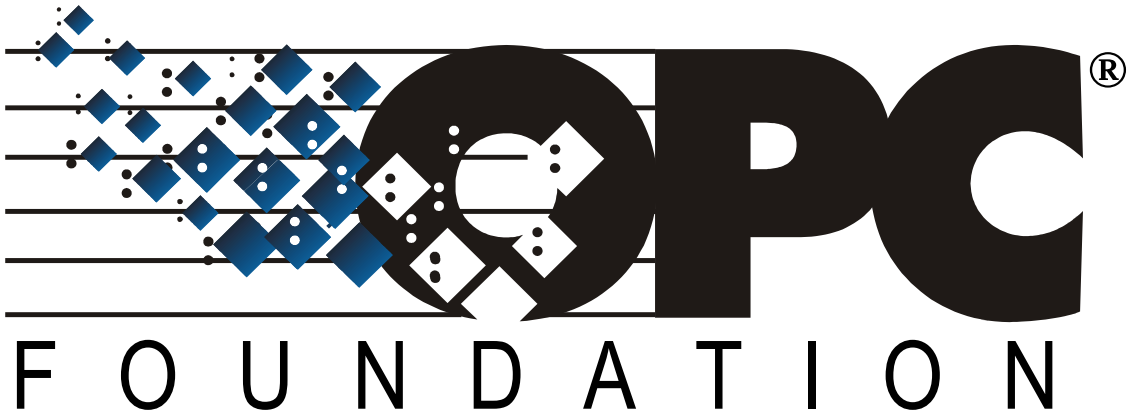

OPC Unified Architecture

Specification

Part 14: PubSub

Release 1.04

February 06, 2018

Specification Type:	Industry Standard Specification	Comments:	
Title:	OPC Unified Architecture Part 14 :PubSub	Date:	February 06, 2018
Version:	Release 1.04	Software:	MS-Word
		Source:	OPC UA Part 14 - PubSub Release 1.04 Specification.docx
Author:	OPC Foundation	Status:	Release

CONTENTS

FIGURES

TABLES

OPC FOUNDATION

UNIFIED ARCHITECTURE –

FOREWORD

This specification is the specification for developers of OPC UA applications. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of applications by multiple vendors that shall inter-operate seamlessly together.

AGREEMENT OF USE

GENERAL PROVISIONS

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of the State of Minnesota, excluding its choice or law rules.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, this specification.

ISSUE REPORTING

The OPC Foundation strives to maintain the highest quality standards for its published specifications, hence they undergo constant review and refinement. Readers are encouraged to report any issues and view any existing errata here: http://www.opcfoundation.org/errata

OPC Unified Architecture Specification

Part 14: PubSub

1 Scope

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

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

The specification consists of

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

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

2 Normative references

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

Part 1: OPC UA Specification: Part 1 – Concepts
http://www.opcfoundation.org/UA/Part1/

Part 2: OPC UA Specification: Part 2 – Security Model
http://www.opcfoundation.org/UA/Part2/

Part 3: OPC UA Specification: Part 3 – Address Space Model
http://www.opcfoundation.org/UA/Part3/

Part 4: OPC UA Specification: Part 4 – Services
http://www.opcfoundation.org/UA/Part4/

Part 5: OPC UA Specification: Part 5 – Information Model
http://www.opcfoundation.org/UA/Part5/

Part 6: OPC UA Specification: Part 6 – Mappings
http://www.opcfoundation.org/UA/Part6/

Part 7: OPC UA Specification: Part 7 – Profiles
http://www.opcfoundation.org/UA/Part7/

Part 8: OPC UA Specification: Part 8 – Data Access

http://www.opcfoundation.org/UA/Part8/

Part 12: OPC UA Specification: Part 12 – Discovery

http://www.opcfoundation.org/UA/Part12/

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

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

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

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

3 Terms, definitions and conventions

3.1 Terms and definitions

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

3.1.1
DataSetClass
template declaring the content of a *DataSet*

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

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

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

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

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

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

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

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

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

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

3.1.7
SubscribedDataSet
configuration for dispatching of received *DataSets*

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

3.2 Abbreviations and symbols

AMQP Advanced Message Queuing Protocol
AS Authorization Service
CA Certificate Authority
CRL Certificate Revocation List
CTL Certificate Trust List
HMI Human Machine Interface
IGMP Internet Group Management Protocol
MIME Multipurpose Internet Mail Extensions
MQTT MQ Telemetry Transport
MTU Maximum Transmission Unit
PCP Priority Code Point
QoS Quality of Service
SKS Security Key Service
STS Security Token Service
UA Unified Architecture
UADP UA Datagram Protocol
UDP User Datagram Protocol
URI Uniform Resource Identifier
URL Uniform Resource Locator
VID VLAN Identifier

4 Overview

4.1 Fields of application

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

The following are some example uses for *PubSub*:

- Configurable peer to peer communication between controllers and between controllers and HMIs. The peers are not directly connected and do not even need to know about the existence of each other. The data exchange often requires a fixed time-window; it may be point-to-point connection or data distribution to many receivers.

- Asynchronous workflows. For example, an order processing application can place an order on a message queue or an enterprise service bus. From there it can be processed by one or more workers.

- Logging to multiple systems. For example, sensors or actuators can write logs to a monitoring system, an HMI, an archive application for later querying, and so on.

- OPC UA *Servers* representing services or devices can stream data to applications hosted in the cloud. For example, backend servers, big data analytics for system optimization and predictive maintenance.

4.2 Abstraction layers

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

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

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

4.3 Decoupling by use of middleware

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

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

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

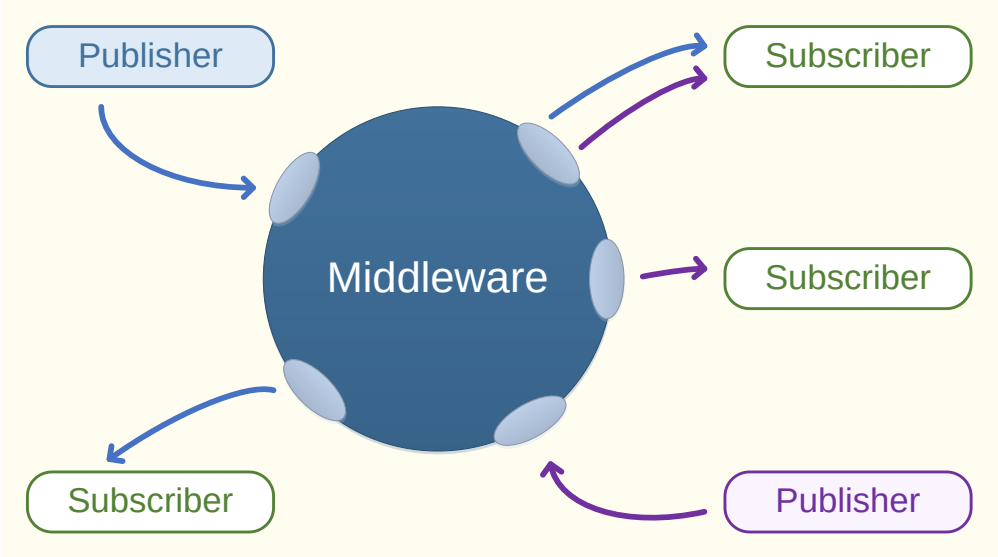

Figure 1 – Publish Subscribe Model Overview

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

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

4.4 Synergy of models

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

5 PubSub Concepts

5.1 Introduction

This clause describes the general OPC UA *PubSub* concepts.

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

The abstract communication parameters are described in clause 6.

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

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

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

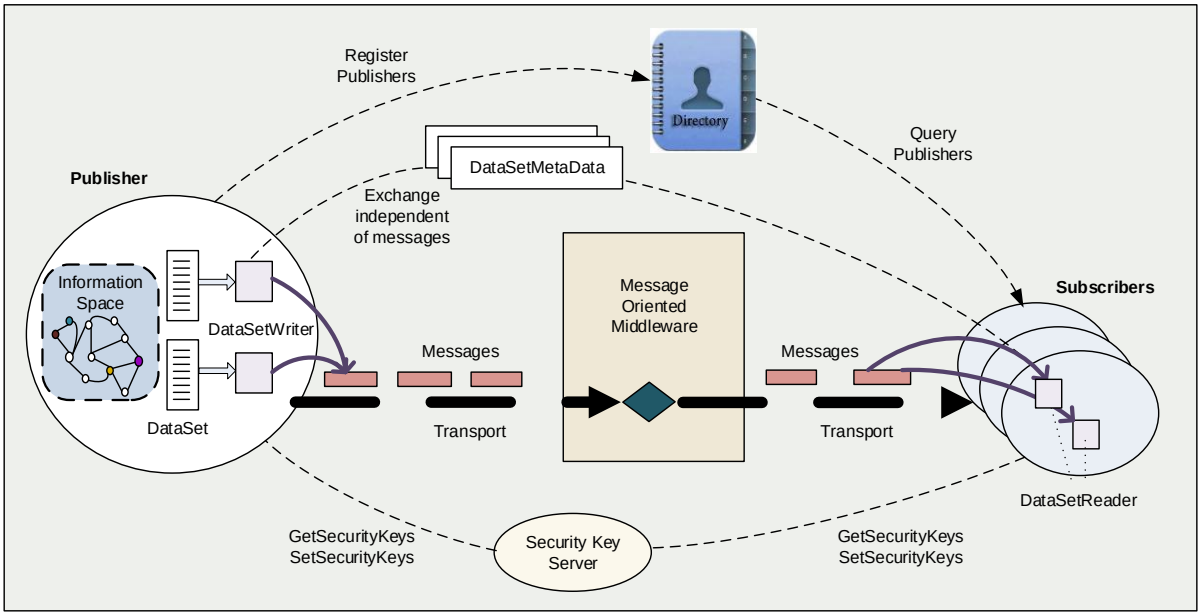

Figure 2 – Publisher and Subscriber entities

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

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

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

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

Note 1: The PubSub directory is an optional entity that allows *Publishers* to advertise their *PublishedDataSets* and their communication parameters. This directory functionality is planned for a future release of this specification.

5.2 DataSet

5.2.1 General

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

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

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

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

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

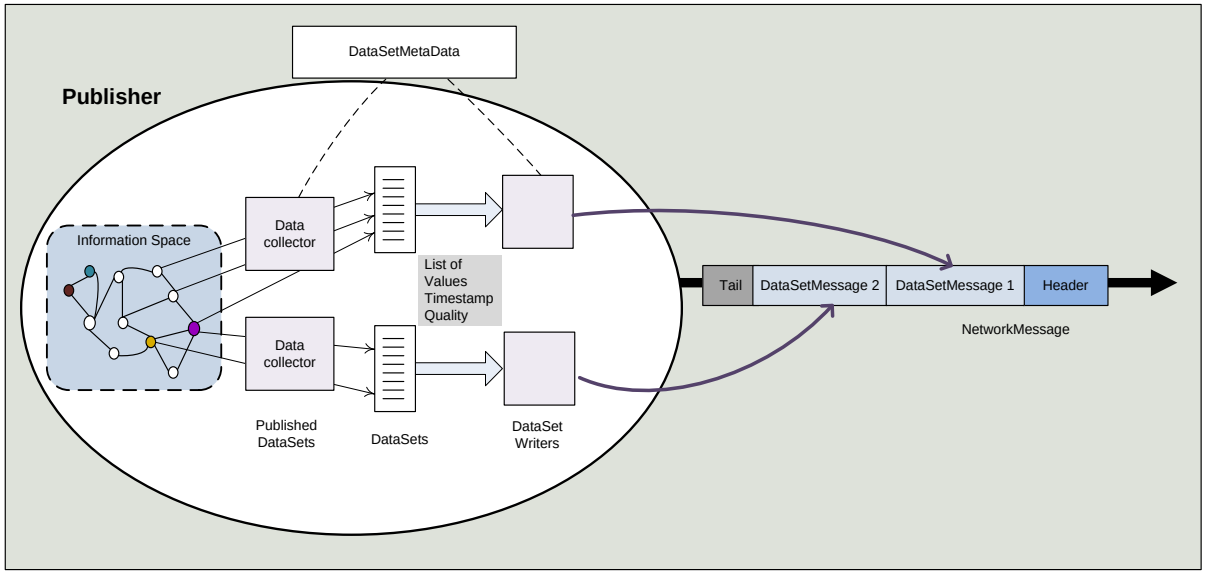

Figure 3 – DataSet in the process of publishing

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

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

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

5.2.2 DataSetClass

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

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

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

5.2.3 DataSetMetaData

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

The *DataSetMetaDataType* is defined in 6.2.2.1.2.

Example description (simplified, in pseudo-language):

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

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

Each *DataSetMessage* also includes the version of the *DataSetMetaData* that it complies with. This allows *Subscribers* to verify if they have the corresponding *DataSetMetaData*. The related *ConfigurationVersionDataType* is defined in 6.2.2.1.5.

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

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

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

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

- The *DataSetMetaData* is sent as a *NetworkMessage* from the *Publisher* to the *Subscriber* before *DataSetMessages* with changed content are sent. The used *Message Oriented Middleware* should ensure reliable delivery of the message. The mapping for the *Message Oriented Middleware* defines a way for the *Subscriber* to request the *DataSetMetaData*. The *Subscriber* goes to an error state if it has not

received the new *DataSetMetaData* that matches the *ConfigurationVersion* of the received *DataSetMessage*.

- The *Subscriber* is automatically updated via the OPC UA configuration *Methods* defined in the *PubSub* configuration model when the *DataSet* in the *Publisher* is updated.
- The *Subscriber* is an OPC UA *Client* and is able to obtain the update from the *Publisher* or a configuration server via the information exposed by the *PubSub* configuration model.
- The *Subscriber* is updated with product specific configuration means when the *DataSet* in the *Publisher* is changed.

5.3 Messages

5.3.1 General

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

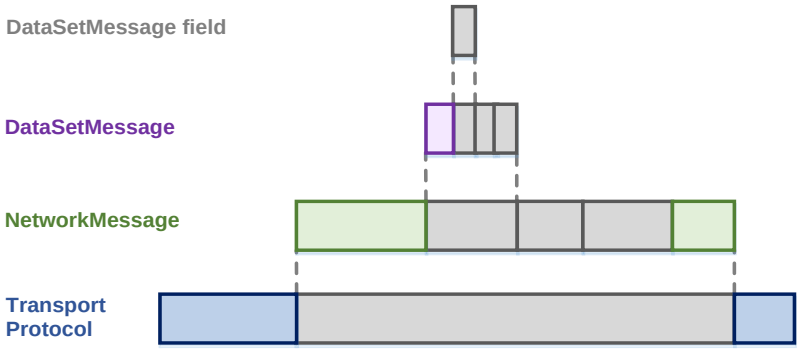

Figure 4 – OPC UA PubSub Message Layers

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

Following is an abstract definition of *DataSetMessage* and *NetworkMessage*. The concrete structure depends on the message mapping and is described in 7.2.

