.9. 10 Codage du champ DataValue Les champs du *DataSet* sont codés en DataValue. Cette option est définie si le *DataSet* est configuré pour envoyer d'autres éléments en plus de la Valeur. 11 Réservé Bit 3: *DataSetMessageSequenceNumber* activé Bit 4: *Statut* activé Bit 5: *ConfigurationVersionMajorVersion* activée Bit 6: *ConfigurationVersionMinorVersion* activée Bit 7: *DataSetFlags2* activé Le bit doit être "False", si DataSetFlags2 est 0.
DataSetFlags2	Octet	Le *DataSetFlags2* doit être omis si le bit 7 du *DataSetFlags1* est "False". Si le champ est omis, l'*Abonné* doit gérer les bits associés comme "False". Plage de bits 0-3: Type de *DataSetMessage* UADP 0000 Image clé de données (voir 7.2.2.3.5) Si le champ *DataSetFlags2* n'est pas renseigné, il s'agit du type de *DataSetMessage* par défaut. 0001 Image delta de données (voir 7.2.2.3.6) 0010 Événement (voir 7.2.2.3.7) 0011 Entretien (voir 7.2.2.3.8) 01xx Réservé 1xxx Réservé Bit 4: *Horodatage* activé Bit 5: *PicoSeconds* incluses dans l'en-tête du *DataSetMessage* Bit 6: Réservé Bit 7: Réservé aux champs de fanion étendus ultérieurs

Nom	Type	Description
DataSetMessage SequenceNumber	UInt16	Numéro de séquence à croissance monolithique stricte affecté par l'éditeur à chaque *DataSetMessage* envoyé. Il convient qu'un récepteur ignore les *DataSetMessage* plus anciens que la dernière séquence traitée s'il ne gère pas le reclassement des *DataSetMessages*. Les récepteurs ont besoin de connaître les limites des numéros de séquence (passage de 65 535 à 0). Pour déterminer si un *DataSetMessage* reçu est plus récent que le dernier *DataSetMessage* traité, la formule suivante doit être utilisée: (65 535 + numéro de séquence reçu – dernier numéro de séquence traité) modulo 65 536. Les résultats inférieurs à 16 384 indiquent que le *DataSetMessage* reçu est plus récent que le dernier *DataSetMessage* traité, et le *DataSetMessage* reçu est traité. Les résultats supérieurs à 49 162 indiquent que le message reçu est plus ancien (ou aussi ancien) que le dernier *DataSetMessage* traité et il convient d'ignorer le *DataSetMessage* reçu si le reclassement des *DataSetMessage* n'est pas nécessaire. Les autres résultats ne sont pas valides et les *DataSetMessages* doivent être ignorés. Le champ doit être omis si le bit 3 du *DataSetFlags1* est "False".
Horodatage	UtcTime	Moment où les données ont été collectées. L'*Horodatage* doit être omis si le bit 4 de *DataSetFlags1* est "False".
PicoSeconds	UInt16	Spécifie le nombre d'intervalles de 10 picosecondes ($1,0 \times 10^{-11}$ seconde) qui doit être ajouté à l'*Horodatage*. Le champ doit être omis si le bit 5 du *DataSetFlags2* est "False".
Statut	UInt16	Statut global du DataSet. Il s'agit de l'entier de poids fort 16 bits du *DataType StatusCode* qui représente la valeur numérique de la *Sévérité* et le *SubCode* du *DataType StatusCode*. Le champ doit être omis si le bit 4 du *DataSetFlags1* est "False".
ConfigurationVersion MajorVersion	VersionTime	Version majeure de la version de configuration du DataSet utilisé comme vérification de cohérence avec les *DataSetMetaData* disponibles côté *Abonné*. Le champ doit être omis si le bit 5 du *DataSetFlags1* est "False".
ConfigurationVersion MinorVersion	VersionTime	Version mineure de la version de configuration du DataSet utilisé comme vérification de cohérence avec les *DataSetMetaData* disponibles côté *Abonné*. Le champ doit être omis si le bit 6 du *DataSetFlags1* est "False".

7.2.2.3.5 DataSetMessage de type image clé de données

Les données du *DataSetMessage* de type image clé de données et les en-têtes associés sont représentés à la Figure 30.

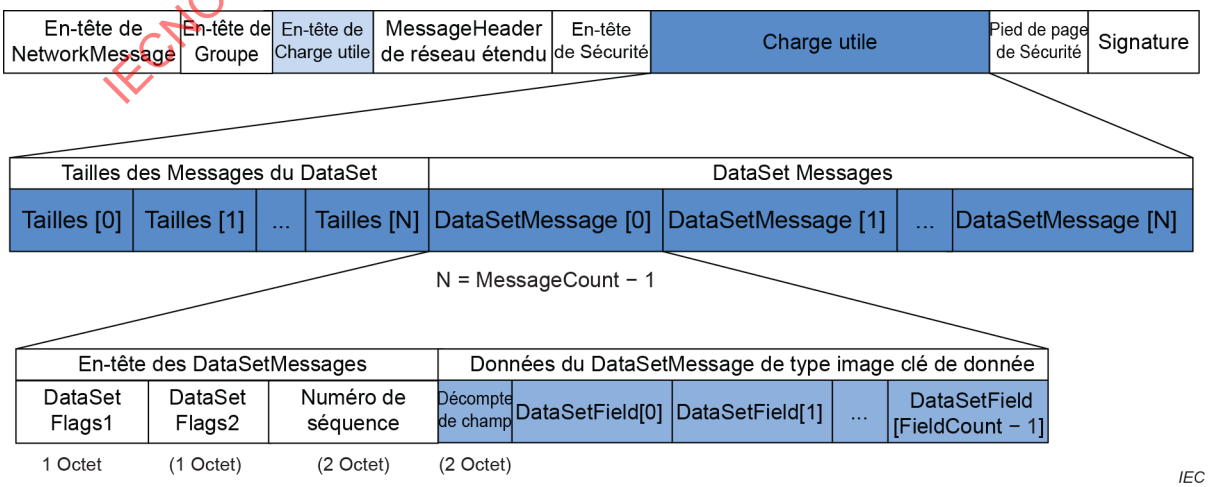


Figure 30 – Données de DataSetMessage de type image clé de données

Le codage de la structure du *DataSetMessage* de type image clé de données est spécifié dans le Tableau 82.

Tableau 82 – Structure de DataSetMessage de type image clé de données

Nom	Type	Description
FieldCount	UInt16	Nombre de champs du *DataSet* contenu dans le *DataSetMessage*. Le *FieldCount* doit être omis si le codage du champ *RawData* est défini dans les *EncodingFlags* spécifiés en 7.2.2.3.4.
DataSetFields	BaseDataType	Valeurs de champ du DataSet. Le codage du champ dépend des *EncodingFlags* de l'en-tête du *DataSetMessage*, définis en 7.2.2.3.4. Le codage par défaut est une *Variante* si les bits 1 et 2 ne sont pas définis.

7.2.2.3.6 DataSetMessage de type image delta de données

Les données du *DataSetMessage* de type image delta de données et les en-têtes associés sont représentés à la Figure 31.

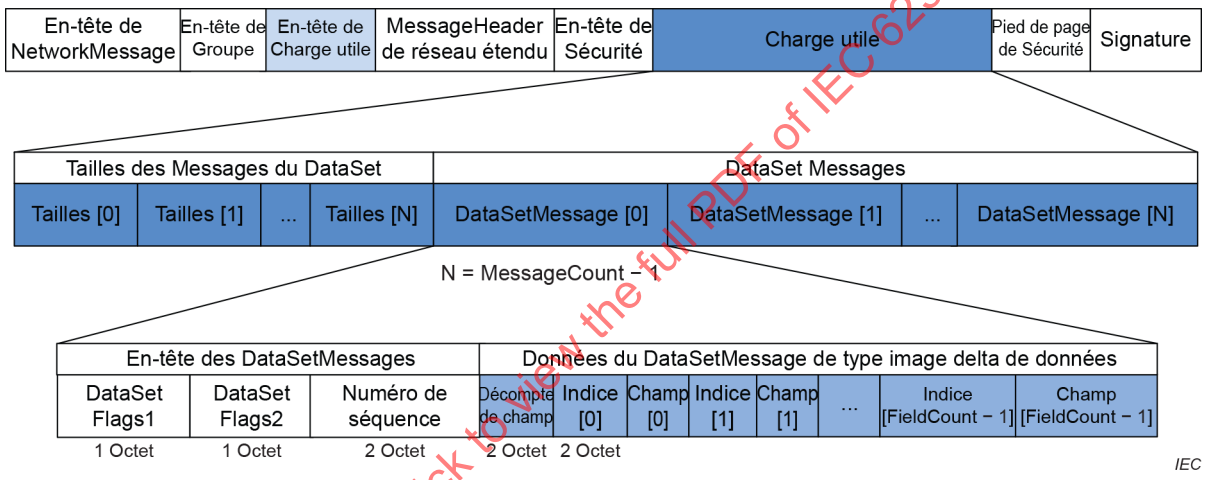


Figure 31 – DataSetMessage de type image delta de données

Les informations sur une valeur isolée des messages de type image delta sont plus importantes en raison de l'indice supplémentaire nécessaire à l'envoi exclusif des données modifiées. L'*Éditeur* doit envoyer un message de type image clé si le message de type image delta est plus grand qu'un message de type image clé.

