

Standard Robotic System Communication Interface

Revision history

Version	Date	Author	Description
1.0	11-03-2025	MIV	Initial version
	26-04-2025	PDA	Release

Review history

Version	Date	Reviewer	Notes / description
1.0	12-03-2025	AEF	Review
	24-06-2025	PDA	Release

Table of Contents

1 Introduction	4
1.1 Feature Overview	4
1.2 References	4
1.3 Glossary	4
1.4 Copyright and right to use under Apache License 2.0	5
1.5 Background / 5G-Robot	5
2 Telegram Format	6
2.1 Telegram Buildup	6
2.2 Formats	6
2.3 Required Properties	6
2.3.1 Interface Version	6
3 Common Data Types	8
3.1 Header	8
3.2 Use of PackML	10
4 Communication	12
4.1 Communication Protocol	12
4.2 Multiple Interfaces for one cell	12
4.3 Specific Equipment Interface Specification	12
4.4 Sequence Numbering and Acknowledgements	12
4.4.1 Sequence Number	13
4.4.2 Acknowledging Incoming Messages	13
4.5 Keep-alive Interval/Timeout	13
4.5.1 Send an Empty Telegram	13
4.5.2 Request an Acknowledgement in a Telegram That Typically Does Not Require Any	14
4.6 Reaction time monitoring	14
5 Communication Telegrams	15
5.1 Protocol Acknowledgement	15
5.2 Empty Telegram	19
5.3 Error Telegram	20

1 Introduction

This document describes the basic elements that are used in telegrams between a High Level Control System and the IT and OT systems.

1.1 Feature Overview

The document defines the data/information that should be exchanged between systems. The interface is protocol-agnostic and can be implemented with various protocols, whether it is event based (eg. MQTT) or request based (eg. REST).

Examples of telegrams will be shown using the JSON schema. How data is transmitted using various communication technologies will be specified in the respective interface documents.

1.2 References

The following references have been used to devise the document.

ID	Document	Description
[1]	Title: ISO 8601 Date and time format	https://www.iso.org/iso-8601-date-and-time-format.html
[2]	Title: PackML	https://www.omac.org/packml
[3]	Title: PackML State Model	https://en.wikipedia.org/wiki/PackML#/media/File:Packml_highres.png

1.3 Glossary

Term	Description
High-Level Control (HLC)	The High-Level Control system is a general designation for systems in the high-level control and integration layers (layer 2 and 3 of the ISA-95 model). The HLC can also be a single system, consisting of a combination of WCS, WMS and MES.
- Equipment - System - Cell	A piece of equipment, an individual system or otherwise an actor using the standard interfaces to communicate with e.g. a HLC.
Information Technology (IT)	IT refers to the enterprise-level network and computer systems used for business processes, distinct from the control systems that directly manage industrial operations.
Operation Technology (OT)	Operational Technology is hardware and software that monitors and controls industrial equipment, assets, and processes. It is commonly found in factories and warehouses.

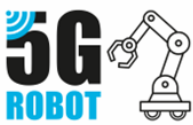
1.4 Copyright and right to use under Apache License 2.0

The copyright on this document and any contained specifications, designs, etc. shall remain with the company: Intelligent Systems A/S, Havnevej 11, DK-9560 Hadsund, DENMARK.

Any referenced 3rd party designs, technologies and or other IP shall remain with the original owners.

The reference architecture, designs and the included open standard integration interfaces are open and free to use under Apache License 2.0. For details please see <https://www.apache.org/licenses/LICENSE-2.0>.

1.5 Background / 5G-Robot



Parts of this document / release was made in the **5G-Robot** project also known under the long name **5G-ENABLED AUTONOMOUS MOBILE ROBOTIC SYSTEMS** - the largest innovation project that has been launched under the Innovation Fund Denmark's (IFD) Grand Solutions program.

The groundbreaking project united Denmark's leading robot, automation and factory digitalization companies as technology vendors, research partners and industry-leading end-user companies.