5.3.2 DataSetMessage field

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

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

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

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

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

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

5.3.3 DataSetMessage

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

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

Additional header information includes:

DataSetWriterId	Identifies the *DataSetWriter* and indirectly the *PublishedDataSet.*
Sequence number	A number that is incremented for each *DataSetMessage*. Can be used to verify the ordering and to detect missing messages.
Timestamp	A timestamp describing when the data in this *DataSetMessage* was obtained.
Version	Version information about the configuration of the *DataSetMetaData.*
Status	Status information about the data in this *DataSetMessage.*
Keep alive	When no *DataSetMessages* are sent for a configured time period, a keep alive *DataSetMessage* is sent to signal the *Subscribers* that the *Publisher* is still alive.

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

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

5.3.4 NetworkMessage

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

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

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

5.3.5 Message Security

Message security in *PubSub* concerns integrity and confidentiality of the published message payload. The level of security can be:

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

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

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

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

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

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

5.3.6 Transport Security

The transport security is specific to the transport protocol mapping.

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

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

5.3.7 SecurityGroup

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

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

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

5.4 Entities

5.4.1 Publisher

5.4.1.1 General

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

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

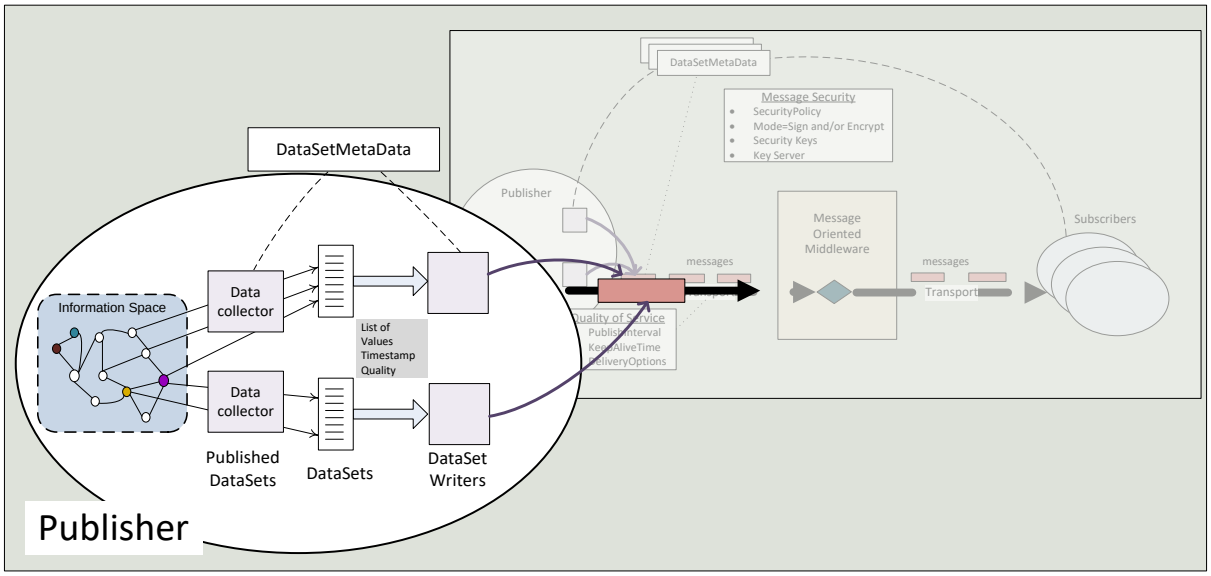

Figure 5 – Publisher details

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

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

5.4.1.2 Message sending

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

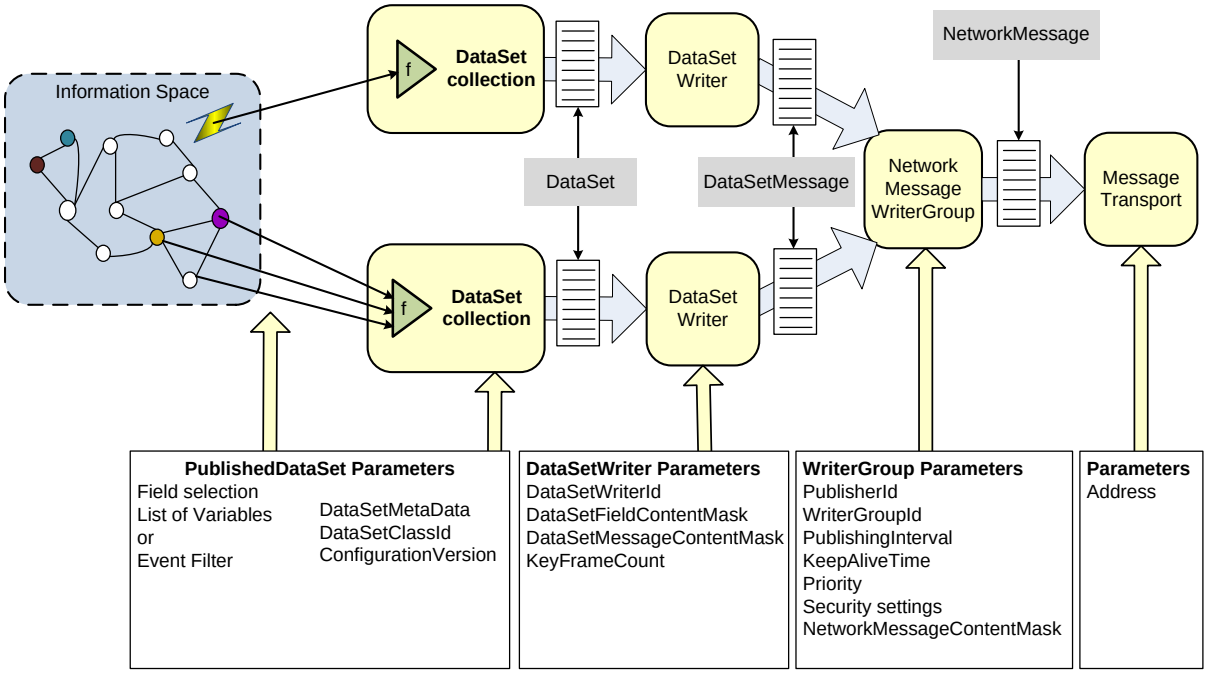

Figure 6 – Publisher message sending sequence

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

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

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

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

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

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

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

5.4.2 Subscriber

5.4.2.1 General

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

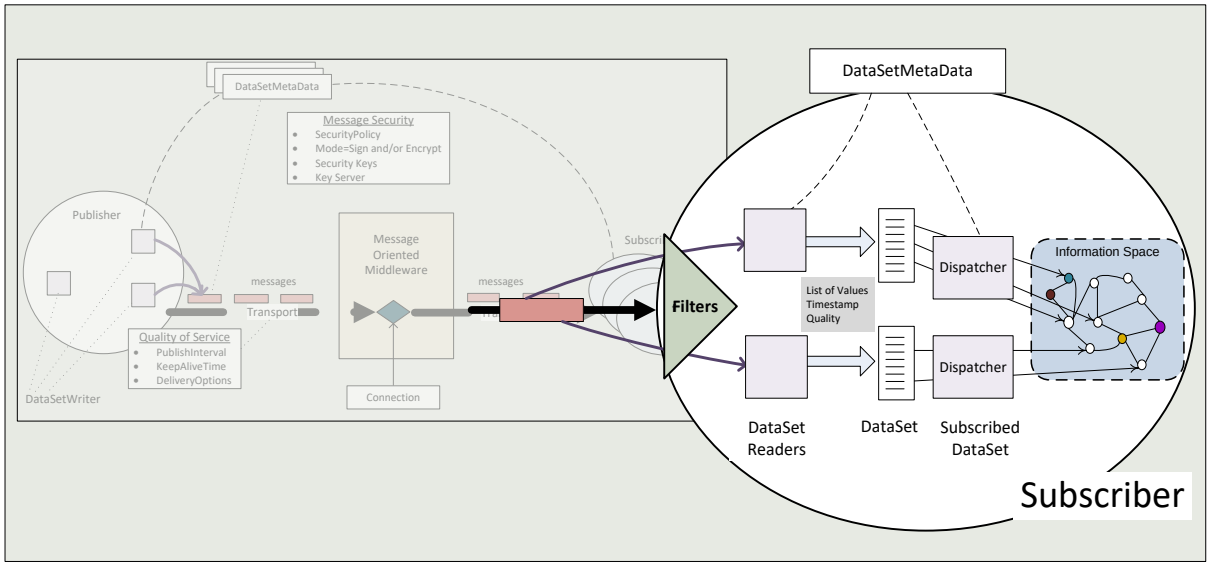

Figure 7 – Subscriber details

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

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

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

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

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

5.4.2.2 Message reception

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

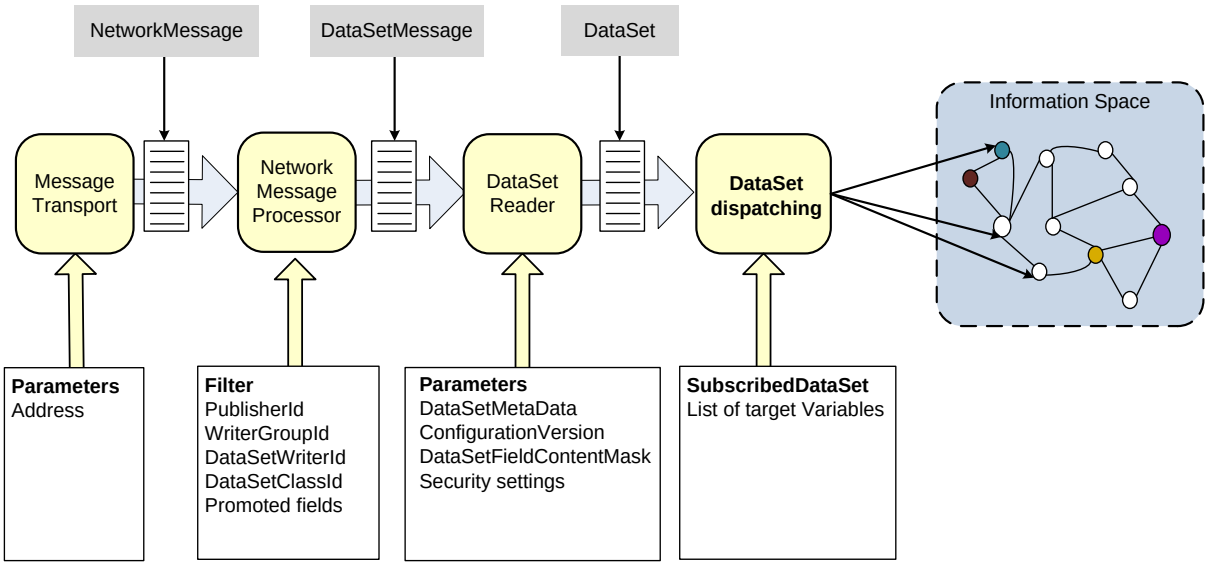

Figure 8 – Subscriber message reception sequence

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

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

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

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

5.4.3 Security Key Service

5.4.3.1 General

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

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

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

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 Part 3. The *User Authorization* model defines the mapping of identities to *Roles* and the mechanism to set *Permissions* for *Roles* on a *Node*. The *Permissions* on a *SecurityGroup Object* is used to determine if a *Role* has access to the keys for the *SecurityGroup*.

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

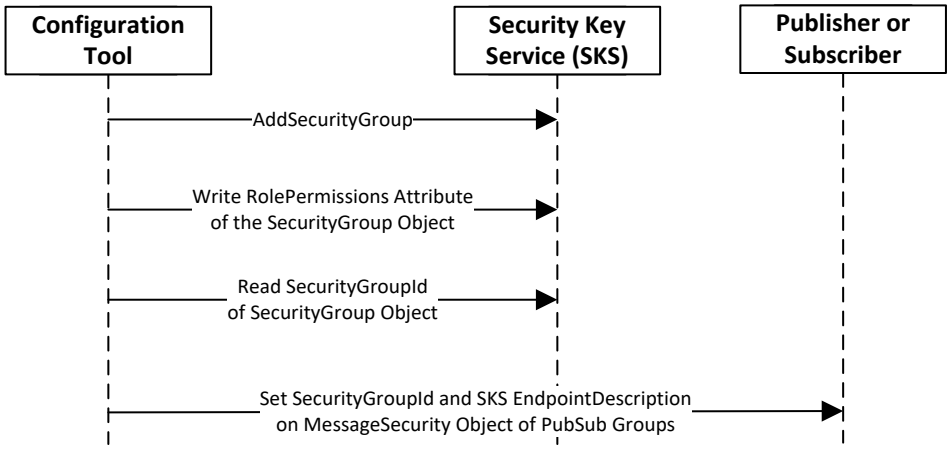

Figure 9 – SecurityGroup Management Sequence

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

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

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

5.4.3.3 Key Acquisition Handshakes

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

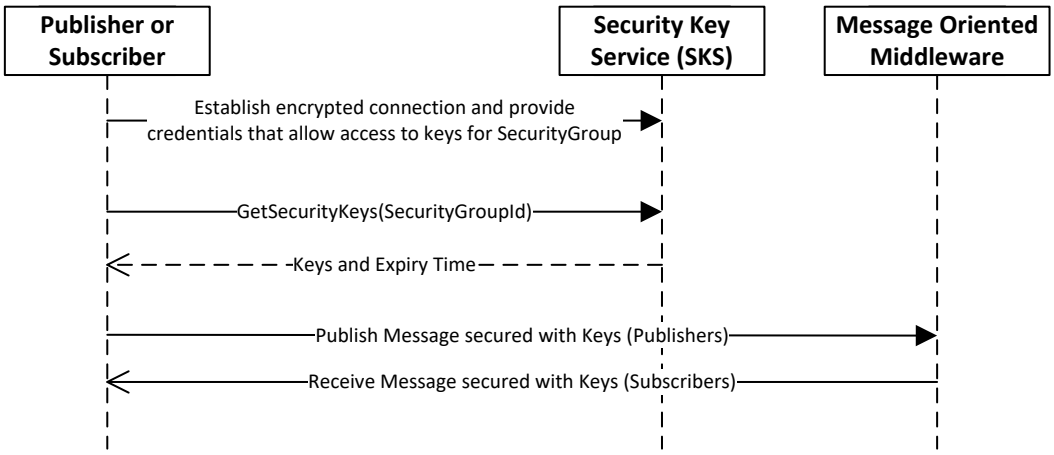

Figure 10 – Handshake used to pull keys from SKS

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 Part 4.

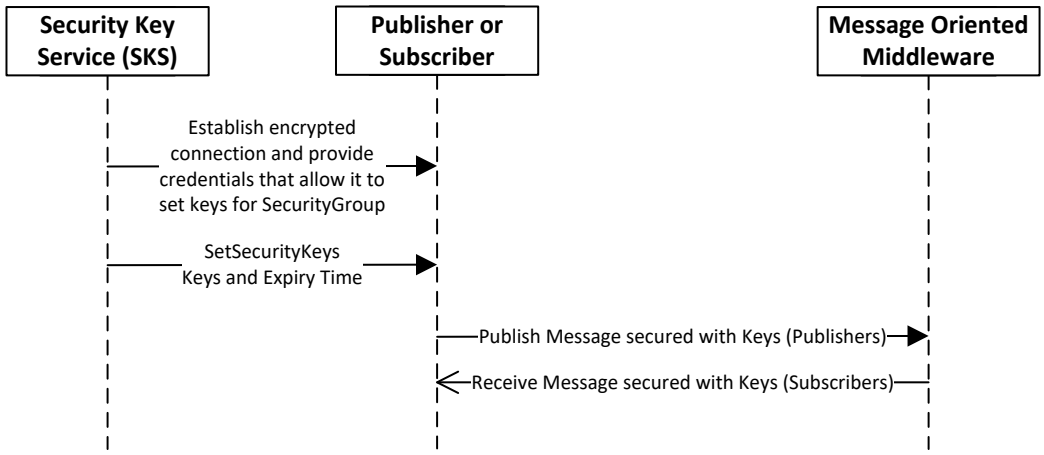

Figure 11 – Handshake used to push keys to Publishers and Subscribers

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.

