INTERNATIONAL STANDARD

ISO 5231

First edition
2022-10

Extended farm management information systems data interface (EFDI) — Concept and guidelines

Reference number
ISO 5231:2022(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 23, *Tractors and machinery for agriculture and forestry*, Subcommittee SC 19, *Agricultural electronics.*

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Participants in the agricultural environment are dealing with the necessity of exchanging data with various systems and interfaces. With increasing demand for process monitoring and control, and just-in-time information on task execution state, and at the same time integration of mobile devices into farm work processes, the need of a standardized way for data exchange arises.

Extended farm management information systems data interface (EFDI) — Concept and guidelines

1 Scope

This document specifies an extensible communication system concept and defines rules for adding new functionalities to cover specific use cases.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

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

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
network
group of two or more participants connected to each other via a server

3.2
endpoint
uniquely addressable instance within a network

Note 1 to entry: An endpoint can be a farm management information system (FMIS), a telemetry unit, a terminal or a complete machine.

Note 2 to entry: The server is also an endpoint.

3.3
client
C
endpoint that communicates with the server

3.4
server
S
central component for communication in a network

Note 1 to entry: All communication is done via the server.

3.5
messaging component
MC
part of the server and network management

Note 1 to entry: The messaging component (MC) keeps track of logged-in endpoints and their capabilities and is also responsible for the delivery and forwarding of messages.

3.6
noun
type of a message

3.7
verb
action which is executed with a specific noun

3.8
scenario
order in which the request and response messages shall be performed

Note 1 to entry: Every request and response consists of a verb and a noun.

3.9
scenario flow
sequence of scenarios

Note 1 to entry: Scenario Flows define how scenarios are related to each other and how they are to be executed in dependence on each other.

3.10
scenario set
group of scenario flows

3.11
A-Step
request that is sent from client A to client B

3.12
B-Step
request that is sent from client B to client A

3.13
streaming
subscription to specific types of messages and subsequently reception of unsolicited information

3.14
onboarding
initial access or registration of an endpoint

3.15
onboarding service
service enabling clients to access a network

3.16
protocol buffers
protobuf
data structures that can be serialized in compact binary representation

Note 1 to entry: In this document protocol buffers version 3 (proto3) applies to all definitions. See also https://developers.google.com/protocol-buffers/docs/proto3

3.17
message queue telemetry transport
MQTT
standardized transport protocol with publish-subscribe paradigm

Note 1 to entry: In this document MQTT 3.1.1 applies to all definitions. See also http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

3.18
hypertext transport protocol secure
HTTPS
set of rules for secure transferring data over the internet or a local network

Note 1 to entry: See also https://datatracker.ietf.org/doc/html/rfc2818

3.19
semantic versioning
concept for defining version numbers to software.

Note 1 to entry: Semantic versioning allows to imply compatibility information to a version number, see also https://semver.org/.

3.20
VDMA Database
contains all ISO5231 definitions for scenario sets, scenario flows, scenarios and message.

Note 1 to entry: Database can be accessed at https://isobus.net/isobus/efdi

3.21
transport layer security
TLS
protocol for secure communication over the internet

3.22
transport layer security certificates
TLS-CERTS
certificates providing information for encryption of communication

3.23
extensible messaging and presence protocol
XMPP
application profile for data exchange

3.24
fully qualified domain name
FQDN
location with all its domain levels

4 Architectural overview

The extended farm management data interface consists of a set of layers packaged on top of the network application layer as illustrated in Figure 1.

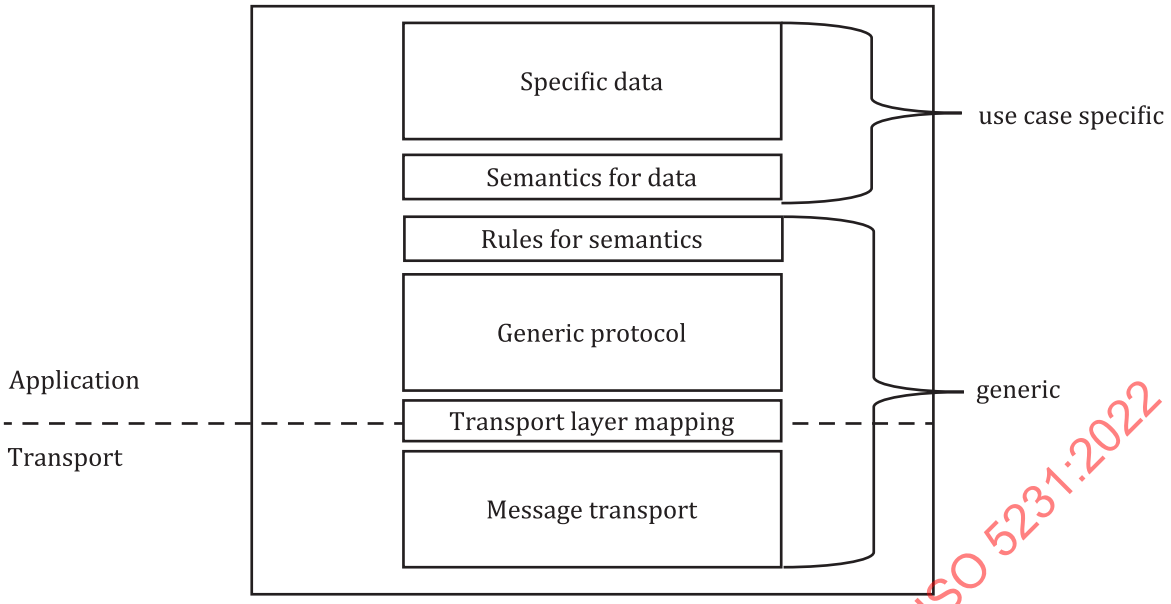


Figure 1 — Layer model

The selected message transport protocol to show how to implement EFDI is based on the MQTT protocol as the message transport layer. However, the application layers (see Figure 1) are not bound to it. Messages are partitioned into a header and a payload. The header shall contain addressing information and may contain additional data for setting up a stateful, persistent session. Depending upon lower-layer transport protocol functionalities, the additional data may be omitted and instead be covered by the transport protocol layer. It is theoretically possible to transport any payload within the EFDI protocol. However, transfer of ISO 11783-10 data is explicitly described within Annex B. There is also a generic file transfer handling described by the basic file functionality scenario set. Apart from the use case specific layer sitting on the top-most position of the stack, EFDI provides a semantic layer to indicate to message receivers what to do with the data (execute task, replace element within task, etc.).

The definition of generic protocol layer is described in Clause 6. The rules for semantics are defined in Clause 7 and the transport layer mapping is described In Clause 8.

5 Network

5.1 Overview

This clause gives an overview about the different components that are part of the network as shown in Figure 2.

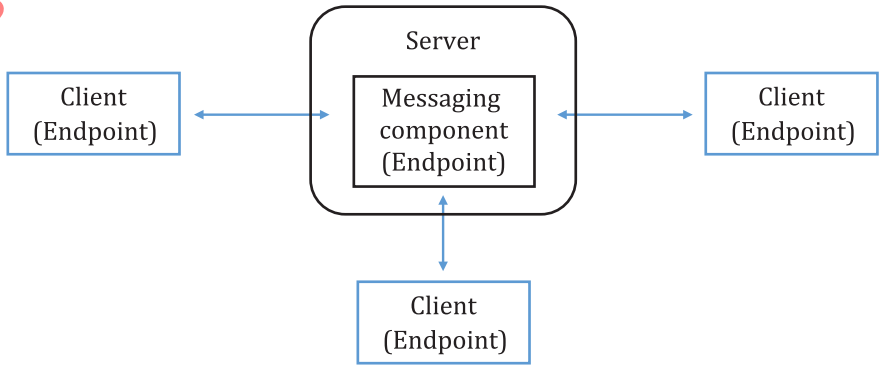


Figure 2 — Components in a network

Clients can establish a connection to the server. All endpoints in the network can communicate to each other.

5.2 Onboarding service

The onboarding service (OS) is part of the server component of the network. It is a service with which an endpoint can connect to the network. Individual IDs and certificates are generated, which are necessary for subsequent communication. The onboarding is part of the registration process of an endpoint.

5.3 Connected networks

Interconnecting multiple networks can be achieved by implementing a server containing both a messaging component and a client.

