TECHNICAL SPECIFICATION

ISO/TS 22133

First edition
2023-03

Road vehicles — Test object monitoring and control for active safety and automated/autonomous vehicle testing — Functional requirements, specifications and communication protocol

Véhicules routiers — Surveillance et contrôle des objets de test pour l'évaluation de la sécurité active et des véhicules automatisés/autonomes — Exigences fonctionnelles, caractéristiques et protocole de communication

Reference number
ISO/TS 22133:2023(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

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 22, *Road vehicles*, Subcommittee SC 33, *Vehicle dynamics and chassis components*.

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

Testing of collision avoidance systems, active safety functions and more advanced autonomous functions in vehicles require testing on proving grounds. The purpose is to expose the vehicle under test to potentially dangerous traffic situations in a safe manner. The evaluation is done during development, and in voluntary and mandatory test procedures.

To orchestrate these traffic scenarios, various impactable targets representing traffic actors are controlled. The number of controlled targets may be one or many depending on the required traffic situation scenario. Several requirements are important ranging from safety, to position and speed precision, to logging capabilities.

This document specifies requirements, functionality and a protocol allowing for multivendor target carrier systems to be controlled according to the required traffic situation scenario, to report expected information for logging purposes and other functions required.

Road vehicles — Test object monitoring and control for active safety and automated/autonomous vehicle testing — Functional requirements, specifications and communication protocol

1 Scope

This document specifies requirements, procedures and message formats for controlling and monitoring of test targets, used for testing of active safety functions and autonomous vehicles.

The document specifies functionality and messaging for monitoring and controlling of test objects by a control centre facilitating an interoperable test object environment. This document defines a communication protocol which allows for the control centre to safely execute tests using test objects from multiple vendors.

This document does not specify the internal architecture of the test object nor control centre.

This document does not specify how testing of the vehicles shall be performed.

2 Normative references

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

ISO 8601 (all parts), *Date and time — Representations for information interchange*

ISO 8855, *Road vehicles — Vehicle dynamics and road-holding ability — Vocabulary*

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
test object
entity, as defined in ISO 34501, part of a traffic scenario under monitoring and control by the *control centre* (3.5)

Note 1 to entry: Two types of test objects exist; moveable and *stationary test objects* (3.4).

Note 2 to entry: A test object can have several attributes, see Figure 1.

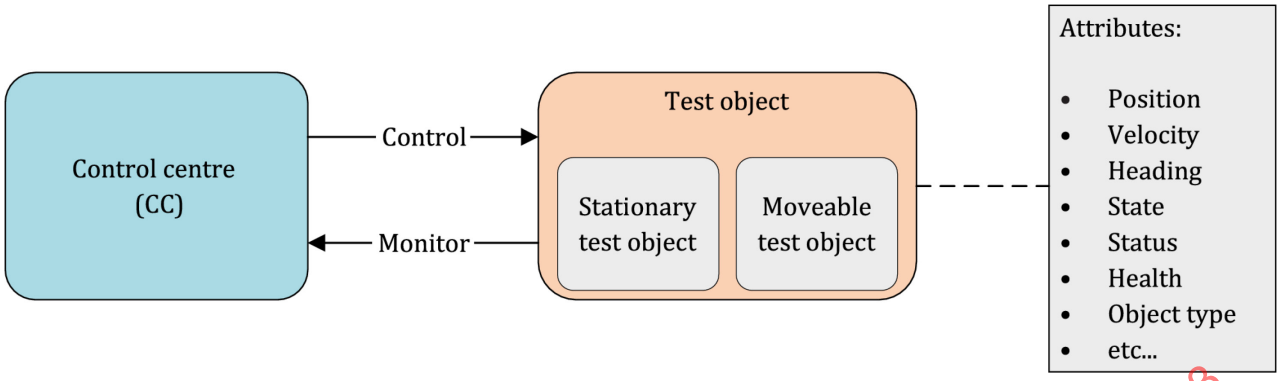


Figure 1 — Relations and inheritance view

3.2
subject vehicle
SV
vehicle to be tested with the system

Note 1 to entry: The subject vehicle, SV, is also known as vehicle under test, VUT.

Note 2 to entry: There may be one subject vehicle (standard setup) or a set of subject vehicles in the specific traffic scenario.

Note 3 to entry: The subject vehicle may be under safety-driver control and/or under *control-centre* (3.5) control.

Note 4 to entry: The subject vehicle is an instantiated *test object* (3.1) in either type of: *stationary test object* (3.4) or *moveable test object* (3.3).

3.3
moveable test object
object under control by the *control centre* (3.5) which has the capability of activation of physical movement

Note 1 to entry: The *subject vehicle* (3.2) is typically a moveable test object.

Note 2 to entry: Various levels of control are possible: open-loop control (using no feedback from the real-time activities of other *test objects* (3.1) or environment) and closed-loop control (taking into account activities of other test objects in real-time and environment). One other type of controlling/triggering is related to test object unique features like: turn indicator, headlight beam, etc.

EXAMPLE Traffic simulation vehicles, also known as traffic support vehicles (TSV), soft crash targets (SCT).

3.4
stationary test object
part of a traffic scenario, which is not moveable but may be under control by the *control centre* (3.5)

Note 1 to entry: A VUT may also be a stationary test object.

Note 2 to entry: Different stationary test objects can exist. Normally they are divided in to two groups: active and passive infrastructure elements (ISE).

EXAMPLE Active ISE: traffic lights, lighting, rain/snow/fog simulator. Passive ISE: elements of construction area, road signs, guardrails, etc.

3.5
control centre
CC
centralized or distributed service for *test-object* (3.1) control and safety monitoring including provision of communication services to the test objects

3.6
path
set of local or global positions in respect to a point of origin and in respect to the orientation to the axes of a defined coordinate system (x,y,z)

3.7
lateral path deviation
position error of the target vehicle relative to the planned *path* (3.6) measured perpendicular from the planned path direction

[SOURCE: ISO 19206-3:2021, 3.8, modified — Notes to entry and figure removed.]

3.8
trajectory
path (3.6) where a *test object* (3.1) is or is intended to move along including test-object dynamics (i.e. including timing)

Note 1 to entry: A trajectory contains a path and additionally the test-object dynamics (such as time, yaw, velocity, acceleration, etc.) for each position on the path.

Note 2 to entry: All test-object parameters related to test-object dynamic (e.g. limitations in acceleration) are described in the corresponding DIDX-file.

3.9
static trajectory
offline pre-planned *trajectory* (3.8) including *test-object* (3.1) position and test-object motion dynamics for a single *moveable test object* (3.3) as part of the traffic scenario

Note 1 to entry: The trajectory is downloaded to the test object utilizing the trajectory object message (TRAJ).

Note 2 to entry: The parameters used by the test object are described in the DIDX-file.

Note 3 to entry: The *control centre* (3.5) extracts information from the trajectory file and builds a trajectory message, TRAJ, that is understandable by the test object.

3.10
dynamic trajectory
trajectory (3.8) which at the start of the scenario execution is not known by the *test object(s)* (3.1) nor the *control centre (CC)* (3.5)

Note 1 to entry: The dynamic trajectory is created based on events and actions during the ongoing test.

Note 2 to entry: The dynamic trajectory is continuously sent by the CC or gated through the CC in smaller pieces/fragments utilizing the trajectory object message (TRAJ).

Note 3 to entry: In the case where the test object(s) or other systems (e.g. a simulation environment) are generating the dynamic trajectory this information shall be gated by sending to the CC and then from the CC to the test object.

3.11
test scenario
scenario, as defined in ISO 34501, including all *test objects* (3.1), moveable and stationary, and the definition of the planned activities over time or/and location

Note 1 to entry: The traffic scenario can be formally represented by two complementary descriptions enabling test execution, a *scenario description* (3.12), and a map of the test area.

3.12
scenario description
textual representation of a scenario

Note 1 to entry: A scenario description can be used in combination with a *scenery description* (3.13) to generate *trajectories* (3.8) (static or dynamic) by the *control centre* (3.5) or at a *test object* (3.1).

EXAMPLE: *Test scenario* (3.11) described using OpenSCENARIO or iSCAML [17].

3.13
scenery description
representation of the test area limited to the road geometry (e.g. lanes and junctions) and *stationary test objects* (3.4) (e.g. road signs and markings)

Note 1 to entry: A scenery description can be used in combination with a *scenario description* (3.12) to generate *trajectories* (3.8) (static or dynamic) by the *control centre* (3.5) or at a *test object* (3.1).

EXAMPLE: Temporary lane-setup on a proving ground, described using OpenDRIVE.

3.14
safety speed limit
maximum allowed speed when repositioning *moveable test objects* (3.3) while not participating in ongoing test

Note 1 to entry: The *test object* (3.1) is manually driven with the remote-control functionality or automatically driven when outside the test execution phase.

3.15
device ID
unique identifier for each entity in the test setup

Note 1 to entry: The device ID number is used as transmitter ID or receiver ID in every message except when tunnelling data.

3.16
message gating
mechanism to gate messages from a sender to a receiver defined by the transmitter ID and receiver ID in the message header

Note 1 to entry: *Control-centre* (3.5) gating messages from one entity to another do not alter the transmitter ID in the process of message gating.

4 Abbreviated terms

ASP	Adaptive Synchronization Point
DIDX	Device Interface Description XML
ECEF	Earth-Centred, Earth-Fixed (geographical coordinate system)
GNSS	Global Navigation Satellite System
GPS	Global Positioning System (one constellation of GNSS)
ISE	Infrastructure Elements
NTP	Network Time Protocol
NTRIP	Networked Transport of RTCM via Internet Protocol
OWD	One-Way Delay / End-to-end delay
PGRS	Proving Ground Reference System
PTP	Precision Time Protocol
RPM	Reference Point Marker (physical reference point on the proving ground)
RTK	Real Time Kinematic
SCT	Soft Crash Targets
SOW	Second Of Week (in respect to GPS time)
SV	Subject Vehicle, also known as VUT

TAI	International Atomic Time
TCP	Transmission Control Protocol – Connection oriented communication between a server and client
TSV	Traffic Simulation Vehicles
UDP	Used Datagram Protocol - Connectionless communication model
UTM	Universal Transverse Mercator (a two-dimensional Cartesian coordinate system for locations on the surface of the Earth)
VUT	Vehicle Under Test
WGS84	World Geodetic System 1984
CC	Control Centre

5 Test scenario illustration

The illustration of a test scenario in Figure 2 is used as an example to explain a possible test scenario with various test objects. The test objects present in the example scenario are categorised in types and sub types in Table 1. The example does not include all possible test objects and scenarios referred to in this document.

To be able to achieve the following scenario, several different mechanisms can be used to control and monitor all different test objects, which will be described later in this document. Some or all test objects can be controlled by a CC but may also be controlled by a safety driver or by an integrated controller. All test objects are always monitored by the CC.

A test scenario is normally described in a story board or segment description.

Table 1 — Test object type categorization example

Test object type	Test object sub type	Illustration
Stationary test objects	Active infrastructure elements (A-ISE)	
	Passive infrastructure elements (P-ISE)	
Moveable test objects	Object controlled/monitored by the CC (TSV/SCT)	
	Subject vehicle (SV/VUT)	

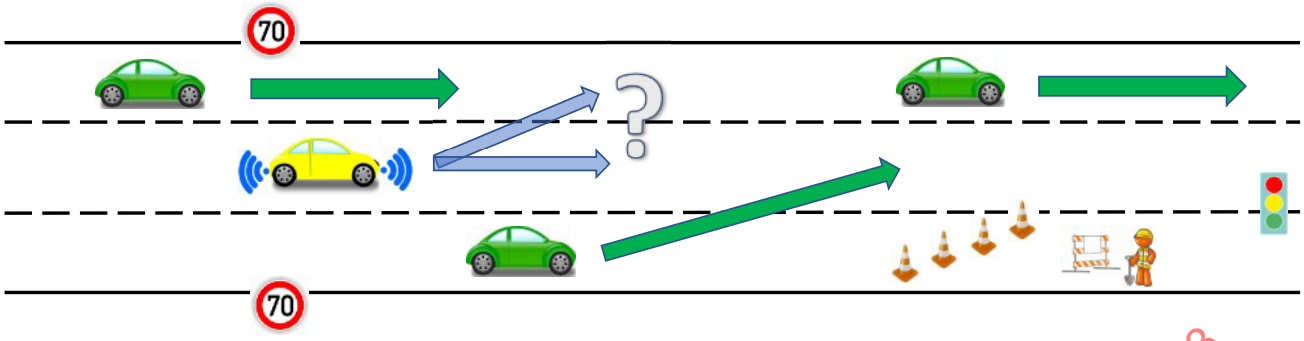


Figure 2 — Example of test scenario

6 General requirements and recommendations

6.1 Function overview

Several techniques and methods used for function and safety testing of autonomous moving test objects are covered by this document and listed below:

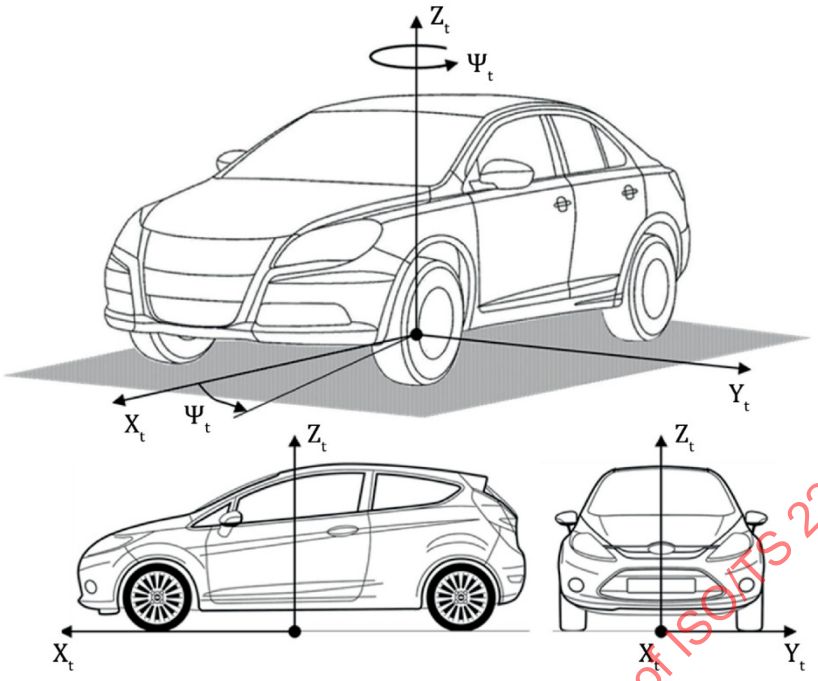
a) test scenario execution,
b) static and dynamic trajectory following,
c) trigger and actions,
d) adaptive synchronization points (master/slave),
e) remote control driving,
f) object-to-object communication,
g) safety features:
   — emergency stop,
   — arm/disarm,
   — time synchronization,
   — geofencing,
   — centralized control.

6.2 Test object coordinate system

6.2.1 Vehicle

Vehicle coordinate system orientation shall follow ISO 8855. See Figure 3.

If not otherwise stated, the geometrical centre which is the centre of the bounding box of the test object projected on the ground is used as the origin of the coordinate system.



NOTE ψ_t is the rotation about the z_t axis.

Figure 3 — Vehicle reference coordinate system (Source: ISO 19206-3:2021, Figure 2)

6.2.2 Moveable test objects other than vehicle

6.2.2.1 Bicyclist

Origin is the centre of pedal crankshaft, projected on the ground, see Figure 4 (see also ISO 19206-4:2020, Figure A.1).

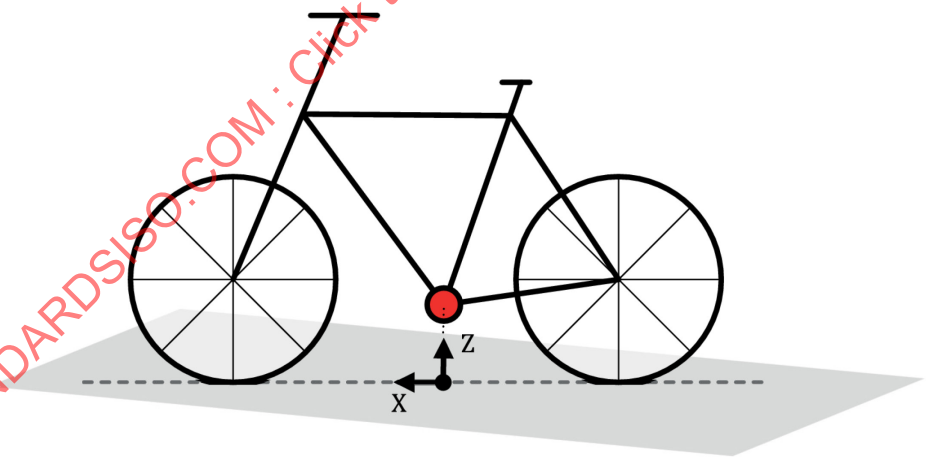


Figure 4 — Bicyclist reference coordinate system

6.2.2.2 Pedestrian

Origin is the centre between the hips, projected on the ground, see Figure 5 (see also ISO 19206-2:2018, Figure A.1).