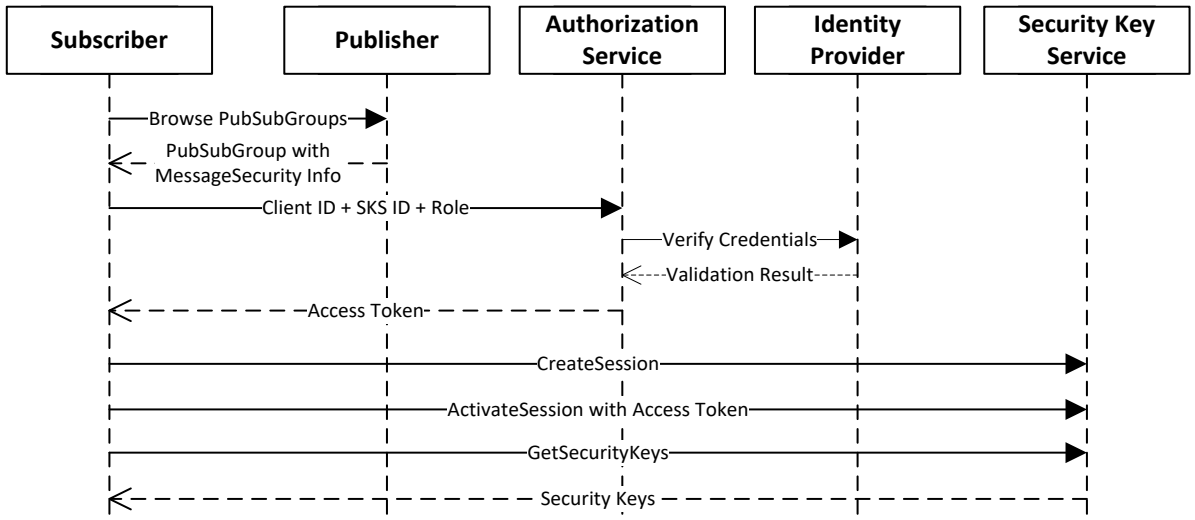

Figure 12 – Handshake with a Security Key Service

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 specification is any infrastructure supporting sending and receiving *NetworkMessage*s 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 part 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 *NetworkMessage*s 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.

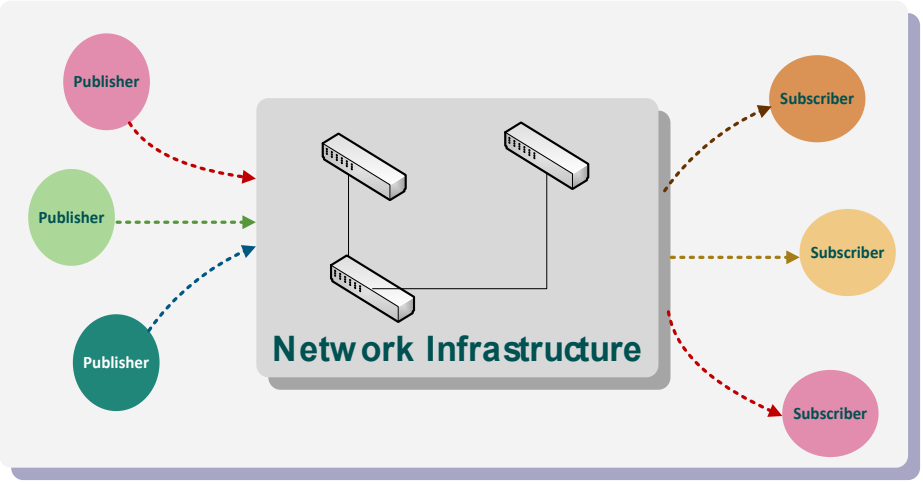

Figure 13 – PubSub using network infrastructure

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*.

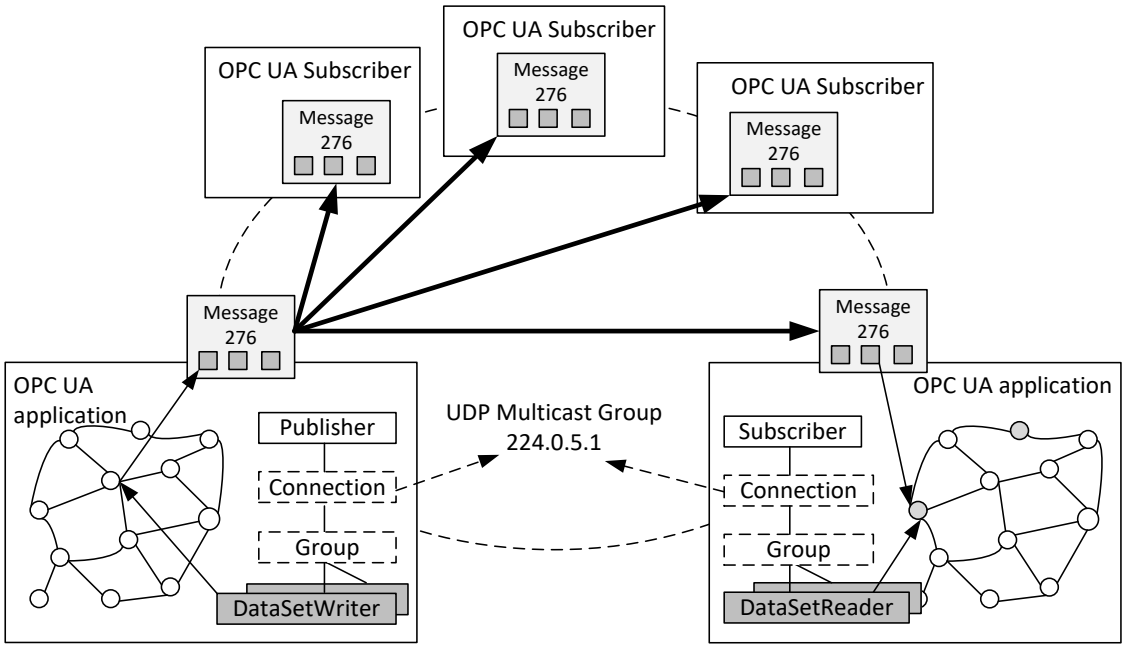

Figure 14 – UDP Multicast Overview

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 *NetworkMessage*s to the right applications based on business criteria ("queue name", "routing key", "topic" etc.) rather than on physical topology (IP addresses, host names).

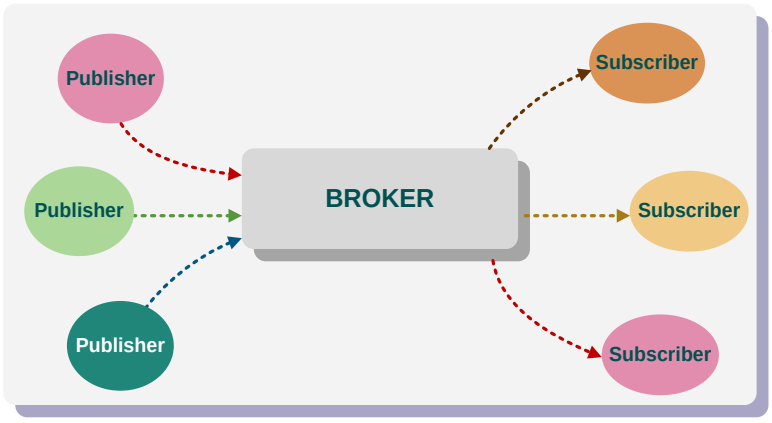

Figure 15 – PubSub using broker

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

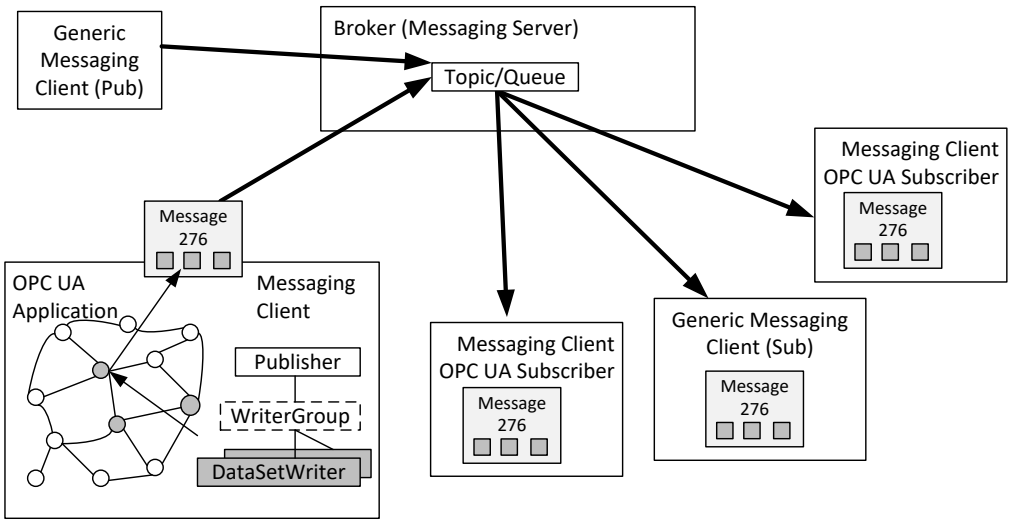

Figure 16 – Broker Overview

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

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

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

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

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

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

6 PubSub Communication Parameters

6.1 Overview

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

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

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

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

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

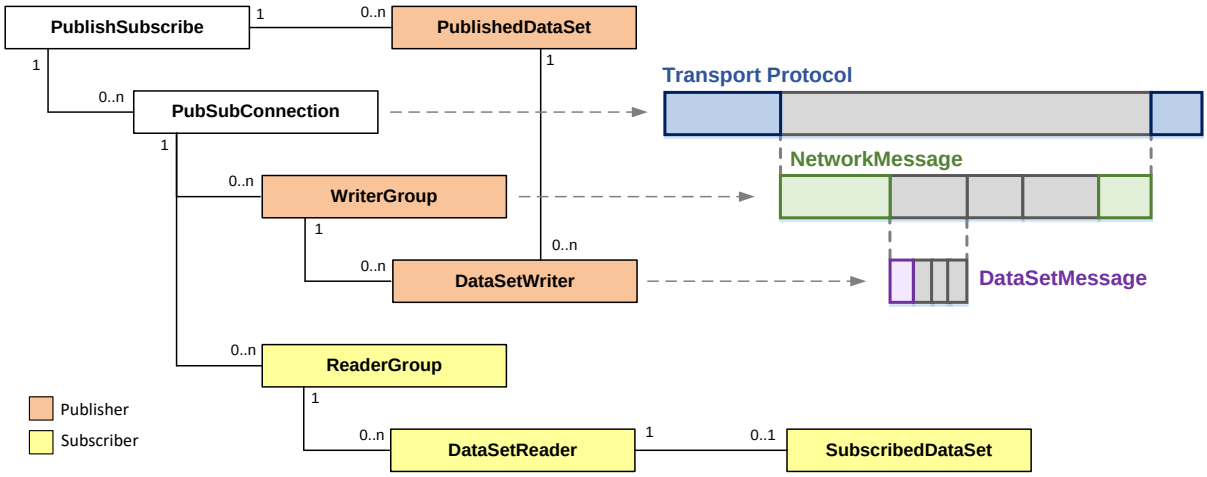

Figure 17 – PubSub Component Overview

The figure shows the following components:

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

- *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.

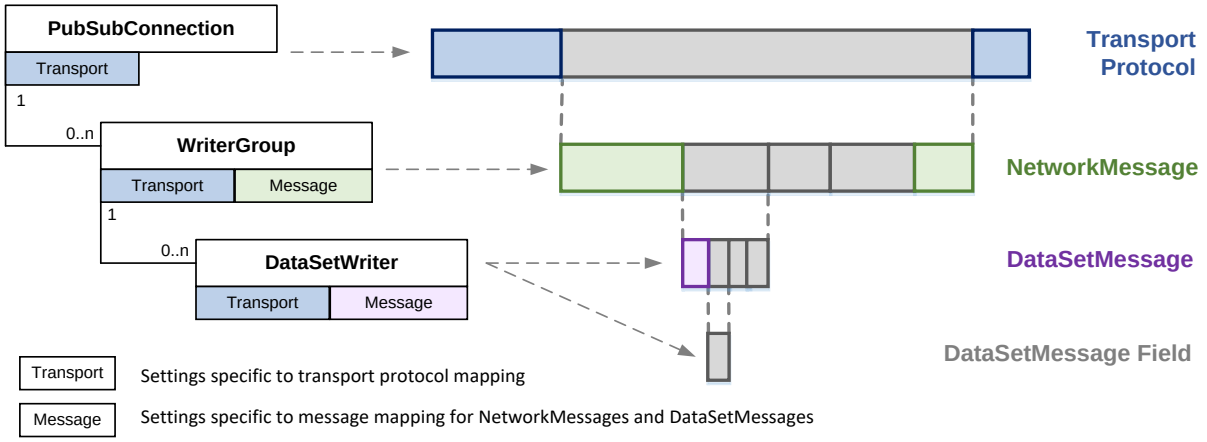

Figure 18 – PubSub Mapping Specific Parameters Overview

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.