In Figure 3, server A handles the directly connected clients via its messaging component. Additionally, server A connects as a client to server B. Through this connection server A indirectly connects to clients through server B. Depending on the provided capabilities of server A's client, server B may also be able to indirectly connect to server A's clients.

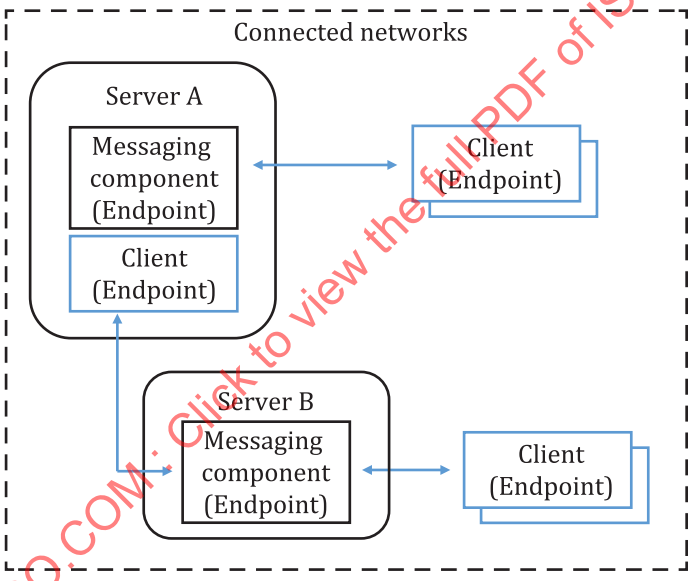


Figure 3 — Components in connected networks

6 Messaging

6.1 Overview

This clause presents various basic concepts of how communication takes place. It includes description of the construction of messages, architectural structures, addressing and streaming.

6.2 Hierarchical structure of scenario sets, scenario flows and scenarios

6.2.1 General

The hierarchical structure of scenario sets, scenario flows and scenarios are depicted in Figure 4. Generally, the scenario set is the topmost item, which contains at least one scenario flow. Each scenario flow contains at least one scenario. A scenario consists of a request and a response. Both a request and a response consist of a combination of a verb and a noun.

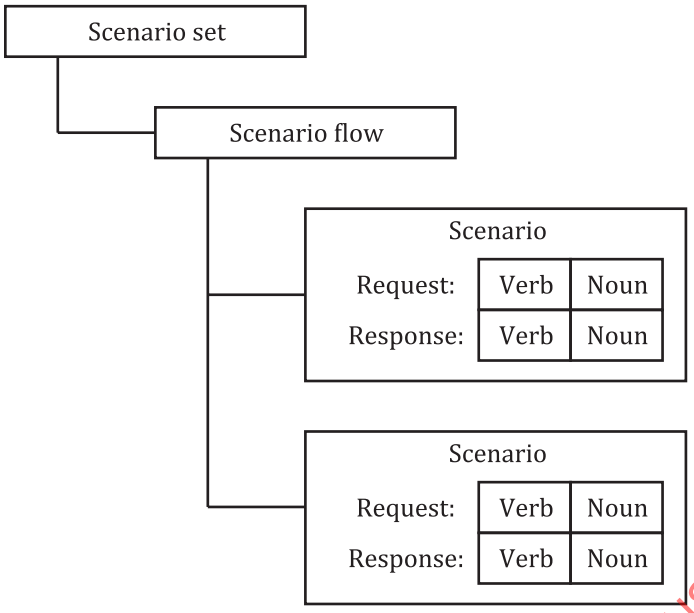


Figure 4 — Hierarchical structure of scenario sets, scenario flows and scenarios

All of the described messaging follows a common process. This common process is described in 6.2.2 to 6.2.5.

6.2.2 Scenario

A sequence of request and response messages is called a scenario. All of the scenarios have a request-response messaging pattern. That means that there is a defined response for a specific request. The request is sent by an endpoint and the response is sent back by another endpoint.

The requests and responses are defined as a combination of verb and noun. This verb-and-noun concept is based on the Business Object Document specified by the Open Applications Group Integration Specification (OAGIS), see Annex A for additional information. The verb defines the action or actions that are desired for the included business information. The business information itself is contained in the other sub-section called a noun.

An example is a scenario for requesting the list of endpoints that are connected to the server shown in Figure 5. The request contains the verb GET and the noun ListEndpoints. Translated this means that the list of connected endpoints is requested. The response contains the verb SHOW and the noun ListEndpointsResponse. Translated, this means that the list of connected endpoints is returned.

Figure 5 — ListEndpoints scenario

The schematic representation can be seen in Figure 6. Additional information and how scenarios shall be described formally can be found in 8.2.

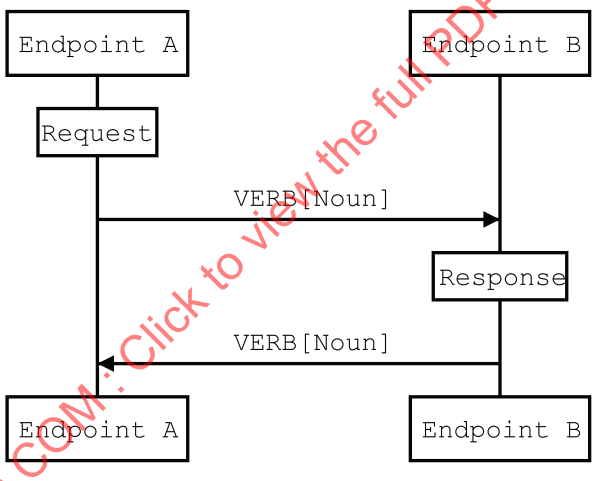


Figure 6 — Schematic representation of scenario

The list of available verbs based on the OAGIS standard has been extended in order to cover new use cases. The following verbs have been added:

— FORWARD: shall only be used by the server, indicates that a message was received and will be forwarded to other endpoints. A more detailed explanation and when this verb is used can be found in 6.2.5;

— STREAM_START: to request streaming data, indicates that the endpoint would like to receive streamed data from now on;

— STREAM_STOP: indicates that the endpoint no longer wishes to receive streamed data;

— STREAM_DATA: necessary to mark streamed data.

More detailed information about the new verbs for streamed data can be found in 7.6.

6.2.3 Scenario flow

A scenario flow consists of at least one scenario. The order of the scenarios is described within the flow. Scenarios in a scenario flow can be optional. If scenarios are declared as optional, they do not need to be executed.

The sender of the first request in a scenario flow is called endpoint A, the other endpoint is called endpoint B for the whole addressed communication in a scenario flow. When A sends the request to B, this is called an A-Step. When B sends the request to A, this is called a B-Step. This is visualized in Figure 7. Additional information and how scenario flows shall be described formally can be found in 7.4.