Le codage de la structure du *DataSetMessage* de type image delta de données est spécifié dans le Tableau 83.

Tableau 83 – Structure de DataSetMessage de type image delta de données

Nom	Type	Description
FieldCount	UInt16	Nombre de champs du DataSet contenu dans le *DataSetMessage*.
DeltaFrameFields	Structure	Sous-ensemble des valeurs de champ du DataSet contenues dans l'image delta.
FieldIndex	UInt16	Indice du Champ du DataSet. L'indice repose sur la position du champ dans les *DataSetMetaData* avec la version de configuration définie dans le champ *ConfigurationVersion*.
FieldValue	BaseDataType	Valeurs de champ du DataSet. Le codage du champ dépend des EncodingFlags de l'en-tête du *DataSetMessage*, définis en 7.2.2.3.4. Le codage par défaut est une variante si les bits 1 et 2 ne sont pas définis.

7.2.2.3.7 *DataSetMessage* d'Événement

Les données du *DataSetMessage d'Événement* et les en-têtes associés sont représentés à la Figure 32.

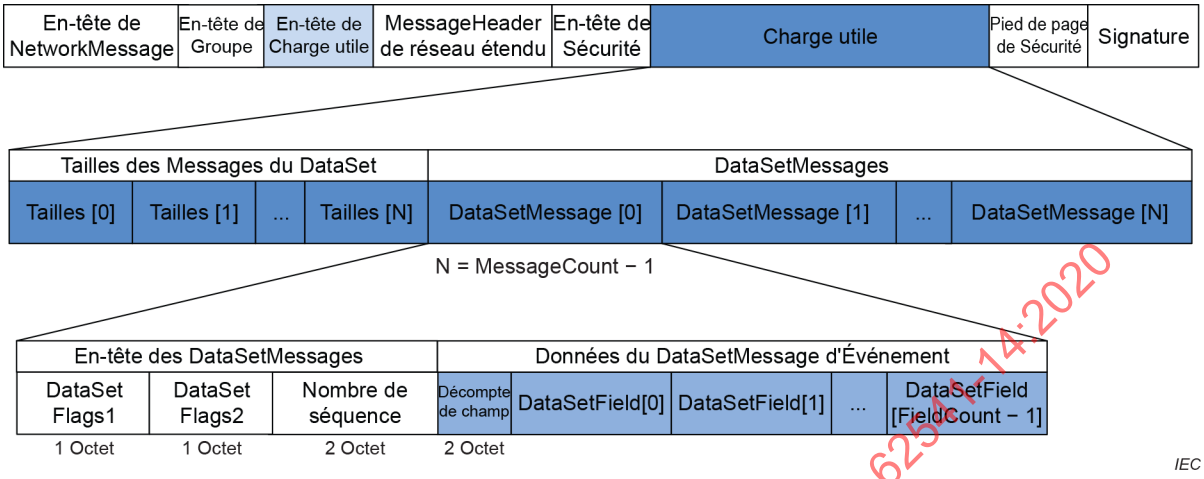


Figure 32 – DataSetMessage d'événement

Le codage de la structure d'un *DataSetMessage d'Événement* est spécifié dans le Tableau 84.

Tableau 84 – Structure de DataSetMessage d'événement

Nom	Type	Description
FieldCount	UInt16	Nombre de champs du *DataSet* contenu dans le *DataSetMessage*.
DataSetFields	BaseDataType	Valeurs de champ du DataSet. Les champs du *DataSetMessage* d'Événement doivent être codés en Variante. Le Codage du Champ *DataSetFlags1* de l'en-tête du *DataSetMessage* (bit 1 et 2) spécifié en 7.2.2.3.4 doit être défini sur "False".

7.2.2.3.8 Message KeepAlive

Le message d'entretien n'ajoute aucun champ supplémentaire. Le message et les en-têtes associés sont représentés à la Figure 33.

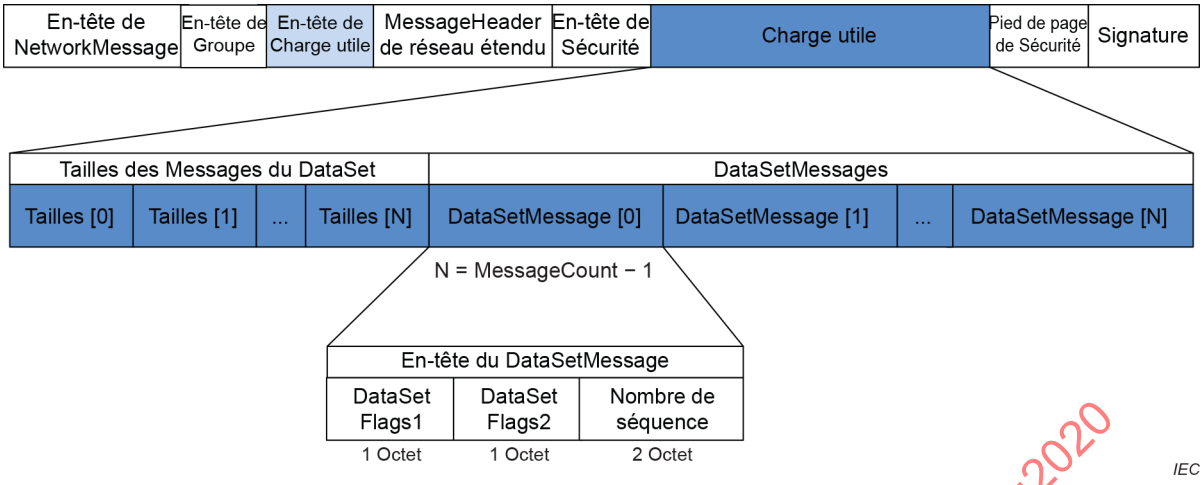


Figure 33 – Message KeepAlive

Le numéro de séquence comporte le prochain numéro de séquence attendu pour le *DataSetWriter*.

7.2.2.3.9 Codage du champ RawData

Le codage des champs *DataSetMessage* est géré comme un *DataType Structure* où les champs *DataSet* sont gérés comme des champs *Structure* et les champs de *DataType Structure* sont gérés comme des structures imbriquées.

L'ensemble des restrictions de codage des *DataTypes Structure* s'applique également au *Codage du Champ RawData*.

Un champ *DataSet* est codé dans le *DataType* et le *ValueRank* spécifiés dans les *DataSetMetaData* pour le *DataSet*. La gestion spéciale suivante doit être appliquée:

- si le *DataType* d'un champ *DataSet* ou d'un champ *Structure* est *Chaîne* ou *ByteString* et que la taille réelle est inférieure à la taille maximale possible indiquée par les dimensions, le champ doit être complété avec des octets dont la valeur est zéro;
- si le *ValueRank* est *OneDimension* (1) ou n > 1 et que la taille réelle est inférieure à la taille maximale possible indiquée par les dimensions, le champ doit être complété avec des octets dont la valeur est zéro.

Les restrictions suivantes s'appliquent au codage du champ *RawData*:

- les dimensions des champs doivent être définies si le *DataType* est *Chaîne* ou *ByteString* ou s'il s'agit d'une matrice. Les champs *Structure* comportant ces *DataTypes* ou *ValueRanks* sont inclus;
- les champs *DataSet* et *Structure* doivent avoir un valueRank concret de valeurs −1 ou n > 0;
- les champs *DataSet* et *Structure* ne doivent pas avoir le *BuiltInType NodeId*, *ExpandedNodeId*, *QualifiedName*, *LocalizedText*, *XmlElement* ou *DataValue*;
- les champs *Structure* ne doivent pas avoir le *builtInType ExtensionObject* ou *Variante*.

Le bit 0 valide du *DataSetMessage* dans *DataSetFlags1* doit être défini sur "False" si les champs ne remplissent pas ces exigences au moment de la création du *DataSetMessage*.

7.2.2.4 Messages de découverte

7.2.2.4.1 NetworkMessage de demande de découverte UADP

7.2.2.4.1.1 Généralités

Les fanions NetworkMessage utilisés avec les messages de demande de découverte doivent utiliser les valeurs de bit suivantes:

- *UADPFlags*: les bits 5 et 6 doivent être "False", les bits 4 et 7 doivent être "True";
- *ExtendedFlags1*: les bits 3, 5 et 6 doivent être "False", les bits 4 et 7 doivent être "True";
- *ExtendedFlags2*: le bit 2 doit être "True", tous les autres bits doivent être "False".

La définition de ces fanions garantit une valeur connue pour les cinq premiers champs du *NetworkMessage* sur l'*Éditeur* comme récepteur. Les paramètres de sécurité réels pour le *NetworkMessage* sont indiqués par le *SecurityHeader*.

7.2.2.4.1.2 Réduction du trafic

Différentes règles sont utilisées pour réduire le volume du trafic sur le réseau en cas de communication en diffusion ou multidiffusion.