Illustration: Project partner logos.

The aim of the project was to revolutionize manufacturing - paving the way to smart production and smart factories and the application of a number of new technologies in production and manufacturing including 5G wireless communication, cloud and edge computing and digital twin.

Intelligent Systems played a leading role in the project, providing the glue that ties the robotic solutions of the partners together making the work as one - i.e. one connected integrated intelligent manufacturing system.

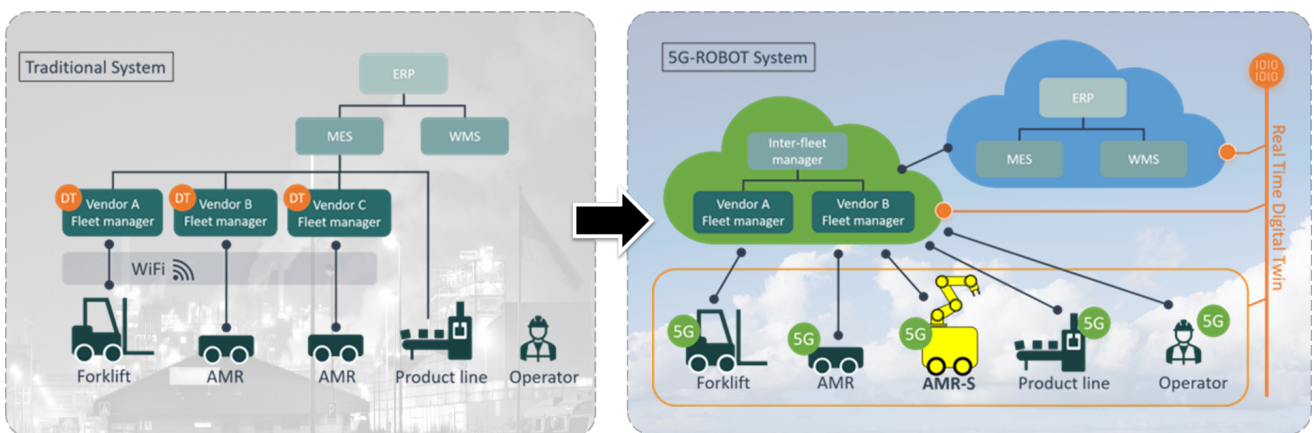


Illustration: The aim is to revolutionize manufacturing paving the way to smart production and smart factories.

Read more about the 5G-Robot project here: <https://www.5gsmartproduction.aau.dk/5g-robot>

2 Telegram Format

This section describes the structure and format of the message telegrams.

2.1 Telegram Buildup

All telegrams comprises of the following:

1. Header
2. Payload

Information in the header is used for control purposes. Some of the information is used solely by the interface implementation and some of the information is meant for the application (for either the sender or receiver).

In order to have the interface be agnostic with regard to communication technologies, the header information has enough data to implement an interface in many technologies. But some parts of the information in the header may be ignored if not relevant to a specific protocol/technology.

The payload is defined in subsequent interface descriptions.

2.2 Formats

The following data format types are supported:

- Integer (default signed 64-bit length unless otherwise specified).
- Decimal (default 64-bit length unless otherwise specified). Used for numbers with decimals.
- String (UTF-8 unless otherwise specified)
- Boolean (lowercase true/false)
- Array/List
- Date (formatted as a String following the ISO 8601 standard [1])
- UUID (formatted as a String)
 - UUID v7 is recommended.
- Null (or the property is omitted)
- Object (defined in subsequent interfaces)
 - Object types may contain many properties, all of which is one of the formats specified here.

2.3 Required Properties

All messages should contain the Header object (see [Header](#)) unless otherwise specified in the telegram's details.

2.3.1 Interface Version

All messages must contain a "version" field containing the version of the interface being used. The value is formatted as a string.

The purpose of this is to be able to communicate with different equipment where each of these may have different interface versions implemented. Adding the version makes it possible to verify that both parties are able to understand the telegrams between them. If a version is not valid, the acknowledgement must show an error and explain that the cause is invalid version. It will be good practice to explain accepted versions in the error message.