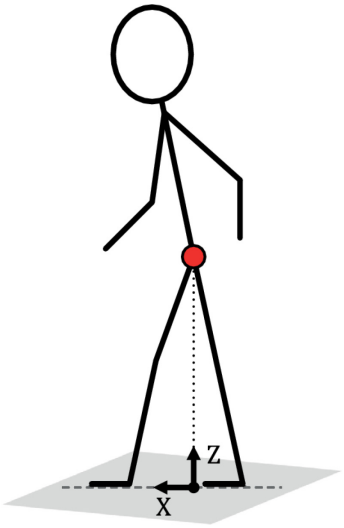


Figure 5 — Pedestrian reference coordinate system

6.2.2.3 Airborne test objects

For test objects not located directly on the ground of the test area such as drones or other aircrafts, the origin shall not be projected on the ground but on the correct altitude/height on the z-axis, see Figure 6.

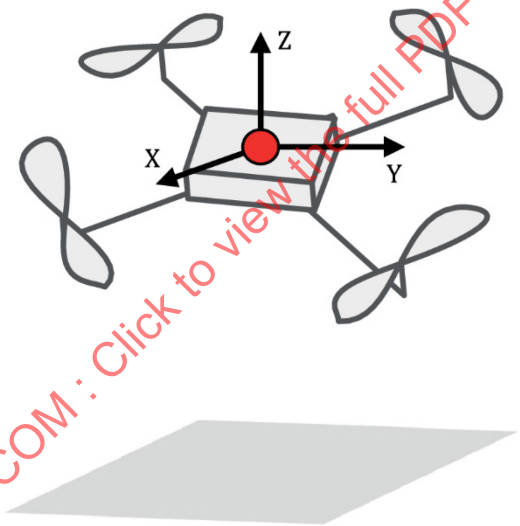


Figure 6 — Aerial test object reference coordinate system

6.2.2.4 Other

Origin on the geometrical centre of the test object, see Figure 7.

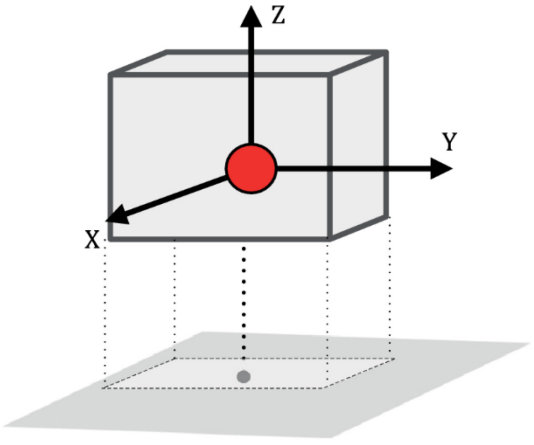


Figure 7 — Other geometrical test object reference coordinate system

6.3 Test scenario coordinate system

6.3.1 Background info: tectonic/continental plate drift

Test object trajectory planning shall take place within a PGRS, proving ground reference system, which is not affected by any creeping or sudden tectonic movements, but referenced to a local reference point.

EXAMPLE In Europe, the absolute movement of the tectonic plate as whole is about 1 cm / year, in the Pacific area values of 10 cm / year are known [15] and after earthquakes sudden significant shifts along faults within the tectonic plate, and/or its margins are observed [11] [12].

WGS84 (as an output of uncorrected GNSS) shall not be used. If used, the infrastructure elements would "move" against the planned trajectory due to the continental plate drift, resulting in significant risk of collision between test objects.

6.3.2 Coordinate system – Tests on proving ground

For tests executed on proving ground areas, the exchange of coordinates between the CC and test objects shall use a local cartesian ENU (east-north-up, where the north direction is defined as true geographical north if the rotation of the PGRS is equal to 0) coordinate system. Test objects shall report their position in the MONR-message with local cartesian coordinates (PGRS) in millimetres (x,y,z) in relation to the local reference point marker (RPM). Only one RPM shall be defined per test.

The coordinates of the RPM can be defined as the origin (values 0 / 0 / 0) which gives the PGRS system defined as a cartesian local coordinate system with its x-axis to east, y-axis to true north and z-axis upwards (ENU). A top-down view of the coordinate system can be found in Figure 8 where the yaw angle Ψ (psi) is measured counter clockwise in degrees, starting with 0° pointing east.

The heading of a test object is measured clockwise in degrees, starting with 0° pointing to local north at the origin (heading = 90° - Ψ).

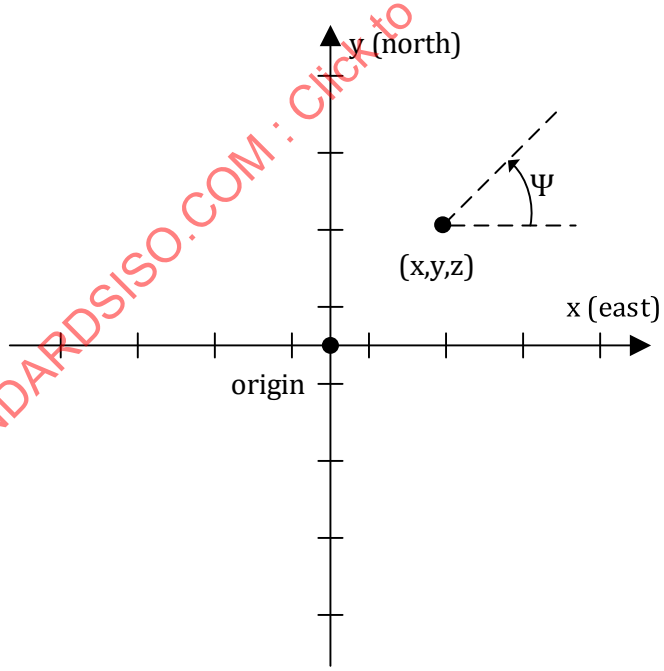


Figure 8 — RPM in PGRS and yaw explanation

6.3.3 Coordinate system – Test on other test areas

For larger areas, or areas outside a limited test area, a global coordinate system using UTM/ECEF shall be used.

Test objects shall report their position in the MONR2-message with global, absolute, coordinates.

Depending on where (on which tectonic plate) the test is executed, a different plate-drift-free coordinate system shall be used according to Table 2.

Table 2 — Geodetic systems on specific tectonic plates

Geographical location of test	Geodetic system
Eurasian plate	ETRS89
North American plate	NAD83
South American plate	ITRF2000
African plate	ITRF2000
Australian plate	ITRF2000
Indian plate	ITRF2000
Arabian plate	ITRF2000
Other plates/fallback /local GNSS correction data	WGS84, locally corrected (see below)

The CC shall transmit information to the test objects about the base coordinate system that shall be used for the test scenario. If this is not applicable or if the CC did not send the information, then the fallback solution is to use either:

— WGS84 (for tests as described in 6.3.3), or
— A local coordinate system (for tests as described in 6.3.2).

It is recommended to use a local GNSS reference station to provide RTK correction data, which is located close to the proving ground's local RPM, using as its initial position the local coordinates being corrected by tectonic plate drift (i.e. taken from the same map, which also provides the coordinates of all other infrastructure elements during the traffic scenario).

This local GNSS reference station shall transmit its RTK correction data to each participating moveable test object. This guarantees that all moveable test objects are located within the same, tectonic plate drift invariant coordinate system.

6.4 Time requirements

6.4.1 General time requirements

All test objects and the CC shall have the same understanding and perception of time. Therefore, it is important that time representation, time resolution, time accuracy and time synchronisation is commonly shared.

6.4.2 Time representation

GPS time shall be used as time base. GPS time is monotonous and does not get affected by leap seconds.

NOTE UTC time is affected by leap seconds, hence is not monotonous and not useable for surveying and control applications with high timing precision requirements.

6.4.3 Absolute time

To represent absolute time, the time is defined as the number of elapsed milliseconds since GPS time began on midnight (00:00:00 UTC) between January 5th and January 6th, 1980.

NOTE The difference between GPS-time and TAI-time is always 19 s (TAI is always 19 s ahead of GPS).

TAI is currently 37 s ahead of UTC (as to date 2021-12-31).

GPS is currently 18 s ahead of UTC (as per date 2021-12-31).

6.4.4 Relative time

The GPS second of week (SOW) shall be used when representing relative time (e.g. during test execution). The GPS week starts at midnight (00h00m00s GPS time) between Saturday and Sunday and consists of 604800 s [16].

All transmitted times shall be in relation to the start of the given GPS week.

6.4.5 Time resolution

When using 32 bits for representing the GPS seconds of week in milliseconds, the time resolution for transmission is 0,25 ms. See Formula (1).

$$604\,800 \times 1\,000 \times 4 = 2\,419\,200\,000 \text{ quarters of ms per week} \quad (1)$$

Representation:

```
GPSSecondOfWeek::=INTEGER
{startOfWeek(0),oneMillisecAfterStartOfWeek(4),unavailable(4294967295)}
(0… 2419199999)
```

NOTE 1 For each millisecond since start of the week, the `GPSSecondOfWeek`-value increases with 4 due to the 0,25 ms resolution.

NOTE 2 The `unavailable` value 4294967295 (dec) corresponds to 0xFFFFFFFF (hex).

EXAMPLE The GPS week roll-over behaviour for the `GPSSecondOfWeek` looks like this at midnight between Saturday and Sunday:

Saturday@23:59:59,99900	`GPSSecondOfWeek=2419199996`
Saturday @23:59:59,99925	`GPSSecondOfWeek=2419199997`
Saturday @23:59:59,99950	`GPSSecondOfWeek=2419199998`
Saturday @23:59:59,99975	`GPSSecondOfWeek=2419199999`
Sunday@00:00:00,00000	`GPSSecondOfWeek=0`
Sunday @00:00:00,00025	`GPSSecondOfWeek=1`
Sunday @00:00:00,00050	`GPSSecondOfWeek=2`
Sunday @00:00:00,00075	`GPSSecondOfWeek=3`
Sunday @00:00:00,00100	`GPSSecondOfWeek=4`
…	
Sunday @00:00:01	`GPSSecondOfWeek=4000`
…	
Sunday @00:01:00	`GPSSecondOfWeek=240000`

6.4.6 Time accuracy (and precision)

Control centre and test object internal clocks shall not deviate from the GPS second of week more than:

- ±125 µs for moveable test objects;
- ±10 ms for infrastructure elements (or slow-moving test objects).

This requires that each test object uses the GPS time for its own time synchronisation or uses a network time server (PTP / NTP) for time synchronisation.

6.4.7 Time synchronization

The time reference for test objects as well as for the CC shall be GPS time. This time is available to all participants operating their own GPS receiver and having sufficient GPS observations available. The GPS observations shall preferably be direct communication with satellites, but for GPS-denied areas a GPS-repeater may be used as a substitute.

For moving test objects in GPS-denied areas (such as indoor) and if the duration of loss of GPS time is outside the stability duration of the clock on the test object shall the time be retrieved from the PTP-server hosted by the CC.

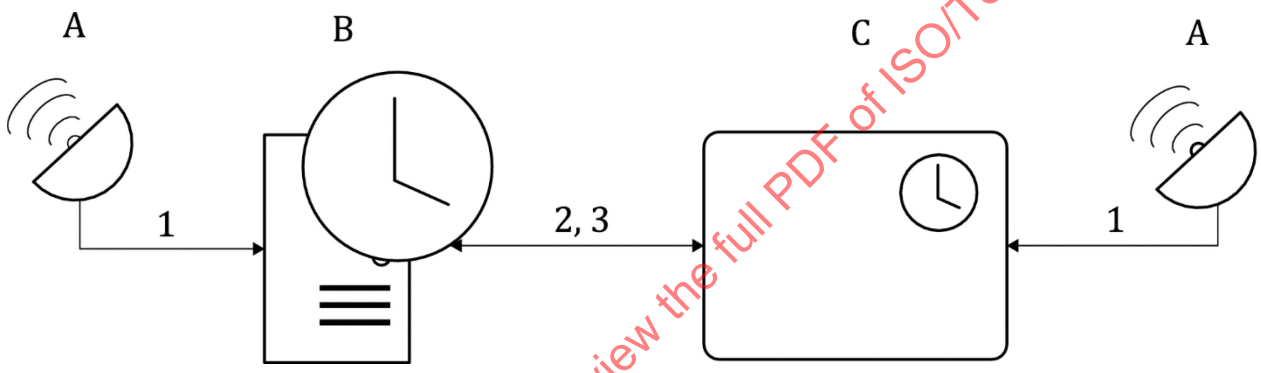
For static or slow-moving test objects in GPS-denied areas (such as indoor), and if the duration of loss of GPS time is outside of the stability duration of the clock on the test object, the time shall be retrieved from the NTP or PTP-server hosted by the CC.

If different time sources are available to the CC and test objects they shall retrieve their time according to the following priority, further visualised in Figure 9:

a) GNSS source (GPS time),
b) PTP,
c) NTP.

If no time source is currently available, `unavailable` shall be sent in messages transmitting time stamped information.

NOTE PTP is using UDP port 319 and 320. NTP is using UDP port 123.



Key

1	GNSS source (GPS time)
2	PTP
3	NTP

Figure 9 — Schematic view of time source priority and NTP/PTP service allocation

6.4.8 Date synchronization

During configuration/setup of a test, the CC shall transmit the current date, using the test object setting (OSEM)-message to the test objects. The test object shall use this time as reference for the test to be executed. The following date and time information shall be included in the message:

[Date, GPSWeek, GPSSecondOfWeek, LeapSeconds]

The date format shall be in accordance with the ISO 8601 series (`YYYYMMDD`).

The CC shall transmit the week-roll-over message to the test objects if a new GPS-week is entered during execution of a test. This message is called GPS second of week roll over message – SOWM.

6.4.9 Network delay

A one-way-delay (OWD) [13] shall be determined after the network has been set up. The determination inside the network is easy because the true time reference is distributed via PTP/NTP (for test objects without GNSS receiver) or the OSEM. Each data packet, being transmitted between any test object and the control centre, contains in its header a timestamp of the time when the data content has been generated.

It is important to recognize, that the OWD is not important for the application, if the total time between data generation on the sender's site and data receiving at the other site is lower than a defined threshold. This threshold is defined, for example, by the heartbeat timeout.

The heartbeat or monitor packets can therefore be used directly for an online monitoring of the over-all latencies of the network.

Current measured network delay can be retrieved from the test object (or from CC) by reading the network-delay-parameter (if available). Other network parameters may also be retrieved. All available parameters are specified in the device interface description XML-file (DIDX).

6.5 Communication requirements and permissions

The following requirements and recommendations apply.

a) Network IPs shall be used for communication among the CC and test objects. The CC and all test objects being part of the same traffic scenario shall operate on the same logical IP network and being addressable (able to communicate with each other).

b) Logical sub-networks shall be supported to allow the execution of several independent traffic scenario tests on the same proving ground without conflicts.

c) The communication system shall support test object status collection. Test object status collection provides the CC with current information about all available test objects, such as battery level, current faults, etc.

d) The subject vehicle shall support sending monitor information to the CC.

e) A traffic scenario may include one or more subject vehicles in a single scenario.

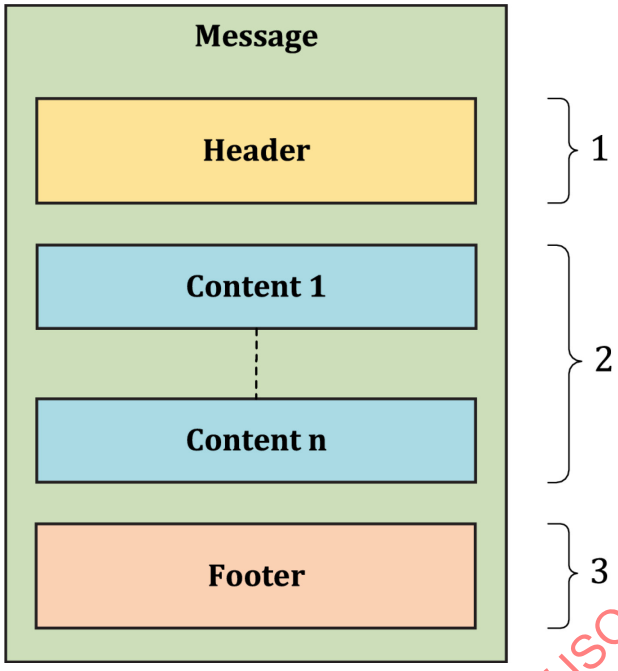
f) Each test object shall be able to report its capabilities via request from the CC.

g) Multicasting of information may be applied if the underlaying network and all entities in the test setup support this.

h) Test objects shall be capable of reading the heartbeat message (see 12.3.5) and send monitor message (see 12.3.6).