Il convient qu'un *Abonné* mette en cache les informations de configuration pour le *PublisherId* et les *DataSetWriterIds* d'intérêt.

Si un *Abonné* exige des informations aux *Éditeurs* après avoir détecté un démarrage ou un changement de version, les demandes de découverte doivent être retardées de manière aléatoire dans une plage de 100 ms à 500 ms. La demande doit être ignorée si les informations ont déjà été reçues pendant ce temps ou qu'un autre *Abonné* a déjà envoyé une demande et que la réponse à celle-ci est utilisée.

Les demandes de découverte pour plusieurs *DataSetWriters* dans un *WriterGroup* doivent être agrégées dans une réponse de découverte.

Un *Éditeur* doit retarder les réponses ultérieures pour une combinaison type de demande-identificateur semblable au *DataSetWriterId* pendant au moins 500 ms. Les demandes de duplication auxquelles il n'a pas encore été apporté de réponse doivent être éliminées par l'*Éditeur*.

L'*Abonné* doit attendre une réponse pendant au moins 500 ms. Tant que toutes les réponses n'ont pas été reçues, l'Abonné demande les informations manquantes. Il doit doubler le délai entre deux demandes successives jusqu'à ce que toutes les réponses nécessaires soient reçues ou rejetées.

7.2.2.4.1.3 En-tête de demande de découverte

Le codage de la structure de l'en-tête d'une demande de découverte est spécifié dans le Tableau 85.

Tableau 85 – Structure de l'en-tête d'une demande de découverte

Nom	Type	Description
RequestType	Octet	Les types de messages de demande de découverte suivants sont définis. 0 Réservé 1 Message de demande d'informations de l'*Éditeur* (voir 7.2.2.4.1.4)

7.2.2.4.1.4 Message de demande d'informations de l'Éditeur

Le codage de la structure du message de demande d'informations de l'*Éditeur* est spécifié dans le Tableau 86.

Tableau 86 – Structure de message de demande d'informations d'éditeur

Nom	Type	Description
InformationType	Octet	Les types de demandes d'informations de l'Éditeur suivants sont définis. 0 Réservé 1 *Points d'extrémité Serveur de l'Éditeur* 2 *DataSetMetaData* 3 Configuration de *DataSetWriter*
DataSetWriterIds	UInt16	Liste des *DataSetWriterIds* pour lesquels les informations sont demandées. Si la demande n'est pas associée au DataSet, la matrice doit être nulle.

7.2.2.4.2 NetworkMessage de réponse de découverte UADP

7.2.2.4.2.1 Généralités

Les fanions NetworkMessage utilisés avec les messages de réponse de découverte doivent utiliser les valeurs de bit suivantes:

- *UADPFlags*: les bits 5 et 6 doivent être "False", les bits 4 et 7 doivent être "True";
- *ExtendedFlags1*: les bits 3, 5 et 6 doivent être "False", le bit 7 doit être "True";
- *ExtendedFlags2*; le bit 1 doit être "False" et le type de *NetworkMessage* doit être réponse de découverte.

La définition de ces fanions garantit une valeur connue pour les cinq premiers champs du *NetworkMessage* pour les *Éditeurs* attendus par l'*Abonné*. Les paramètres de sécurité réels pour le *NetworkMessage* sont indiqués par le *SecurityHeader*.

7.2.2.4.2.2 En-tête de réponse de découverte

Le codage de la structure de l'en-tête d'une réponse de découverte est spécifié dans le Tableau 87.

Tableau 87 – Structure de l'en-tête d'une réponse de découverte

Nom	Type	Description
ResponseType	Octet	Les types de messages de réponse de découverte suivants sont définis. 0 Réservé 1 Message de point d'extrémité de l'Éditeur (voir 7.2.2.4.2.3) 2 Message de métadonnées du DataSet (voir 7.2.2.4.2.4) 3 Message de configuration de DataSetWriter (voir 7.2.2.4.2.5)
SequenceNumber	UInt16	Numéro de séquence à croissance monolithique stricte affecté à chaque réponse de découverte envoyée dans le domaine d'application d'un *PublisherId*.

7.2.2.4.2.3 Message de point d'extrémité de l'Éditeur

Le codage des *Points d'extrémité* disponibles d'un *Éditeur* est spécifié dans le Tableau 88.

Tableau 88 – Structure de message de point d'extrémité d'éditeur

Nom	Type	Description
Endpoints	EndpointDescription	*Points d'extrémité* du *Serveur* OPC UA de l'*Éditeur*. L'*EndpointDescription* est définie dans l'IEC 62541-4.
statusCode	StatusCode	Code de statut indiquant la capacité de l'*Éditeur* à fournir des *Points d'extrémité*.

7.2.2.4.2.4 Message DataSetMetaData

Le codage de la structure du message de métadonnées du *DataSet* est spécifié dans le Tableau 89. Il comporte la présentation actuelle et les *DataSetMetaData* du *DataSet*.

La *ConfigurationVersion* dans l'en-tête du *DataSetMessage* doit correspondre à la *ConfigurationVersion* des *DataSetMetaData*.

L'*Éditeur* doit envoyer ce message sans demande de découverte correspondante si les *DataSetMetaData* du *DataSet* ont été modifiées.

Tableau 89 – Structure de message DataSetMetaData

Nom	Type	Description
DataSetWriterId	UInt16	*DataSetWriterId* du *DataSet* décrit avec les *MetaData*.
MetaData	DataSetMetaDataType	Métadonnées du *DataSet* actuel pour le *DataSet* associé au *DataSetWriterId*. Le *DataSetMetaDataType* est défini en 6.2.2.1.2.
statusCode	StatusCode	Code de statut indiquant la capacité de l'*Éditeur* à fournir les *MetaData* pour le DataSetWriterId.

7.2.2.4.2.5 Message de configuration du DataSetWriter

Le codage de la structure du message de données de configuration du *DataSetWriter* est spécifié dans le Tableau 90. Il comporte la configuration actuelle du *WriterGroup* et du *DataSetWriter* pour le *DataSet*.

L'*Éditeur* doit envoyer ce message sans demande de découverte correspondante si la configuration du WriterGroup a été modifiée.

Tableau 90 – Structure de message de configuration de DataSetWriter

Nom	Type	Description
DataSetWriterIds	UInt16[]	*DataSetWriterIds* contenus dans les informations de configuration.
DataSetWriterConfig	WriterGroupDataType	Paramètres actuels du *WriterGroup* et du *DataSetWriter* pour le *DataSet* associé au *DataSetWriterId*. Le *WriterGroupDataType* est défini en 6.2.5.6. Le champ *DataSetWriters* du *WriterGroupDataType* doit uniquement contenir l'entrée correspondant aux *DataSetWriters* du *WriterGroup* demandés ou modifiés.
statusCodes	StatusCode[]	Code de statut indiquant la capacité de l'*Éditeur* à fournir les informations de configuration pour les *DataSetWriterIds*. La taille de la matrice doit correspondre à la taille de la matrice des *DataSetWriterIds*.

7.2.3 Mapping de message JSON

7.2.3.1 Généralités

JSON est un format utilisant un texte lisible par l'homme. Il est défini dans le IETF RFC 7159.

Le mapping de message JSON permet aux *Applications* OPC UA d'interagir avec les logiciels Web et professionnels qui utilisent ce format.

7.2.3.2 NetworkMessage

Chaque *NetworkMessage* JSON comporte un ou plusieurs *DataSetMessages* JSON. Un *NetworkMessage* JSON est un objet JSON comportant les champs définis dans le Tableau 91.

Tableau 91 – Définition de NetworkMessage JSON

Nom	Type	Description
MessageId	Chaîne	Identificateur globalement unique pour le message. Cette valeur est obligatoire.
MessageType	Chaîne	Cette valeur doit être "ua-data". Cette valeur est obligatoire.
PublisherId	Chaîne	Identificateur unique pour l'*Éditeur*. Identifie la source du message. Cette valeur est facultative. La présence de cette valeur dépend de la définition du *JsonNetworkMessageContentMask*. La source est le *PublisherId* sur une *PubSubConnection* (voir 6.2.6.1).
DataSetClassId	Chaîne	*DataSetClassId* associé aux *DataSets* dans le *NetworkMessage*. Cette valeur est facultative. La présence de cette valeur dépend de la définition du *JsonNetworkMessageContentMask*. Si cela est spécifié, tous les *DataSetMessages* dans le *NetworkMessage* doivent avoir le même *DataSetClassId*. La source est le *DataSetClassId* sur le *PublishedDataSet* (voir 6.2.2.2) associé aux *DataSetWriters* ayant produit les *DataSetMessages*.
Messages	*	Matrice JSON des *DataSetMessages* JSON (voir 7.2.3.3). Cette valeur est obligatoire.

Tous les champs pour lesquels un *DataType* concret a été défini sont encodés à l'aide du *Codage de Données* JSON OPC UA réversible défini dans l'IEC 62541-6.

Les champs du *NetworkMessage* JSON sont contrôlés par le *NetworkMessageContentMask* du mapping de *NetworkMessage* JSON (voir 6.3.2.1.1).