The value is formatted as "v<MAJOR>.<MINOR>".

Example

```
{  
  "version": "v1.0"  
}
```

3 Common Data Types

3.1 Header

Data type: **Header**

The header is a mandatory property and must be included in all messages.

Note that the Acknowledgement-telegram has exceptions for this header. Please refer to the [Protocol Acknowledgement](#) section for details.

Property	Type	Description
system-id	String	An identifier of the system that the HLC (or similar) is communicating with. This means that telegrams sent to / received from the system will use the same identifier. For example, if several production cells communicate with the same HLC, the system-id is used to distinguish between them.
telegram-type-id	String	Name/type of the telegram. This allows for decode the telegram correctly.
version	String	Designates which version of the interface is used. This will be the version of the most specialized interface being used (that builds upon/incorporates other interfaces). For example, a production cell interface utilizes the following interfaces: <ul style="list-style-type: none"> - Standard Interface v1.0 - Standard Equipment Interface v2.1 - Standard Transport Interface v1.0 - Standard Production Cell Interface v2.4 The production cell interface is the most specialized and targets this cell directly, and incorporates/refers to the other interfaces, including their versions. Therefore, the value for all telegrams for this production cell should be v2.4.
message-id	UUID	Unique identifier for the message. A new identifier is generated for each non-retransmitted message. This is used to distinguish between messages (without solely relying on the seq-no), and allows for referring to a specific message.
request-id	UUID	Optional. Only provided if the telegram is sent as a response to another telegram, i.e., response to a request or an error response . Is set to the message-id of the telegram it is a response to.
timestamp	Date	Timestamp for when the telegram was sent.
seq-no	Integer	Sequence number for the message. From 0 - 65535. Initially starts at "1". Incremented by the sender. Reset to "1" after sending the value "65535". Value "0" is used for special cases and is always accepted.