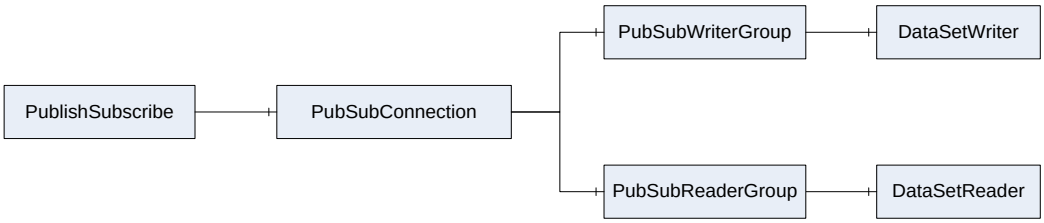

Figure 19 – PubSub Component State Dependencies

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

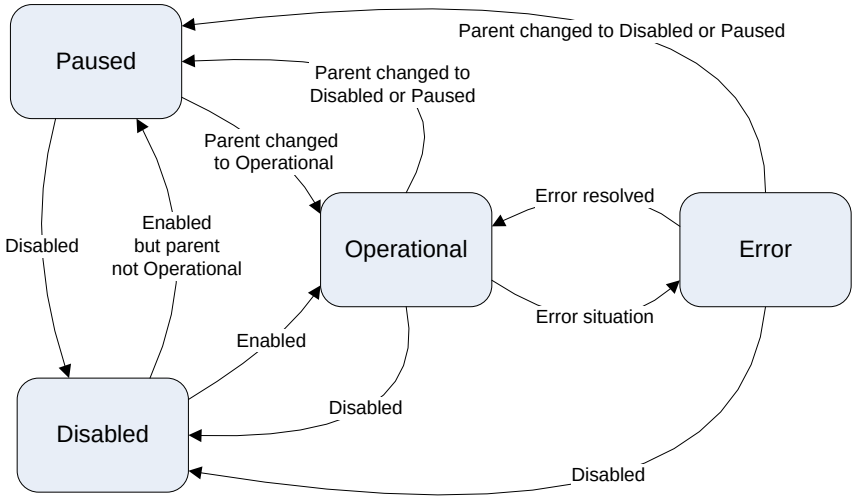

Figure 20 – PubSubState State Machine

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 Part 6. All data types are transferred in *DataSetMessages* as one of the built-in data types. In most cases the identifier of the *DataType NodeId* matches the built-in type. The following special cases must be handled in addition: (1) Abstract types always have the built-in type *Variant* since they can result in different concrete types in a *DataSetMessage*. The *dataType* field may provide additional restrictions e.g. if the abstract type is *Number*. Abstract types shall not be used if the field is represented as *RawData* set by the *DataSetFieldContentMask* defined in 6.2.3.1. (2) *Enumeration DataTypes* are encoded as *Int32*. The *Enumeration* strings are defined through a *DataType* referenced through the *dataType* field. (3) *Structure* and *Union DataTypes* are encoded as *ExtensionObject*. The encoding rules are defined through a *DataType* referenced through the *dataType* field. (4) *DataTypes* derived from built-in types have the *BuiltInType* of the corresponding base DataType. The concrete subtype is defined through the *dataType* field. (5) *OptionSet DataTypes* are either encoded as one of the concrete *UInteger DataTypes* or as an instance of an *OptionSetType* in an *ExtensionObject*.
dataType	NodeId	The *NodeId* of the *DataType* of this field. If the *DataType* is an *Enumeration* or an *OptionSet*, the semantic of the *Enumeration DataType* is provided through the *enumDataTypes* field of the

Name	Type	Description
		DataSetMetaData. If the *DataType* is a *Structure* or *Union*, the encoding and decoding description of the *Structure DataType* is provided through the *structureDataTypes* field of the *DataSetMetaData*.
valueRank	Int32	Indicates whether the *dataType* is an array and how many dimensions the array has. It may have the following values: *n* > 1: the *dataType* is an array with the specified number of dimensions. OneDimension (1): The *dataType* is an array with one dimension. OneOrMoreDimensions (0): The *dataType* is an array with one or more dimensions. Scalar (−1): The *dataType* is not an array. Any (−2): The *dataType* can be a scalar or an array with any number of dimensions. ScalarOrOneDimension (−3): The *dataType* can be a scalar or a one dimensional array. NOTE All *DataTypes* are considered to be scalar, even if they have array-like semantics like *ByteString* and *String*.
arrayDimensions	UInt32[]	This field specifies the maximum supported length of each dimension. If the maximum is unknown the value shall be 0. The number of elements shall be equal to the value of the v*alueRank field*. This field shall be null if v*alueRank* ≤ 0. The maximum number of elements of an array transferred on the wire is 2147483647 (max Int32). It is the total number of elements in all dimensions based on the UA Binary encoding rules for arrays.
maxStringLength	UInt32	If the *dataType* field is a *String* or *ByteString* then this field specifies the maximum supported length. If the maximum is unknown the value shall be 0. If the dataType field is not a *String* or *ByteString* the value shall be 0. If the *valueRank* is greater than 0 this field applies to each element of the array.
dataSetFieldId	Guid	The unique ID for the field in the *DataSet*. The ID is generated when the field is added to the list. A change of the position of the field in the list shall not change the ID.
properties	KeyValuePair[]	List of *Property* values providing additional semantic for the field. If at least one *Property* value changes, the *MajorVersion* of the *ConfigurationVersion* shall be updated. If the *Property* is *EngineeringUnits*, the unit of the *Field Value* shall match the unit of the *FieldMetaData*. The *KeyValuePair* DataType is defined in Part 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 Part 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 Part 4. To assure interoperability, the *Subscriber* has to use *DataSetMetaData* for decoding with a *MajorVersion* that matches the *MajorVersion* in *DataSetMessages* sent by the *Publisher*. Removing fields from the *DataSet* content, reordering fields, adding fields in between other fields or a DataType change in fields shall result in an update of the *MajorVersion*. If at least one *Property* value of a *DataSetMetaData* field changes, the *MajorVersion* shall be updated. There can be situations where older configurations of a *Publisher* are loaded and changed with product specific configuration tools. In this case the *MajorVersion* shall be updated if the configuration tool is not able to verify if the change only extends the configuration and does not change the existing content. Additional criteria for changing *MajorVersion* or *MinorVersion* are defined in this specification.
minorVersion	VersionTime	The *MinorVersion* reflects the time of the last change. Only the *MinorVersion* shall be updated if fields are added at the end of the *DataSet* content. If the *MajorVersion* version is updated, the *MinorVersion* is updated to the same value as *MajorVersion*.

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 standard organisation 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 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 Part 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 Part 3. The *IntegerId DataType* is defined in Part 4. The *IntegerIds* for the *Attributes* are defined in Part 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 Part 4) Percent_2 – PercentDeadband (This type is specified in Part 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 Part 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 Part 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 Part 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 Part 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 *Publisher* side and the decoded DataSet field on the Subscriber side. The *DataSetFieldContentMask* controls the representation of the DataSet fields in a *DataSetMessage*.

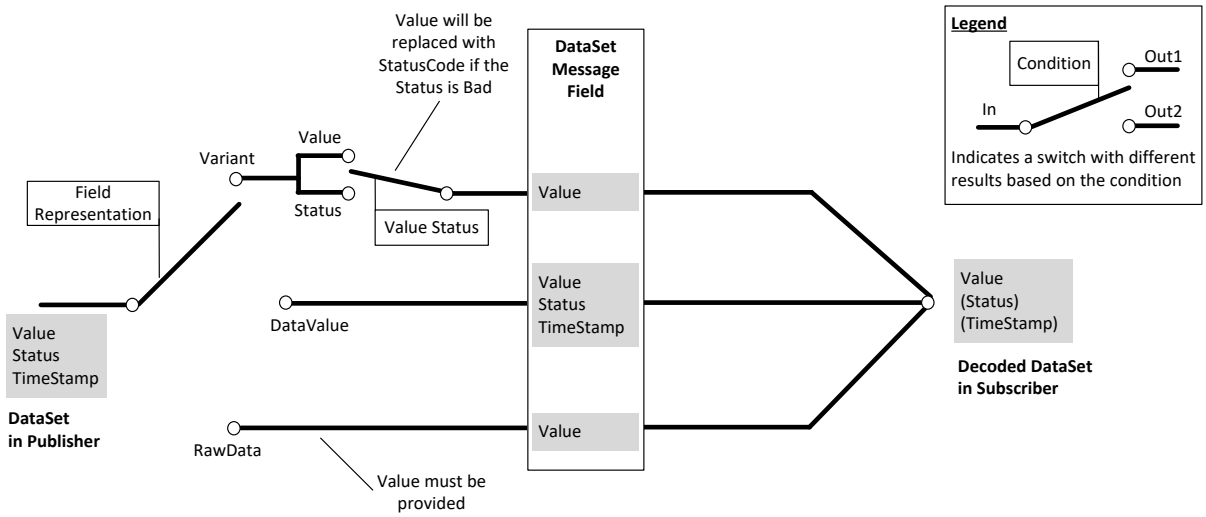

Figure 21 – PubSub Information Flow dependency to field representation

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(4)**		**Value**	**Status(4)**	**Value**	**Status(4)**
Value 1	Good_*	Variant	Value 1	N/A (1)	Value 1	N/A (1)
Value 1	Uncertain_*		Value 1		Value 1	
Null	Bad_*		Bad_* (1)		Null	Bad_*
Value 1	Good_*	DataValue	Value 1	Good_*	Value 1	Good_*
Value 1	Uncertain_*		Value 1	Uncertain_*	Value 1	Uncertain_*
Null	Bad_*		Null	Bad_*	Null	Bad_*
Value 1	Good_*	RawData	Value 1	N/A	Value 1	N/A
Value 1	Uncertain_*		Value 1(2)		Value 1	
Null	Bad_*		DefaultValue(3)		DefaultValue	

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

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 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 Part 5 and consists of a *QualifiedName* and a value of *BaseDataType*.

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

6.2.3.5 DataSetWriter Structure

6.2.3.5.1 DataSetWriterDataType

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

Table 17 – DataSetWriterDataType Structure

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

6.2.3.5.2 DataSetWriterTransportDataType

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

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

Table 18 – DataSetWriterTransportDataType Definition

Attributes	Value			
BrowseName	DataSetWriterTransportDataType			
IsAbstract	True			
References	**NodeClass**	**BrowseName**	**IsAbstract**	**Description**
Subtype of Structure defined in Part 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 Part 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 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 Part 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 Part 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. Part 4 describes *UserTokenPolicies* in more detail.

6.2.4.5 MaxNetworkMessageSize

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

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

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

6.2.4.6 GroupProperties

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

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

6.2.4.7 PubSubGroup Structure

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

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 Part 5.				
HasSubtype	DataType	WriterGroupDataType	FALSE	Defined in 6.2.5.6.1.
HasSubtype	DataType	ReaderGroupDataType	FALSE	Defined in 6.2.7.2.1.

6.2.5 WriterGroup Parameters

6.2.5.1 WriterGroupId

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

6.2.5.2 PublishingInterval

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

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

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

6.2.5.3 KeepAliveTime

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

6.2.5.4 Priority

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

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

The lowest priority is zero and the highest is 255.

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 Part 3 for more detail on *LocaleId*.

6.2.5.6 WriterGroup Structures

6.2.5.6.1 WriterGroupDataType

This *Structure DataType* is used to represent the configuration parameters for *WriterGroups*. It is a subtype of *PubSubGroupDataType* defined in 0.

The *WriterGroupDataType* is formally defined in Table 22.

Table 22 – WriterGroupDataType Structure

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

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

Table 23 – WriterGroupDataType Definition

Attributes	Value		
BrowseName	WriterGroupDataType		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**IsAbstract**
Subtype of PubSubGroupDataType defined in 0.			

6.2.5.6.2 WriterGroupTransportDataType

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

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

Table 24 – WriterGroupTransportDataType Definition

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

6.2.5.6.3 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 Part 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 Part 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* specifies additional properties for the configured connection. The *KeyValuePair* type is defined in Part 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 versions of this specification 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 sections 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 Part 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 Part 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 sections 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 sections 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 0.			

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 Part 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 Part 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 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.

6.2.8.8 SecurityGroupId

The parameter is defined in 6.2.4.3.

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

6.2.8.9 SecurityKeyServices

The parameter is defined in 6.2.4.4.

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

6.2.8.10 DataSetReaderProperties

The *DataSetReaderProperties* parameter is an array of *DataType KeyValuePair* that specifies additional properties for the configured *DataSetReader*. The *KeyValuePair DataType* is defined in Part 5 and consists of a *QualifiedName* and a value of *BaseDataType*.

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

6.2.8.11 DataSetReader Structure

6.2.8.11.1 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.
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.
securityMode	MessageSecurityMode	Defined in 6.2.8.7.
securityGroupId	String	Defined in 6.2.8.8.
securityKeyServices	EndpointDescription[]	Defined in 6.2.8.9.
dataSetReaderProperties	KeyValuePair[]	Defined in 6.2.8.10.
transportSettings	DataSetReaderTransportDataType	Transport specific DataSetReader parameters. The abstract base type is defined in 6.2.8.11.2. The concrete subtypes are defined in the sections for transport mapping specific parameters
messageSettings	DataSetReaderMessageDataType	DataSetMessage mapping specific DataSetReader parameters. The abstract base type is defined in 6.2.8.11.3. The concrete subtypes are defined in the sections for message mapping specific parameters.
subscribedDataSet	SubscribedDataSetDataType	The SubscribedDataSet specific parameters. The abstract base type and the concrete subtypes are defined 6.2.9.

6.2.8.11.2 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 Part 5.				
HasSubtype	DataType	BrokerDataSetReaderTransportDataType	FALSE	Defined in 6.4.2.4.6.

6.2.8.11.3 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 Part 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 Part 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 Part 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* where to write the received *DataSetMessage* field value to.
attributeId	IntegerId	Id of the *Attribute* to write e.g. the *Value Attribute*. This shall be a valid *AttributeId*. The Attributes are defined in Part 3. The *IntegerId DataType* is defined in Part 4. The *IntegerIds* for the *Attributes* are defined in Part 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 Part 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 useable 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*.

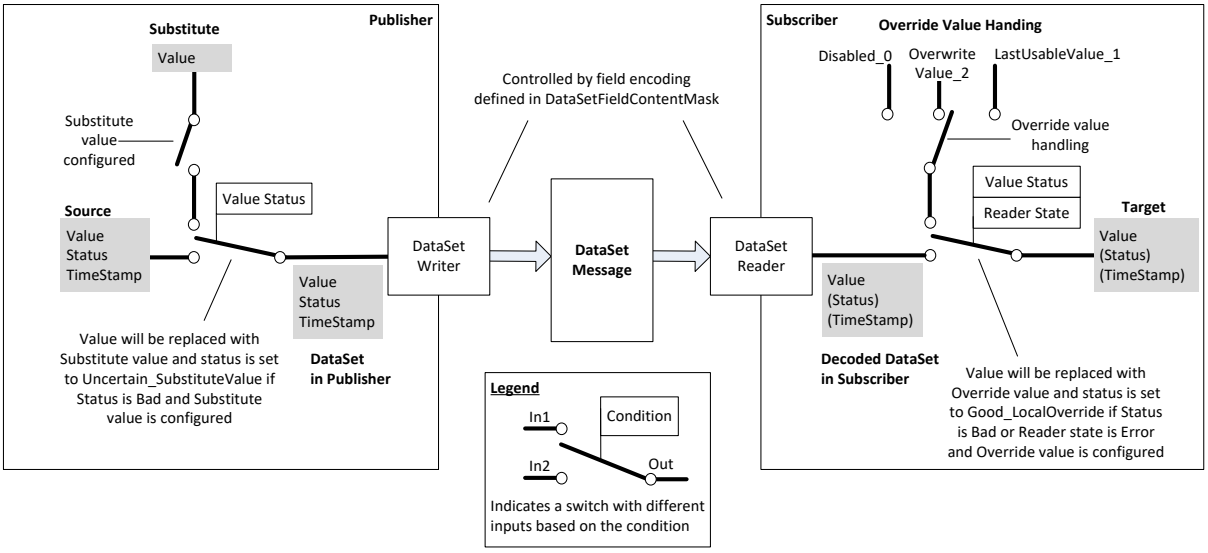

Figure 22 – PubSub Information Flow

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[(1)]**		**Value**	**Status[(1)]**
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_*
Note 1: 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[(1)]**				**Value**	**Status[(1)]**
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[(2)]	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[(2)]	Uncertain_LastUsableValue
		Disabled_0	Null		Null	Bad_OutOfService
		OverrideValue_2	Value 2	Error_3	Value 2	Good_LocalOverride
		LastUsableValue_1	Null		LastValue[(2)]	Uncertain_LastUsableValue
		Disabled_0	Null		Null	Bad_NoCommunication
Note 1: 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. Note 2: 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*.