Si le bit *NetworkMessageHeader* du *NetworkMessageContentMask* n'est pas défini, le *NetworkMessage* correspond au contenu du champ *Messages* (une matrice JSON de *DataSetMessages*, par exemple).

Si le bit *DataSetMessageHeader* du *NetworkMessageContentMask* n'est pas défini, le contenu du champ *Messages* est une matrice de contenu du champ *Payload* pour chaque *DataSetMessage* (voir 7.2.3.3).

Si le bit *SingleDataSetMessage* du *NetworkMessageContentMask* est défini, le contenu du champ *Messages* est un objet JSON contenant un unique *DataSetMessage*.

Si les bits *NetworkMessageHeader* et *DataSetMessageHeader* ne sont pas définis et que le bit *SingleDataSetMessage* est défini, le *NetworkMessage* est un objet JSON contenant l'ensemble de paires nom-valeur définies pour un unique *DataSet*.

Si la taille du *NetworkMessage* codé JSON dépasse les limites du *Courtier*, le message est abandonné et un *Événement PubSubTransportLimitsExceeded* est signalé.

7.2.3.3 DataSetMessage

Un *DataSetMessage* est produit par un *DataSetWriter* et comporte une liste des paires nom-valeur spécifiées par le *PublishedDataSet* associé au *DataSetWriter*. Le contenu du *DataSetMessage* est formellement défini par un *Objet DataSetMetData*. Un *DataSetMessage* est un objet JSON comportant les champs définis dans le Tableau 92.

Les *DataSetWriters* peuvent régulièrement fournir des messages d'entretien qui sont des *DataSetMessages* sans champ *Payload*.

Tableau 92 – Définition de DataSetMessage JSON

Nom	Type	Description
DataSetWriterId	UInt16	Identificateur du *DataSetWriter* qui a créé le *DataSetMessage*. Cette valeur est obligatoire. Cet identificateur est unique dans le domaine d'application de l'*Éditeur*.
SequenceNumber	UInt32	Numéro de séquence à croissance monolithique stricte affecté au *DataSetMessage* par le *DataSetWriter*. Cette valeur est facultative. La présence de cette valeur dépend de la définition du *JsonDataSetMessageContentMask*.
MetaDataVersion	ConfigurationVersionDataType	Version des *DataSetMetaData* qui décrivent les contenus de la *Charge utile*. Cette valeur est facultative. La présence de cette valeur dépend de la définition du *JsonDataSetMessageContentMask*.
Horodatage	DateTime	Horodatage qui s'applique à toutes les valeurs contenues dans le *DataSetMessage*. Cette valeur est facultative. La présence de cette valeur dépend de la définition du *JsonDataSetMessageContentMask*.
Statut	StatusCode	Code de statut qui s'applique à toutes les valeurs contenues dans le *DataSetMessage*. Cette valeur est facultative. La présence de cette valeur dépend de la définition du *JsonDataSetMessageContentMask*.
Payload	Objet	Objet JSON contenant les paires nom-valeur spécifiées par le *PublishedDataSet*. Le format de la valeur dépend du *DataType* du champ et des fanions spécifiés par le *DataSetMessageContentMask*.

Tous les champs contenant un *DataType* concret sont codés à l'aide du *Codage de Données* JSON OPC UA réversible défini dans l'IEC 62541-6.

Les champs du *DataSetMessage* sont spécifiés par le *DataSetFieldContentMask* dans les paramètres du *DataSetWriter*.

Le *DataSetFieldContentMask* spécifie le format des valeurs de champ de la *Charge utile* conformément aux règles suivantes:

- si le *DataSetFieldContentMask* aboutit à une représentation de RawData, la valeur de champ est une *Variante* codée au moyen du *Codage de Données* JSON OPC UA non réversible défini dans l'IEC 62541-6;
- si le *DataSetFieldContentMask* aboutit à une représentation de *DataValue*, la valeur du champ est une *DataValue* codée au moyen du *Codage de Données* JSON OPC UA non réversible défini dans l'IEC 62541-6;
- si le *DataSetFieldContentMask* aboutit à une représentation de *Variante*, la valeur du champ est une *Variante* codée au moyen du *Codage de Données* JSON OPC UA réversible défini dans l'IEC 62541-6.

7.2.3.4 Messages de découverte

7.2.3.4.1 Généralités

Le mapping de message JSON définit seulement un message de découverte facultatif pour l'échange des *DataSetMetaData*. L'objectif principal est l'échange d'informations supplémentaires non contenues dans les *DataSetMessages*, comme les *Propriétés* des champs du *DataSet*.

7.2.3.4.2 DataSetMetaData

Les *DataSetMetaData* décrivent le contenu d'un *DataSet* publié par un *DataSetWriter*. Plus précisément, il spécifie les noms et types de données des valeurs qui doivent apparaître dans la *Charge utile* d'un *DataSetMessage*.

Lorsque les DataSetMetaData d'un DataSet varient, le *DataSetWriter* peut être configuré pour publier la valeur mise à jour au moyen du mécanisme défini par le mapping de protocole de transport.

Les champs *DataSetWriterId* et *Version* d'un *DataSetMessage* sont utilisés pour corréler un *DataSetMessage* avec des *DataSetMetaData*.

Un *DataSetMetaData* est un objet JSON comportant les champs définis dans le Tableau 93.

Tableau 93 – Définition de DataSetMetaData JSON

Nom	Type	Description
MessageId	Chaîne	Identificateur globalement unique pour le message. Cette valeur est obligatoire.
MessageType	Chaîne	Cette valeur doit être "ua-metadata". Cette valeur est obligatoire.
PublisherId	Chaîne	Identificateur unique pour l'*Éditeur*. Identifie la source du message. Cette valeur est obligatoire.
DataSetWriterId	UInt16	Identificateur du *DataSetWriter* qui a publié les *DataSetMetaData*. Cette valeur est obligatoire. Cet identificateur est unique dans le domaine d'application de l'*Éditeur*.
MetaData	DataSetMetaDataType	Métadonnées définies en 6.2.2.1.2. Cette valeur est obligatoire.

Tous les champs contenant un *DataType* concret sont codés à l'aide du *Codage de Données* JSON OPC UA réversible défini dans l'IEC 62541-6.

7.3 Mappings de protocole de transport

7.3.1 Généralités

Le paragraphe 7.3 répertorie les protocoles normalisés qui ont été choisis pour le présent document ainsi que leurs combinaisons possibles avec les mappings de message.

7.3.2 UDP OPC UA

UDP OPC UA est un protocole UDP simple utilisé pour le transport des *NetworkMessages* UADP.

La syntaxe de l'URL du protocole de transport UDP utilisée dans le paramètre d'*Adresse* défini en 6.2.6.3 prend la forme suivante:

opc.udp://<hôte>[:<accès>]

L'hôte est une adresse IP ou un nom enregistré comme un nom d'hôte ou de domaine. Les adresses IP peuvent être des adresses de monodiffusion, de multidiffusion ou de diffusion. Il s'agit de la destination du datagramme UDP.

L'accès OPC UA enregistré par l'IANA pour la communication UDP est 4840. Il s'agit de l'accès par défaut recommandé pour la communication en diffusion, en monodiffusion et en multidiffusion. Des accès de remplacement peuvent être utilisés.

Le transport d'un *NetworkMessage* UADP dans un paquet UDP est défini dans le Tableau 94.

Tableau 94 – Message UADP transporté par UDP

Nom	Description
En-tête de trame	En-tête de la trame.
En-tête IP	L'en-tête IP de la trame comporte des informations comme l'adresse IP source et l'adresse IP de destination. La taille de l'en-tête IP dépend de la version utilisée.
En-tête UDP	L'en-tête UDP de la trame comporte l'accès source, l'accès de destination, la longueur et la somme de contrôle. Chaque champ a une longueur de deux octets. La taille totale de l'en-tête UDP est de 8 octets.
NetworkMessage UADP	Le NetworkMessage UADP est envoyé comme données UDP.
Pied de page de trame	Pied de page de la trame.

Pour le protocole UDP OPC UA, il est recommandé de limiter la *MaxNetworkMessageSize* avec les en-têtes supplémentaires à la taille de la MTU. Le nombre de trames utilisées pour un *NetworkMessage* UADP influence la probabilité de perdre un *NetworkMessages* UADP.

Pour le protocole UDP OPC UA, la *MaxNetworkMessageSize* et les en-têtes supplémentaires doivent être limités à 65 535 octets.

Il est recommandé d'utiliser des commutateurs avec prise en charge IGMP pour limiter la distribution du trafic en multidiffusion vers les participants intéressés. Les *Applications* OPC UA doivent émettre un rapport d'adhésion IGMP.

NOTE L'IGMP (Internet Group Management Protocol) est un protocole normalisé utilisé par les hôtes pour signaler leur adhésion aux groupes de multidiffusion IP; il est essentiel de le mettre en œuvre par tout hôte souhaitant recevoir des datagrammes de multidiffusion IP. Les messages IGMP sont utilisés par les routeurs de multidiffusion afin de savoir quels groupes de multidiffusion disposent de membres sur leurs réseaux associés. Les messages IGMP sont également utilisés par les commutateurs capables de prendre en charge l'"IGMP snooping", lequel permet au commutateur d'écouter les messages IGMP et d'envoyer uniquement les *NetworkMessages* de multidiffusion vers les accès ayant rejoint le groupe de multidiffusion.