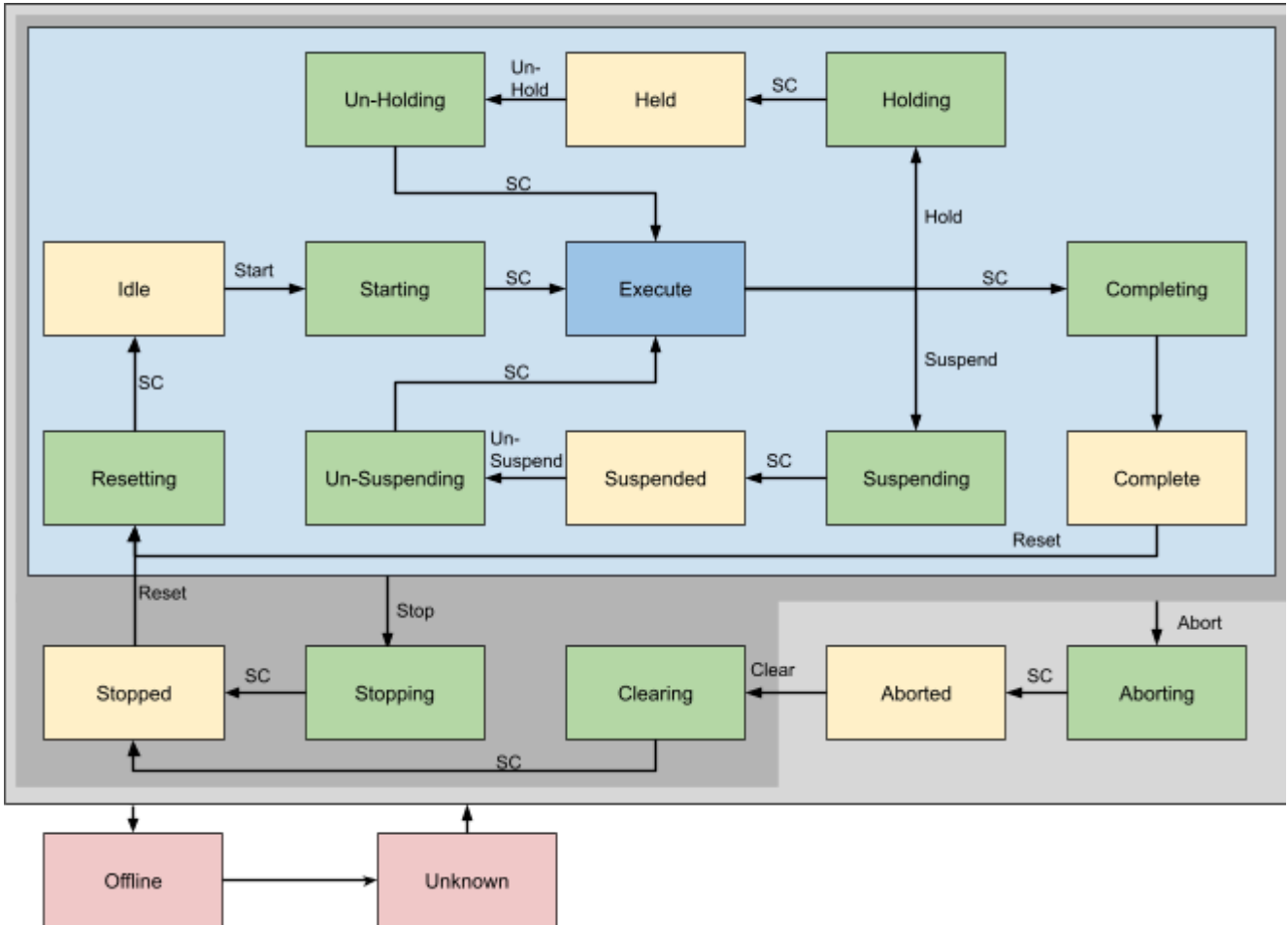
		Omitted for acknowledge telegrams. For more information, see Sequence Number .
need-ack	Boolean or null (default true)	Optional. Interpreted as true if omitted or null. Used to specify that if an acknowledgement is required. Note: Always false for acknowledge telegrams.
retransmitted	Integer or null	Only used if the acknowledgement is not received within a time limit. Number indicates how many times the telegram has been retransmitted (1 = one retransmission).

Example

```
{
  "header": {
    "system-id": "ProdCell-7A",
    "telegram-type-id": "BatchInformation",
    "version": "v2.0",
    "message-id": "3c378b3a-7b3b-4846-b18d-7a62309e4d89",
    "timestamp": "2023-01-19T09:01:50Z",
    "seq-no": 1574,
    "need-ack": false,
    "retransmitted": 13
  }
}
```

3.2 Use of PackML

PackML [2] is used to represent the states of the system in the standard interface.



Modified PackML State model. For the original, see [3].

The PackML standard illustrated above is commonly used for equipment, however, some additional states have been added for the standard interfaces:

Offline: This extra state will be used to show that there is no connection/communication with the equipment, and therefore none of the PackML states are to be considered valid. In some cases, equipment continues to operate without communication and the Offline state will indicate that the HLC is unable to receive information from the cell.

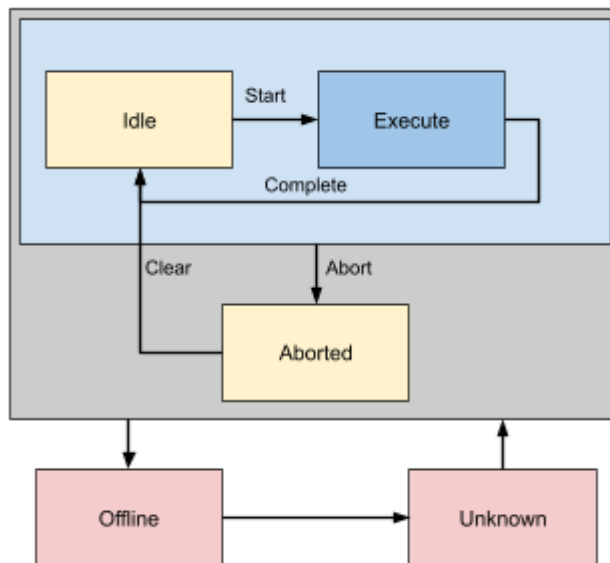
Unknown: This state is only used until the first state has been received from the equipment. It indicates that the equipment has not communicated with the HLC yet, and therefore, it is not possible to determine a valid state.