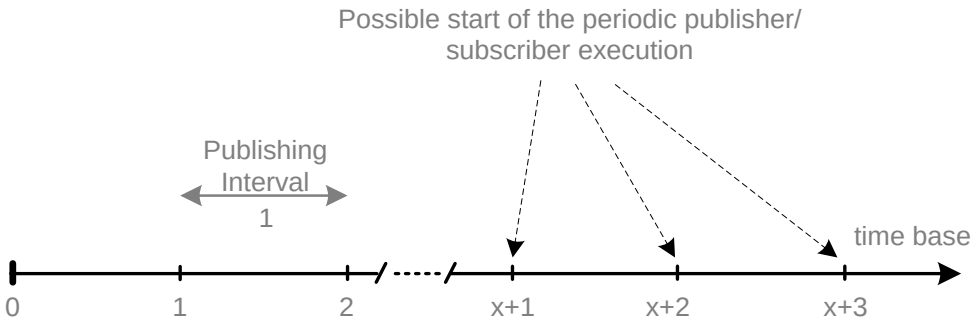

Figure 23 – Start of the periodic publisher execution

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.

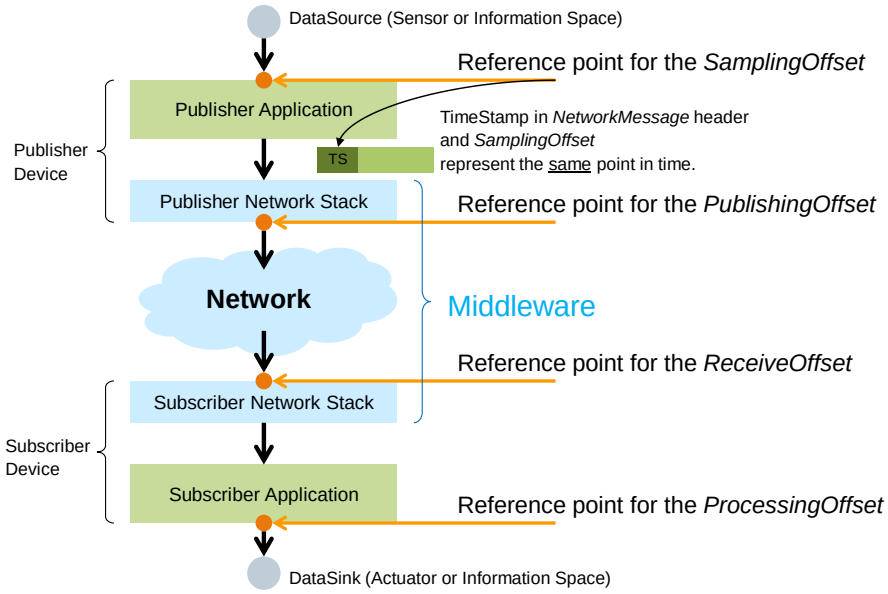

Figure 24 – Timing offsets in a PublishingInterval

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 Part 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* has to 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 Part 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*.

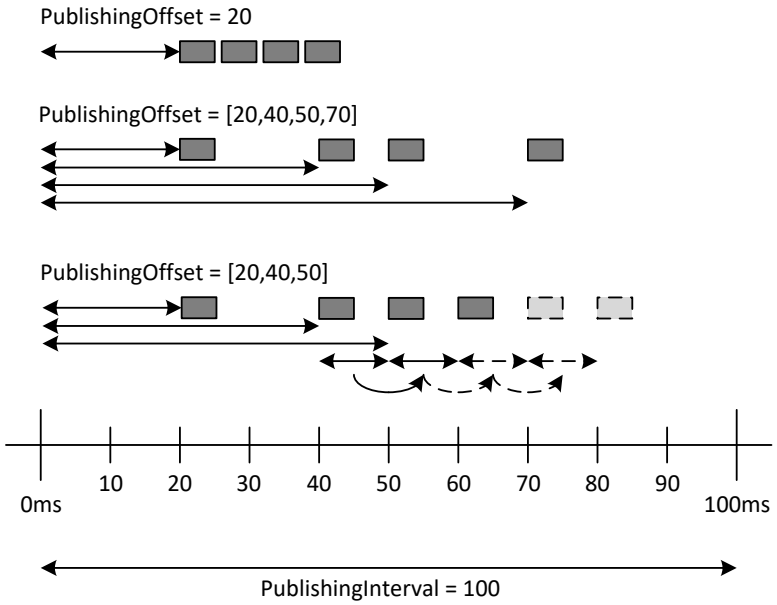

Figure 26 – PublishingOffset options for multiple *NetworkMessages*

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 *NetworkMessage*s are sent immediately after each other. If more *NetworkMessage*s are available for sending than offset values in the array, the offset for the remaining *NetworkMessage*s are 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 6.2.5.6.3.

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 1: 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 Part 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 1 to entry: 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 65535.

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 must be processed by the application in the *Subscriber*.

The different timing offsets inside a *PublishingInterval* cycle on *Publisher* and *Subscriber* side 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 6.2.8.11.3.

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* are ignored (see 6.3.2.2.1).
SingleDataSetMessage	2	Each JSON *NetworkMessage* contains only one *DataSetMessage*.
PublisherId	3	The *PublisherId* is included in the *NetworkMessage*s.
DataSetClassId	4	The *DataSetClassId* is included in the *NetworkMessage*s.
ReplyTo	5	The *ReplyTo* is included in the *NetworkMessage*s.

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 Part 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 6.2.5.6.3.

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	1	If this flag is set, a DataSetWriterId shall be included in the *DataSetMessage* header.
MetaDataVersion	2	If this flag is set, the *ConfigurationVersion* is included in the *DataSetMessage* header.
SequenceNumber	3	If this flag is set, the DataSetMessageSequenceNumber is included in the DataSetMessage header.
Timestamp	4	If this flag is set, a timestamp shall be included in the *DataSetMessage* header.
Status	5	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 Part 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 6.2.8.11.3.

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 6.2.6.4.

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 6.2.5.6.2.

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 Part 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 6.2.6.4.

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 Part 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 71.

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 must 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 6.2.5.6.2.

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 Part 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 Data Type 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 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 meta data 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 Part 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 6.2.8.11.2.

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

This clause 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 tranposrt 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 Part 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 Part 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 Part 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.

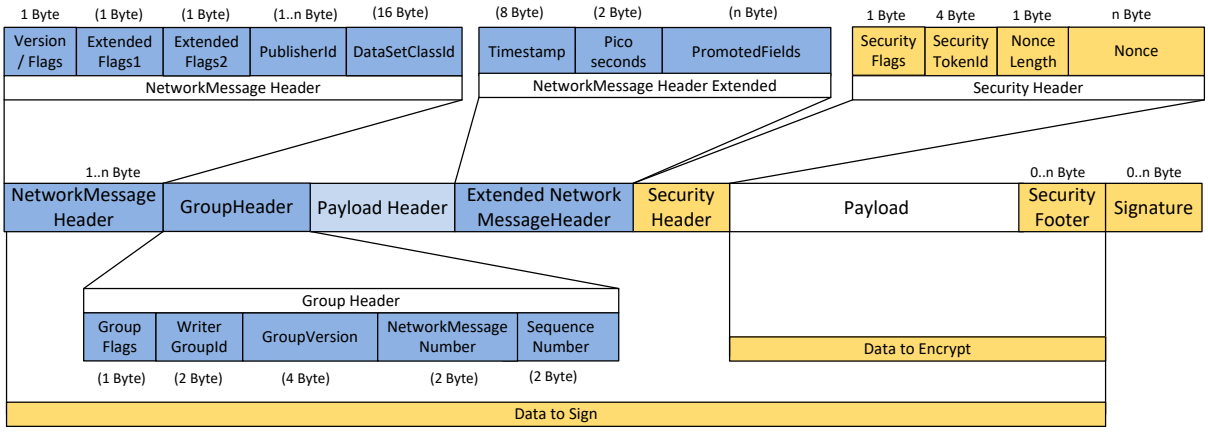

Figure 27 – UADP NetworkMessage

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.
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.

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 0-3. 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 e-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 4 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*.
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 must 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, that 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 2-5.
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 of 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 4294967295 to 0). To determine whether a received *NetworkMessages* is newer than the last processed *NetworkMessages* the following formula shall be used: (4294967295 + received sequence number – last processed sequence number) modulo 4294967296. Results below 1073741824 indicate that the received *NetworkMessages* is newer than the last processed *NetworkMessages* and the received *NetworkMessages* is processed. Results above 3221225472 indicate that the received message is older (or same) than the last processed *NetworkMessages* and the received *NetworkMessages* 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 the 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 Part 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 has 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 *Subscribers* 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 received if *ChunkOffset* plus the size of the current chunk equals *TotalSize*. The reassembled *NetworkMessage* payload can be processed after all 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 on *NetworkMessage*.

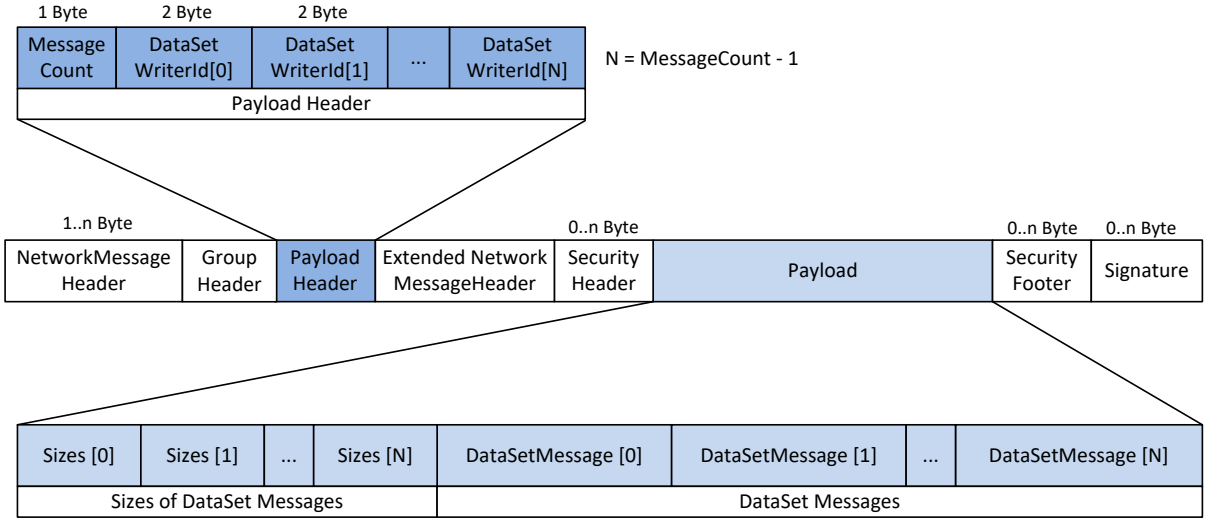

Figure 28 – UADP DataSet Payload

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 *DataSetMessages* 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 65535, 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.

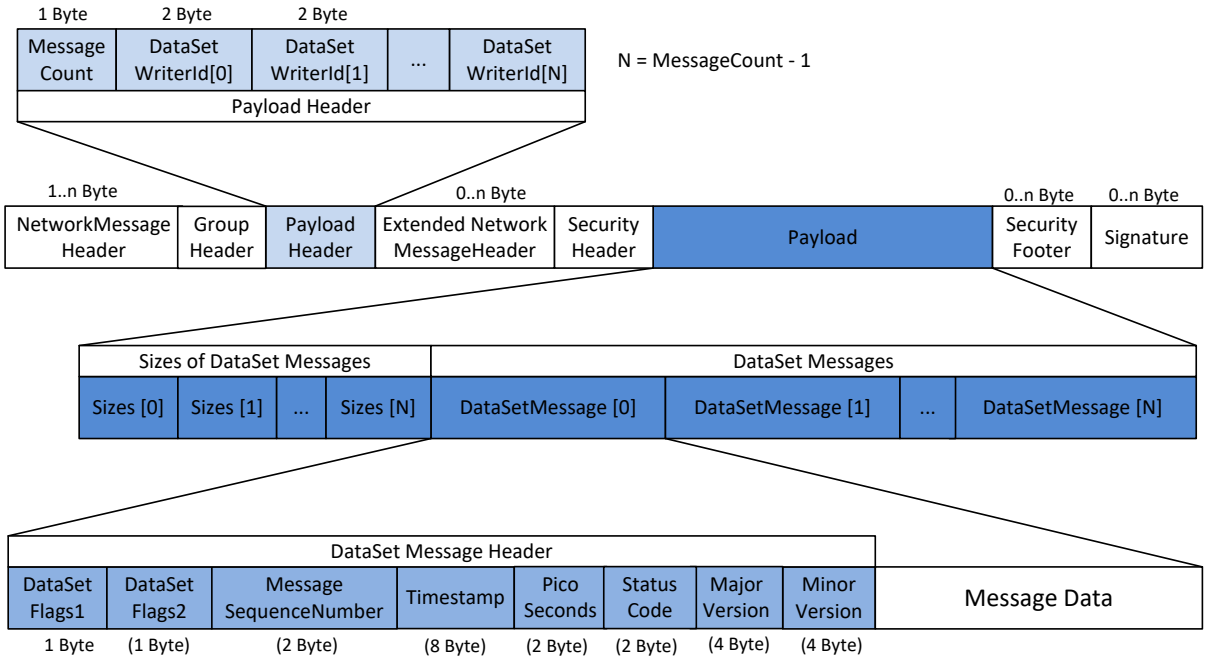

Figure 29 – DataSetMessage Header Structure

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 *DataSet* fields are encoded in the DataTypes specified in the *DataSetMetaData* for the *DataSet*. The encoding 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*. 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 65535 to 0). To determine whether a received *DataSetMessage* is newer than the last processed *DataSetMessage* the following formula shall be used: (65535 + received sequence number – last processed sequence number) modulo 65536 Results below 16384 indicate that the received *DataSetMessage* is newer than the last processed *DataSetMessage* and the received *DataSetMessage* is processed. Results above 49162 indicate that the received message is older (or same) than 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 2 of *DataSetFlags1* is false.
Timestamp	UtcTime	The time the Data was collected. The *Timestamp* shall be omitted if Bit 3 of *DataSetFlags1* is false.
PicoSeconds	UInt16	Specifies the number of 10 picoseconds (1,0 e-11 seconds) intervals which shall be added to the *Timestamp*. The field shall be omitted if Bit 4 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	Version Time	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	Version Time	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.