Structure and package of data are done according to Figure 10. A message is the highest element consisting of header, content data and footer. This is called flexible message structure.

A message can contain one or more content(s). Each message can be built up to contain dynamic content, meaning that addition to a defined message is possible. Pre-defined content is specified further down in this document, and addition to this should be used with care, since it may affect the latency of communication timing expected.



Key

1 message header metadata: `ID, counter, version, length, etc...`

2 one or several message content(s) containing: `valueID, length, data`

3 message footer: `CRC checksum`

Figure 10 — Message structure

7 Safety and risk assessment requirements and recommendations

7.1 General

The control centre shall support safety monitoring of moveable test objects. This includes support for emergency stop of all moveable test objects.

The control centre as well as all moveable test objects shall have support to enforce one or more virtual fences.

For every fence an appropriate function shall be supported, low speed fence, stop fence, shutdown fence, geofence, etc. See 7.2 for details and terminology about local and global geofence.

a) The system should support a safe low-speed mode for remote transport.

b) The system should support a safe low-speed mode for safe return/re-position.

c) The system shall support an arming function (ready to start test – active).

d) The system shall support an emergency stop.

e) The system may support `soft stop` – controlled braking – also called `normal stop`.

If a test object receives an emergency stop signal or the condition for initiating an emergency stop is fulfilled, the test object shall stop as quickly as possible without considering extensive usage of tires, etc. This type of stopping should only be used for emergency stop, compared to a soft stop where test object can use a longer period for stopping, if preferred.

7.2 Local test object fence and global geofence

7.2.1 Common requirements and recommendations

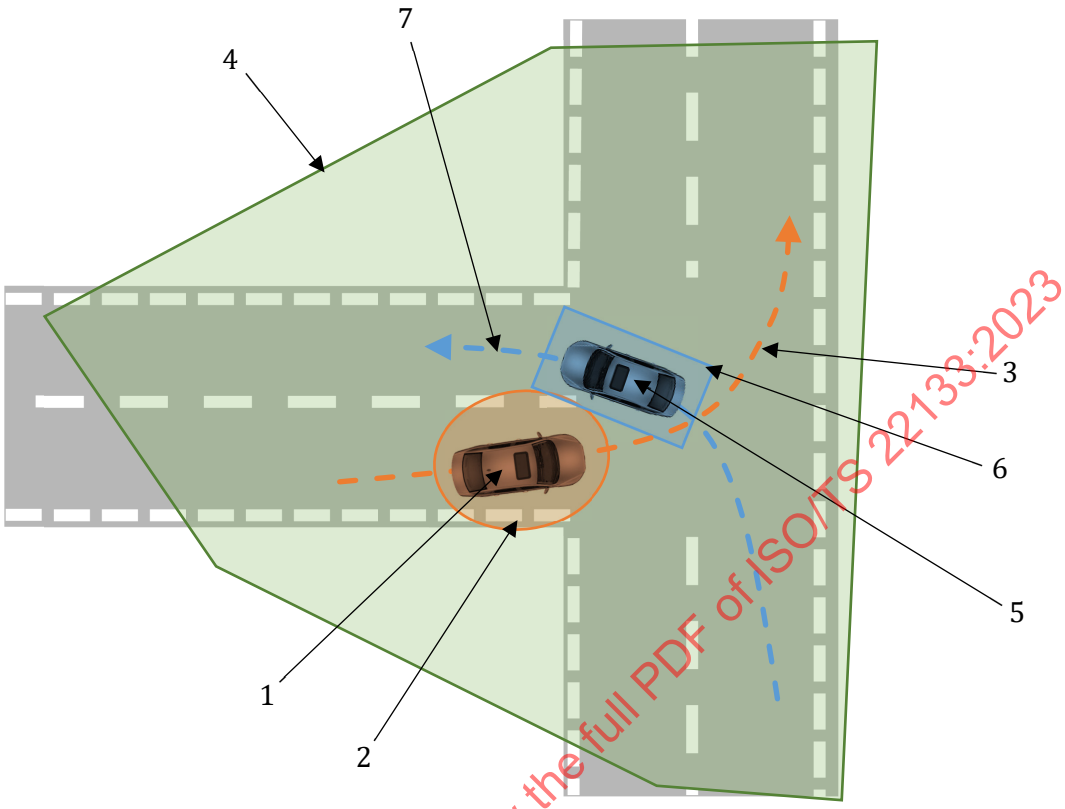
Virtual fences are used as criteria of whether a moveable test object meets certain safety requirements. The `global geofence` defines the area on the proving ground (or on the public road or other test area) where the moveable test objects are allowed, or not allowed, to move. The `local geofence` is defined individually for each moveable test object and defines criteria (such as position, speed, type of test) which shall be fulfilled by each moveable test object in respect to its trajectory.

7.2.2 Global geofence

The virtual boundary around the allowed test area is called global geofence. The test object shall evaluate the position of itself in real-time against the geofence. If the test object exceeds (enters or exits) the boundary, a related measure shall be executed. The control centre may additionally evaluate the position of each test object against the geofence.

7.2.3 Local test object fence

The CC shall inform moveable test objects how much deviation from their trajectories is allowed. This is called `trajectory deviation fence` or `local test object fence`. This information is sent to the test object before the test is executed, but in principle the local fence can also depend on specific situations and scenarios and the fence-tolerances may be updated during a running test scenario. Figure 11 illustrates a combination of both global and local virtual fences during a test scenario.



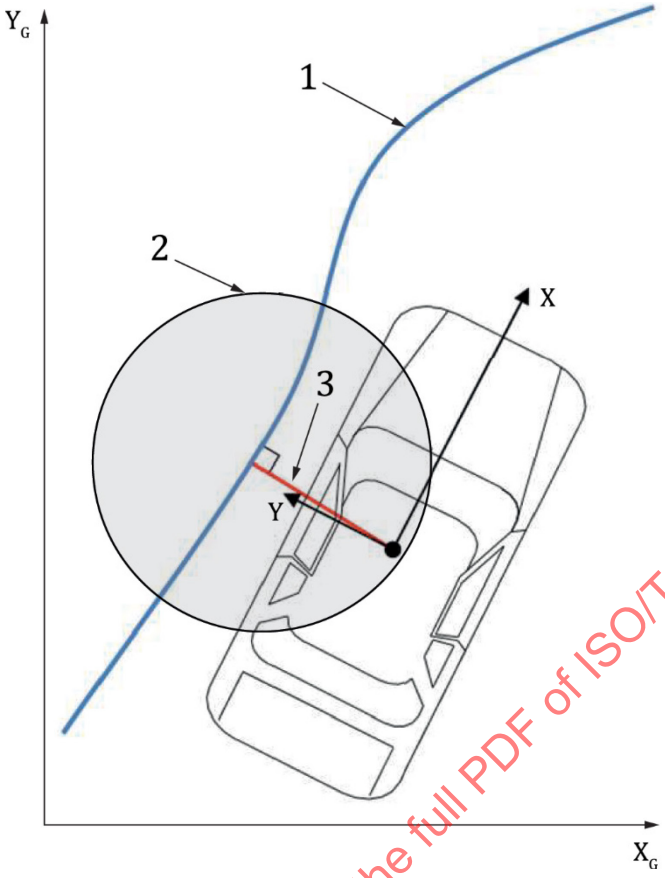
Key

1 test object 1
2 test object 1 – local virtual fence
3 test object 1 – trajectory
4 test object 1 – global virtual fence
5 test object 2
6 test object 1 – local virtual fence
7 test object 1 – trajectory

Figure 11 — Test scenario with geofence and local test object fences

The test object trajectory deviation shall not exceed the maximum value(s) as sent in the configuration message for local fences (e.g. `MaxWayDeviation`, `MaxLateralDeviation`, `MaxYawDeviation`). If such deviation is identified by the test object, it shall abort the test, it shall send an `abortRequest` to the control centre and in case of receiving no feedback in time, it shall perform a predefined emergency action.

Figure 12 illustrates the lateral path deviation in the context of geofencing.



Key

1 planned path

2 local virtual fence

3 lateral path deviation

Figure 12 — Lateral path deviation (Source ISO 19206-3:2021, Figure 1)

NOTE The position error is the deviation between the desired position and the current position for each specific time stamp.

If the current moveable test object positioning accuracy (measured by the test object itself by its estimated position standard deviation) exceeds the threshold `MinPosAccuracy` (i.e. its estimated local position error is larger than the required accuracy), then this shall be reported to the CC.

8 Communication security requirements

To avoid unauthorised access and control of the CC or test objects by any untrusted source, encryption over the communication layer between all test objects and the CC may be implemented, but a non-encrypted mode shall be available for development and debugging purposes.

9 Architecture and interfaces

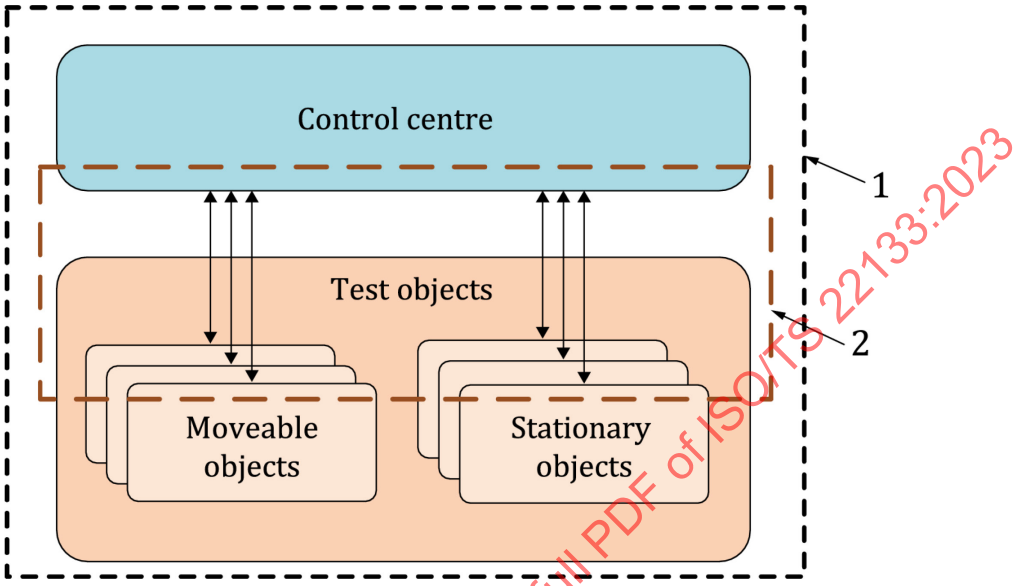
9.1 General

The test system consists of the control centre and test objects. Test objects can be of type `moveable` or `stationary`.

Figure 13 shows basic communication setup between the CC and test objects and the boundary of this communication protocol document. Direct communication between test objects may also be implemented.

The supporting system such as communication networks and NTRIP caster and their respectively integration towards the test object or CC is not described in this document.

A fast, low latency data communication between the server and all test objects shall be assured by the architecture. Also, receiving of broadcasted GNSS correction data (via RTCM-SC104 format) from a NTRIP Caster shall be supported by all moveable test objects. NTRIP server is preferably part of the CC infrastructure.



Key

1 system boundary

2 communication protocol boundary

Figure 13 — Overview of architecture

9.2 Control centre and test object states

9.2.1 General

To describe the overall operational states and transitions between them, state diagrams are provided for test objects and the CC. Guard conditions are listed in the tables below to describe the condition that needs to be fulfilled to execute a dedicated transition.

9.2.2 Test objects state diagram

The test object state diagram is illustrated in Figure 14, and further detailed in Tables 3 and 4.

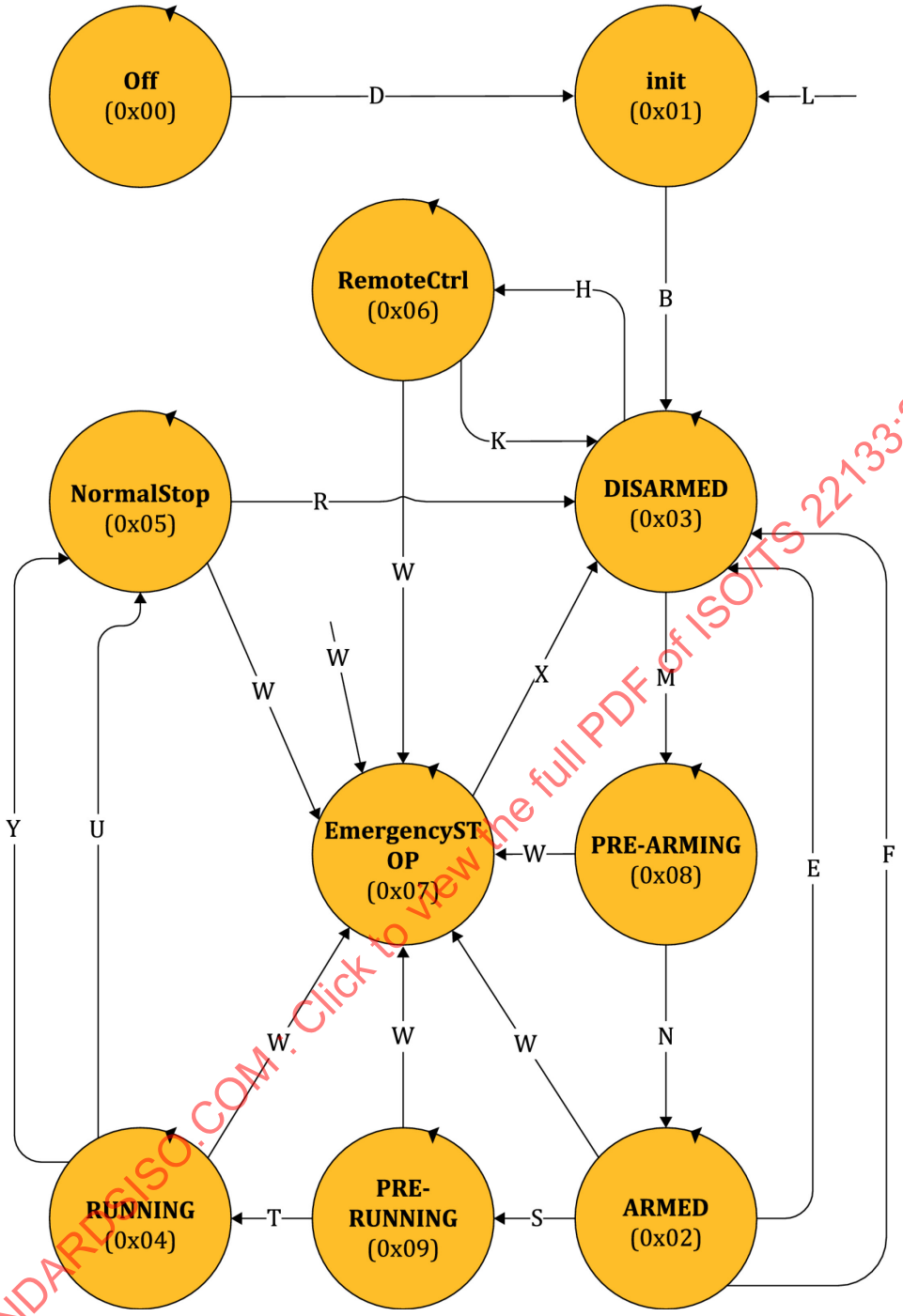


Figure 14 — Test object states

Table 3 — Test object state description

State ID	State Name	State Description
0x00	OFF	Test object turned off
0x01	INIT	Test object in initializing phase, e.g. setting up supporting system
0x02	ARMED	Test object ready to execute the test
0x03	DISARMED	Test object connected to the CC
0x04	RUNNING	Test scenario is executing

State ID	State Name	State Description
0x05	NORMAL_STOP	Test scenario has been executed/finished
0x06	REMOTECONTROL	Test object is manually driven by the CC or external remote control
0x07	EMERGENCY_STOP	Test object is currently emergency stopping
0x08	PRE-ARMING	Test object is currently preparing for arm
0x09	PRE-RUNNING	Test object is waiting for running or preparing for running

9.2.3 Test object state change conditions

The following list corresponds to the state transition conditions in Figure 14.