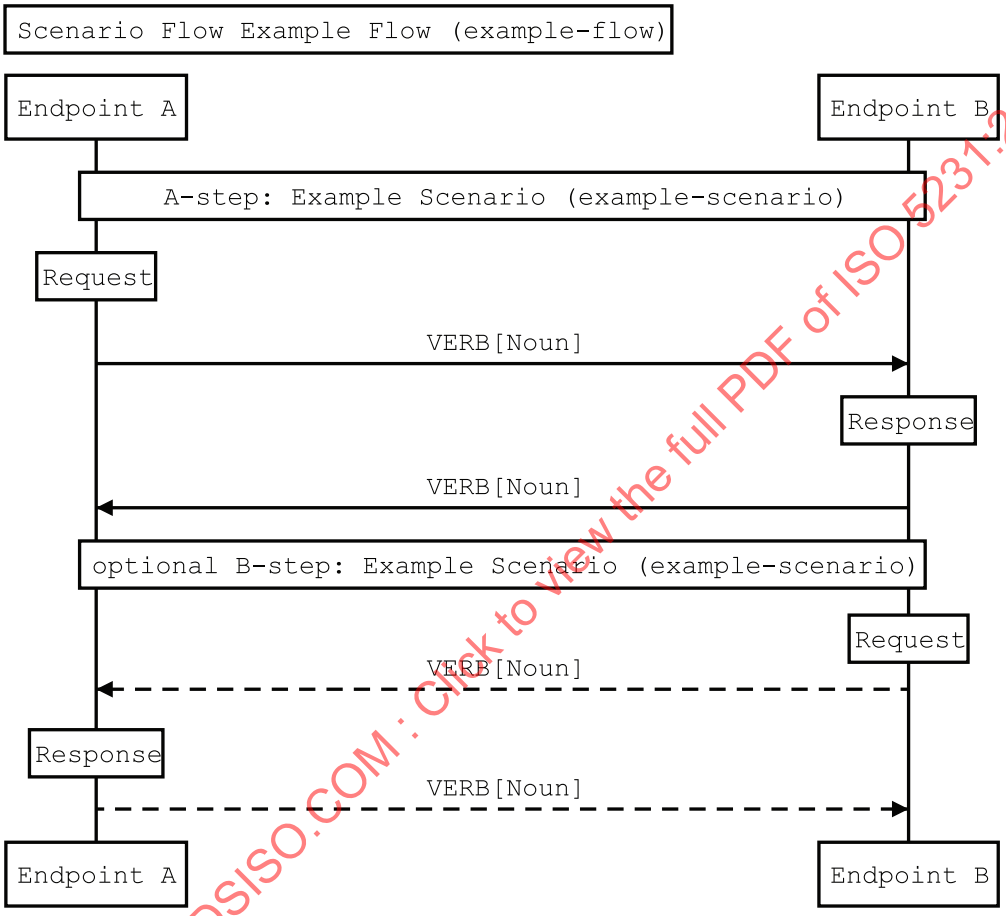


Figure 7 — Schematic representation of a scenario flow

6.2.4 Scenario set

A scenario set consists of at least one scenario flow. A scenario set is a grouping that can be used, e.g. for certification, if necessary. Additional information and how scenario flows shall be described formally can be found in 7.5.

Currently, there is only one scenario set defined. This scenario set is called basic and contains all scenario flows necessary for basic communication.

6.2.5 Detailed view on a scenario

From an application point of view, an example scenario is shown in Figure 7. A request is sent by one endpoint and the response is sent back by another endpoint. If we take a more detailed look at it, we get a different result. Endpoints do not communicate directly with each other like it is shown in 5.1. Instead, endpoints exchange messages between each other using the server/messaging component. The simplified representation, which only consists of one request and response pair, is actually split into

four sub-scenarios. Figure 8 shows sub-scenarios that are implicitly being executed in scenarios where information is exchanged between two endpoints via the server/messaging component.

The four sub-scenarios are the following which can also be seen in Figure 8:

— Sub-Scenario Endpoint Request (Sub_1)
  — Request (Endpoint A to Server): VERB [Noun]
  — Response (Server to Endpoint A): FORWARD [ForwardInfo]
— Sub-Scenario Forward Endpoint Request (Sub_2)
  — Request (Server to Endpoint B): VERB [Noun]
  — Response (Endpoint B to Server): CONFIRM [ ]
— Sub-Scenario Endpoint Response (Sub_3)
  — Request (Endpoint B to Server): VERB [Noun]
  — Response (Server to Endpoint B): FORWARD [ForwardInfo]
— Sub-Scenario Forward Endpoint (Sub_4)
  — Request (Server to Endpoint A): VERB [Noun]
  — Response (Endpoint A to Server): CONFIRM [ ]

The contents of Figure 8 are described in detail.

1) Endpoint A sends the request of the scenario that is described within the scenario definition. This request is sent to the server. The server directly sends a ForwardInfo message as response. The server's response contains, among other things, the information that the message has been forwarded.

2) The server sends a message to endpoint B. This message contains the verb and noun of the message the server received from endpoint A. The reception of this message shall be confirmed by the recipient endpoint B. The confirmation contains the verb CONFIRM without a noun. The CONFIRM message is intended to confirm the reception of a message sent between the server and endpoint A. In the end, endpoint A will receive the verb-noun message of endpoint B as response.

3) Endpoint B creates the response message described in the scenario definition. This message will be sent as a request message to the server. The server directly sends a ForwardInfo message as response. The server's response contains, among other things, the information that the message has been forwarded.

4) The server sends a message to endpoint A. This message contains the verb and noun of the message the server received from endpoint B. The reception of this message shall be confirmed by the recipient endpoint A. The confirmation contains the verb CONFIRM without a noun. The CONFIRM message is intended to confirm the reception of a message sent between the server and endpoint B.

All these steps can be abstracted from the scenarios, since it is the communication in the layer below the scenario definitions. For this reason, the scenarios are always presented in a simplified form – as it can be seen in Figure 7. But all the mentioned sub-scenarios are necessary for successful communication.

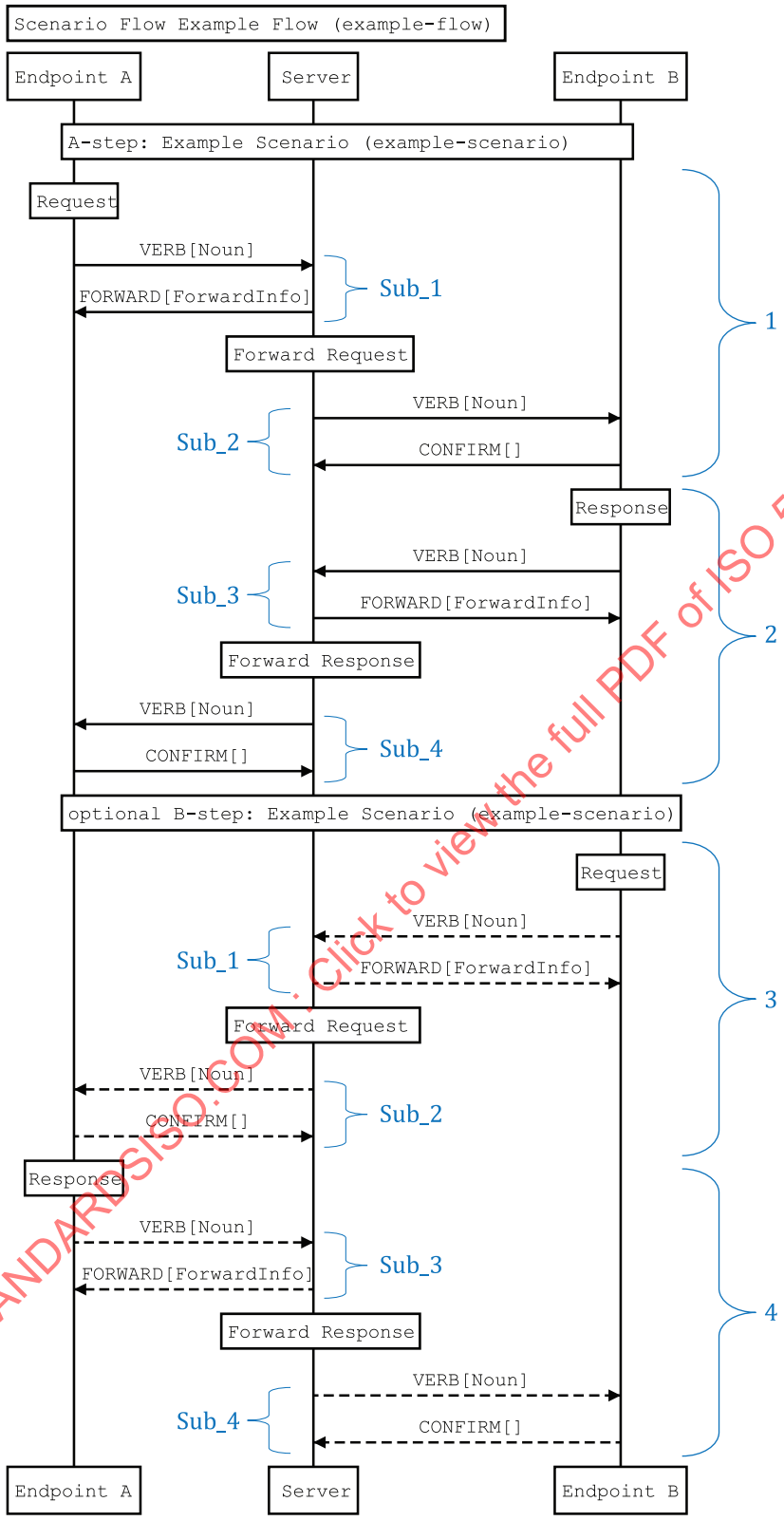


Figure 8 — Detailed schematic representation of scenario flow

In a scenario where the response contains the verb CONFIRM the response will not be forwarded to the endpoint that sent the request of this scenario. In this case the message with the verb FORWARD shall be considered as confirmation of successful transfer. Otherwise a CONFIRM would be forwarded that would be CONFIRMed and so on. Figure 9 shows the sub-scenarios that use the verb CONFIRM in the response.