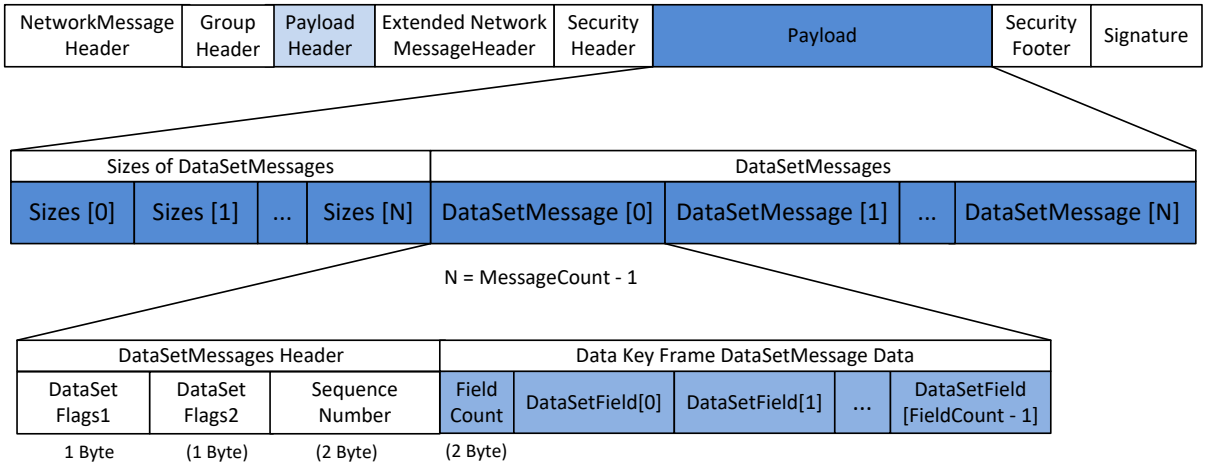

Figure 30 – Data Key Frame DataSetMessage Data

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 0 and 1 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.

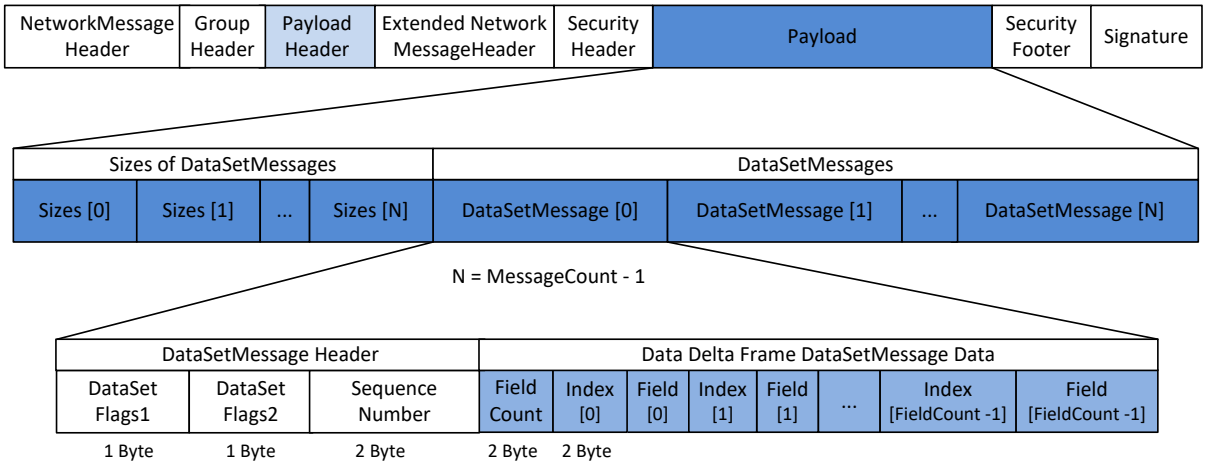

Figure 31 – Data Delta Frame DataSetMessage

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 2 and 3 are not set.

7.2.2.3.7 Event *DataSetMessage*

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

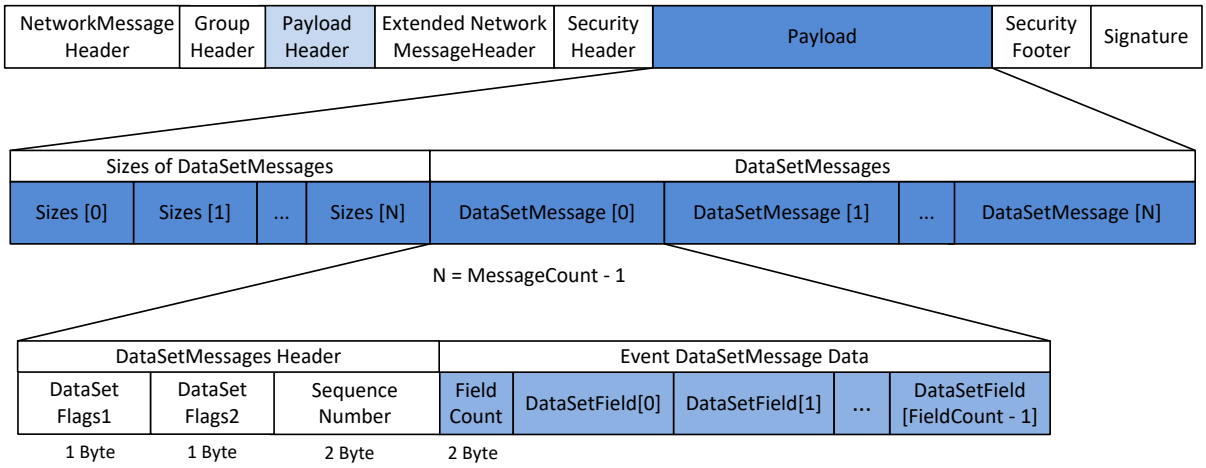

Figure 32 – Event DataSetMessage

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.

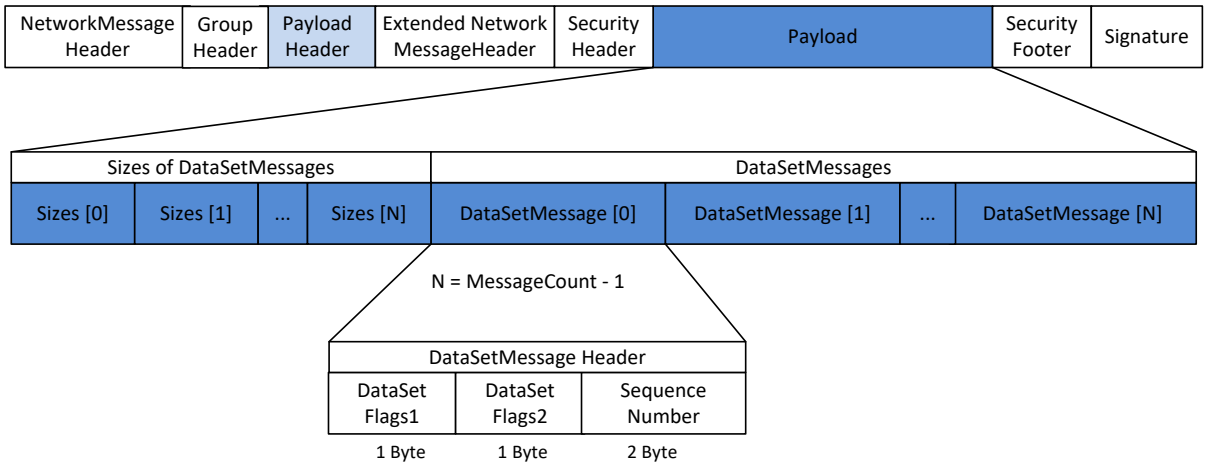

Figure 33 – KeepAlive Message

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

7.2.2.4 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 100-500 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 Part 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 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 *DataSetMessage*s (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 Part 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 Objects.* 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	String	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	ConfigurationVersion DataType	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 Part 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 Part 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 Part 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 Part 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	DataSetMeta DataType	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 Part 6.

7.3 Transport Protocol Mappings

7.3.1 General

This clause lists the standard protocols that have been selected for this specification 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 65535 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 must 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 RFC1112.
- IGMP V2 is defined in RFC2236.
- IGMP V3 is defined in RFC3376.

RFC2236 and RFC3376 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.

IGMP Membership Report Messages
OPC UA devices shall 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 1522 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 specification 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 *Broker*s have an upper limit on message size. The mechanism for handling *NetworkMessage* that exceed the *Broker* limits depend 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 or 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 subsections 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 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 Part 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 is = 0 the value is encoded as an AMQP ‘string’ type using the format for a NodeId defined in Part 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 Part 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 Part 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 specification 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 primary 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 must 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 choses 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 meta data 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

This chapter 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.

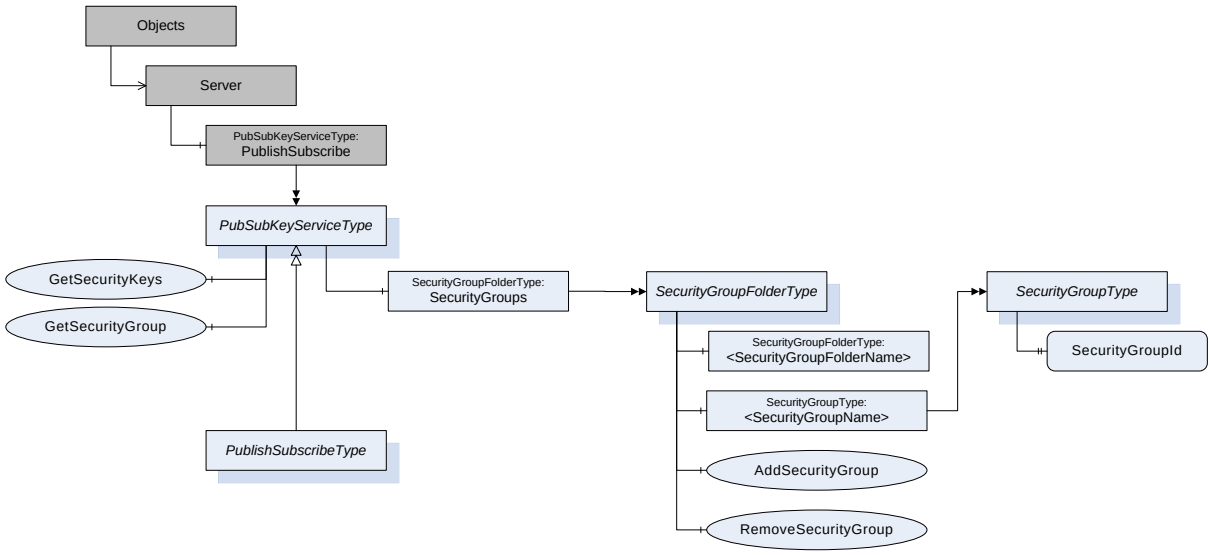

Figure 34 – PublishSubscribe Object Types Overview

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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
ComponentOf the *Server Object* defined in Part 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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* must 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 Part 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* must expect 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* must 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* must 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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 Part 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	**Node Class**	**BrowseName**		**TypeDefinition**	**Modelling Rule**
Subtype of FolderType defined in Part 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.

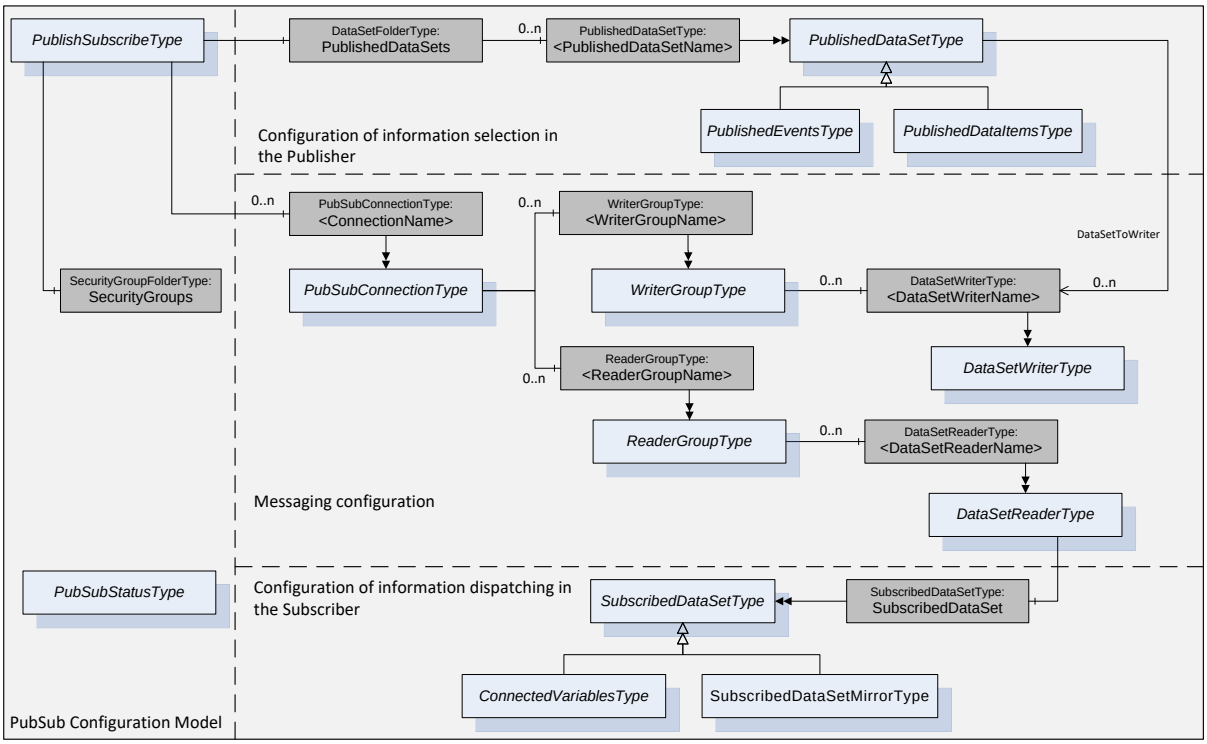

Figure 35 – PubSub Configuration Model Overview

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*.

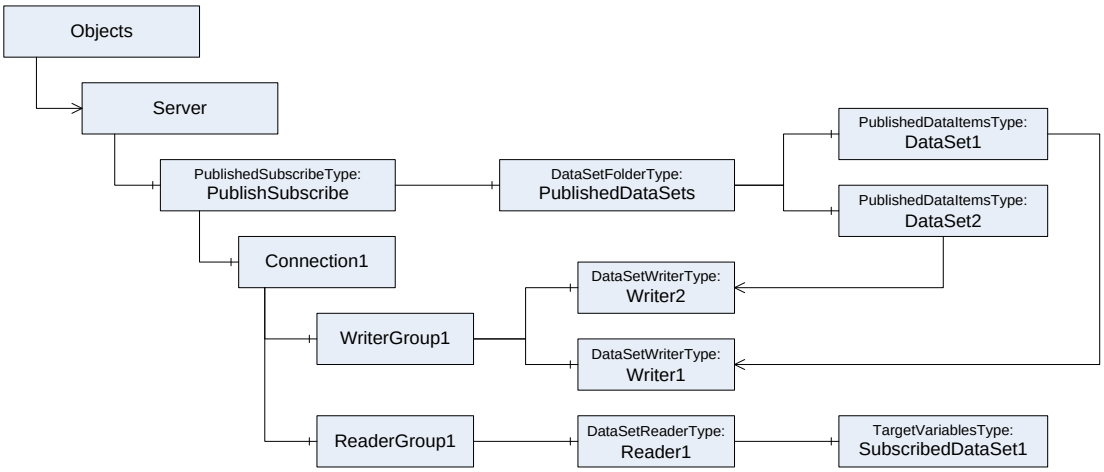

Figure 36 – PubSub Example Objects

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*.