Note that the states in the figure have been grouped with colored boxes. These boxes group related states, but are also used to indicate that any state in a given group can lead to the states that are connected to the group. For example, from any state in the blue box, the system can go to state aborting, i.e., the states *Idle*, *Held*, *Execute*, etc. can all lead to state aborting. Likewise, any of the standard PackML states can lead

to the state *Offline*. For the state *Unknown*, the system can go to any of the standard PackML states on the first received state from the system.

When using PackML states, it is possible to limit the states to be used in each instance (e.g. in one case omitting *Suspending*, *Suspended* and *Un-Suspending*). Thus, it is not necessary to implement all states if they are not relevant.

The minimum state machine consists only of *Idle*, *Execute*, *Aborted*, *Offline* and *Unknown*. See the figure below for an example.



4 Communication

This section describes how systems communicate with each other using the Open Standard Interface. The application data itself, i.e. the data transferred between the two systems, will be through the telegrams described in the following chapters.

4.1 Communication Protocol

This interface defines the methods for sending information across a communication channel and outlines the available features. The features defined within this interface and its specializations provide all the necessary information to implement it for a specific protocol (e.g., MQTT, socket, REST). Details on how these features are used in a particular protocol implementation are documented separately for that protocol.

For information on how to implement the standard interfaces using a specific protocol such as MQTT, REST, etc. or a socket connection, a document exists that provides implementation details for how to use the telegram's header and payload should be formatted.

4.2 Multiple Interfaces for one cell

Each interface is designed with a specific primary focus. However, a system can integrate features from multiple standard interfaces, meaning the communication channel might carry telegrams defined by different interface specifications. Importantly, each telegram will function exactly as described within its originating interface. Utilizing these telegrams may also involve processing their corresponding responses as defined in their respective interfaces. Therefore, incorporating additional interfaces into a system grants access to all the telegrams and their associated responses defined within those interfaces.

An example is a production cell that uses the standard production cell interface. During specification of this cell it is determined that a telegram in the standard transport interface is required. This one telegram has 2 other related telegrams for responses to the desired telegram. So the production cell will be required to implement 3 telegrams from the standard transport interface in order to use the one desired.

4.3 Specific Equipment Interface Specification

To address partial interface implementations, vendors must provide a specification detailing the supported parts. However, certain interdependent telegrams cannot be separated.

For instance, a production cell vendor might choose to implement only three telegrams from the production cell interface. This is acceptable if these telegrams do not depend on other unimplemented telegrams. Therefore, utilizing telegrams from standard interfaces does not necessitate implementing all telegrams within those interfaces. Instead, vendors must clearly define the specific portions of the interfaces they will use.

When specifying the interfaces to be used, it is crucial to indicate their versions. Standard interfaces may evolve with added functionality, leading to new versions. Specifying a particular version of a standard interface ensures that the relationships between telegrams remain consistent as they were in that specific version. Future requirements for a telegram are only relevant if included in the specified interface version.

4.4 Sequence Numbering and Acknowledgements

To ensure synchronized communication and ordered delivery of the telegrams, the interface contains a sequence and an acknowledgement concept.

4.4.1 Sequence Number

All telegrams must contain a sequence number, except for acknowledgements (see [Protocol Acknowledgement](#) for more details). The sequence number can have the values 0-65535 (uint16), and is unique for each direction (one counter for one system, another counter for the other). The sequence number starts at 1 and is incremented by one for each telegram. After the sequence number "65535" is sent, the following number to be used is "1". The sender is responsible for this and as both systems can act as sender, they have their own sequence numbers.