Table 4 — Test object state change description

State transition ID	Precondition	Action/condition
D	Test object powered off	System boot / power on
L	Any state	CC connection lost, i.e. If HEAB timeout expires
B	Test object Initialized	IP connection to the CC is established
H	Test object disarmed	`OSTM.StateChangeRequest(remoteControl)` received
K	Remote control active	`OSTM.StateChangeRequest(disarm)` received
M	Test object disarmed	`OSTM.StateChangeRequest(arm)` received
F	Test object armed	`OSTM.StateChangeRequest(disarm)` received
E	Test object armed	Test object internal error detected
S	Test object armed	Start message received
U	Test scenario is running	Normal stop message received
W	Test scenario is running	Emergency stop message received or test object detects by itself to stop (from any state except OFF or INIT)
Y	Test scenario is running	Test scenario executed
R	Test is executed	Test object stopped
X	Test object emergency stopped	Emergency stop released/reset

9.2.4 Control centre state diagram

The control centre state diagram is illustrated in Figure 15, and further detailed in Tables 5 and 6.

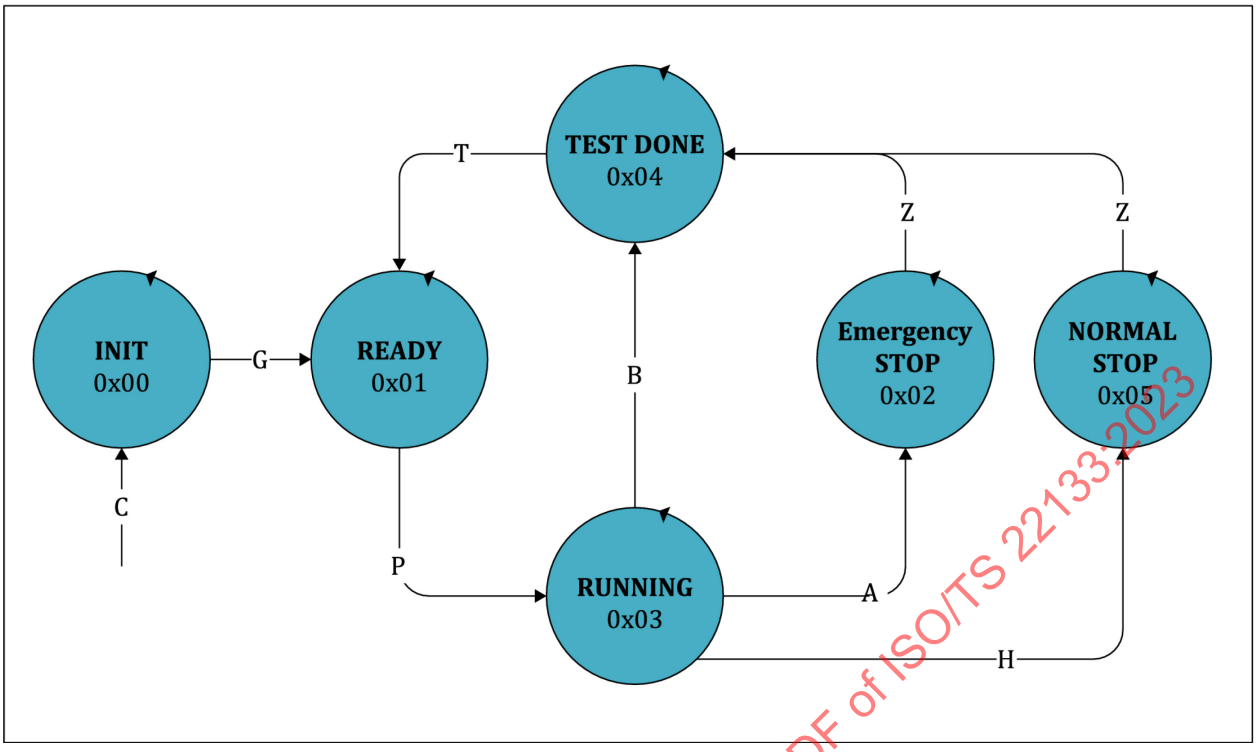


Figure 15 — Control centre states

Table 5 — Control centre state description

State ID	State name	State description
0x00	INIT	CC is initializing.
0x01	READY	CC ready/idle. This state is where the CC sends ARM and DISARM to test objects.
0x02	ABORT	Emergency stop is sent to all test objects.
0x03	RUNNING	Test scenario is executing.
0x04	TEST_DONE	Test scenario has been executed/finished.
0x05	NORMAL_STOP	Normal stop is sent to test objects.

9.2.5 Control centre state change triggers

The following list corresponds to the state change conditions in Figure 15.

Table 6 — Control centre state change description

State transition ID	Precondition	Action/condition
C	CC off	CC booting/starting
G	CC started	Initialized
P	CC ready	Start test
A	Test running	Abort test
H	Test running	Stop test-request received
T	Test is executed.	Test successfully executed. Test objects stopped.

State transition ID	Precondition	Action/condition
B	Test running	Test successfully executed
Z	Test stopped/aborted	Test object stopped

9.3 Communication setup

9.3.1 General

TCP communication is used to exchange commands between the CC and test objects. The TCP channel is called `control channel`.

UDP communication is used to exchange information (data) about status (position, health, etc.) periodically between the CC and test objects. The UDP channels is called `process channel`.

UDP communication is also used to find or discover ISO/TS 22133 compatible test objects and the CC in the network. This UDP channel is called the `discovery channel`.

The communication setup is summarised in Table 7.

Table 7 — Port numbers

Communication channel	Port number	TCP or UDP	Description
Process channel port	53240	UDP	Periodically process data during test
Control channel port	53241	TCP	Exchange of control commands
Discovery channel port	53242	UDP	Test object and control discovery

9.3.2 Test object discovery

The test object discovery is used to find ISO/TS 22133 test objects in the network. The CC shall send a discovery request (DREQ) as a broadcast and a test object(s) shall answer with a discovery response (DRES). Additionally, the test object shall send the discovery response after the bootup phase as a broadcast message.

The test object discovery shall be on port 53242 *(*discovery channel*)*.

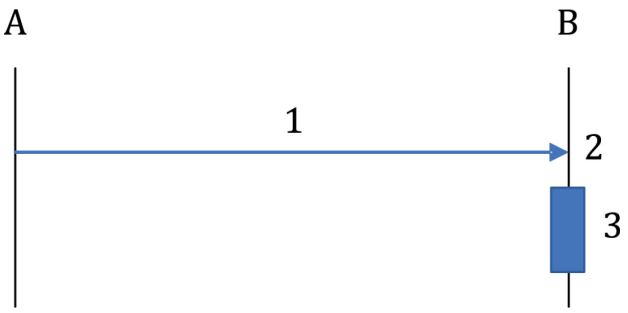
NOTE For the cases where broadcast messages are not supported by the communication layer the CC can send addressed (unicast or multicast) DREQ messages to the test objects. This can be done by manually specifying the IP addresses of the test objects for the CC.

9.3.3 TCP communication setup (control channel)

The test object shall act as TCP server. The CC shall be the TCP client. The IP number of the test object shall be preconfigured and known by the CC before initiating the communication to test object.

1) The test object (server) shall listen for incoming connections.
2) The CC (client) shall send a connection request to the test object (server) to the default control channel port 53241. A different port can be used if required.
3) The test object (server) shall accept the connection and save the IP- and port number of the sender (client).

The above procedure is visualized in Figure 16 and shall only be performed once per connection phase.



Key

1	TCP request
2	default TCP port is 53241
3	test object shall save IP and port number of the control centre from the TCP request
A	control centre (TCP client)
B	test object (TCP server)

Figure 16 — TCP communication setup

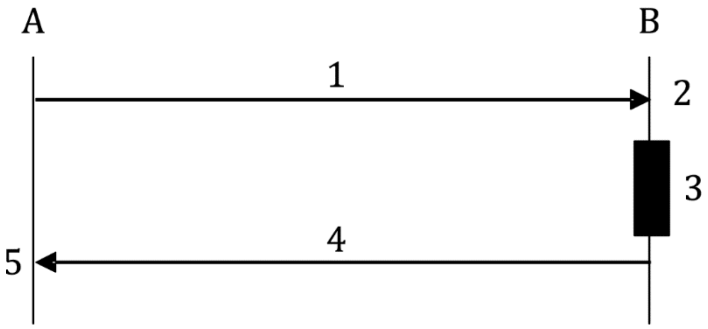
9.3.4 UDP communication setup (process channel)

The UDP communication between the CC and a test object is initialized through the following procedure:

1) test object shall be idle, listening to incoming messages on port 53240;
2) CC shall send a HEAB (heartbeat) message to the test object on port 53240;
3) every test object shall save the IP- and port number of the sender;
4) test object starts sending MONR (Monitor) message to the CC.

This procedure (saving the IP address of CC) shall only be performed on the first received HEAB-message (since IP number is not expected to change during ongoing test).

The setup is visualized in Figures 17 and 18.



Key

1	HEAB (UDP)
2	MONR (UDP)
3	default UDP port is 53240
4	test object saves IP and port number of the control center in the HEAB message
5	return port number received in HEAB message
A	control centre
B	test object

Figure 17 — UDP communication setup

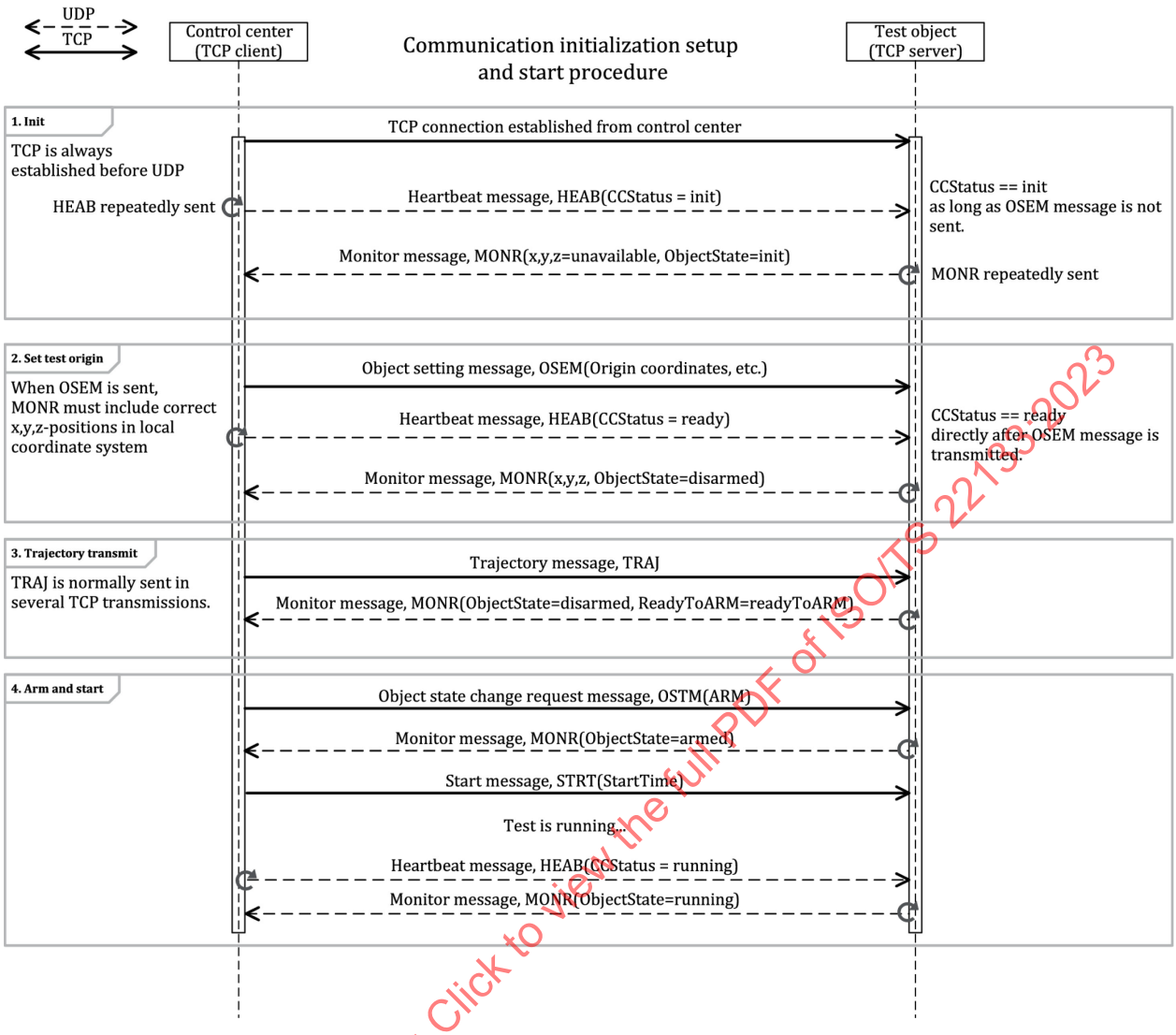


Figure 18 — Typical communication initialization setup

9.3.5 File transfer protocol (FTP)

If a file transfer is required, FTP may be used.

For a file transfer, for example, to load a Log-file from the test object, the test object shall support FTP (file transfer protocol). For better security a test object may support FTPS (FTP over SSL).

It is recommended to not use FTP during ongoing test (RUNNING), since this would increase the network load.

NOTE The FTP is not intended to be used to transfer trajectory or other real time relevant information.

9.4 Control functionalities

9.4.1 Automated and non-automated drive

Control functionalities are separated in automated and non-automated drive. A drive is called automated once there are no human interactions required to change the dynamic state of a test object. A drive is called non-automated once there is human interaction required to change the dynamic state of a test object (e.g. during remote control manoeuvring). The automated drive and non-automated drive may be used by different scenarios according to Clause 10.

Every non-stationary automatic driving test object shall be able to control its actuators without human remote-control impact to follow a trajectory in time and space.

Non-automated driving test object is meant to move by safety driver or by remote control via the remote-control manoeuvring message (RCMM), e.g. to/from the storage container.

9.4.2 Remote-control manoeuvring

9.4.2.1 Remote-control manoeuvring - General information

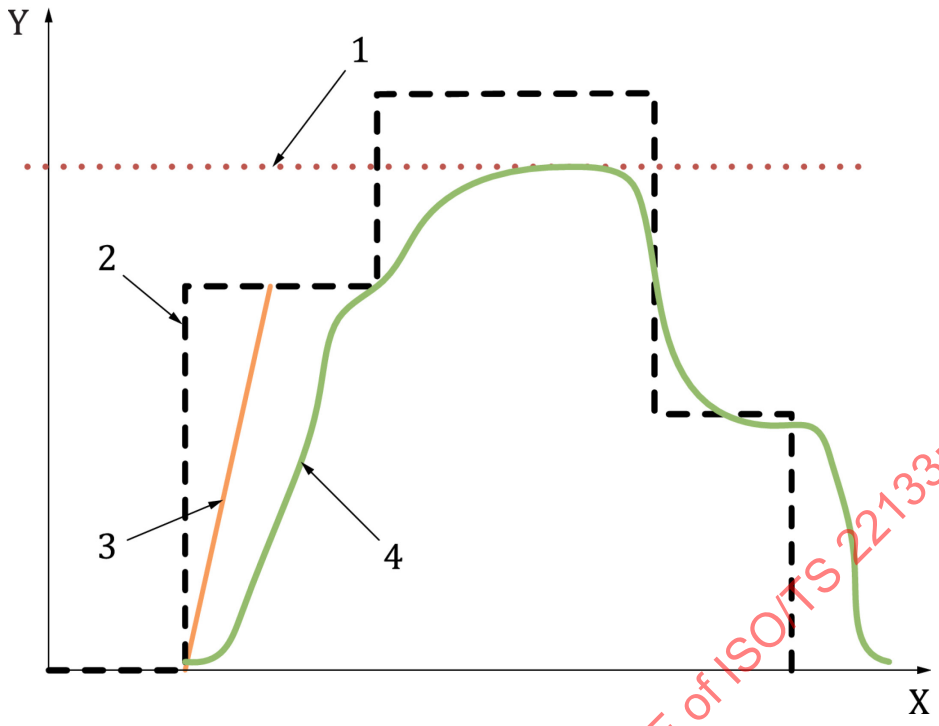
The remote-control (or sometimes referred as low-level control) functionality is used to manually control the test object for transportation or relocation in non-test situations. The control message for this is continuously transmitted during operation and not via a planned trajectory.

The CC (or optionally the remote control) will repeatedly send control instructions to the test object online, which will change the current movement for throttle and steering.