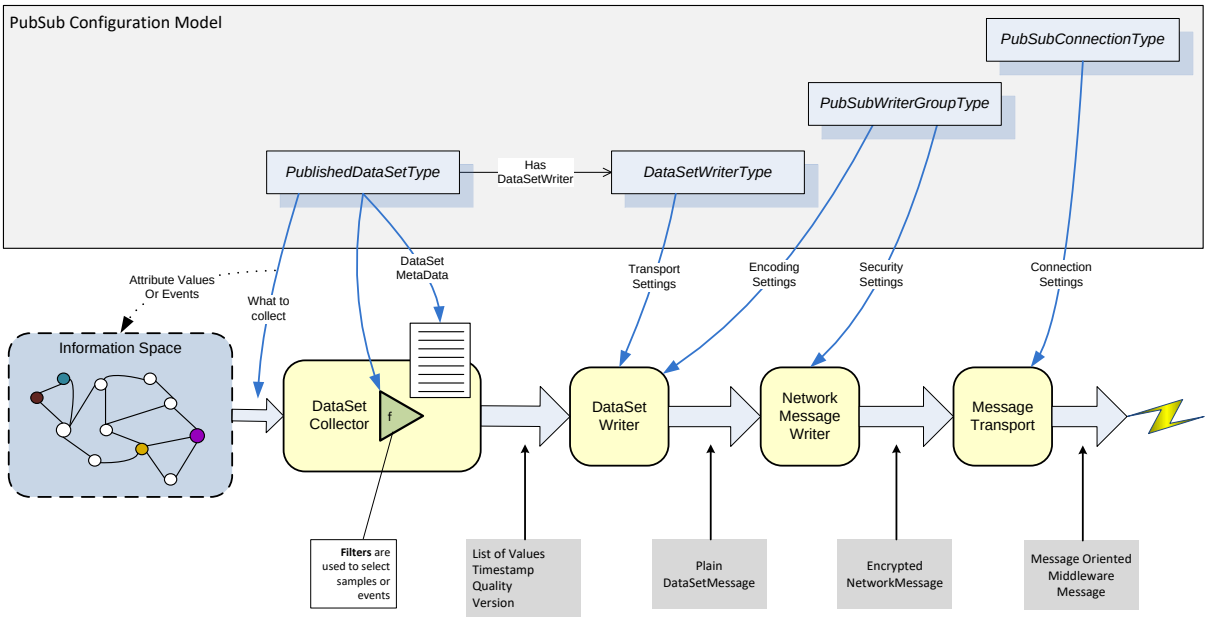

Figure 37 – PubSub Information Flow

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 specification.

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 standard. 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.

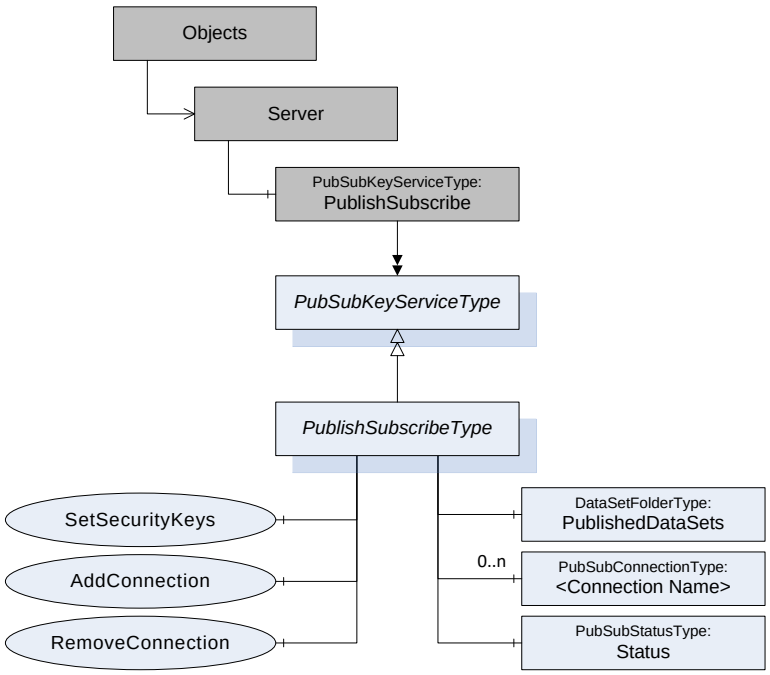

Figure 38 – PublishSubscribe Object Types Overview

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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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 Part 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 Part 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 a 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 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* must expect 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* must 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* must 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 character.
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 group, *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 Part 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.

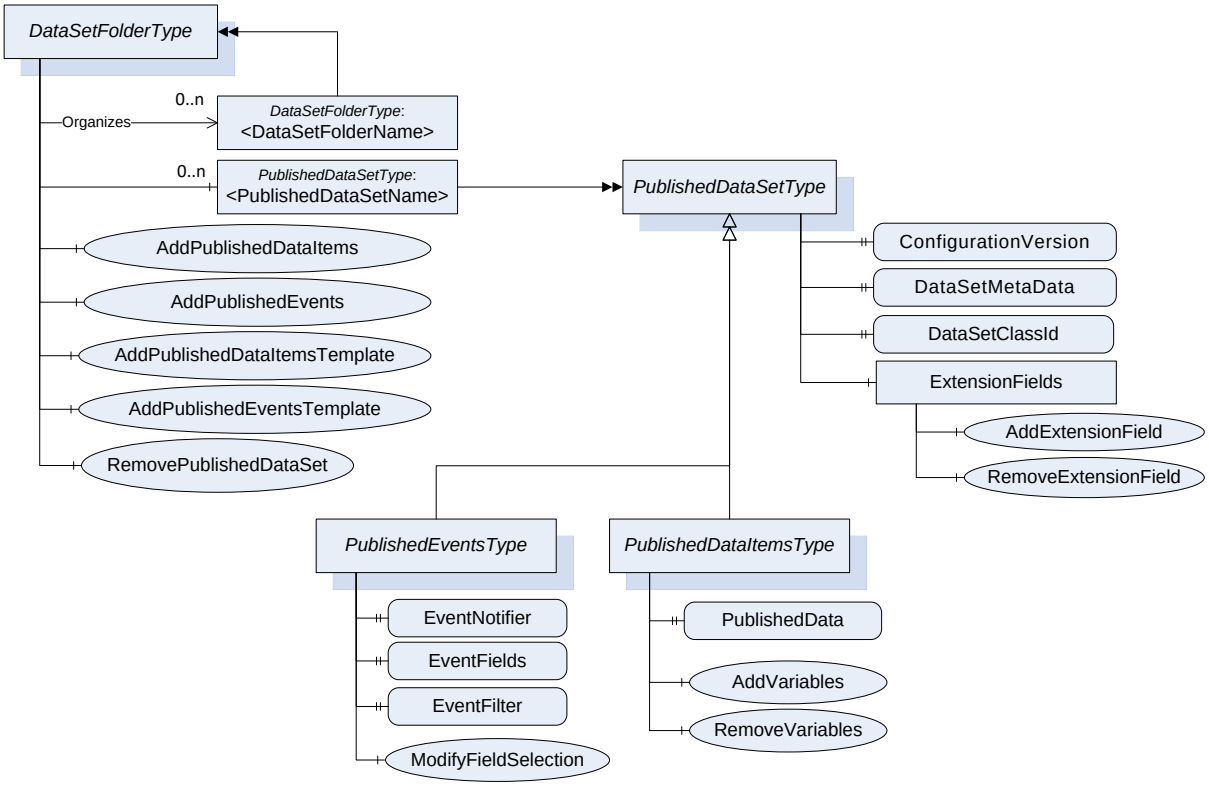

Figure 39 – Published DataSet Overview

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* defines 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 5.					
DataSetToWriter	Object	<DataSetWriterName>		DataSetWriterType	Optional Placeholder
HasProperty	Variable	ConfigurationVersion	Configuration VersionDataType	PropertyType	Mandatory
HasProperty	Variable	DataSetMetaData	DataSetMeta DataType	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* must 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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* must be added to the *DataSet* after creation.

Metadata that normally appear in message headers can be included to 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 Part 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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 must 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 passed in.
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 Part 4 for the description of this result code.
Bad_NodeIdUnknown	See Part 4 for the description of this result code.
Bad_IndexRangeInvalid	See Part 4 for the description of this result code.
Bad_IndexRangeNoData	See Part 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 passed in through *VariablesToRemove* must 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 passed in.
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 passed 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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 must 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 must 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 Part 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of FolderType defined in Part 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 passed in 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 Part 4 for the description of this result code.
Bad_NodeIdUnknown	See Part 4 for the description of this result code.
Bad_IndexRangeInvalid	See Part 4 for the description of this result code.
Bad_IndexRangeNoData	See Part 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 passed in 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 meta-data 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* that is used to collect *Events* from.
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 Part 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 Part 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 passed in 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 Part 4 for the description of this result code.
Bad_NodeIdUnknown	See Part 4 for the description of this result code.
Bad_IndexRangeInvalid	See Part 4 for the description of this result code.
Bad_IndexRangeNoData	See Part 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 passed in 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* that is used to collect *Events* from.
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 Part 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 Part 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.

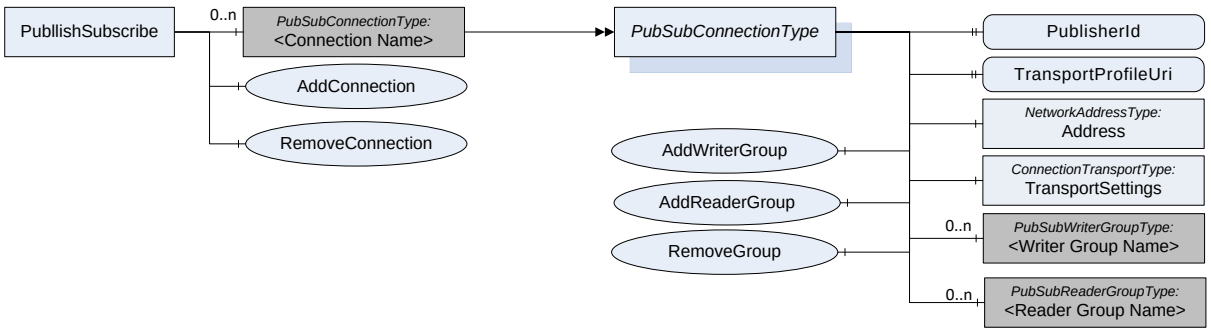

Figure 40 – PubSubConnectionType Overview

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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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		PubSubDiagnosticsConnectionType	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 Part 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 setting 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 character.
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 character.
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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.

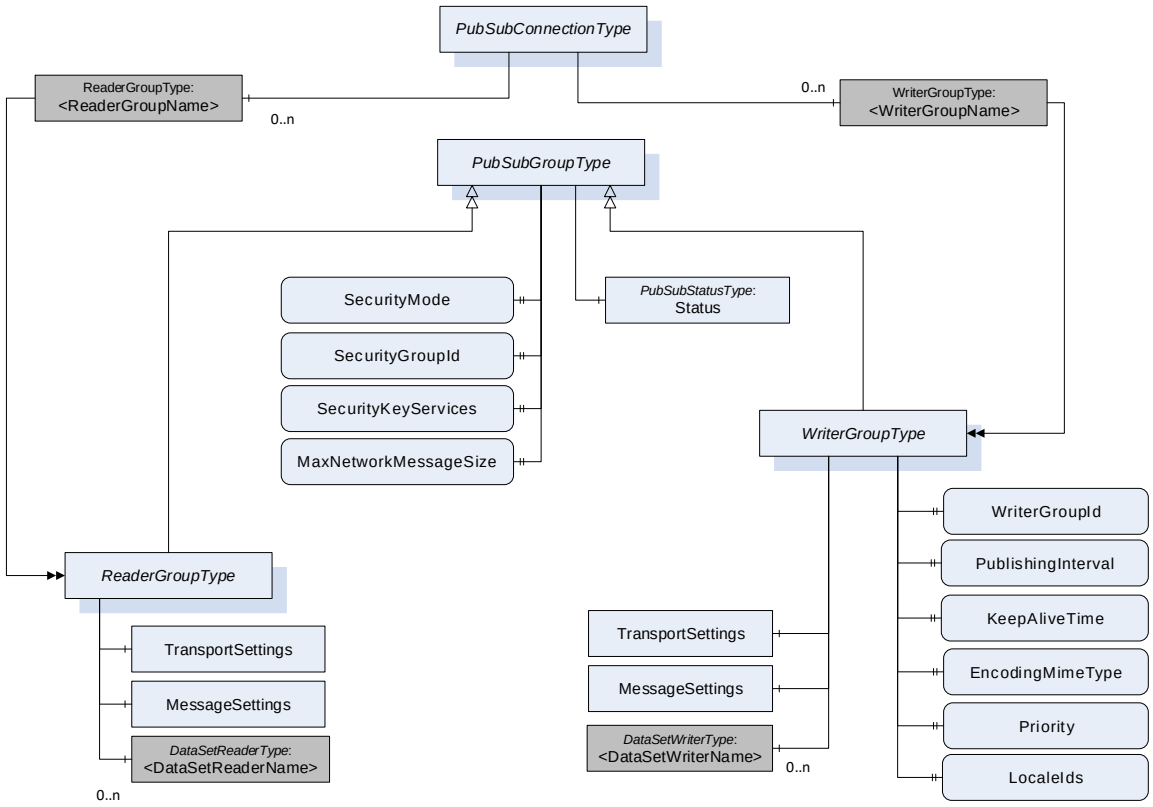

Figure 41 – PubSubGroupType Overview

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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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
HasComponent	Object	TransportSettings		WriterGroupTransportType	Optional
HasComponent	Object	MessageSettings		WriterGroupMessageType	Optional
HasDataSetWriter	Object	<DataSetWriterName>		DataSetWriterType	OptionalPlaceholder
HasComponent	Object	Diagnostics		PubSubDiagnostics WriterGroupType	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 setting 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 setting 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 character.
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 Part 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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 114.

Table 119 – ReaderGroupType Definition

Attribute	Value				
BrowseName	ReaderGroupType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
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 setting 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 setting 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 Part 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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.