Il existe trois versions d'IGMP:

- IGMP V1 défini dans l'IETF RFC 1112;
- IGMP V2 défini dans l'IETF RFC 2236;
- IGMP V3 défini dans l'IETF RFC 3376.

L'IETF RFC 2236 et l'IETF RFC 3376 traitent des exigences applicables à l'hôte et au routeur pour l'interopération avec les précédentes versions d'IGMP.

Étant donné que les dispositifs OPC UA font un usage intensif de la multidiffusion IP pour le transport UDP, une utilisation cohérente de l'IGMP par les dispositifs OPC UA est essentielle pour créer des réseaux d'*Application* OPC UA parfaitement fonctionnels.

Les dispositifs OPC UA émettent un message de rapport d'adhésion (V1, V2 ou V3, selon le cas) lors de l'ouverture d'une connexion UDP sur laquelle ils recevront des *NetworkMessages* en multidiffusion.

7.3.3 OPC UA Ethernet

OPC UA Ethernet est un protocole Ethernet simple qui utilise l'EtherType B62C et est utilisé pour le transport des *NetworkMessages* UADP comme charge utile de la trame Ethernet II sans en-tête IP ou UDP.

La syntaxe de l'URL du protocole de transport Ethernet utilisée dans le paramètre d'*Adresse* défini en 6.2.6.3 prend la forme suivante:

opc.eth://<hôte>[:<VID>[.PCP]]

L'hôte est une adresse MAC, une adresse IP ou un nom enregistré comme un nom d'hôte. Le format d'une adresse MAC est constitué par six groupes de chiffres hexadécimaux séparés par des tirets (01-23-45-67-89-ab, par exemple). Un système peut également accepter les noms d'hôtes et/ou les adresses IP s'il fournit des solutions pour les résoudre en une adresse MAC (DNS et Reverse-ARP, par exemple).

Le VID est l'ID du VLAN (un numéro).

Le PCP est le Point de Code de Priorité (un numéro à un chiffre).

Le transport d'un *NetworkMessage* UADP dans une trame Ethernet II est défini dans le Tableau 95.

Tableau 95 – Message UADP transporté par Ethernet

Nom	Description
En-tête de trame	En-tête de la trame avec un EtherType de 0xB62C.
NetworkMessage UADP	Le NetworkMessage UADP est envoyé comme charge utile Ethernet.
Pied de page de trame	Pied de page de la trame.

Pour l'OPC UA Ethernet, la *MaxNetworkMessageSize* et les en-têtes supplémentaires doivent être limités à la taille d'une trame Ethernet, soit 1 522 octets.

L'EtherType OPC UA enregistré par l'IANA pour la communication UDP est 0xB62C.

7.3.4 AMQP

7.3.4.1 Généralités

L'AMQP (Advanced Message Queuing Protocol) est un protocole de couche d'application normalisé ouvert pour l'*Intergiciel Orienté Message*. L'AMQP est souvent utilisé avec un *Courtier* qui relaie les messages entre les applications qui ne peuvent pas communiquer directement.

Les *Éditeurs* envoient des messages AMQP aux points d'extrémité AMQP. Les abonnés écoutent les points d'extrémité AMQP pour les messages entrants. Si un *Courtier* est utilisé, il peut conserver les messages de sorte qu'ils puissent être livrés même quand l'abonné n'est pas en ligne. Les *Courtiers* peuvent également permettre l'envoi de messages à plusieurs Abonnés.

Le protocole AMQP définit un codage binaire pour tous les messages avec un en-tête et un corps. L'en-tête permet aux applications d'insérer des informations supplémentaires telles que les paires nom-valeur sérialisées par le codage binaire AMQP. Le corps est un blob binaire opaque qui peut contenir toutes données sérialisées à l'aide d'un codage déterminé par l'application.

Le présent document définit deux mappings de message possibles pour le corps de message AMQP: le mapping de message UADP défini en 7.2.2 et le mapping de message JSON définie en 7.2.3. Les *Courtiers* AMQP sont associés à une limite de taille de message supérieure. Le mécanisme de gestion des *NetworkMessages* qui dépassent les limites du *Courtier* dépend du codage.

Avec l'AMQP, la sécurité est principalement assurée par une connexion TLS entre l'*Éditeur* ou l'*Abonné* et le *Courtier* AMQP; toutefois, cela exige que le *Courtier* AMQP soit approuvé. C'est pourquoi il peut être nécessaire de fournir une sécurité de bout en bout. Il est nécessaire que les applications qui exigent une sécurité de bout en bout avec l'AMQP utilisent les *NetworkMessages* UADP et le codage de message binaire défini en 7.2.2.2. Les corps de message codés JSON reposent sur les mécanismes de sécurité fournis par l'AMQP et le *Courtier* AMQP.

7.3.4.2 Adresse

La syntaxe de l'URL du protocole de transport AMQP utilisée dans le paramètre d'*Adresse* défini en 6.2.6.3 prend la forme suivante:

amqps://<nom de domaine>[:<accès>][/<chemin>]

L'accès par défaut est 5671.

La syntaxe de l'URL AMQP sur les WebSockets prend la forme suivante:

wss://<nom de domaine>[:<accès>][/<chemin>]

L'accès par défaut est 443.

7.3.4.3 Authentification

L'authentification doit s'effectuer conformément au *AuthenticationProfileUri* configuré des entités de *PubSubConnection*, *DataSetWriterGroup*, *DataSetWriter* ou *DataSetReader*.

En l'absence d'informations d'authentification fournies sous la forme d'un *ResourceUri* et d'un *AuthenticationProfileUri*, une authentification SASL Anonyme est utilisée.

Si le profil d'authentification spécifie une authentification SASL PLAIN, une connexion séparée est exigée pour chaque nouveau paramètre d'authentification.

7.3.4.4 Propriétés de connexion

L'AMQP permet l'envoi de propriétés dans le cadre de l'ouverture d'une connexion, de l'établissement d'une session et de l'ajout de liens.

Les propriétés de connexion s'appliquent à toute connexion, toute session ou tout lien créé(e) dans le cadre de la *PubSubConnection* ou des entités de configuration subordonnées telles que le *WriterGroup* et le *DataSetWriter*.

Les propriétés sont définies par la matrice *KeyValuePair* dans les ConnectionProperties, les *WriterGroupProperties* et les *DataSetWriterProperties*. Le *NamespaceIndex* du *QualifiedName* de la *KeyValuePair* doit être 0 pour les propriétés AMQP normalisées. Le *Nom* du *QualifiedName* est constitué à partir d'un préfixe et du nom de propriété AMQP avec la syntaxe suivante.

Nom = <préfixe cible>-<nom de propriété AMQP>

Le préfixe cible peut avoir les valeurs suivantes:

- connexion;
- session;
- lien.

La *Valeur* de la *KeyValuePair* est convertie en type de données AMQP à l'aide des règles définies dans le Tableau 98. S'il n'existe aucune règle définie pour ce type de données, la propriété ne doit pas être incluse.

Les propriétés de connexion sont destinées à être utilisées occasionnellement afin d'optimiser l'interopérabilité avec les points d'extrémité des courtiers existants.

7.3.4.5 RequestedDeliveryGuarantee

Un rédacteur négocie la garantie de livraison de son lien à l'aide de la politique de règlement snd-settle-mode (réglé, non réglé, mixte) qu'il utilise et de la politique rcv-settle-mode (premier, second) de courtier souhaitée.

Inversement, le lecteur négocie la garantie de livraison à l'aide de la politique rcv-settle-mode (premier, second) qu'il utilise et de la politique snd-settle-mode (réglé, non réglé) de courtier souhaitée.

Cela correspond aux valeurs de *BrokerTransportQualityOfService* suivantes:

- *AtMostOnce_1*: les messages sont préréglés au niveau du point d'extrémité de l'expéditeur et ne sont pas renvoyés. Les messages peuvent être perdus lors du transit. Il s'agit du paramètre par défaut;
- *AtLeastOnce_2*: les messages sont reçus et réglés au niveau du récepteur, sans attendre le règlement par l'expéditeur;
- *ExactlyOnce_3: les messages sont reçus, l'expéditeur procède au règlement, puis le récepteur procède au règlement.*

7.3.4.6 Limites de transport et entretien

Si le *KeepAliveTime* est défini sur un *WriterGroup*, il convient d'utiliser une valeur légèrement supérieure à la valeur configurée pour le groupe comme délai d'inactivité de la connexion afin de s'assurer que la connexion est déconnectée lorsque le message d'entretien n'a pas été envoyé par un rédacteur. Sinon, lorsqu'aucun *KeepAliveTime* n'est spécifié, il convient que la mise en œuvre définisse une valeur par défaut raisonnable.

Lors de la définition de la taille de message maximale pour le lien, la *MaxNetworkMessageSize* du *PubSubGroup* doit être utilisée. Si cette valeur est 0, la mise en œuvre détermine une valeur maximale raisonnable.