The responsibility for keeping the speed limit below the safety speed limit (used during manual driving) lies on the test object and person controlling and configuring the test object.

The maximum allowed speed/safety speed limit, acceleration, etc. for each test object shall be specified in the DIDX-file, see Clause 11 and Annex D.

The test object shall under no circumstances exceed the stipulated `safety speed` limit. Even if the CC is requesting a higher (faster) value than allowed, the test object shall keep its speed below the safety speed limit. The test object is also responsible for complying with the permissible acceleration. Figure 19 illustrates an example of how the requested `SpeedValue` (dashed black line), the `safety speed limit` (dotted red line) and the `max acceleration` (solid orange line) is affecting the `actual speed` (solid green line).



Key

X	time (t)
Y	SpeedValue [%] or [m/s]
1	safety speed limit / maximum allowed speed
2	requested SpeedValue
3	maximum allowed acceleration
4	maximum allowed acceleration

Figure 19 — Permissible acceleration and maximum speed in relation to requested SpeedValue

The test object shall be in the state `remoteControlled` when performing remote manoeuvring.

Possible parameters to control shall include the following two possibilities:

— speed and steering (absolute values);

— throttle, brake, steering and longitudinal direction (relative values).

The message for controlling this is the RCMM (remote-control manoeuvring message). See 12.3.11 for details on how the message is structured.

The RCMM is a periodic message during remote control sessions. It shall be sent from the CC with a default period of 10 Hz. If message timeout occurs, or messages are lost, then the test object shall immediately perform a soft stop.

Due to operational safety, one person (operator) should only control one test object at the same time.

9.4.2.2 Remote-control manoeuvring - Absolute values

It shall be supported by the test objects to use absolute values (in SI units) for controlling the speed (m/s) and steering angle (°) to be independent on arbitrary chosen relative limits.

9.4.2.3 Remote-control manoeuvring - Relative values

It shall be supported by the test object to use relative requests (percent based). The range for throttle goes from 0 % (no request) to 100 % (maximum propulsion power). The requested direction movement (forward/reverse) is determined by longitudinal direction request.

The range for the requested brake force applied by friction brake actuators goes from 0 % (no brake force) to 100 % (maximum brake).

The range for steering goes from -100 % (right) to +100 % (left).

10 Trajectory and scenario-based testing

10.1 Introduction to trajectory and scenario-based testing

Trajectories are a collection of waypoints to define a path of a single test object during test, added by a time stamp for each waypoint (location and time) and test-object dynamics (e.g. yaw, velocity, acceleration).

All test objects shall support the use of static and may also support the use of dynamic trajectories.

Test scenarios constitute of a collection of trajectories and other parameters necessary for the test objects participating in the scenario.

Test scenarios may also be defined by a scenario-description file, i.e. a textual representation in file format, using a scenario definition language. The scenario description covers the details of the triggers the test object is sensitive to, actions initiated by these triggers and more information relevant for this specific scenario.

When orchestrating a test scenario, the CC shall define and communicate what test mode to be used by the test object(s) using OSEM (see 12.3.2). The subsequent test scenario information shall match the selected test mode. The commonalities and differences between different test modes and corresponding information are given through 10.2 to 10.4.

10.2 Static trajectories

A static (or pre-planned) trajectory is information intended to be used by a moveable test object to follow a predefined trajectory in space and time, and it holds all necessary data for a moveable test object to control the dynamic state of a test object movement. It is allowed to have several trajectories downloaded to the same test object.

The static trajectory is sent from the CC to the test object prior to the test utilizing the trajectory message (TRAJ).

If a test object is delayed in motion due to a trigger/action event, this affects the timing but not the path.

10.3 Dynamic trajectories

A dynamic trajectory (online planned) is information intended to be used by a moveable object to follow a specific trajectory in space and time, and it holds all necessary data for a moveable test object to control the position and movement.

The online planned trajectory can be generated in the CC or on the test object itself. It can be generated based on a scenario description and with respect to real-time events and gated through the CC.

The dynamic (or online planned) trajectory may not be known at the start of a specific scenario execution neither by the test object nor by the CC, but it is calculated during execution of the test.

If generated in the CC, the dynamic trajectory is sent from the CC to the test object during the ongoing test in smaller pieces or chunks utilizing the trajectory message (TRAJ), if supported by test object and by the CC.

It is recommended that the dynamic trajectory is sent with a sufficient frequency for not exceeding the networks maximum transmission unit.

To increase safety during test, each new piece of the dynamic trajectory sent to the test object could include a safe-way-out. The safe-way-out is the last part of the updated trajectory, describing how the test object safely can end its participation in the scenario if no further trajectory message is received.

10.4 Scenario-description languages

Optionally this document supports the use of scenario-description languages, where they are compatible with all test objects. Standardised description of test scenarios like the OpenSCENARIO is well known and industry standard but in some cases, it does not cover all the needs of the proving ground.

For illustrative purposes, this document contains reference to a scenario metalanguage, iSCAML[17] (independent SCenArio description Meta Language) that may be used in combination with this document. Other scenario-description languages can be used, and this document is designed to be scenario-description language agnostic.

Test scenarios defined by a scenario-description file, i.e. a textual representation in file format, can include details of the triggers the test object is sensitive to, actions initiated by these triggers and more information relevant for this specific scenario. Scenario-description files can be used to cover the following three cases (and permutations of them):

a) static trajectories (CC transmit prior to test);

b) dynamic trajectories, generated at moveable test object;

c) dynamic trajectories, generated in the CC and transmitted on time to the moveable test object.

Scenario descriptions are complemented by descriptions of the static surrounding environment (scenery, e.g. lanes). To execute a test scenario on proving ground, both descriptions may be necessary. In the Clause 11 a possible use case is detailed..

The scenario-description file and corresponding scenery description is sent from the CC to every test object prior to the “ARM” request. The scenario description used is given in iSCAML [17], and the scenery is described using OpenDRIVE. Transfer of scenario information is sent using a GEDM (see 12.3.24), or by FTP. Each test object shall act as defined by this file once the start messages are received from the CC. It shall behave (i.e. drive, act and react) as defined in the scenario-description file once a trigger event becomes valid. If an emergency stop message is received, or such condition discovered by the test object itself, the scenario shall be aborted immediately, overriding remaining instructions provided by the scenario description.

If an updated scenario-description file is sent to the test object it is defined when / how to replace the existing scenario-description file on the test object in the GEDM.

Also, several scenario-description files can be stored on the moveable test object. The selection of a specific scenario-description file is controlled by the events of the traffic scenario itself.

11 Functional requirements

11.1 Device interface description XML (DIDX)

To be able to read and write configuration parameters to/from a test object, a request and response scheme is provided. A read or write request is initiated from the control centre. The test object supported parameters shall be defined in the DIDX-file, in accordance with Annex D.

The DIDX describes the supported messages with their structure for a test object. Additional information, limitations and parameters of the test object are also included in the DIDX.

The vendor shall provide a DIDX-file for each test object.

The DIDX-file is an XML-file and that is divided into seven sections. The file shall contain the DIDX-structure as specified in this document.

a) **Information:** this clause contains some general information about the file and the test object. All information specified in this document shall be included in the DIDX-message. Vendor-specific information may be added.

b) **Limitations:** this clause contains the limitations of the test object. The limitations and the corresponding units are specified in this document. Vendor-specific limits may be added.

c) **Messages:** for each message that the test object supports, a data entry shall be defined in the DIDX. There are two types of messages:

   1) **ISO/TS 22133 message:** for all messages which are defined by this document and that the test object supports.

   2) **Vendor message:** all messages which are not defined by this document.

d) **Triggers and actions:** This clause contains what triggers and actions are supported by the test object.

e) **Datatypes:** Special datatypes are specified in this clause and can be referenced by another datatype or a content of a message. A datatype can be a structure, an enumeration or a bitfield.

f) **Parameters:** Describes the supported parameters of the test object which can be access via the CC. Each parameter has a name, parameter id, access-type and datatype.

g) **Error codes:** Defines the error codes from the MONR message

h) **Protocol tunnel:** Vendor-specific messages.

Further description of the DIDX file structure to be followed, and an example file, is given in Annex D.

11.2 Control centre requirements and recommendations

The following requirements and recommendations apply.

a) Static IP addressing shall be supported and be configurable.

b) The CC or proving ground should provide RTK position correction using NTRIP. If correction data are not available or distributed from the CC, then it shall be ensured that at least all test objects are operating in the same coordinate system with the same minimum accuracy of correction data.

c) The CC shall perform safety monitoring. Safety monitoring is achieved by monitoring the position, speed, yaw, etc. for all dynamic test objects being part the traffic scenario or being operated within the defined geo-fence of the traffic scenario.

d) The CC shall enforce test area boundary (geofence).

e) The CC shall support the feature “emergency stop” when any test object is out of test area boundary. The CC should also support “soft stop”. See Clause 7.

f) The CC shall support test mode for executing static trajectories.

g) The CC may support test modes for executing dynamic trajectories and scenario descriptions.

11.3 Stationary test object requirements

The following requirements apply.

a) Static IP addressing shall be supported and be configurable.

b) A stationary test object shall support the possibility to adjust the test object name.

Examples of stationary test object control:

— weather machine (rain, fog, snow, etc.);

— traffic light.

11.4 Moveable test object requirements and recommendations

The following requirements and recommendations apply.

a) Static IP addressing shall be supported and be configurable.

b) A moveable test object may be controlled by a safety driver, if needed. A fully automated operation can be supported if the safety requirements for the operation within the traffic scenario are fulfilled.

c) A movable test object shall report its position, speed, yaw, status, etc. using the monitoring message. This message shall be sent periodically.

d) A movable test object shall execute a proper safety measure when detecting a communication outage, according to communication timeout (see 12.3.2).

e) A moveable test object should support safe torque off (switch off torque to the propulsion system). For electrically powered test objects IEC 61800-5-2 is relevant.

f) A moveable test object shall support a soft stop (slow down to zero velocity according to a defined scenario/trajectory).

g) A moveable test object shall support an emergency stop (maximum breaking deceleration at constant steering angle).

h) A moveable test object shall support a local memory storage which is used to store all relevant local data (such as time, position, velocity, yaw, estimated own deviation, test object status) for the entire test scenario to be able to search for problems in post-mission even when the communication network failed significantly.

i) A moveable test object may support manual transport mode (the remote-control functionality) to reposition test objects when not in testing mode.

j) If manual transport mode is supported there shall exist a safety speed limit.

k) A movable test object shall support the possibility to adjust the test object name.

l) A movable test object shall support test mode for executing static trajectories.

m) A movable test object may support test modes for executing dynamic trajectories and scenario descriptions.

11.5 Functions with behaviour description

11.5.1 Arm and disarm test object

Each test object can be armed and disarmed by request from the CC. To change a test object's state from disarmed to armed or vice versa the message OSTM is used. Acknowledgement of a successful changes of state is received via MONR message (which is continuously sent from all test objects to CC). The procedure for arming and disarming a test object is illustrated in Figure 20.

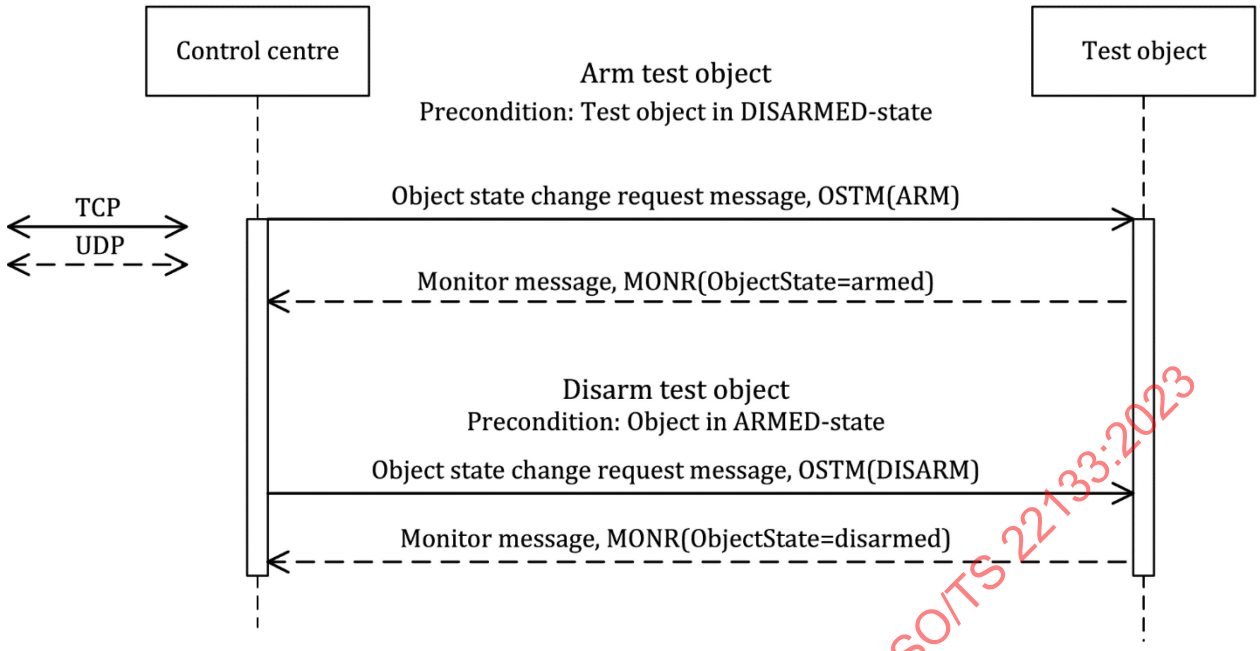
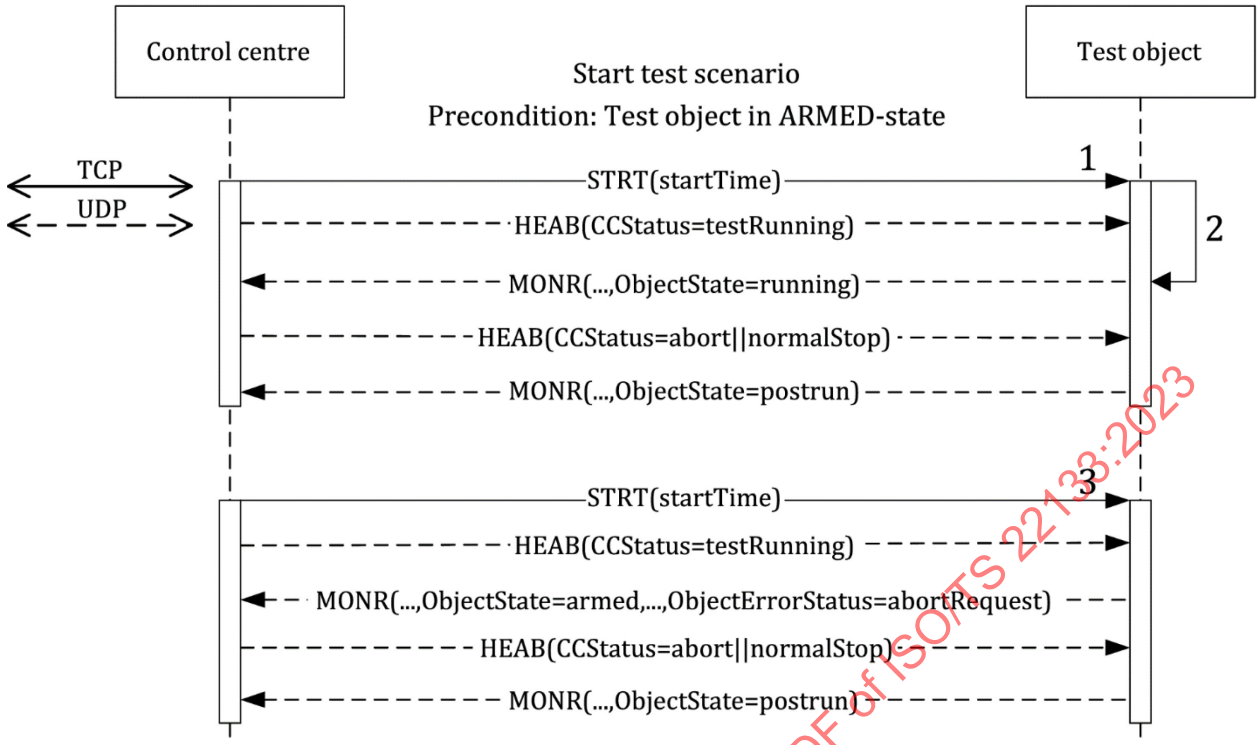


Figure 20 — Arm and disarm test object

11.5.2 Start test

All test objects participating in a test scenario can be started at the same time via a STRT message. The message contains a parameter with the start time. The STRT message shall be sent to the test object in good time before the specified `start time`.