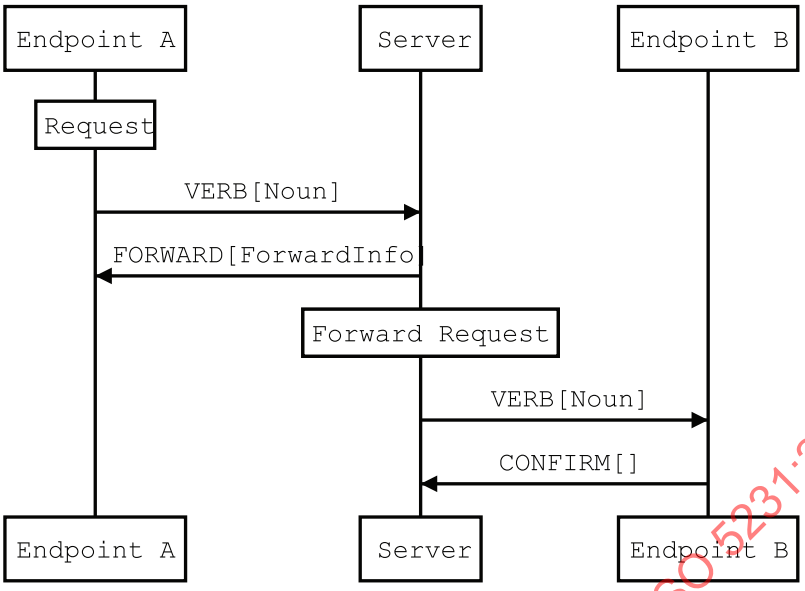


Figure 9 — Detailed schematic representation of scenario with CONFIRM Response

With a view to messages exchanged between endpoints and the server, the detailed representation is a different story again. In this case, the messages are sent as they were defined in the scenario. Only the reception of the last message sent by the server shall be confirmed by the endpoint with the verb CONFIRM. This can be seen in Figure 10. This process has been slightly modified compared to the scenarios described from endpoint to endpoint, see Figure 9. If the communication takes place directly from endpoint to server, the server no longer needs to forward the message to any other endpoint. For this reason, the server does not respond using the verb FORWARD. Instead, the server's response to the scenario request shall be regarded as confirmation of receipt.

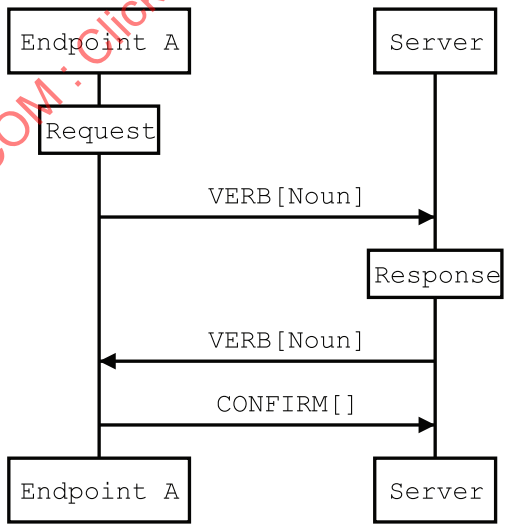


Figure 10 — Detailed schematic representation of scenario with Server

6.3 Addressing

6.3.1 General

This subclause describes the concepts of "Generic protocol" layer of Figure 1. The specific message definitions are described in Annex B.

In general, every endpoint gets a unique address. With this address, every endpoint can be identified unambiguously. The addressing is managed by the messaging component. The basic concept is to send as few messages as possible without prior request so that no information is sent initially by an endpoint.

Messages can be addressed directly to one or more endpoints. They can also be published to send the message to all subscribed endpoints. It is flagged in the message if the message is directly sent, published, or if both techniques shall be used. Messages are not delivered to the same endpoint twice.

There are several modes that can be used to send messages:

— DIRECT: for directed communication to a certain endpoint, the desired recipients of this message shall be specified

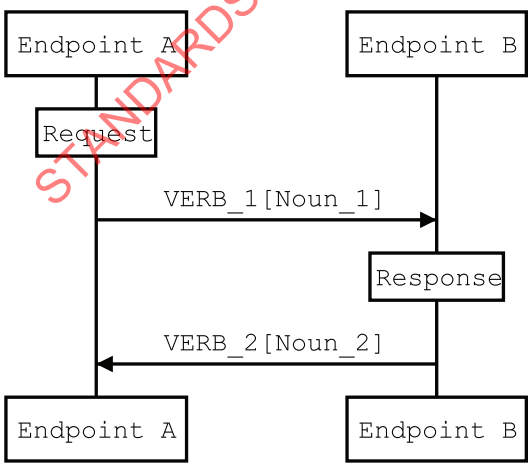
— PUBLISH: in this undirected communication the message is delivered to all subscribed endpoints, no recipients shall be named

— PUBLISH_WITH_DIRECT: it is a combination of the modes DIRECT and PUBLISH, messages will be delivered to all subscribed endpoints as well as to all desired recipients specified within the message

6.3.2 Endpoint capabilities

Each endpoint has a series of abilities and skills that are called capabilities hereafter. In order for the messaging component to know which messages can be sent to and received from endpoints, each endpoint shall provide its capabilities. The capabilities are also based on the noun and verb concept, which has been explained in 6.2.2. In summary, the capabilities of an endpoint indicate which scenarios the endpoint supports. Scenarios that are not supported by the endpoint cannot be received by the endpoint, nor is the endpoint allowed to send these messages. The messaging component shall not send messages to an endpoint that the endpoint does not support. In addition, the messaging component shall not forward messages that the endpoint cannot send according to the endpoint's capabilities.

When a client wants to announce that it supports a scenario, it shall support the request and the response defined in the corresponding scenario. Request and response consist of a combination of verb and noun each. An example is shown in Figure 11. There are two different endpoints: endpoint A and endpoint B. To be sender of the request and receiver of the response (endpoint A), the client shall be able to SEND the verb-noun-combination of the request and to RECEIVE the noun-verb-combination of the response. To be receiver of the request and sender of the response (endpoint B), the client shall be able to RECEIVE the noun-verb-combination of the request and to SEND the noun-verb-combination of the response. With this information, a client can determine which clients support which scenarios.



Capabilities Endpoint A

Direction	Verb	Noun
SEND	VERB_1	Noun_1
RECEIVE	VERB_2	Noun_2

Capabilities Endpoint B

Direction	Verb	Noun
RECEIVE	VERB_1	Noun_1
SEND	VERB_2	Noun_2

Figure 11 — Necessary capabilities to support a scenario

The capabilities are not sent without prior request. Instead, a message containing the status of the endpoint is sent. This message is sent when the endpoint is connected to the network and contains

a reference to the endpoint's capabilities. If the messaging component already knows this unique reference and the capabilities behind it, the messaging component already knows what the endpoint is capable of. If the messaging component does not yet know this unique reference and the associated capabilities, it can request them. When an endpoint connects to the network it shall send a status message indicating that it is online. The endpoint shall always produce the same reference for a certain set of capabilities. Conversely, this also means that the reference output shall be different when the set of capabilities of the endpoint is different.

The message with the abilities can be quite voluminous under certain circumstances. All scenarios are semantically versioned. If an endpoint now supports multiple versions and many scenarios, the message about the capabilities becomes very large. If this message is always sent when the endpoint connects to the network without being prompted, it creates a lot of unnecessary traffic.

If an endpoint has connected to the network, it shall report that it is online. If it disconnects from the network on a regular basis, it shall report that it is expected to be offline. In the case that the endpoint unintentionally loses the connection to the network, it shall indicate that it will unexpectedly be offline.