Les autres limites dépendent de la mise en œuvre et des capacités du système d'exploitation ou des fonctionnalités du dispositif sur lequel l'*Éditeur* ou l'*Abonné* s'exécute; elles peuvent être rendues configurables par des extensions de modèle de configuration ou par d'autres moyens.

7.3.4.7 En-tête de message

L'en-tête d'un message AMQP comporte plusieurs champs normalisés. Le Tableau 96 décrit comment ces champs doivent être renseignés lorsque le message AMQP est constitué.

Tableau 96 – Champs d'en-tête AMQP normalisé

Nom de champ	Source
message-id	Valeur globalement unique créée par le *DataSetWriter*.
subject	Les valeurs valides sont "ua-data" ou "ua-metadata".
content-type	Type MIME pour le corps de message. Les types MIME sont spécifiés en 7.3.4.8.1 et 7.3.4.8.2, relatifs au corps de message.

Le sujet définit le type de message contenu dans le corps AMQP. Une valeur "ua-data" spécifie que le corps contient un *NetworkMessage* UADP ou JSON. Une valeur "ua-metadata" spécifie que le corps contient un *Message* de *DataSetMetaData* à codage UA Binaire ou JSON. Le "content-type" spécifie si les données du message sont binaires ou JSON.

L'en-tête de message AMQP doit inclure des champs supplémentaires définis dans le *WriterGroup* ou le *DataSetWriter* par le biais de la matrice *KeyValuePair* dans les *WriterGroupProperties* et les *DataSetWriterProperties*. Le *NamespaceIndex* du *QualifiedName* de la *KeyValuePair* doit être 0 pour les propriétés de message AMQP normalisées. Le *Nom* du *QualifiedName* est constitué à partir d'un préfixe de message et du nom de propriété AMQP avec la syntaxe suivante.

Nom = message-<nom de propriété AMQP>

Le Tableau 97 définit les propriétés de messages AMQP normalisées.

Tableau 97 – Mappings du QualifiedName et du Nom de l'en-tête AMQP OPC UA normalisé

Nom de la propriété AMQP normalisée	DataType OPC UA	Type de données AMQP
to	Chaîne	*
user-id	ByteString	binary
reply-to	Chaîne	string
correlation-id	ByteString	*
absolute-expiry-time	DateTime	timestamp
group-id	Chaîne	string
reply-to-group-id	Chaîne	string
creation-time	DateTime	timestamp
content-encoding	Chaîne	symbol

L'hypothèse retenue est que tout nom qui ne se trouve pas dans le tableau est une propriété d'application. Dans ce cas, l'espace de noms fourni dans le cadre du *QualifiedName* doit être l'*ApplicationUri*.

L'en-tête de message AMQP doit inclure les champs promus supplémentaires du *DataSet* comme une liste de paires nom-valeur. Les champs de *DataSet* avec le fanion *PromotedField* défini dans les *fieldFlags* des *FieldMetaData* sont copiés dans l'en-tête AMQP. La *Structure* des *FieldMetaData* est définie en 6.2.2.1.3. Les champs promus doivent toujours être inclus dans l'en-tête, et ce même si le corps du *DataSetMessage* comporte une image delta et que le champ du *DataSet* n'est pas inclus dans l'image delta. Dans ce cas, la dernière valeur connue est envoyée dans l'en-tête.

Lorsqu'un champ est ajouté à l'en-tête, il est converti en type de données AMQP à l'aide des règles définies dans le Tableau 98. S'il n'existe aucune règle définie pour ce type de données, le champ ne doit pas être inclus.

Tableau 98 – Règles de conversion des champs de l'en-tête AMQP OPC UA

DataType OPC UA	Règles de conversion en types de données AMQP
Booléen	Type AMQP "boolean".
SByte	Type AMQP "byte".
Octet	Type AMQP "ubyte".
Int16	Type AMQP "short".
UInt16	Type AMQP "ushort".
Int32	Type AMQP "int".
UInt32	Type AMQP "uint".
Int64	Type AMQP "long".
UInt64	Type AMQP "ulong".
Float	Type AMQP "float".
Double	Type AMQP "double".
Chaîne	Type AMQP "string".
ByteString	Type AMQP "binary".
DateTime	Type AMQP "timestamp". Cette conversion peut aboutir à une perte de précision sur certaines plateformes. Les règles de gestion des pertes de précision sont décrites dans l'IEC 62541-6.
Guid	Type AMQP "uuid".
QualifiedName	Le QualifiedName est codé comme un type AMQP "string" avec le format <NamespaceUri>'#'<Name>.
LocalizedText	Non pris en charge; le champ associé est éliminé.
NodeId	Si le NamespaceIndex = 0, la valeur est codée comme un type AMQP "string" utilisant le format pour le NodeId défini dans l'IEC 62541-6. Si le NamespaceIndex est > 0, la valeur est convertie en un ExpandedNodeId avec un NamespaceUri et codée comme un type AMQP "string" utilisant le format pour l'ExpandedNodeId défini dans l'IEC 62541-6.
ExpandedNodeId	Si le NamespaceUri n'est pas fourni, les règles applicables au NodeId sont utilisées. Si le NamespaceUri est fourni, la valeur est codée comme un type AMQP "string" utilisant le format pour l'ExpandedNodeId défini dans l'IEC 62541-6.
StatusCode	Type AMQP "uint".
Variante	Si la valeur dispose d'un datatype pris en charge, il utilise cette conversion; sinon, il n'est pas pris en charge et le champ associé est éliminé.
Structure	Non pris en charge; le champ associé est éliminé.
Structure avec champs facultatifs	Non pris en charge; le champ associé est éliminé.
Matrice	Non pris en charge; le champ associé est éliminé.
Union	Non pris en charge; le champ associé est éliminé.

7.3.4.8 Corps de message

7.3.4.8.1 JSON

Un corps JSON est codé comme pour le mapping de message JSON défini en 7.2.3.

Le type MIME correspondant est application/json.

7.3.4.8.2 UADP

Un corps UADP est codé comme pour le mapping de message UADP défini en 7.2.2.

Le type MIME correspondant est application/opcua+uadp.

Si la taille du message AMQP codé dépasse les limites du *Courtier*, il doit être décomposé en plusieurs blocs, comme décrit en 7.2.2.2.4.

Il est recommandé de configurer le *MetaDataQueueName* décrit en 6.4.2.3.6 comme une sous-rubrique du *QueueName* correspondant avec le nom "$Metadata".

7.3.5 MQTT

7.3.5.1 Généralités

Le MQTT (Message Queue Telemetry Transport) est un protocole de couche d'application normalisé ouvert pour l'*Intergiciel Orienté Message*. Le MQTT est souvent utilisé avec un *Courtier* qui relaie les messages entre les applications qui ne peuvent pas communiquer directement.

Les *Éditeurs* envoient des messages MQTT aux courtiers MQTT. Les abonnés s'abonnent aux courtiers MQTT pour les messages. Un *Courtier* peut conserver les messages de sorte qu'ils puissent être livrés même quand l'abonné n'est pas en ligne. Les *Courtiers* peuvent également permettre l'envoi de messages à plusieurs *Abonnés*.

Le protocole MQTT définit un protocole binaire utilisé pour envoyer et recevoir des messages aux/des rubriques. Le corps est un blob binaire opaque qui peut contenir toutes données sérialisées à l'aide d'un codage déterminé par l'application.

Le présent document définit deux codages possibles pour le corps de message: le *DataSetMessage* à codage binaire défini en 7.2.2 et le *DataSetMessage* à codage JSON défini en 7.2.3. Le MQTT ne fournit pas de mécanisme de spécification du codage des messages MQTT, ce qui signifie que les *Abonnés* doivent être configurés à l'avance avec la connaissance du codage attendu. Il convient que les *Éditeurs* publient uniquement les *NetworkMessages* en utilisant un seul codage vers un nom de rubrique MQTT unique.

Avec le MQTT, la sécurité est principalement assurée par une connexion TLS entre l'*Éditeur* ou l'*Abonné* et le serveur MQTT; toutefois, cela exige que le serveur MQTT soit approuvé. C'est pourquoi il peut être nécessaire de fournir une sécurité de bout en bout. Il est nécessaire que les applications qui exigent une sécurité de bout en bout avec le MQTT utilisent les *NetworkMessages* UADP et le codage de message binaire défini en 7.2.2. Il est essentiel que les corps de message codés JSON reposent sur les mécanismes de sécurité fournis par le MQTT et le serveur MQTT.

7.3.5.2 Adresse

La syntaxe de l'URL du protocole de transport MQTT utilisée dans le paramètre d'*Adresse* défini en 6.2.6.3 prend la forme suivante:

mqtts://<nom de domaine>[:<accès>][/<chemin>]

L'accès par défaut est 8883.

La syntaxe de l'URL MQTT sur les WebSockets prend la forme suivante:

wss://<nom de domaine>[:<accès>][/<chemin>]

L'accès par défaut est 443.

7.3.5.3 Authentification

L'actuel mapping de transport MQTT prend uniquement en charge l'authentification simple par Nom d'utilisateur/Mot de passe.

7.3.5.4 ConnectionProperties

L'actuel mapping de transport MQTT ne prend pas en charge le concept de propriétés de connexion et tout paramètre configuré des propriétés de connexion doit être discrètement éliminé.