The sequence number "0" (zero) represents a reset of the counter in that direction (e.g. HLC-> equipment) and is always accepted. After acknowledging this message, the next expected sequence number is set to "1". A reset will not influence the sequence number in the opposite direction.

When establishing a connection for the first time, or after a connection loss, it is recommended to reset the sequence number by sending a telegram with sequence number 0 as the first message.

4.4.2 Acknowledging Incoming Messages

Each message must be acknowledged unless otherwise specified in the specific telegram. Examples of telegrams that do not require acknowledgement are fire-and-forget telegrams that may be sent frequently, containing a payload that is obsolete within a short period of time. Acknowledgements should be sent as described in section [Protocol Acknowledgement](#).

Note that only one telegram can be sent at a time (one in each direction). If required by the telegram, an acknowledgement must be received before sending the next telegram. The telegram is retransmitted if an acknowledgment is not received within a time limit (default 1 second). When retransmitting a telegram, the telegram is a copy of the first telegram in all details (including sequence number), except for the addition of a retransmission flag and updated timestamp for which the telegram was sent.

Acknowledgement-messages themselves are *not* included in this restriction; they can be sent by a client at any time, even when that client is waiting for another acknowledgement.

If no acknowledgement is required, the next message may be sent immediately.

4.5 Keep-alive Interval/Timeout

It is both sides' responsibility to monitor both receiving and sending telegrams. If there is a period of no communication, or if the communication is one-way, both sides must take action to verify that the other party is still connected. To this end these following features may be used.

There will be different ways of handling keep-alive signals depending on the chosen protocol. Two features are added to the interface to make it possible to implement keep-alive functionality - one or both of these features may be used. The features are:

1. Send an empty telegram.
2. Request an acknowledgement in a telegram that typically does not require any.

How to use the features are described as follows.

4.5.1 Send an Empty Telegram

If there are no telegrams being sent or received, a keep-alive telegram can be sent, which is functionally an [empty telegram](#) - called an empty telegram here. Sending this telegram allows for requesting an acknowledgement telegram from the opposite party and thus verify that there is a connection both ways.

4.5.2 Request an Acknowledgement in a Telegram That Typically Does Not Require Any

If the communication becomes one-sided for a period of time (i.e. only one party is actively sending telegrams), either party is allowed to send an empty telegram to verify the connection. Instead of this extra message, in case one party only transmits telegrams which do not require acknowledgements, that party can explicitly require acknowledgements for one of the messages to request a response. This way telegrams already being sent are used instead of sending an additional telegram.

4.6 Reaction time monitoring

As part of keep alive functionality, timing can also be monitored on the interface. In case a keep-alive telegram is not acknowledged in time an alarm is raised, but once it is received the message will then be able to help determine the delay on the communications channel.

Before sending a message, the timestamp is saved. A rolling window of a number of telegrams and their sequence numbers is saved in a table in memory. The timestamp is to be saved for telegrams that request an acknowledgement from the recipient.

When receiving a telegram, the timestamp is saved so that the reception time can be stamped into the acknowledgement telegram - if one is to be sent. After handling the message an acknowledgement telegram is sent back (if requested) with the reception time registered.

When receiving an acknowledgement telegram using the sequence number to look up the original telegram's timestamp, it is possible to calculate the time it took to handle the telegram.

This feature helps in determining if the communication is lost or if it is just slower than expected.

5 Communication Telegrams

5.1 Protocol Acknowledgement

Telegram Type ID: ack

When a telegram is received, an acknowledge telegram is to be returned immediately, unless the need-ack is set to false in the header of the telegram. For more information, see [Sequence Numbering and Acknowledgements](#).

For acknowledgement messages, the header uses the sequence number of the message to acknowledge and acknowledgment required or retransmitted fields are omitted.