6.3.3 Endpoint subscriptions

Subscriptions are used to subscribe to a list of verb and nouns (scenarios). Messages that are published (mode PUBLISH or PUBLISH_WITH_DIRECT) are forwarded to endpoints that are subscribed to the verb-noun-combination contained in the message. Subscriptions can be defined for specific nouns and verbs. Each new subscription list sent by an endpoint deletes old subscriptions. After changing the capabilities, the endpoint needs to resend its subscriptions. An unsubscription message will delete the listed subscriptions.

As a sender of a message does not always know the relevant endpoints, it can send the message as a public message. Every other endpoint can subscribe to any message type that is part of its capabilities. An endpoint cannot subscribe to scenarios that it does not support according to its capabilities.

6.3.4 Messaging component capabilities

Just as the endpoints have capabilities, the messaging component also has capabilities. However, these are slightly different from the endpoints. The capabilities of the messaging component are not only about which scenarios are supported, but more about how long and to what extent the messaging component can store messages. These are called the minimum capabilities of the messaging component defined in the message MinimumCapabilities.

The endpoint capabilities contain information about how many messages and how long messages can be stored. It can also contain the messaging component's memory size. The messaging component does not have to reveal all of these statements, but the more information about the messaging component is known, the more deterministic its behaviour is and the better the endpoints can adapt to it. At least one of these information shall be provided:

— number of messages;

— time of storage;

— size of storage;

— optionally, the maximum message size and the maximum number of individual messages in a total message can also be specified. The default capabilities of the messaging component are MinimumCapabilities;

— ListEndpoints (optional).

StreamingCapabilities (optional). In addition to the minimum capabilities of the messaging component there are optional capabilities the messaging component can support. Listing of endpoints and the stream mechanism. If streaming is supported, all verbs mentioned in 6.7 can be exchanged with

the messaging component. If streaming is not part of the capabilities, streaming is not part of the capabilities, streaming is not possible with this messaging component.

The messaging component capabilities can be requested by every client. The messaging component is always addressed with mode DIRECT and the list of recipients shall be empty. How to request these capabilities is explained in the messaging component capabilities scenario flow. The ListEndpoints message contains an entry with an empty endpoint address for each supported scenario of the messaging component.

In connected networks (as shown in Figure 3) a server A can connect to a server B. The client of server A that is connected to server B shall announce that it is part of a network that is not connected to server B directly. In order to do so, this client shall specify in its capabilities that it supports the scenario Send List Endpoints. It shall be able to RECEIVE a GET with ListEndpoints and to SEND a SHOW with ListEndpointsResponse. Additionally, this client can also be able to support other scenarios mentioned in the Messaging Component Capabilities scenario flow.

6.4 Endpoint architecture

6.4.1 Overview

The architecture of an endpoint aims at receiving and forwarding messages according to a combination of capabilities, subscriptions, public or direct addressed messaging. How this architecture can be mapped to a transport protocol is described in Clause 8.

6.4.2 Endpoint's inbox and outbox

Inside the server there is an inbox and an outbox for each endpoint. The endpoint communicates with the inbox when messages are sent to the server. To receive messages from the server, the endpoint communicates with its outbox. If, for instance, the request of a scenario is sent from an endpoint, this message arrives in the inbox. The response of this scenario is received via the outbox. A schematic representation of how the design inside the server and how the message flow from the endpoint and to the endpoint is, can be seen in Figure 12.

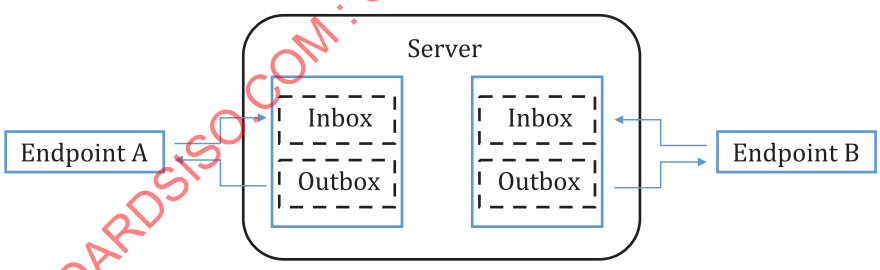


Figure 12 — Architecture of endpoints in a network

Figure 13 shows how a message is sent from endpoint A to endpoint B. The message is sent from endpoint A to its inbox. Within the server this message is forwarded to endpoint B by putting the message in the outbox of endpoint B. This message can be retrieved from there by endpoint B.

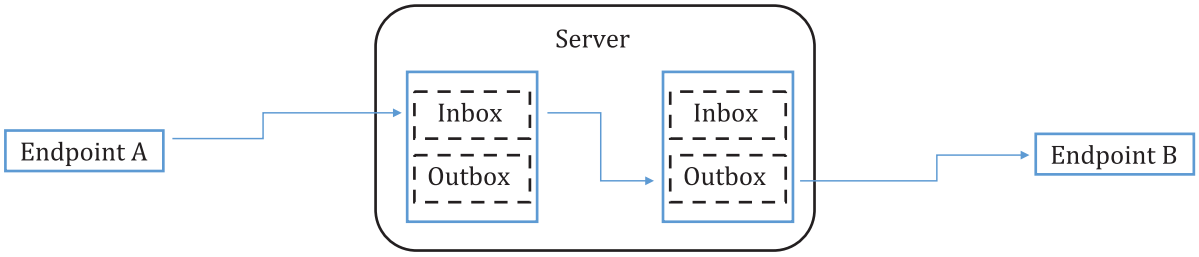


Figure 13 — Message sent from Endpoint A to Endpoint B

This is the minimal possible architecture. Within the server component, there shall be an inbox and an outbox for each endpoint.

6.4.3 Endpoint's feed

6.4.3.1 General

Additionally, it is possible that there is a feed for each endpoint within the server component. This feed is used as a storage for messages. Whether there is a feed for each endpoint can be seen in messaging component capabilities. The messaging component capabilities can also be used to determine how large the memory is, how many messages can be stored and for how long. When the maximum storage capacity of the server is reached – whether the number of stored messages or the size of messages per endpoint is exceeded – the oldest messages shall be discarded and removed from the endpoint's feed.

Figure 14 shows how the inbox, outbox and feed of an endpoint can be seen schematically within the server. The communication still takes place via the in- and outbox with the only difference that it is possible that outbox data may be stored in the feed.

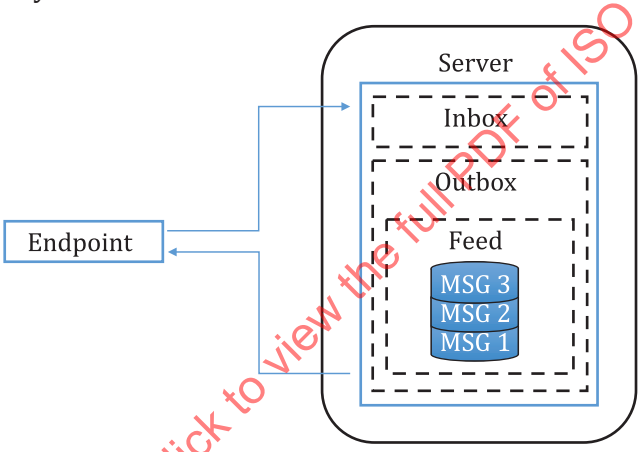


Figure 14 — Architecture of Endpoint with feed

Clause 8 describes in detail how the communication for querying for feed messages looks like – based on MQTT.

6.4.3.2 Feed configuration

After the initial connection establishment – the so called onboarding – no messages from other endpoints will be sent to the onboarded endpoint. The endpoint shall send its capabilities and shall configure the feed explicitly. After this, messages from other endpoints can be received. An endpoint only appears in the list of endpoints after performing the onboarding, sending its capabilities and configuring its feed.

Messages that are exchanged with the messaging component, and are part of a scenario that only happen with the messaging component, are delivered directly to the endpoint. Also, the ForwardInfo of the messaging component will be sent directly to the endpoint. The feed is not used for this communication.

The way how to receive messages that are sent by other endpoints shall be configured. On the one hand, messages can be stored in the feed. In this case, every message shall be requested from the feed. Every received message from the feed shall be confirmed. The confirmation of reception deletes the message from the feed. On the other hand, messages can be delivered directly. In this case, all messages will be delivered directly. If the endpoint is offline, it cannot receive any message and the messages are getting lost. Therefore, the endpoint has to be online to exchange messages successfully.