Il convient que les mises en œuvre tentent de se reconnecter aux sessions existantes (CleanSession=0) et de reprendre le transfert de message au niveau de QoS spécifié.

7.3.5.5 RequestedDeliveryGuarantee

Les valeurs de *BrokerTransportQualityOfService* sont mises en correspondance vers les paramètres de QoS MQTT de publication et abonnement comme suit:

- AtMostOnce_1 est est associée par mapping à la QoS MQTT 0;
- AtLeastOnce_2 est est associée par mapping à la QoS MQTT 1;
- ExactlyOnce_3 est est associée par mapping à la QoS MQTT 2.

7.3.5.6 Limites de transport et entretien

Si le *KeepAliveTime* est défini sur un *WriterGroup*, il convient d'utiliser une valeur légèrement supérieure à la valeur configurée (en secondes) pour le groupe comme délai d'entretien MQTT afin de s'assurer que la connexion est déconnectée lorsque le message d'entretien n'a pas été envoyé par un rédacteur dans les délais spécifiés.

La mise en œuvre détermine les limites de taille du paquet et du message en fonction des capacités du système d'exploitation ou des fonctionnalités du dispositif sur lequel l'application s'exécute. Elles peuvent être rendues configurables par des extensions de modèle de configuration ou par d'autres moyens.

7.3.5.7 En-tête de message

L'actuel mapping de transport MQTT ne prend pas en charge les en-têtes de message. Tout champ promu ou supplémentaire défini dans le *WriterGroup* ou le *DataSetWriter* doit être discrètement éliminé. Les mises en œuvre ne doivent pas définir le fanion MQTT RETAIN, sauf pour les messages de métadonnées publiés au *MetaDataQueueName*, comme décrit en 6.4.2.3.6.

7.3.5.8 Corps de message

7.3.5.8.1 JSON

Un corps JSON est codé comme pour le mapping de message JSON définie en 7.2.3.

7.3.5.8.2 UADP

Un corps UADP est codé comme pour le mapping de message UADP définie en 7.2.2.

Il est prévu que le logiciel utilisé pour recevoir les *NetworkMessages* UADP puisse traiter le corps sans avoir besoin d'en connaître le mode de transport.

Si la taille du message MQTT codé dépasse les limites du *Courtier*, il est décomposé en plusieurs blocs, comme décrit en 7.2.2.2.4.

Il est recommandé de configurer le *MetaDataQueueName* décrit en 6.4.2.3.6 comme une sous-rubrique du *QueueName* correspondant avec le nom "$Metadata". Le fanion MQTT RETAIN doit être défini pour les messages de métadonnées.

8 Modèle de service de clés de sécurité PubSub

8.1 Vue d'ensemble

L'Article 8 spécifie le *Modèle d'Information* OPC UA pour un *Service de Clé de Sécurité* (SKS). La fonctionnalité et le comportement d'un SKS sont décrits en 5.4.3. Il définit le cadre de distribution des clés cryptographiques utilisées pour la sécurité de messages.

Le SKS peut être un service réseau utilisé pour gérer les clés de tous les *Éditeurs* et *Abonnés* ou faire partie d'un *Éditeur* pour gérer les clés pour les *NetworkMessages* envoyés par cet *Éditeur*.

La Figure 34 représente les *ObjectTypes* et leurs composants utilisés pour représenter l'Objet *PublishSubscribe*.

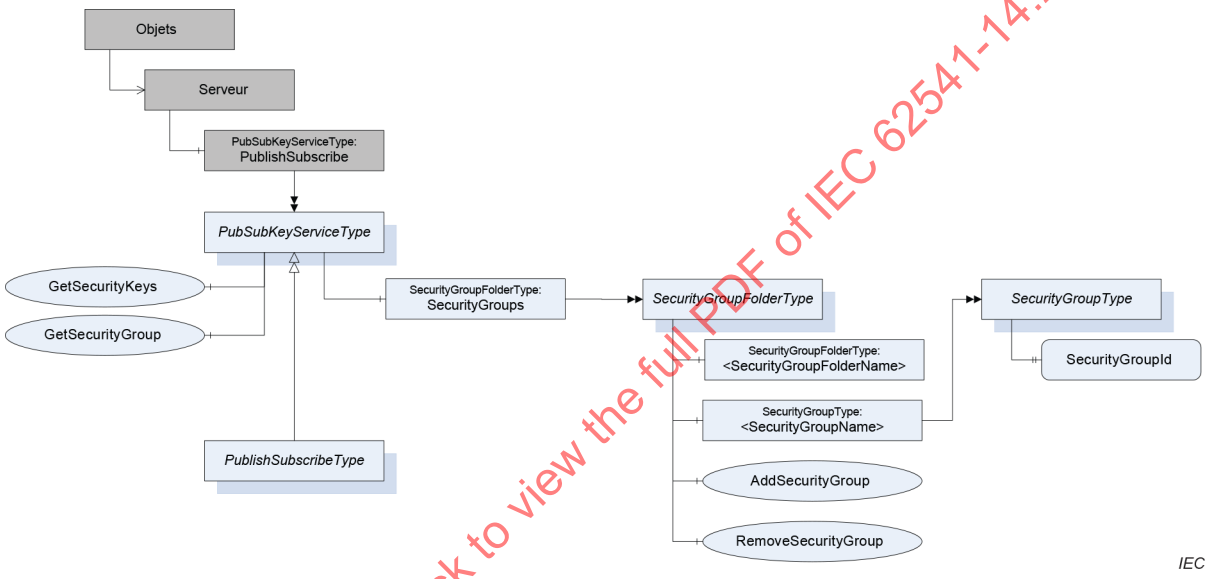


Figure 34 – Vue d'ensemble des Types d'Objets PublishSubscribe

L'*Objet PublishSubscribe* est le nœud racine de tous les *Objets* de configuration *PubSub* associés. Il s'agit d'une instance du *PubSubKeyServiceType* ou du *PublishSubscribeType* et d'un composant de l'*Objet Serveur*.

Le *PubSubKeyServiceType* définit la *Méthode* d'accès aux clés de sécurité et la gestion correspondante des *SecurityGroups*. Cet *ObjectType* est utilisé pour l'*Objet PublishSubscribe* si seule la fonctionnalité de *Service de Clé de Sécurité* est fournie. Si la fonctionnalité de configuration *PubSub* est fournie, le *PublishSubscribeType* est utilisé à la place.

Les *SecurityGroups* sont organisés par le *SecurityGroupFolderType* et représentés par les instances du *SecurityGroupType*.

Le *PublishSubscribeType* comporte les points d'entrée du modèle de configuration PubSub défini à l'Article 9.

8.2 Objet PublishSubscribe

Pour assurer l'interopérabilité entre les *Éditeurs*, les *Abonnés, les Services de Clé de Sécurité* et les outils de configuration, tous les *Objets PubSub* doivent être exposés au moyen d'un *Objet* intitulé "PublishSubscribe" du type *PubSubKeyServiceType* ou d'un sous-type. Cet *Objet* doit être un composant de l'*Objet Serveur*. Il est défini de manière formelle dans le Tableau 99.

Tableau 99 – Définition de l'Objet PublishSubscribe

Attribut	Valeur				
BrowseName	PublishSubscribe				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Composant de l'*Objet Serveur* défini dans l'IEC 62541-5.					
HasTypeDefinition	ObjectType	PubSubKeyServiceType			

8.3 PubSubKeyServiceType

Le *PubSubKeyServiceType* est défini de manière formelle dans le Tableau 100.

Tableau 100 – Définition de PubSubKeyServiceType

Attribut	Valeur				
BrowseName	PubSubKeyServiceType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de BaseObjectType défini dans l'IEC 62541-5.					
HasComponent	Méthode	GetSecurityKeys	Défini en 8.4.		Facultative
HasComponent	Méthode	GetSecurityGroup	Défini en 8.7.		Facultative
HasComponent	Objet	SecurityGroups		SecurityGroupFolderType	Facultative

L'*ObjectType PubSubKeyServiceType* est un type concret qui peut être utilisé directement.

Le dossier *SecurityGroups* organise les *Objets* représentant la configuration du *SecurityGroup*.

8.4 Méthode GetSecurityKeys

Cette *Méthode* est utilisée pour récupérer les clés de sécurité d'un *SecurityGroup*.

Cette *Méthode* est exigée pour accéder aux clés de sécurité d'un *PubSubGroup* où le *SecurityGroup* gère les clés de sécurité des *PubSubGroups*. L'*Objet MessageSecurity* de l'*Objet PubSubGroup* comporte le *SecurityGroupId* qui doit être transmis à cette *Méthode* afin d'accéder aux clés du *PubSubGroup*. Noter que plusieurs *PubSubGroups* peuvent partager un même *SecurityGroupId*.