If the `start time` in the message is already passed (`current time > start time`) when the message is received, then the test object shall immediately abort test and request the CC to abort the test. See Figure 21.



Key

1 if `currentTime` < `startTime` when STRT message is received test object goes to running state when the specified `startTime` is reached

2 waits until `currentTime` >= `startTime`

3 if `currentTime` > `startTime` when STRT message is received, it does not start test if `startTime` has passed when STRT message is received

Figure 21 — Start test

11.5.3 Emergency stop of test scenario (initiated by the CC)

To initiate an emergency stop of test scenario, the CC will send a status flag "Emergency abort" in the heartbeat (HEAB) message to the test objects. Once a test object has received and interpreted the request it shall perform an emergency stop and send `aborting` in the monitor message, MONR. Once the test object is stopped it shall send `postrun` in MONR. The procedure for emergency stop initiated by the CC is illustrated in Figure 22.

The CC shall send ABORT when any of the following occurs:

a) one or more test objects are deviating from the path outside the global geofence;

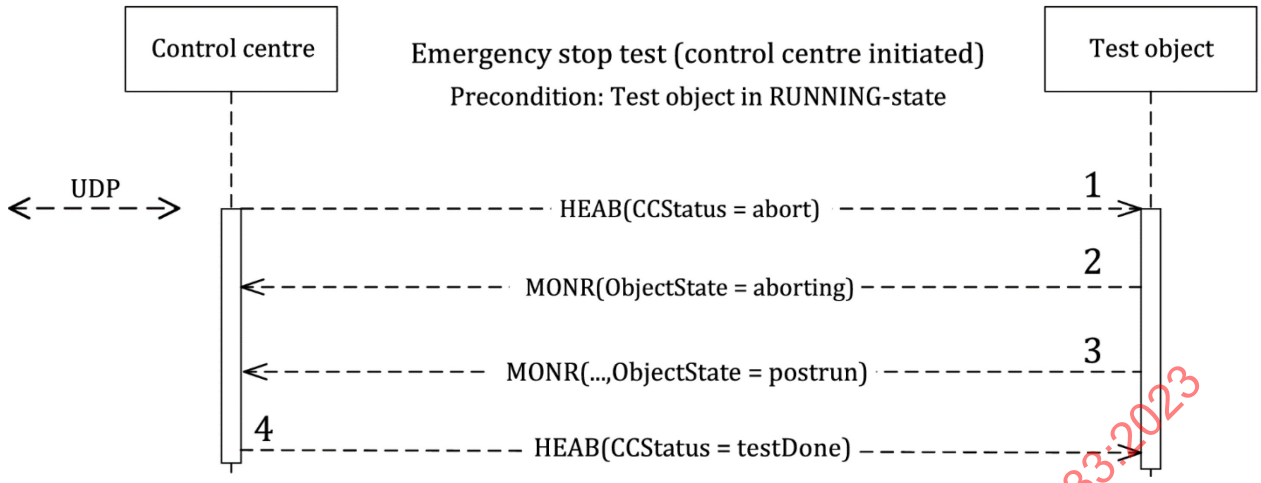
b) test object is moving outside its local virtual fence (i.e. when the CC receives MONR with ObjectErrorStatus `(outsideGeoFence==TRUE);`

c) a predefined number(n) of consecutive MONR messages from a test object are missing/not received OR older than a predefined timeout threshold;

d) any test object is sending `abortRequest` signal `(abortRequest==TRUE);`

e) unexpected collision risk between test objects detected;

f) emergency stop has a higher priority than normal stop.



Key

1	start break
2	stopping/breaking
3	test object stopped
4	test finished

Figure 22 — Emergency stop / abort test (CC initiated)

11.5.4 Emergency stop of test scenario (upon request from test object)

Test object shall abort the test and send `abortRequest` (in the `ObjectErrorStatus`-signal inside the MONR-message) to the CC if one or more of the following conditions are fulfilled:

a) the test object is deviating from its planned trajectory more than the allowed deviation tolerance;

b) the test object is moving outside its local virtual fence;

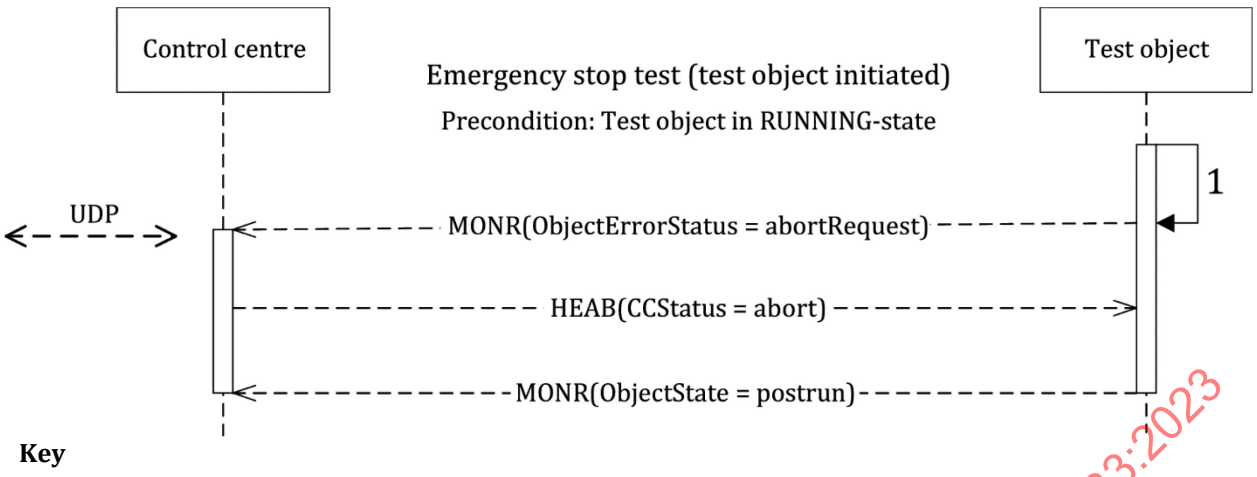
c) a predefined number (`n`) of consecutive HEAB messages from the CC are not received;

d) the timestamp of `n` consecutive HEAB messages are older than a predefined time error threshold value;

e) positioning accuracy is low;

f) internal test object faults are detected (e.g. battery issues, motor issues, etc.).

The procedure for emergency stop initiated by the test object is illustrated in Figure 23.



Key

1 error detected, emergency stop required

Figure 23 — Emergency stop (initiated by test object)

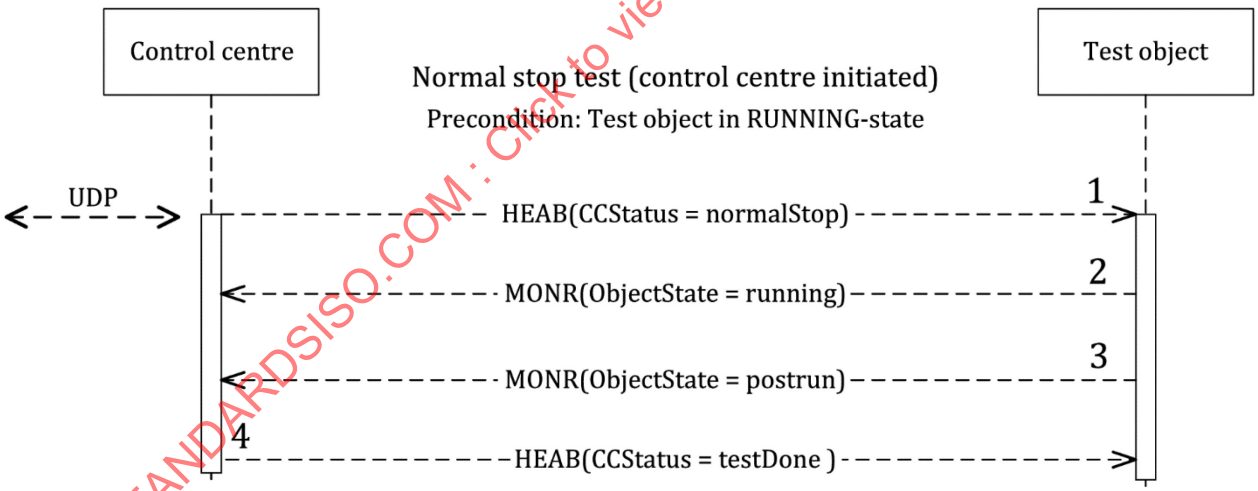
11.5.5 Normal stop of test scenario

Normal stop is a way to stop the test object in a controlled way but with less efficiency than an emergency stop. The intention of this functionality is to reduce tension on mechanical parts due to applying the brakes in a gentler way.

— Normal stop has a lower priority than an emergency stop.

— Normal stop command is in the attribute `CCStatus` in HEAB message sent from CC.

The procedure for normal stop initiated by the CC is illustrated in Figure 24.



Key

1 start breaking

2 stopping/breaking

3 test object stopped

4 test finished

Figure 24 — Normal stop (initiated by control centre)

11.5.6 Download static (pre planned) trajectories

To download one or several pre-planned trajectory file(s) from the CC to a test object, the test object runs/follows this trajectory once it enters the running state. The OSEM is always sent before the TRAJ message, as illustrated in Figure 25.

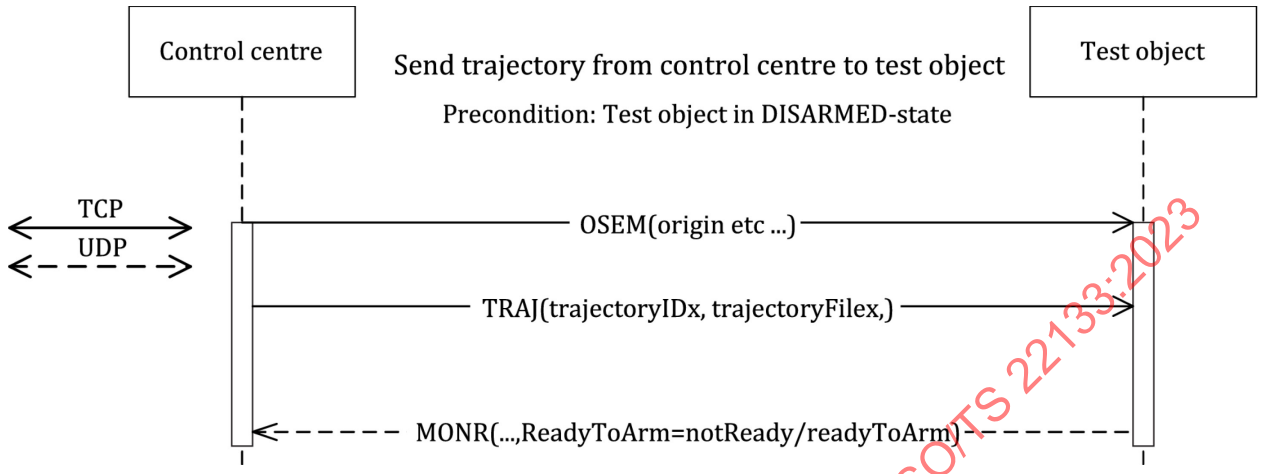


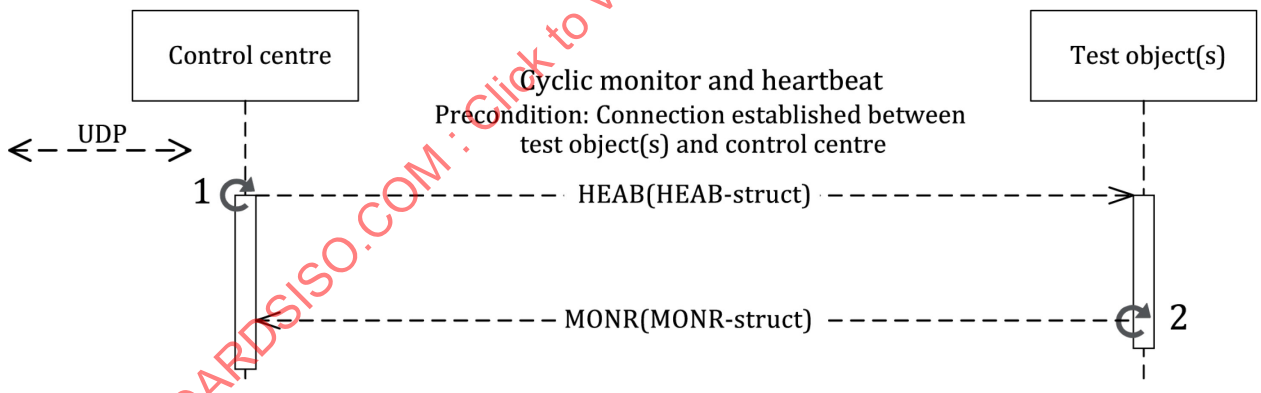
Figure 25 — Configure trajectories

11.5.7 Cyclic monitor and heartbeat

Cyclic messages, MONR (monitor message), are sent with status information from each test object to the CC.

Cyclic messages, HEAB (heartbeat message), are sent with status information and commands from the CC to each test object individually (no broadcast).

The procedure for periodic monitoring is illustrated in Figure 26.



Key

1 HEAB repeatedly sent

2 MONR repeatedly sent

Figure 26 — Periodic monitoring

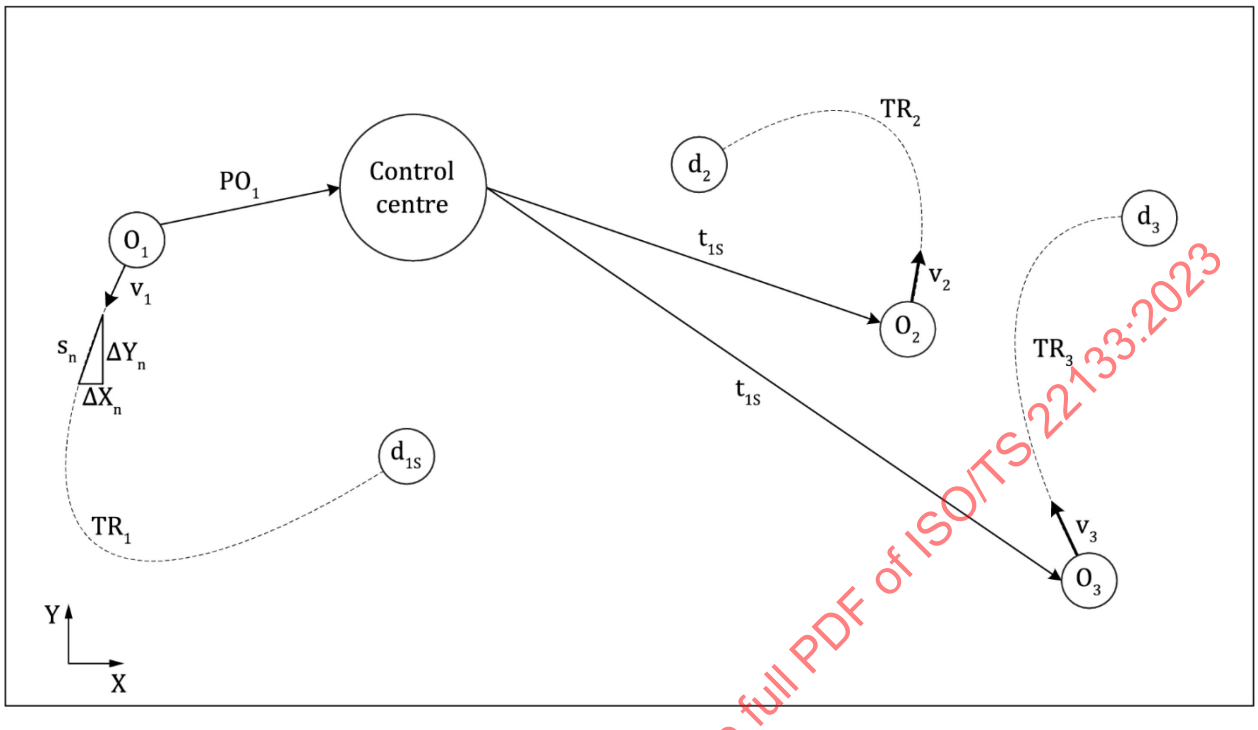
11.5.8 Adaptive synchronization point

11.5.8.1 Adaptive synchronization point – General information

Synchronization points are used to control the velocity of a test object (slave) compared to another test object (master). An adaptive synchronization point is defined as:

All test objects need to reach the respective destination point d_p at the same time as the synchronization test object O_s reaches the synchronization point ds.