Additionally, there is the possibility that the messages will be stored in the endpoint's feed but will also be delivered directly. Messages will be delivered directly if the endpoint is online. Furthermore, these

messages will be stored in the feed. When a directly delivered message is confirmed the message will be automatically removed from the endpoint's feed. This way, all the messages you can receive from your feed are those messages, you did not yet receive directly.

There are three configurations of the client's feed, of which only one can be active at one point in time:

— storage of messages inside the feed;

— direct delivery of messages with storage inside the feed;

— direct delivery of messages without storage inside the feed (default behaviour).

6.4.3.3 Request feed message

Messages stored in the feed shall be explicitly requested by the endpoint. This is illustrated in Figure 15. Incoming messages are sent by endpoint A and delivered to endpoint B. These messages are stored in the feed of endpoint B (1). The request for messages from the feed is sent by endpoint B to the server via its inbox. The requested messages are now taken from the feed and placed in the outbox of endpoint B (2). From this point the messages can be retrieved from endpoint B (3).

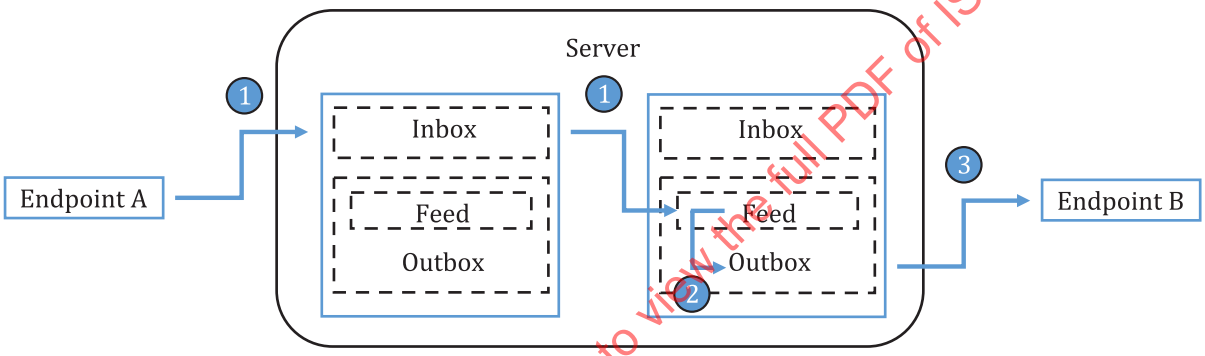


Figure 15 — Visualization of message retrieval from feed

If messages have been discarded and removed from the endpoint's feed, the number of discarded messages shall be specified in the response message.

6.5 Connection establishment

The server system shall provide the registration code to the user who wants to onboard the client system. It is up to the server system how the registration code is provided. It is also up to the server system to decide in what form the registration code is provided - whether it is numeric, alphanumeric or anything else that a human can read and enter into a connecting client system. The server system can also define whether the validity period of the registration code may be limited. In addition, the server system can also define whether a provided registration code may only be used once. The client system shall give the user the possibility to enter the registration code when a registration code is used.

The connection establishment begins with a request to the onbarding service according to the OnbardingProcess (see https://isobus.net/isobus/efdi/#messaging.v0.m0.onboarding-process). This process provides information to subsequently exchange information with the system.

According to the OnboardingResponse the messaging component shall allow client connections with only the provided authentication and connection criteria.

Endpoints can also be removed from the network. This process is called Offboarding Process (see https://isobus.net/isobus/efdi/#messaging.v0.m0.offboarding-processAfter the offboarding process, all messages in the endpoint's feed are deleted and the endpoint is no longer addressable in the system. The system shall also provide the ability to perform server-side offboarding. The consequences are the same as if an endpoint would offboard itself.

6.6 Authorization

The authorization for communication to other endpoints is up to the server system. The authorization can be environment and client specific. The authorization information is provided during the OnbardingProcess (see https://isobus.net/isobus/efdi/#messaging.v0.m0.onboarding-process).

6.7 Streaming

The messaging component can also handle streams. This ability is included in the capabilities of the messaging component. A messaging component without this ability is not able to handle streamed data.

For streaming data the following information is important

— The following verbs
  — STREAM_START
  — STREAM_STOP
  — STREAM_DATA
— Stream description / reference

If the messaging component receives a STREAM_START message with a noun, it shall send a message with the verb STREAM_START and the requested noun to all recipients mentioned in the request message once. When another endpoint requests a STREAM_START with the same noun, only recipients that have not received a STREAM_START with the requested noun shall receive such a message.

If an endpoint receives a STREAM_START for a specific noun, it shall start streaming data. This means that the client shall now send out the requested data as soon as this data is available. The time interval at which this happens can be endpoint-specific. When receiving messages with the verb STREAM_DATA, the messaging component shall forward the messages with the noun to recipients that

a) are capable to receive the verb STREAM_DATA in combination with the noun, and

b) are subscribed to the above mentioned combination, and

c) have requested streaming data by sending the verb STREAM_START in combination with the noun.

An endpoint can request the streaming to stop by sending the verb STREAM_STOP with the specific noun. This STREAM_STOP shall be sent explicitly if the client does not want to receive data anymore. Offboarding of an endpoint implies a STREAM_STOP for all streams of the offboarded endpoint. If the endpoint is offline the streaming of data shall continue and corresponding messages shall sent to the receiving endpoint's feed, if available.

If there are no recipients left for STREAM_DATA of a specific endpoint, the messaging component shall inform the streaming endpoints by sending a STREAM_STOP message to the specific endpoint.

The messaging component shall trigger the requested endpoints to start streaming data and to stop streaming data. The messaging component shall ensure that the streaming endpoints only receive the start and stop requests once. Moreover, the messaging component shall forward the incoming streamed data to the endpoints that have requested this data.

The data that is sent with the verb STREAM_DATA shall contain a reference. This reference refers to the description of the stream. Without this description, the streamed data cannot be interpreted correctly.

EXAMPLE When ISO 11783-10 TimeLog messages are sent, the corresponding Device Descriptor Object Pool (DDOP) is not known, these TimeLogs cannot be interpreted. When sending ISO 11783-10 TimeLogs, a reference to the DDOP is provided. The endpoint which receives these ISO 11783-10 TimeLogs and the corresponding reference can now decide whether it already knows this reference or not. If it already knows the reference, it can interpret the data. If it does not know them, it can request the data that are referenced by the reference.

6.8 Chunking

For large data, endpoints shall implement a concept for chunked data transfer. The endpoints shall control the segmentation.

If the endpoint wants to send a message that exceeds the maximum message size provided in MinimumCapabilities message it needs to chunk the message into smaller chunks. Afterwards the endpoint envelopes these chunks into the EndpointMessages and sends those chunked messages each as a single message. The endpoint shall ensure that the ChunkComponent information within the message is filled correctly. The server transports the messages without changing the content to the receiver. The recipient receives those chunks message by message. Afterwards the recipient can reassemble the binary content again. This is shown in Figure 16.