Properties

Property	Type	Description
seq-no	Integer	The same sequence number as in the telegram being acknowledged.
error-code	String or null	An error code indicating a faulty package. If omitted it corresponds to NO_ERROR.
received-timestamp	Date	Timestamp of reception of telegram.

Error codes

Error code	Description
NO_ERROR	Telegram was accepted without any issues.
REPEATED_TELEGRAM	Sent when a telegram is – based on sequence number - considered “repeated”. Telegram <u>will not</u> be processed.
SEQ_TOO_HIGH	Sent when a sequence number is larger than current value is received. The telegram <u>will</u> be processed and the sequence counter will be adjusted by the receiver.
SEQ_TOO_LOW	Sent when the sequence is too small. Telegram is considered too small when deviating more than -1 (default) from the expected sequence number. Else it is considered as a repeat telegram. Telegram <u>will not</u> be processed.

Examples

Successful acknowledgement

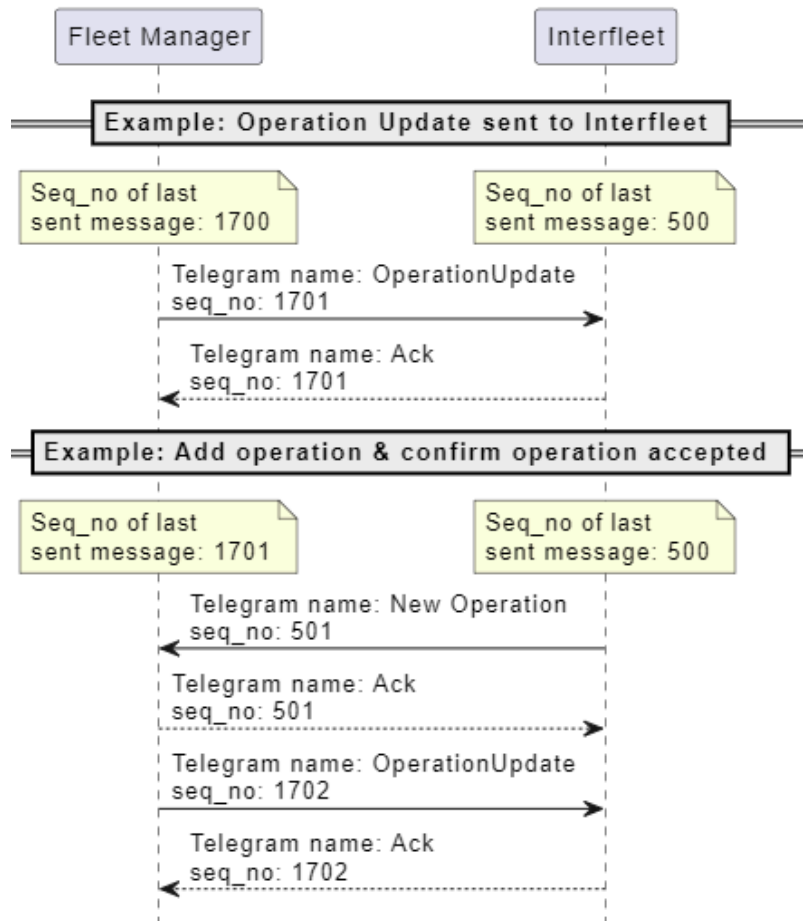
Successful acknowledgement (implicit no error)

```
{
  "header": {
    "message-id": "3c378b3a-7b3b-4846-b18d-7a62309e4d89",
    "need-ack": false,
    "timestamp": "2023-01-19T09:01:50Z",
    ... // Other header information
  },
  "seq-no": 2255
}
```

Successful acknowledgement (explicit no error)

```
{
  "header": {
    "message-id": "3c378b3a-7b3b-4846-b18d-7a62309e4d89",
    "need-ack": false,
    "timestamp": "2023-01-19T09:01:50Z",
    ... // Other header information
  },
  "seq-no": 2255,
  "error-code": "NO_ERROR"
}
```

The following figure illustrates an example of using acknowledgements between an interfleet and fleet manager (telegram names are examples). This example is to show how acknowledgement and sequence numbers work.