The slave and master test object will follow their planned path, but the slave shall adapt its own velocity, so it reaches the defined `slave synchronization point` when the master test object reaches the `master synchronization point`.



Key

d_{1s}	synchronization point of test object 1	TR_2	trajectory for test object 2
O_1	test object 1 (travelling towards d_{1s})	O_2	test object 2
PO_1	position test object 1	v_2	speed of test object 2
v_1	speed of test object 1	d_3	destination point of test object 3
TR_1	trajectory for test object 1	TR_3	trajectory for test object 3
t_{1s}	time to synchronization point for test object 1 (MTSP)	O_3	test object 3
d_2	destination point of test object 2	v_3	speed of test object 3

Figure 27 — Synchronization point setup

Figure 27 shows a setup with three different test objects O_1, O_2, O_3. Each test object has its own trajectory specified in TR_n and those are distributed to the test objects by the CC. The destination point d_{1s} is configured in the setup as the synchronization point and O_1 is heading for this point. When O_1 travels towards d_{1s} is it constantly sending its position and speed to the server in PO_1. The CC uses PO_1 information (MONR) to calculate the momentaneous time t_{1s} until O_1 reaches d_{1s} and sends t_{est} to O_2 and O_3 in MTSP. The time t_{est} is the time until the O_1 reaches d_{1s}.

O_2 and O_3 uses time information in MTSP to adapt the needed speed to reach their destinations at the same time as O_1 reaches d_{1s}.

In an ASP-test, there is always one master, and one or more slaves. The master does not know about the fact that it is a master test object. The t_{est} is calculated by the CC and sent to the slaves in the MTSP-message. The MTSP-message is periodically sent with and updated `EstSyncPointTime`-value.

NOTE 1 A typical message interval for MTSP is 100 ms / 10 Hz but is possible to adjust it.

NOTE 2 Other means are available to ensure relative timing and position, such an alternative is to include constraints and relationships between test objects in an executable scenario description.

11.5.8.2 Adaptive synchronization point - Message sequence

The message sequence is described below and is visualized in Figure 28.

a) Configuration:
   Before executing the test scenario, synchronization point(s) along the trajectories shall be defined for the master and slave test object(s). The slave synchronisation point is sent to the slave utilizing the SYPM, synchronization point configuration message. The master synch point is internally configured in the CC.
b) Execution:
   The master and slave test objects start moving along respectively trajectory. Slave adjusts its speed, so it reaches the synch point at the `EstSyncPointTime`.
c) Ending/cancellation:
   1) When the master test object reaches its sync point, the CC stops sending MTSP to the slave. The slave test object shall continue along its trajectory. The ASP ends.
   2) When the `EstSyncPointTime` - `currentTime` < `FreezeTime` the slave test object shall end the attempt to align to the ASP, and instead continue to follow the trajectory. The ASP ends.
   3) If a decision is taken to end the ASP, the CC shall send an updated SYPM-message for the `SyncPointTime` in question. The `FreezeTime` shall be set to 0 to cancel/end this sync point. The ASP is cancelled.
   4) Other events might also end the ASP-adaption in the slave. This is up to the slave. If this occurs the `adaptSypPoEnded` bit in `ObjectErrorStatus` in the MONR-message shall be set to TRUE.

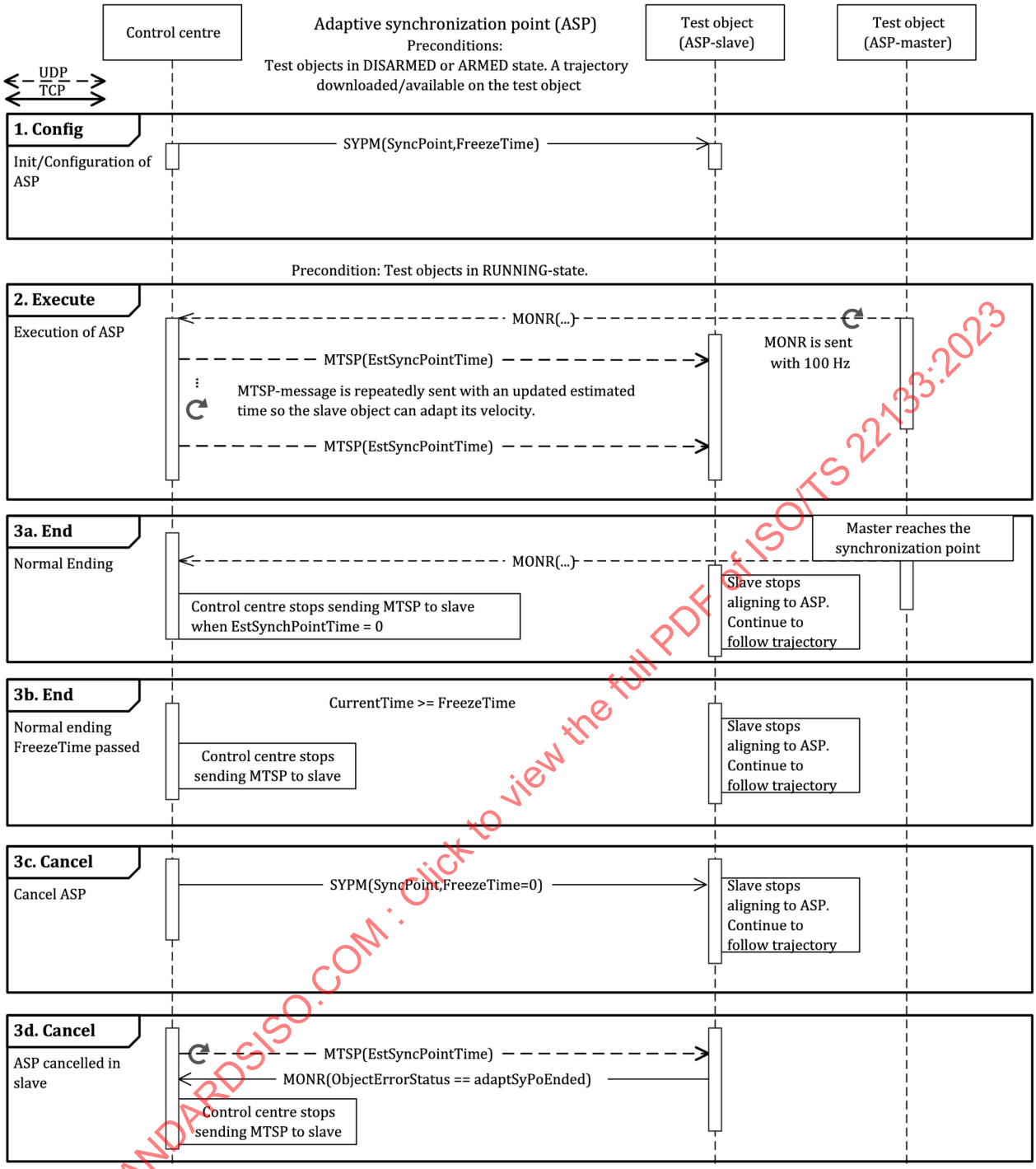


Figure 28 — Adaptive synchronization point sequence diagram

11.5.9 Remote-control manoeuvring

This is a way of manually drive/control test objects from the CC. A message is sent from the CC to a test object with a frequency of 10 Hz during remote-control manoeuvring sessions, see Figure 29.

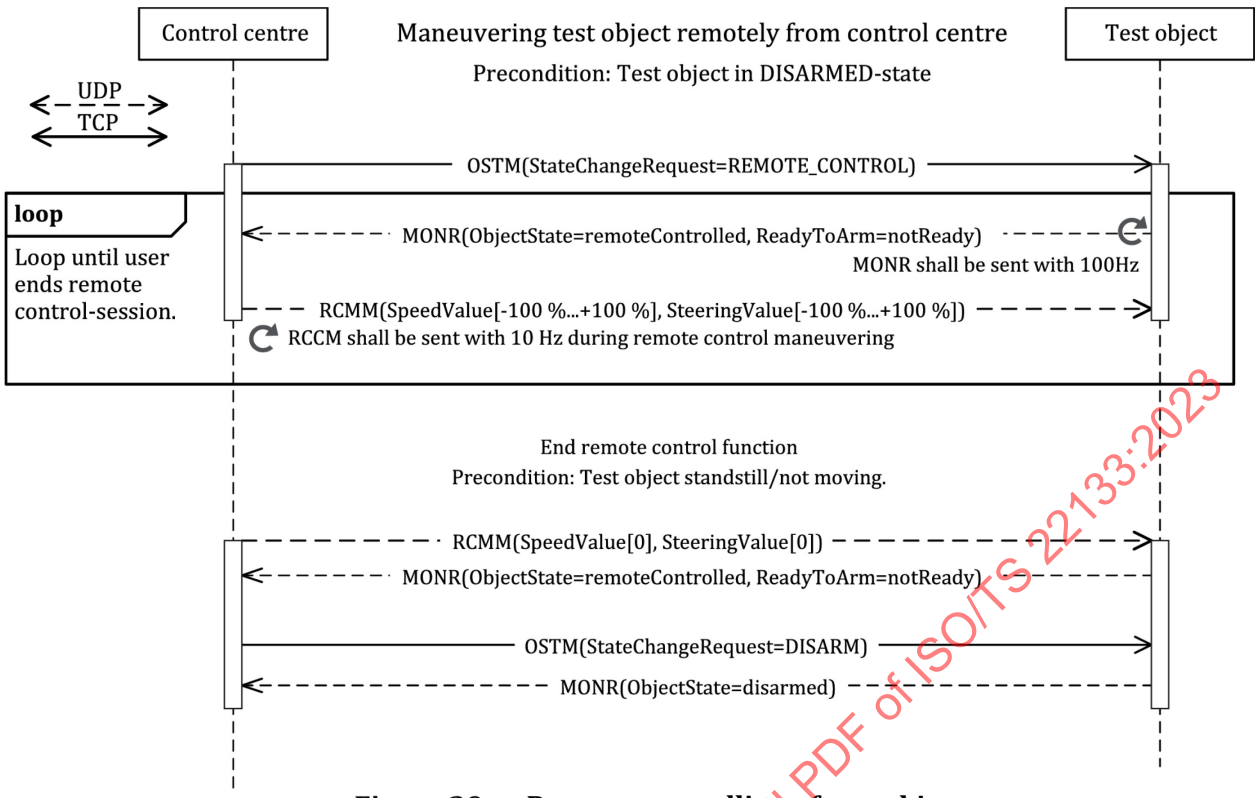


Figure 29 — Remote controlling of test object

11.5.10 Trigger and action

The trigger and action fundamentals are defined by trigger and action events that may be necessary to execute a test scenario. The trigger and action events can be extracted from a test scenario and communicated as described in the following section. If a test object supports direct test execution utilising a distributed scenario description, triggers and actions may also be executed using this method.

A trigger is one or several predefined conditions that are fulfilled. The configuration of the trigger event is sent from the CC to the `TriggerEvent` reporting test object during initializing phase of the test scenario.

A `TriggerEvent` can be reported by a test object or the CC, depending on what and how the configuration is setup through the trigger configuration message. Only one instance is allowed to report a `TriggerEvent` for a specific `TriggerID`.

An `ExecuteAction` is sent to the test object(s) that is expected to execute the requested action, specified in the action configuration message. One or several test objects may be the action executor.

When setting up the TriggerType and ActionType, there are a number of ValueID:s used for configuring the trigger condition as well as the action event. If the `TriggerType`, `TriggerTypeParameters`, `ActionType` and `ActionTypeParameters` are not sufficient for configuring the trigger and action event, then it is appropriate to use vendor specific ValueID:s.

An example list of typical TriggerTypes and ActionTypes can be found in Annex C.

Figure 30 describes the message sequence for configuring, executing and cancelling triggers and actions.

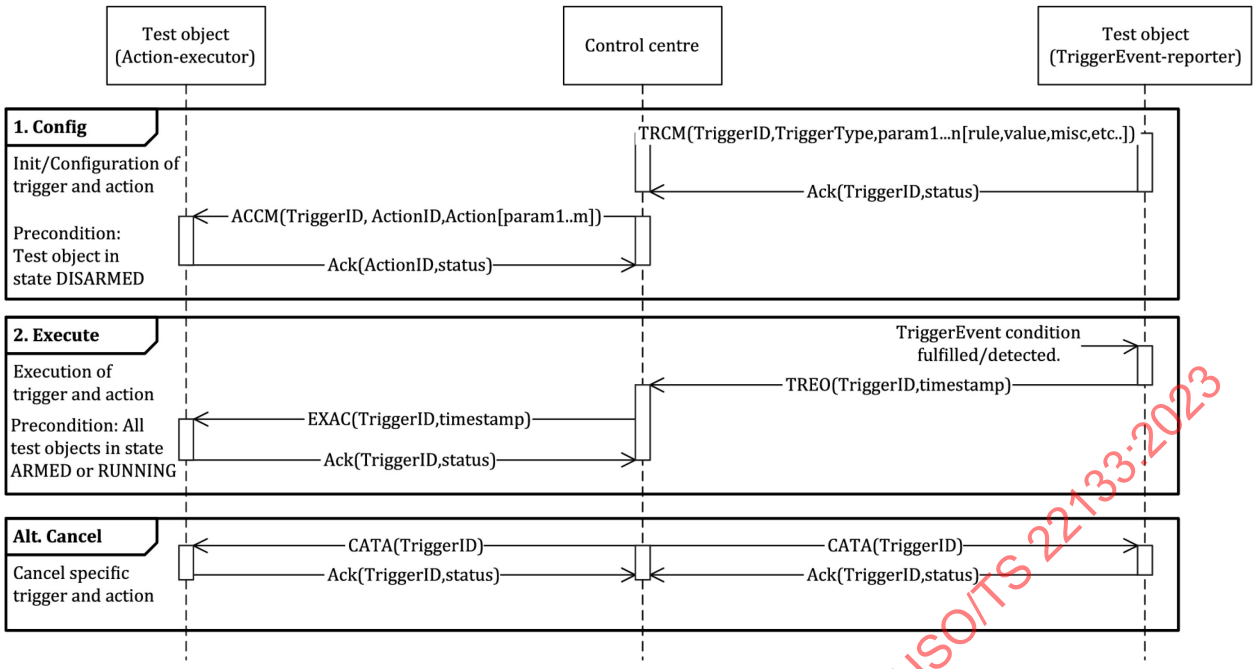


Figure 30 — Trigger and action sequence diagram

Trigger and action messages:

- trigger configuration message (TRCM);
- action configuration message (ACCM);
- trigger event occurred message (TREO);
- execute action message (EXAC);
- cancel trigger and action message (CATA).

12 Interface requirements

12.1 General

To identify and ease communication a common layer reference is used within this document, the layer model described in Table 8 is used.

Table 8 — Layer model

Layer 3: Application-specific extensions	
Private data types Extended state, configuration data, testing, debugging, etc.	**Private messages** Device-specific functions setup, calibration, etc.

Layer 2: Application services	
Application functions Adaptive synch points, remote control, etc. **Application messages** All defined mandatory and none-mandatory messages	**Application data types** Data types used for communication between the CC and test objects **Device state** Exchange of generic test object state
Layer 1: Basic services	

Session handling Attaching / detaching of devices Managing communication **Time synchronization** Define and provide all devices with time synchronization.	**Position correction data** Provide all test objects with global and common position correction data. **Message header definition** A common header definition for all messages
Layer 0: Transport layer TCP/UDP/IP	

12.2 Message

12.2.1 Message structure

A message contains header, content(s) and footer and is sent via TCP/UDP/IP. See Figure 31.

12.2.2 Sequential byte order

The sequential order of the byte arrangements shall be according to little-endian-format. That means that components are ordered from the little end with the least significant bit first (LSB first).