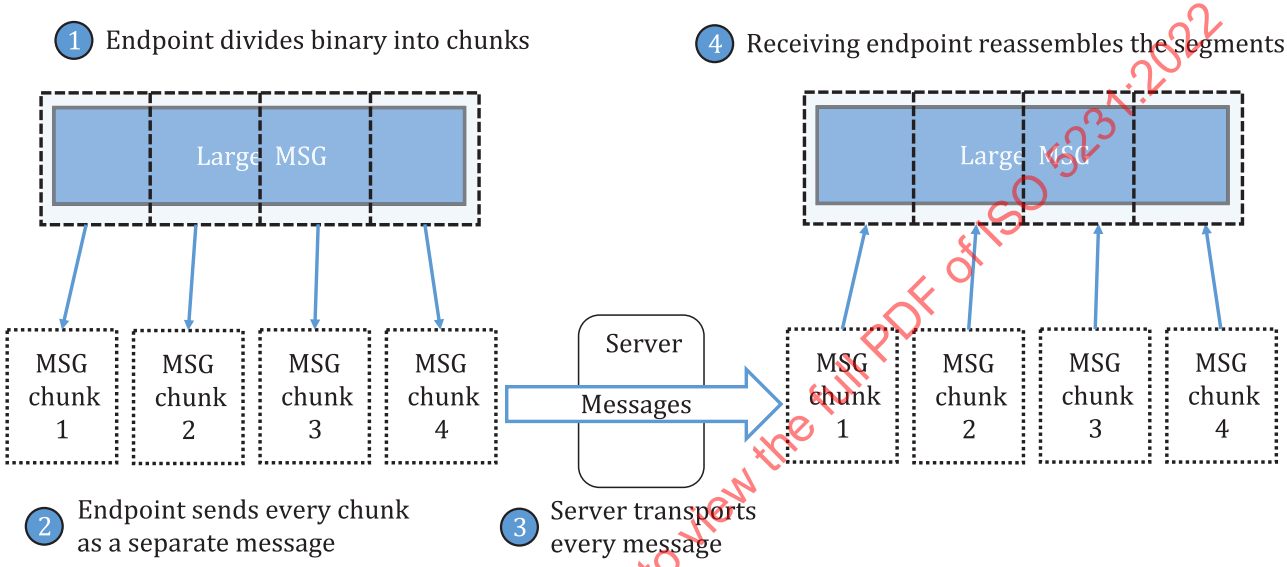


Figure 16 — Chunking of messages

6.9 Error handling

Inside the wrapping messages, ServerMessages that are sent by the server and EndpointMessages that are sent by any other endpoint, generic errors are defined. In case of a generic error, the response of a scenario does not contain the defined response message of this scenario. It only contains the generic error information. Additional error messages are defined for each scenario. The reason for this is that specific error codes are necessary for each scenario. These error codes can be found inside the response of each scenario and is called ResponseCode. Every response of a scenario shall define possible errors as ResponseCode. Technically, the ResponseCode is an enumeration. The value 0 always means OK and indicates that everything can be processed successfully.

7 Formal and semantic description

7.1 General

The list of messages, scenarios, scenario flows and scenario sets is extensible. These definitions are publicly available at the VDMA Database.

Every message, scenario, scenario flow and scenario set have a package name consisting of a unique identification including semantic versioning. The semantic versioning consists of the MAJOR version and the MINOR version. MAJOR will be increased if incompatible changes are published and MINOR will be increased as new functionalities compatible with the existing definitions are released. MAJOR versions are indicated by “v<major-version>” e.g. “v1”. MINOR versions are indicated by “m<minor-version>” e.g. “m4”. The general structure has the following layout:

<unique-namespace>.v<version-major>.m<version-minor>.<name>

EXAMPLE The identification of the basic messaging scenario set: ‘messaging.v0.m0.basics’

All definition files shall be encoded using UTF-8.

7.2 Noun

Nouns are messages that are defined within the protobuf definitions. Each message is contained within a package. Each message has an URL that describes its type. This URL is called type_url. A type_url always looks like the following: ‘iso.org/5231/<package-name>.<message-name>'. The <package-name> can be found at the beginning of each proto file after the label ‘package’, e.g. ‘messaging.v0.m0.messages’. The <message-name> can be found in each proto file after the label ‘message’, e.g. ‘Onboarding’.

EXAMPLE For the onboarding message the type_url looks like this: ‘iso.org/5231/messaging.v0.m0.messages.Onboarding'.

In this case, the filled and serialized protobuf message is the data of the scenario.

7.3 Scenario

All scenario flows are described in files with the extension “scenario”. In general, all scenario fields shall have the following structure: “@scenario {attribute-name} <attribute-value>”. All fields are mandatory. Each scenario description consist of :

- Scenario ID
  - Description: machine readable identification of this scenario, contains semantic versioning, ID shall be unique. <scenario-id> shall be replaced by the scenario ID.
  - Formal representation: @scenario {id} <scenario-id>
  - Example: @scenario {id} messaging.v0.m0.send-feed-header
- Scenario Name
  - Description: human readable name for this scenario, without semantic versioning. <scenario-flow-name> shall be replaced by the scenario name.
  - Formal representation: @ scenario {name} “<scenario-name>”
  - Example: @scenario {name} "Send Feed Header"
- Scenario Request Verb
  - Description: verb to be used for the request of this scenario. <scenario-request-verb> shall be replaced by the verb used in this request.
  - Formal representation: @ scenario {request.verb} <scenario-request-verb>
  - Example: @scenario {request.verb} GET
- Scenario Request Noun
  - Description: noun to be used for the request of this scenario, defined by the type_url of the used message. <scenario-request-noun> shall be replaced by the noun used in this request.
  - Formal representation: @scenario {request.noun} <scenario-request-noun>

  - Example: @scenario {request.noun} messaging.v0.m0.messages.FeedHeader
- Scenario Response Verb
  - Description: verb to be used for the response of this scenario. <scenario-response-verb> shall be replaced by the verb used in this response.
  - Formal representation: @scenario {response.verb} <scenario-response-verb>
  - Example: @scenario {response.verb} SHOW
- Scenario Response Noun
  - Description: noun to be used for the response of this scenario, defined by the type_url of the used message. <scenario-response-noun> shall be replaced by the noun used in this response.
  - Formal representation: @scenario {response.noun} <scenario-response-noun>
  - @scenario {response.noun} messaging.v0.m0.messages.FeedHeaderResponse

To fill the required information (verb, noun and data), the fields verb and noun of the endpoint and server proto messages shall be used. The field noun is represented as google protobuf Any. The "Any" message type lets you use messages as embedded types without having their proto definition. The Any consists of a field type_url and a field value. The noun shall be put inside the field type_url of the "Any". The data shall be put inside the field value of the "Any". The data is the actual content to be transmitted within this message. Scenarios may not contain any content.

7.4 Scenario flow

All scenario flows are described in files with the extension “flow”. In general, all scenario flow fields have the following structure: “@flow {attribute-name} <attribute>”. All fields are mandatory. Each scenario flow shall be described by the following attributes:

- Scenario Flow ID
  - Description: machine readable identification of this scenario flow, contains semantic versioning, ID shall be unique. <scenario-flow-id> shall be replaced by the scenario flow ID.
  - Formal representation: @flow {id] <scenario-flow-id>
  - Example: @flow <id> messaging.v0.m0.delete-messages
- Scenario Flow Name
  - Description: human readable name for this scenario flow, without semantic versioning. <scenario-flow-name> shall be replaced by the scenario flow name.
  - Formal representation: @flow {name] “<scenario-flow-name>”
  - Example: @flow {name} "Delete Messages"
- Scenario Flow Step
  - Description: scenario that is to be executed in this step, identifiable by its unique scenario ID. For optional scenarios the scenario ID is surrounded by square brackets. If a scenario flow contains several scenarios, there shall be several of these flow step entries. The chronological order of the scenarios is defined by the order in which they are defined from top to bottom. It also indicates whether it is an A-step or a B-step. <scenario-id> shall be replaced by the used scenario ID. <step direction> shall be replaced with “<step a>” if it is an A-step or replaced with “<step b>” if it is a B-step.
  - Formal representation: @flow {step.direction} <scenario-id>

- Example:
  - Optional scenario: @flow {step.a} [messaging.v0.m0.send-feed-header]
  - Required scenario: @flow {step.a} messaging.v0.m0.send-feed-message-delete

7.5 Scenario set

All scenario sets are described in files with the extension “set”. In general, all scenario set fields have the following structure: “@set <attribute-name> <attribute>”. All fields are mandatory. Each scenario set shall be described by the following attributes:

- Scenario Set ID
  - Description: machine readable identification of this scenario set, contains semantic versioning, ID shall be unique. <scenario-set-id> shall be replaced by the scenario set ID.
  - Formal representation: @set {id} <scenario-set-id>
  - Example: @set <id> messaging.v0.m0.basics
- Scenario Set Name
  - Description: human readable name for this scenario set, without semantic versioning. <scenario-set-name> shall be replaced by the scenario set name.
  - Formal representation: @set {name} “<scenario-set-name>”
  - Example: @set <name> "Basics"
- Scenario Set Flow
  - Description: scenario flow that is part of this scenario set, identifiable by its unique scenario ID. <scenario-flow-id> shall be replaced by the used scenario flow ID.
  - Formal representation: @set {flow} <scenario-flow-id>
  - Example: @set <flow> messaging.v0.m0.delete-messages

7.6 Documentation structure

For the source files at the VDMA Database, that contain the messages and scenario related definitions there is a distinction between the messages – these are located as protobuf files in the subfolder “messages” – and the definitions – these are located in the subfolder “definitions”. Every file with the extension “proto” shall contain one message definition and shall follow the protobuf 3 schema. Every scenario, scenario flow and scenario set definition can be found inside the “definitions” folder.