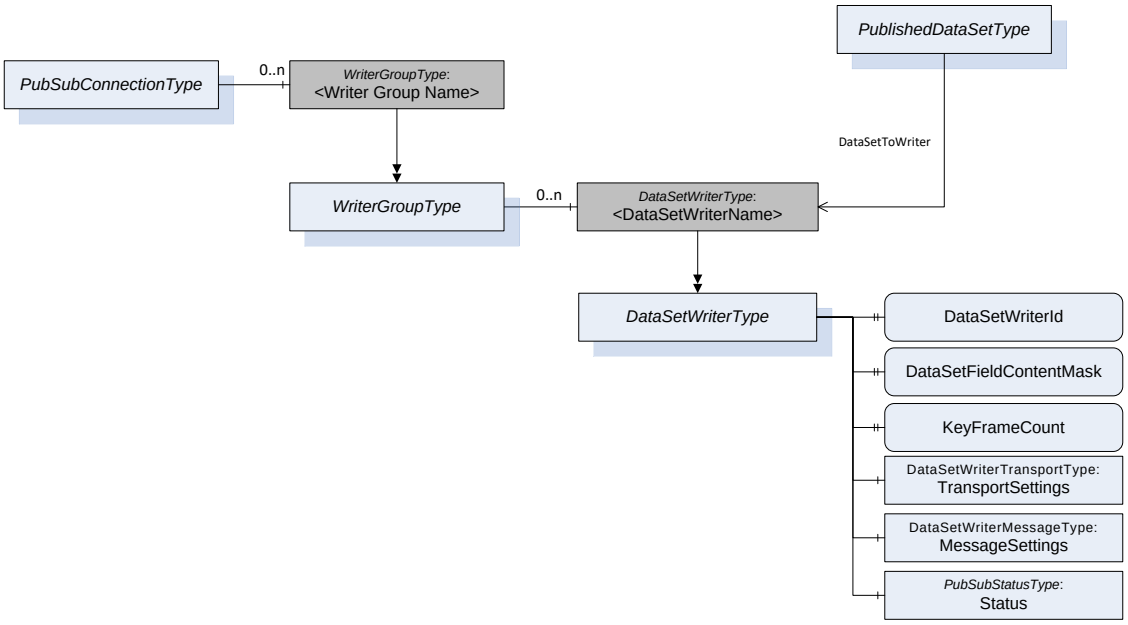

Figure 42 – DataSet Writer Model Overview

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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 5					
HasProperty	Variable	DataSetWriterId	UInt16	PropertyType	Mandatory
HasProperty	Variable	DataSetField ContentMask	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 setting 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 setting 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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.

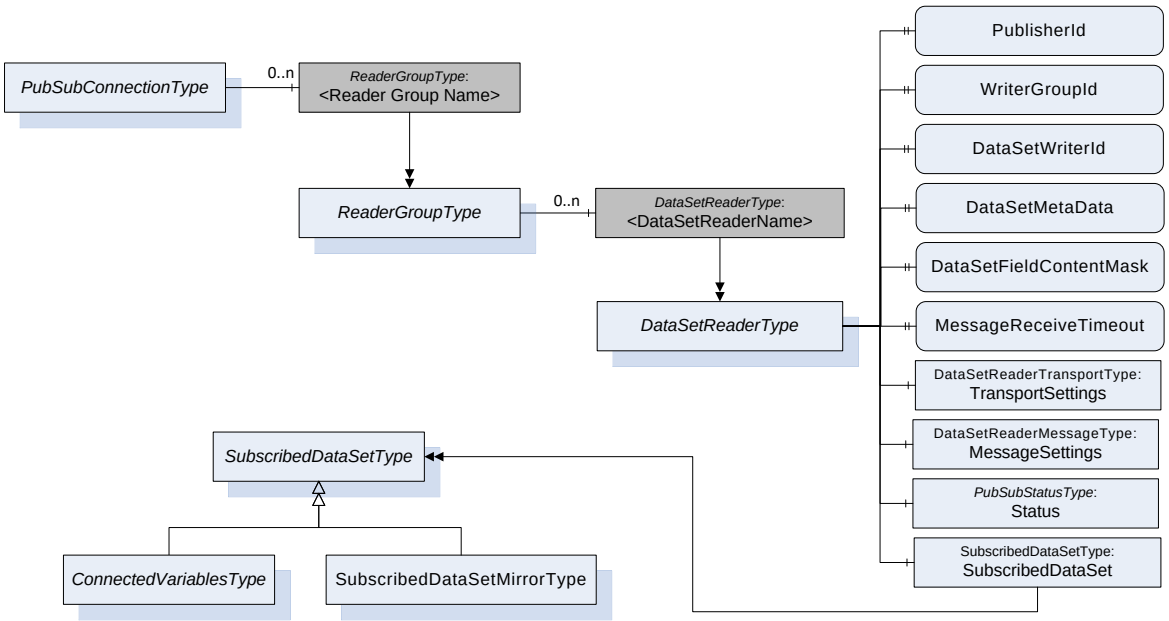

Figure 43 – DataSet Reader Model Overview

9.1.8.2 DataSetReaderType

This *ObjectType* defines receiving behaviour of *DataSetMessages* and the decoding to *DataSets*. The *DataSetReaderType* is formally defined in Table 105.

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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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	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 6.2.8.8.

The *SecurityKeyServices* is defined in 6.2.8.9.

The *DataSetReaderProperties* is defined in 6.2.8.10.

The transport protocol mapping specific setting 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 setting 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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 passed in through *CreateTargetVariables* must 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 passed in.
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 Part 4 for the description of this result code.
Bad_NodeIdUnknown	See Part 4 for the description of this result code.
Bad_IndexRangeInvalid	See Part 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 Part 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 of 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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 passed in through *AddDataConnections* must 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 passed in.
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 Part 4 for the description of this result code.
Bad_NodeIdUnknown	See Part 4 for the description of this result code.
Bad_IndexRangeInvalid	See Part 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 Part 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 of 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 passed in through *RemoveDataConnections* must 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 passed in.
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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 is 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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.

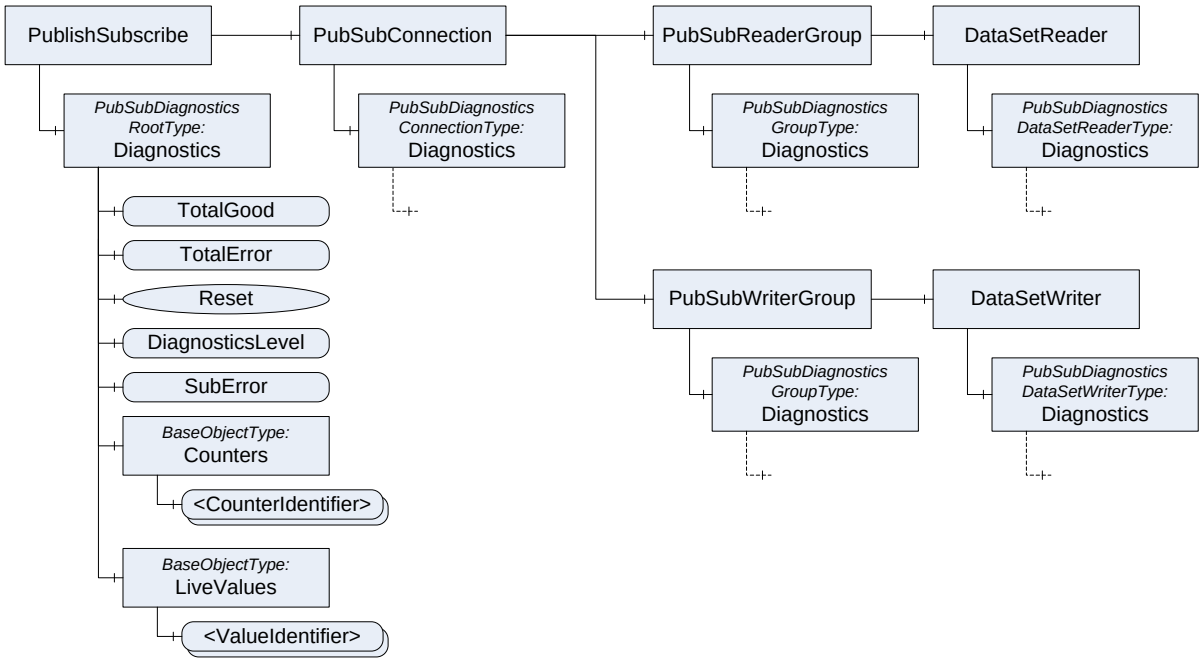

Figure 44 – PubSub Diagnostics Overview

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.

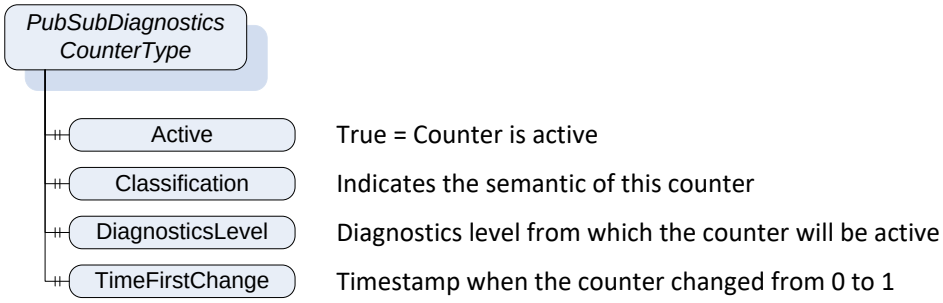

Figure 45 – PubSubDiagnosticsCounterType

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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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 this in the *Object* diagnostics counters with classification *Information_0*.

The *TotalError Variable* provides the sum of all counters in this 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 *PubSubDiangosticsType*.

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 have

to be provided. This mechanism provides the user 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 the standard.

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 Part 5.					
HasProperty	Variable	Active	Boolean	PropertyType	Mandatory
HasProperty	Variable	Classification	PubSubDiagnostics CounterClassification	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	Modelling Rule	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	Modelling Rule	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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of SystemEventType defined in Part 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 Part 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	UadpNetworkMessage ContentMask	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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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 167.

Table 165 – DatagramWriterGroupTransportType Definition

Attribute	Value				
BrowseName	DatagramWriterGroupTransportType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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	BrokerTransportQuality OfService	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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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 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 Part 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 Part 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 Part 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 Part 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 contend of the file is the *UABinaryFileDataType* encoded as *ExtensionObject*.

The file specific meta data 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	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of BaseObjectType defined in Part 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 Part 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 an URL *String*. The *NetworkAddressUrlType* is formally defined in Table A.13.

Table A.13 – NetworkAddressUrlType Definition

Attribute	Value				
BrowseName	NetworkAddressUrlType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
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*.

This Annex contrasts the *Subscription* functionality available in the *Client Server* communication model with the data distribution mechanism of *PubSub*. See Part 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*. I.e., 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 for *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.46.

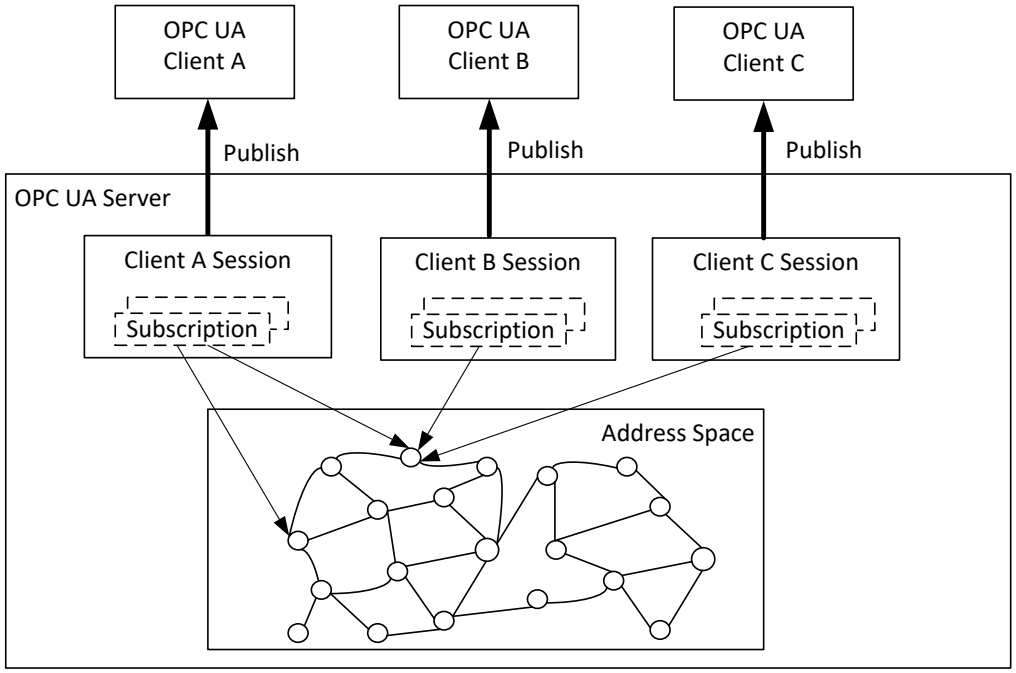

Figure B.46 – Subscriptions in OPC UA Client Server Model

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.47 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.

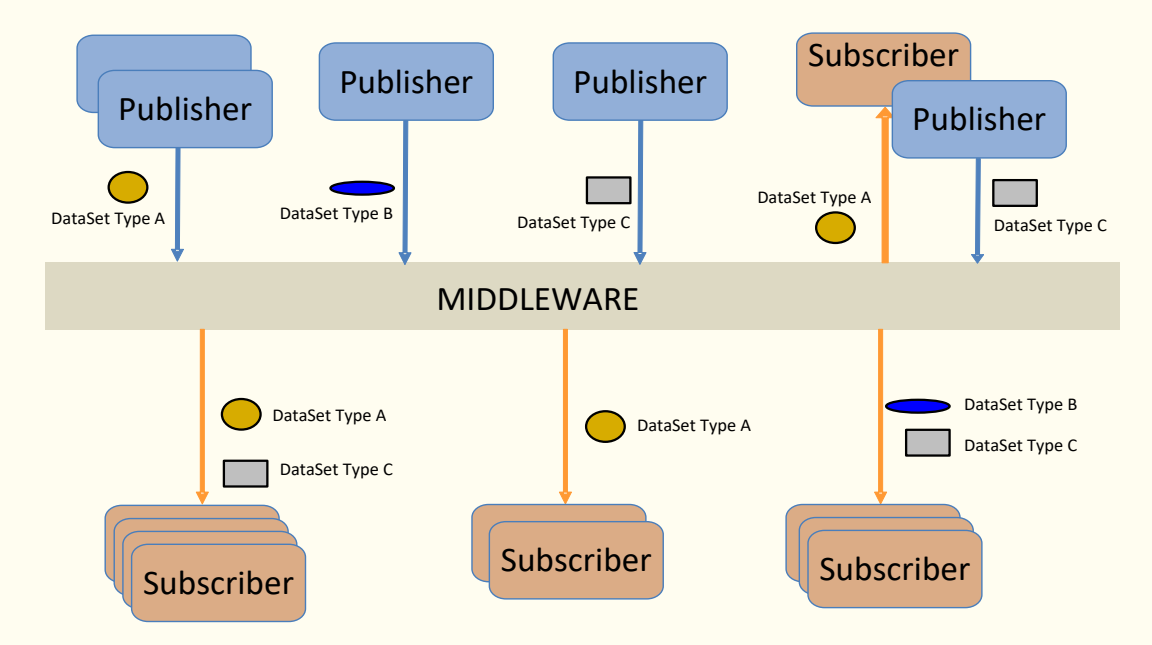

Figure B.47 – Publish Subscribe Model Overview

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. 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.

_____