Sequence number is larger than expected

Example: The last telegram received had sequence number 42 and was acknowledged and processed. The next telegram is expected to have sequence number 43, but a telegram with sequence number 45 is received. The telegram is acknowledged with the following acknowledgement telegram and **will** be processed by the receiver.

If telegrams with sequence numbers 43 and 44 are received later, they will be rejected due to the sequence number being smaller than expected.

```

{
  "header": {...},
  "seq-no": 45,
  "error-code": "SEQ_TOO_HIGH"
}
  
```

Sequence number is smaller than expected

Example: The last telegram received had sequence number 42 and was acknowledged and processed. The next telegram is expected to have sequence number 43, but a telegram with sequence number 39 is received. The telegram is acknowledged and but **NOT** processed by the receiver with the following acknowledgement telegram.

```
{  
  "header": {...},  
  "seq-no": 39,  
  "error-code": "SEQ_TOO_LOW"  
}
```

The sender should resend the package with a valid sequence number, e.g. by resetting the counter (sending the number "0") or by incrementing the sequence number by a large number.

Repeated telegram

If the sequence number deviates from the expected with exactly -1 (or other project-specific value) the telegram is considered as a repeating telegram. It has already been processed and will **not** be processed again, but acknowledged with an error code:

```
{  
  "header": {...},  
  "seq-no": 39,  
  "error-code": "REPEATED_TELEGRAM"  
}
```

Incorrect acknowledge

If the sender receives an acknowledgement with a sequence number not corresponding to the expected (last sent telegram), the sender ignores the acknowledgement and waits for the correct acknowledgement. If not received within the time-out the telegram is retransmitted.

5.2 Empty Telegram

Telegram Type ID: empty

Telegram transmitted by the HLC or equipment. The receiver should respond with an acknowledgement with a timestamp.

These messages are primarily used to verify that a system is receiving telegrams and how long telegrams are underway. This may be used as a keep-alive telegram or to monitor time-critical communication with other systems.

Properties

Property	Type	Description
		(This telegram does not contain any additional properties)

Example

```
{  
  "header": {...}  
}
```

5.3 Error Telegram

Telegram Type ID: error

Telegram transmitted by the HLC or equipment. Used when an error occurs and contains a pre-defined error reason, and optionally a reference to a resource.

Note that for communication-related errors, see [Sequence Numbering and Acknowledgements](#) and [Protocol Acknowledgement](#).

If this error was caused due to a request, the error should include the Telegram ID (see [Header](#)) of the request. The error and reason provided in the error telegram must be based on the options that are noted in the specific telegram that the error message is a response to.

The error is sent asynchronously; i.e. it is not synchronous as the [Protocol Acknowledgement](#) is.

Additionally, the message-id from the header of the original message must be set as request-id in the [Header](#) of the error message. This is to ensure that the client that receives the error message knows which telegram that the error message relates to.

Properties

Property	Type	Description
error	String	String containing the error ID. See the specific client's interface for valid error IDs.
reason	String or null	Optional. String containing a human-readable error message or null.
resource-id	String or null	Optional. String containing a reference (e.g. an ID) to a resource that is relevant to the error.
request-telegram-type-id	String or null	Optional. Used if the error was caused due to a request. If so, it should be the same as the Telegram ID that caused this error.

Example

```
{
  "header": {
    "telegram-type-id": "error",
    "message-id": "504c198d-6140-4b69-b54d-c21eafb732de",
    "request-id": "ebe1134e-ea29-45bb-874c-f30c1d9c1748",
    ... // Other header information
  },
  "error": "NOT_FOUND",
  "reason": "The entity with the provided name was not found",
  "resource-id": "res1957-ab12",
  "request-telegram-type-id": "system.locations"
}
```