Les *Autorisations* de l'*Objet SecurityGroupType* pour le *SecurityGroupId* contrôlent l'accès aux clés de sécurité du *SecurityGroupId*. Si l'utilisateur utilisé pour appeler cette *Méthode* ne dispose pas de l'*Autorisation d'Appel* définie pour l'*Objet SecurityGroupType* correspondant, le *Serveur* doit renvoyer une erreur *Bad_UserAccessDenied* pour cette *Méthode*. Le *SecurityGroupType* est défini en 8.6. Le chiffrement est exigé pour cette *Méthode*. Cette *Méthode* doit renvoyer une erreur *Bad_SecurityModeInsufficient* si la communication n'est pas chiffrée.

Les informations nécessaires pour accéder au *Serveur* qui met en œuvre la *Méthode GetSecurityKeys* pour le *SecurityGroup* sont également contenues dans l'*Objet MessageSecurity* de l'*Objet PubSubGroup*.

La *Méthode GetSecurityKeys* peut être mise en œuvre par un *Éditeur* ou par un SKS central. Dans les deux cas, les *NodeIds* notoires de l'*Objet PublishSubscribe* et la *Méthode GetSecurityKeys* correspondante sont utilisés pour appeler la *Méthode GetSecurityKeys*.

Si l'*Éditeur* met en œuvre la *Méthode GetSecurityKeys* et la gestion du *SecurityGroup* associé, les clés sont invalidées immédiatement après le retrait d'un *SecurityGroup* ou la révocation des clés d'un *SecurityGroup*.

Si un SKS central met en œuvre la *Méthode GetSecurityKeys* et la gestion du *SecurityGroup* associé, les clés ne sont plus valides après le retrait d'un *SecurityGroup* ou la révocation des clés d'un *SecurityGroup*. Toutefois, les *Abonnés* doivent être prêts pour les *Éditeurs* utilisant des clés invalidées jusqu'à ce qu'ils aient appelé la *Méthode GetSecurityKeys*. Les *Éditeurs* qui utilisent un SKS central doivent appeler la méthode *GetSecurityKeys* à une période correspondant à la moitié de la *KeyLifetime*.

Signature

```
GetSecurityKeys (
    [in]  String           SecurityGroupId
    [in]  UInt32           StartingTokenId
    [in]  UInt32           RequestedKeyCount
    [out] String           SecurityPolicyUri
    [out] IntegerId        FirstTokenId
    [out] ByteString[]     Keys
    [out] Duration         TimeToNextKey
    [out] Duration         KeyLifetime
    );
```

Argument	Description
SecurityGroupId	Identificateur du *SecurityGroup*. Il doit être unique au sein du *Service de Clé de Sécurité*.
StartingTokenId	L'actuel jeton est demandé en transmettant 0. Il peut s'agit d'un ancien *SecurityTokenId* afin d'obtenir une clé valide pour les messages précédemment envoyés. Si le *StartingTokenId* est inconnu, les jetons les plus anciens disponibles sont renvoyés.
RequestedKeyCount	Nombre de clés demandées qu'il convient de renvoyer dans la réponse. Si 0 est demandé, aucune future clé n'est renvoyée. Si l'appelant demande un nombre supérieur à celui admis par le *Service de Clé de Sécurité*, le SKS doit renvoyer le nombre maximal de clés admis.
SecurityPolicyUri	URI de l'ensemble d'algorithmes et de longueurs de clé utilisé pour sécuriser les messages. Les *SecurityPolicies* sont définies dans l'IEC 62541-7.
FirstTokenId	*SecurityTokenId* de la première clé de la matrice de clés renvoyées. Le *SecurityTokenId* apparaît dans l'en-tête des messages sécurisés à l'aide d'une *Clé*. Il commence à 1 et est incrémenté de 1 à chaque fois que la *KeyLifetime* expire même si aucune clé n'a été demandée. Si le *SecurityTokenId* est incrémenté au-delà de la valeur maximale de *UInt32*, il redémarre à 1. Si l'appelant dispose des informations de clé issues des appels précédents de la *Méthode GetSecurityKeys*, le *FirstTokenId* est utilisé pour mettre en correspondance la liste existante avec la liste extraite et pour éliminer les doublons. Si le *FirstTokenId* est inconnu, la liste existante doit être éliminée et remplacée.
Clés	Liste classée des clés utilisées lorsque la *KeyLifetime* expire. Si la clé actuelle a été demandée, la première clé de la matrice est utilisée pour sécuriser les messages. Cette clé n'est pas directement utilisée étant donné que le protocole associé aux *PubSubGroups* spécifie un algorithme pour générer des clés distinctes pour les différents types d'opérations de cryptographie. Des informations supplémentaires sont données en 7.2.2.2.3. Le *SecurityTokenId* associé à la première clé de la liste est le *FirstTokenId*. Toutes les clés suivantes possèdent un *SecurityTokenId* incrémenté de 1 pour chaque clé renvoyée.
TimeToNextKey	Délai (en millisecondes) prévu avant que la *CurrentKey* n'expire. Si un *Éditeur* utilise cette *Méthode* pour obtenir les clés auprès d'un SKS, le *TimeToNextKey* et la *KeyLifetime* sont utilisés pour calculer le moment où l'*Éditeur* doit utiliser la clé suivante. Le *TimeToNextKey* défini le moment auquel passer de la *CurrentKey* à la *FutureKey* et la *KeyLifetime* définit le moment auquel passer d'une future clé à la suivante. Pour un *Abonné*, le *TimeToNextKey* et la *KeyLifetime* sont utilisés pour calculer le temps que l'*Abonné* attend avant que les *Éditeurs* utilisent la prochaine clé. En raison de la latence réseau, de la livraison désordonnée et de l'usage de clés pour plusieurs *Éditeurs*, il est essentiel que l'*Abonné* prévoit une période de chevauchement au cours de laquelle les *NetworkMessages* reçus utilisent la clé précédente ou la clé suivante. Le *TimeToNextKey* et la *KeyLifetime* sont également utilisés pour calculer le délai jusqu'à ce que l'*Éditeur* et l'*Abonné* doivent rechercher de nouvelles clés.
KeyLifetime	Durée de vie d'une clé en millisecondes. Les clés renvoyées peuvent expirer plus tôt si celles-ci sont rejetées pour quelque raison que ce soit. La rotation non planifiée des clés est indiquée dans l'en-tête du *NetworkMessage* avant que la clé suivante soit utilisée afin d'offrir à l'*Abonné* un délai lui permettant de rechercher de nouvelles clés. Si le *CurrentTokenId* du message n'est pas reconnu, le récepteur doit de nouveau appeler cette *Méthode* pour obtenir de nouvelles clés.

Codes de résultat de méthode

ResultCode	Description
Bad_NotFound	Le *SecurityGroupId* est inconnu.
Bad_UserAccessDenied	L'appelant n'est pas autorisé à demander les clés pour le SecurityGroup.
Bad_SecurityModeInsufficient	Le canal de communication n'utilise pas de chiffrement.

8.5 Méthode GetSecurityGroup

Cette *Méthode* permet une recherche directe du *NodeId* d'un *Objet SecurityGroupType* à partir d'un *SecurityGroupId*. Elle est utilisée par un outil d'administration de sécurité pour obtenir l'*Objet SecurityGroup* à des fins de configuration des autorisations d'accès des clés.

Le *SecurityGroupId* est l'identificateur du *SecurityGroup* dans les *Éditeurs*, les *Abonnés* et le *Serveur* de clés. Cette *Méthode* renvoie le *NodeId* du *Nœud* d'*Objet SecurityGroup* correspondant qui fournit les options de configuration et de diagnostic pour un *SecurityGroup*.

Signature

```
GetSecurityGroup (
    [in]  String      SecurityGroupId
    [out] NodeId      SecurityGroupNodeId
    );
```

Argument	Description
SecurityGroupId	*SecurityGroupId* du *SecurityGroup* à rechercher.
SecurityGroupNodeId	*NodeId* de l'*Objet SecurityGroupType*.

Codes de résultat de méthode

ResultCode	Description
Bad_NoMatch	Le SecurityGroupId ne peut être trouvé sur le *Serveur*.

8.6 SecurityGroupType

Le *SecurityGroupType* est défini de manière formelle dans le Tableau 101.

Les *Autorisations* de l'*Objet SecurityGroupType* contrôlent l'accès aux clés de sécurité du *SecurityGroup* au moyen de la *Méthode GetSecurityKeys*. La *Méthode GetSecurityKeys* est définie en 8.4.

Tableau 101 – Définition de SecurityGroupType

Attribut	Valeur				
BrowseName	SecurityGroupType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de BaseObjectType défini dans l'IEC 62541-5.					
HasProperty	Variable	SecurityGroupId	Chaîne	PropertyType	Obligatoire
HasProperty	Variable	KeyLifetime	Durée	PropertyType	Obligatoire
HasProperty	Variable	SecurityPolicyUri	Chaîne	PropertyType	Obligatoire
HasProperty	Variable	MaxFutureKeyCount	UInt32	PropertyType	Obligatoire
HasProperty	Variable	MaxPastKeyCount	UInt32	PropertyType	Obligatoire

La *Propriété SecurityGroupId* comporte l'identificateur du *SecurityGroup* utilisé dans l'échange de clés des *Méthodes GetSecurityKeys* et *SetSecurityKeys* du *PubSubGroupType*.

La *Propriété KeyLifetime* définit la durée de vie d'une clé en millisecondes.