Messages can be found under the following path: <package-name>/messages

Definitions can be found under the following path: <package-name>/definitions

7.7 Structure of protobuf files

7.7.1 General structure

A template protobuf file is shown in Figure 17. Every protobuf file shall use Protocol Buffers Version 3. To ensure this, at the beginning of the file the label syntax = “proto3” shall be used. Hereafter, the package specifier shall be added by adding the package label and the package name itself. Also, the import statements to use and import other protobuf definitions shall be added. In addition, every message shall contain a field extension. This field has the field number 2048 and can be used for proprietary purposes.

Every message and every field should contain comments. To add documentation comments for message definitions, a comment block should be inserted before the start of the message definition. This comment block shall start with a forward slash followed by two asterisks. The end of the comment block is marked by one asterisk symbol followed by a forward slash. To add documentation comments for message fields an inline comment after the end of the field definition may be used. Such comment shall start with three forward slashes. The end of an inline comment block is the end of line.

```
syntax="proto3";
package messaging.v1.m2.examples;
import "google/protobuf/any.proto";

/**
 * Message comment for an example message.
 */

message ExampleMessage {

  string endpoint_id = 1;                          /// This is a field comment

  repeated google.protobuf.Any extension = 2048; /// For proprietary purposes

}
```

Figure 17 — General structure of protobuf files

7.7.2 Error code structure

Every message used as response noun of a scenario shall contain a response code. The structure is the same for each message and can be seen as a template in Figure 18. For that purpose, an enumeration with the name ResponseCode shall be included in every message. The field shall be named "response_code" and use the field index "1". This enumeration contains message-specific information about whether processing was successful or whether errors occurred. The entry with the value 0 always has the meaning OK. This is the default case. Conversely, this means that if the ResponseCode is not equal to 0, something could not be completed with the status OK. The field with the number 1 has the name response_code and the type ResponseCode.

```
syntax="proto3";
package messaging.v1.m2.examples;
import "google/protobuf/any.proto";

/**
 * Message comment for a template scenario response noun.
 */

message TemplateScenarioResponseNoun {

  enum ResponseCode {

    OK = 0;                                          /// No error occurred

  }

  ResponseCode response_code = 1;                    /// Contains processing information

  repeated google.protobuf.Any extension = 2048; /// For proprietary purposes

}
```

Figure 18 — Scenario response protobuf template

8 Transport layer mapping

8.1 General

Messages sent by the messaging component have a format that is described within the ServerMessage. Messages sent to the messaging component have a format that is described within the EndpointMessage.

The protocol is technically not bound to MQTT. In this revision of this standard only MQTT shall be used as transport protocol so the usage of MQTT will be described in the following paragraphs.

Figure 19 shows the communication with the server when using MQTT. In this case, the endpoint will publish the protobuf serialized EndpointMessages on the 'inbox' topic and the server subscribes to that topic. Messages that are sent to an endpoint will be publish by the server as protobuf serialized ServerMessage on the 'outbox' topic. The endpoint shall subscribe to that topic. Both topics are specified within the onboarding response. No polling is required.

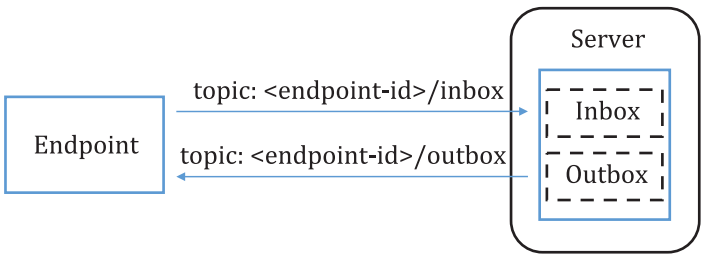


Figure 19 — Communication with Server using MQTT

The MQTT Quality of Service Level for publishing and subscribing shall be 2.

8.2 Service discovery

8.2.1 General

The EFDI over MQTT shall be announced as the service name “efdi-onboarding”. Depending on the scope of the used Ethernet network the service discovery shall be used by using the DNS-SD [DNS-SD] for connection to internet based systems. For local area networks, the mDNS [MDNS] mechanism shall be used to discover the availability of the onboarding service.

8.2.2 Internet based

EFDI onboarding service uses the same DNS-SRV-Lookup mechanism which is described in section 3.2.1 of [XMPP] . The preferred process for FQDN resolution is to use [DNS-SRV] records as follows:

a) The initiating entity constructs a DNS-SRV query whose inputs are:

   1) a Service of “efdi-onboarding"

   2) a Proto of “tcp"

   3) a Name corresponding to the “origin domain" TLS-CERTS [TLS] of the FMIS-Exchange service to which the initiating entity wishes to connect (e.g., ”example.net" or ”flow.example.com")

b) The result is a query such as _efdi-onboarding._tcp.example.net.

   or

   _efdi-onboarding._tcp.flow.example.com.

c) If a response is received, it will contain one or more combinations of a port and FQDN, each of which is weighted and prioritized as described in DNS-SRV. (However, if the result of the DNS-SRV lookup is a single resource record with a Target of ".", i.e., the root domain, then the initiating entity shall abort SRV processing at this point because according to DNS-SRV such a Target means that the service is decidedly not available at this domain.

d) The initiating entity chooses at least one of the returned FQDNs to resolve (following the rules in DNS-SRV), which it does by performing DNS A or AAAA lookups on the FQDN; this will result in an IPv4 or IPv6 address.

e) The initiating entity uses the IP address(es) from the successfully resolved FQDN (with the corresponding port number returned by the DNS-SRV lookup) as the connection address for the receiving entity.

f) If the initiating entity fails to connect using that IP address but the A or AAAA lookups returned more than one IP address, then the initiating entity uses the next resolved IP address for that FQDN as the connection address.

g) If the initiating entity fails to connect using all resolved IP addresses for a given FQDN, then it repeats the process of resolution and connection for the next FQDN returned by the DNS-SRV lookup based on the priority and weight as de_ned in DNS-SRV.

h) If the initiating entity receives a response to its DNS-SRV query but it is not able to establish an onboarding service connection using the data received in the response, it SHOULD NOT attempt the fallback process described in the next section (this helps to prevent a state mismatch between inbound and outbound connections).

i) If the initiating entity does not receive a response to its DNS-SRV query, it SHOULD attempt the fallback process described in the next section.

If the service discovery fails the connecting system shall provide a way to manually configure the necessary connection information.

8.2.3 Local Area Network

In accordance to the internet based definition of DNS records the mDNS service discovery method shall use the same records.

Annex A
(informative)

OAGIS Verbs

The Open Applications Group, Inc. (OAGi), provided the Open Applications Group Integration Specification (OAGIS) verb list and verb message/response pattern which is used to describe scenarios. The OAGIS verb list contains these verbs:

— Acknowledge

— Cancel

— CancelAcknowledge

— Change

— ChangeAcknowledge

— Confirm

— Get

— Notify

— Process

— Show

— Sync

— SyncResponse

An example and the message/response pattern can be seen in Table A.1.

Table A.1 — OAGIS message/response pattern

Message Verb	Verb Response	Example
Process	Acknowledge	Request: ProcessPurchaseOrder Response: AcknowledgePurchaseOrder
Change (with support for add, change, delete, or replace)	ChangeAcknowledge	Request: ChangePurchaseOrder Response: ChangeAcknowledgePurchaseOrder
Cancel	CancelAcknowledge	Request: CancelPurchaseOrder Response: CancelAcknowledgePurchaseOrder
Sync (with support for add, change, delete, or replace)	SyncResponse	Request: SyncCustomerAddress Response: SyncResponseCustomerAddress
Get	Show	Request: GetPurchaseOrder Response: ShowPurchaseOrder
Notify	*No response required*	NotifyRemittanceAdvice

How the OAGIS verbs are used can be seen in 6.2. Hierarchical structure of Scenario Sets, Scenario Flows and Scenarios. The used and extended definitions can be seen in 6.2.2 Scenario.