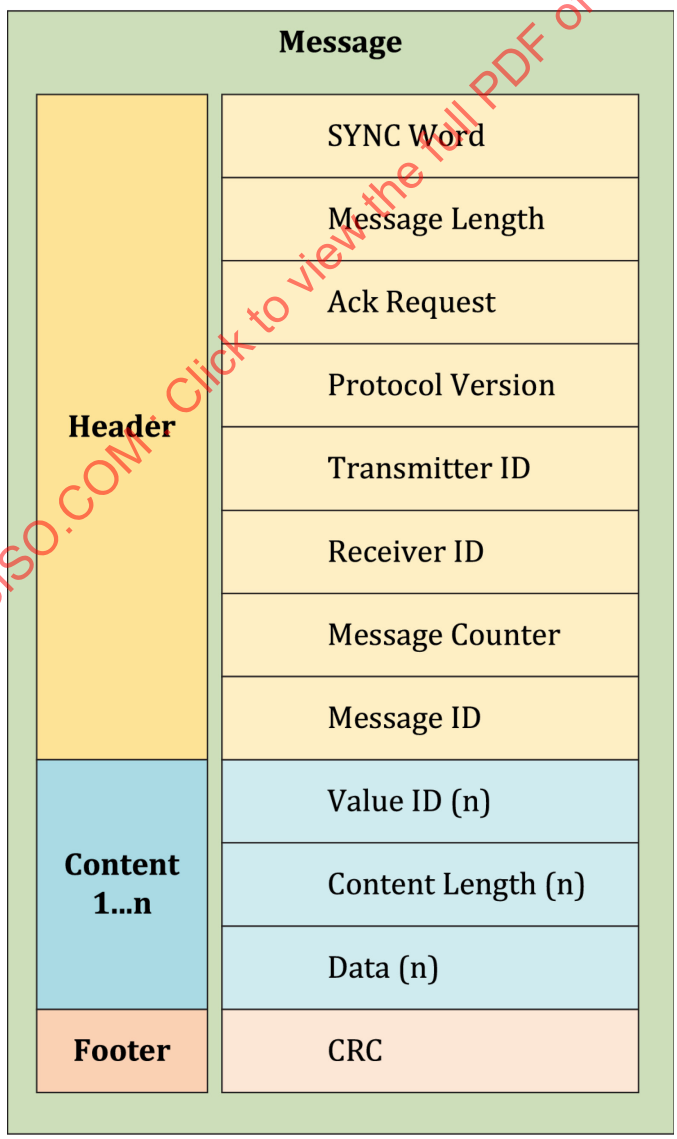


Figure 31 — Message structure

12.2.3 Message header

12.2.3.1 Message header - Structure

The message header is always 18 bytes long and its data are structured according to Figure 32. The data name, data length and its representation can also be seen in Figure 32.

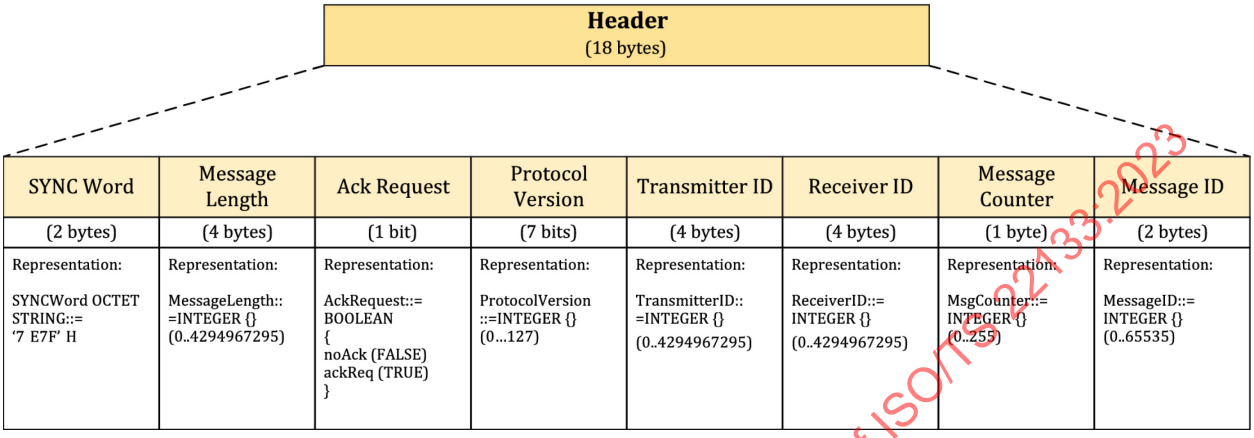


Figure 32 — Message header structure

12.2.3.2 Message header - SYNC word

The sync word/preamble indicates the start of the message stream. Two bytes with value 0x7E7F.

NOTE An asymmetric sync word makes it easier to find endianness errors.

12.2.3.3 Message header - Message length

The total length of all subsequent contents in bytes (excluding package header and package footer (CRC)).

The maximum allowed message length shall not exceed `MaxMessageLength` bytes defined in OSEM (see Table 17, struct order number 10).

12.2.3.4 Message header - Ack request

The ack request bit indicates whether the sender of a message is expecting the receiver to acknowledge the message. The GREM shall be used as the acknowledge message with appropriate content.

— `FALSE` : no ack required. Receiver shall not acknowledge the message.

— `TRUE` : ack required. Receiver shall acknowledge the message.

12.2.3.5 Message header - Protocol version

ISO/TS 22133 protocol version identifier: the current edition of this document specifies version 2. Version 1 is deprecated as it was used during the development.

A higher/greater number indicates a later edition of the implementation. 7 bits are used. Resulting in possible range from 0-127.

A protocol change on a high level such as changing header or message structure will result in an increased protocol version. Minor changes in specific messages may not result in any protocol version number change.

12.2.3.6 Message header - Transmitter ID

When a message is sent from an entity the sender shall use its device ID as a transmitter ID. The test objects will get their transmitter ID to use from the CC in the OSEM directly after the TCP connection is established.

If two or more test objects are connected behind the same IP-address, then the transmitter ID shall be different for each test object and be distinguished by `Sub-Object ID` in OSEM. `Sub-Object ID` can be retrieved by using DREQ and DRES mechanism before establishing the TCP connection to the test object.

— A device ID equal to zero is not allowed. The CC shall manage all device IDs in the setup. The device ID is distributed to the test objects in the OSEM.

— A test object has no active device ID before OSEM is received. Device ID 0xFFFFFFFF may be used to indicate an invalid/unknown ID.

12.2.3.7 Message header - Receiver ID

The receiver ID is an identifier that informs about the intended receiver of the message. A receiver will be a unique number (device ID) defined and managed by a CC.

If the receiver ID is zero, the message is intended to be received and used by the TCP connected CC.

The message will be gated through the CC if the receiver ID is not equal to zero and not equal to the TCP connected CC device ID. It is optional for the CC to support message-gating functionality. The CC shall send the message on the appropriate communication channel of the message to the device ID (defined by receiver ID in the header) without changing the transmitting ID field in the message.

12.2.3.8 Message header - Message counter

Message counter represents the sequential number of messages sent per direction between the CC and test objects. Each test object manages its own message counter, and the CC manages unique counter per test object. The message counter resets upon reaching its maximum value 255, starting again at 0.

The purpose with the CC is to be able to detect missing messages or duplicates of messages, which implicitly can be interpreted as some type of error in the transmitting device.

12.2.3.9 Message header - Message ID

This defines which message identifier to be sent. See Table 9.

NOTE Vendor-specific message IDs are available. See Table 9 for the range.

12.2.4 Message content

12.2.4.1 Message content - Structure

The message body can contain one or several content blocks. In Figure 33 the content structure is defined, where the `ValueID` is given first, followed by the content length and then the actual data content. If several contents are sent in the same message, the contents will follow each other without any additional delimiter (except for the `ValueID` and `content length`).

Even if message contents are specified as structs (e.g. MONR and HEAB) the ValueID and `content length` shall be included in the message.

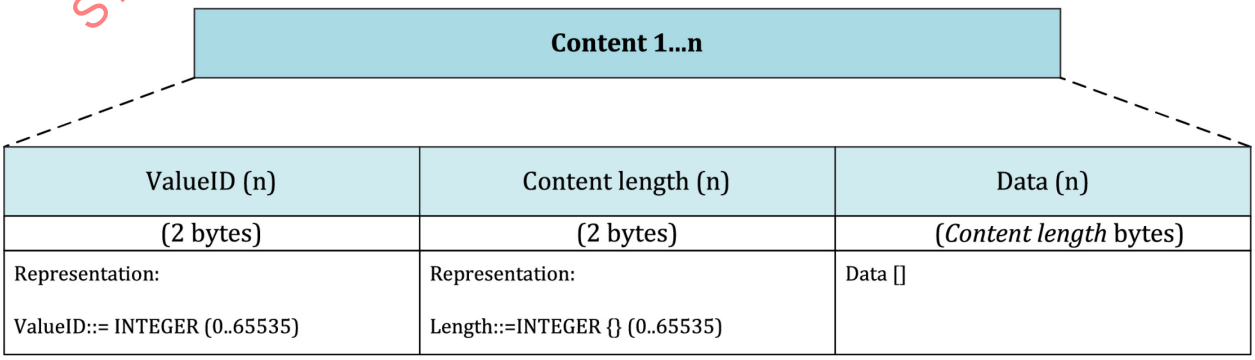


Figure 33 — Message content structure

12.2.4.2 Message content - ValueID

This is the ID for a value representation within the message context. The ValueID values are only locally unique within a specific message. The same ValueID can exist in other messages, but with a completely different name, purpose, context and implication.

NOTE Vendor-specific ValueID ranges are available for some messages. See ValueID table for respective message.

12.2.4.3 Message content - Content length

This is the length of the subsequent data.

NOTE The 2+2 bytes (ValueID + content length) is not included in the value of content length.

12.2.4.4 Message content - Data

This is the actual payload data. Content is described in the message overview subclause. All specified data are mandatory, if not used the `unavailable` option shall be sent.

12.2.5 Message footer

12.2.5.1 Message footer - Structure

The footer indicates the end of the message. It consists of only one data field: CRC. See Figure 34.

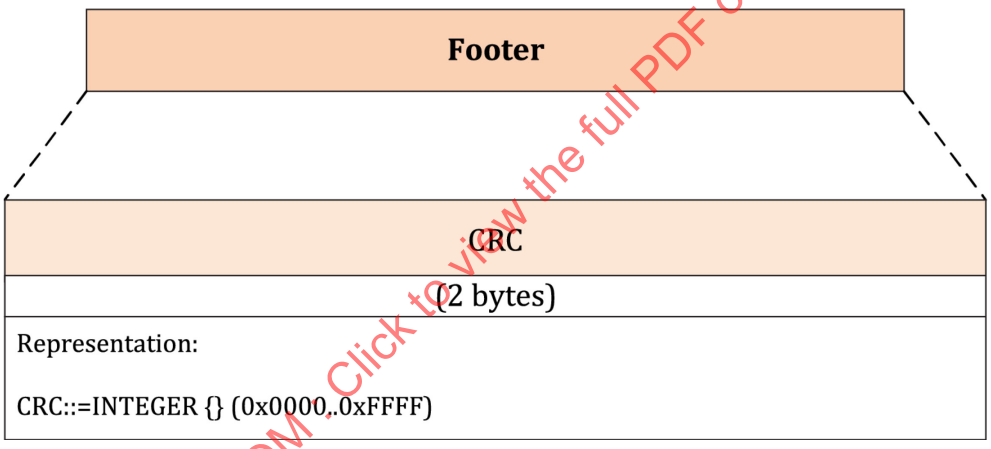


Figure 34 — Message footer structure

12.2.5.2 Message footer - CRC

This is the checksum calculation based on the principles from CRC16-CCITT/ITU-T. See Annex B for example of a CRC calculation implementation.

The CRC checksum shall be calculated based on the data in `Header + Content`.

12.2.6 Protocol tunnel

The protocol tunnel is to be used to tunnel non-ISO/TS 22133 protocol through the ISO/TS 22133 network protocol from/to a test object. To achieve a valid tunnel, the protocol shall include the ISO/TS 22133 header and footer. The data can be protocol specific. Protocol tunnels are fixed to a MessageID on the test object. This MessageID and other information like protocol vendor, protocol name and the protocol version are specified in the DIDX File.

It may happen that two vendors support the same protocol but on different MessageIDs. It is the CC's responsibility to communicate with the correct MessageIDs to the test object.

One way to implement a protocol tunnel is to use the GEDM.

12.2.7 Vendor-specific messages

Vendor-specific messages are messages which are not defined in this document but follow the message format in 12.2.1 to 12.2.5. Vendors-specific messages can be defined in the DIDX file (vendorMessage) to be able to implement this message into the CC or test object.

12.3 Collective message overview

Table 9 presents an overview of all the available messages.

Table 9 — Overview of messages, (M) is mandatory to support, (O) is optional to support

Message name	Abbrev	MessageID	Frequency	Direction	Channel	Explanation
Trajectory Object Message (M)	TRAJ	0x0001	Non-periodic	Control centre to test object	TCP	Message to control the trajectory of a dynamic test object
Object Setting Message (M)	OSEM	0x0002	Non-periodic	Control centre to test object	TCP	Message to transfer settings to test object
Object State Message (M)	OSTM	0x0003	Non-periodic	Control centre to test object	TCP	Message to request a state change in the test object
Start Message (M)	STRT	0x0004	Non-periodic	Control centre to test object	TCP	Message to start a test
Heartbeat Message (M)	HEAB	0x0005	Periodically 10-100 Hz	Control centre to all test objects	UDP	Message to send test status to test objects
Monitor Message (M)	MONR	0x0006	Periodically 1-100 Hz	Test object to control centre as unicast	UDP	Message to supervise and visualize test object
Monitor Message (global, absolute position) (O)	MONR2	0x0007	Periodically or event based 1-100 Hz	Test object to control centre	UDP	Message to supervise and visualize test object
GPS Second of Week Roll Over Message (O)	SOWM	0x0008	Non-periodic	Control centre to all test objects	TCP	Message to inform about current GPS week and to inform about when GPS second of week roll over happens
GeoFence Message (O)	GEOF	0x0009	Non-periodic	Control centre to test object	TCP	Message to download the geo fence to the test the test object
Remote Control Manoeuvring Message (O)	RCMM	0x000A	Periodically during remote-control manoeuvring	Control centre to one test object at a time	UDP	Message to manually control (drive/steer) the test object
Remote Control Manoeuvring Message 2 (O)	RCMM2	0x0016	Periodically during remote-control manoeuvring	Control centre to one test object at a time	UDP	Message to manually control (drive/steer) the test object
General Data Message (M)	GEDM	0x0017	Non-periodic during DISARMED state	Control centre to test object and vice versa	TCP/UDP	Transfers a miscellaneous data

Message name	Abbrev	MessageID	Frequency	Direction	Channel	Explanation
General Response Message (M)	GREM	0x0018	Non-periodic	Control centre to test object and vice versa	TCP/UDP	General response status
Discovery Request Message (M)	DREQ	0x0010	Non-periodic	Control centre to test object	UDP	
Discovery Response Message (M)	DRES	0x0011	Periodic before connected-state	Test object to control centre	UDP	
Parameter Request Message (M)	PREQ	0x0012	Non-periodic	Control centre to test object	TCP	
Parameter Response Message (M)	PRES	0x0013	Non-periodic	Test object to control centre	TCP	
Synchronization Point Configuration Message (O)	SYPM	0x000B	Non-periodic	Control centre to test object	TCP	Message for configuring synchronization point information to slave test object
Master time to synchronization point message (O)	MTSP	0x000C	Periodically	Control centre to test object	UDP	Message for sending remaining time to synchronization point
Trigger Configuration Message (O)	TRCM	0x0021	Non-periodic	Control centre to test object	TCP	Message to enable and configure the TriggerEvent in the test object
Action Configuration Message (O)	ACCM	0x0022	Non-periodic	Control centre to test object	TCP	Message to enable and configure the ActionEvent in the test object
Trigger Event Occurred Message (O)	TREO	0x0023	Non-periodic		TCP	Message from the test object that the trigger occurred
Execute Action Message (O)	EXAC	0x0024	Non-periodic	Control centre to test object	TCP	Message to activate/execute the preconfigured action
Cancel or Delete Trigger Action Message (O)	CADE	0x0025	Non-periodic	Control centre to test object	TCP	Message to cancel/abort the trigger and action function (before or during test)
Action Performed Message (O)	APEM	0x0026	Non-periodic		TCP	Message that informs about an executed action
Reserved for future use	-	0x0100-0x0FFF	-	-	-	-
Protocol tunnel	-	0x1000-0x1FFF	Protocol specific	Protocol specific	UDP/TCP	For tunnelling a protocol through the ISO stack
Vendor Specific Range	-	0x2000-0x2FFF	Vendor specific	Vendor specific	UDP/TCP	Vendor-specific ISO messages

Message name	Abbrev	MessageID	Frequency	Direction	Channel	Explanation
Reserved for future use	-	0xF000-0xFFFF	-	-	-	-

12.3.1 Trajectory object message - TRAJ (MsgID 0x0001)

The trajectory message is used for transferring trajectories from the CC to the test object. The ID of the trajectory shall not be zero and shall be unique on the test object. The name of the trajectory has only an informational character.

If a trajectory on the test object should be deleted, the CC shall send the trajectoryID as well as the trajectoryInfo with the delete bit set to true. If the trajectoryID zero is send and the delete bit is set to true, all trajectories shall be deleted on the test object.

If only chunks of a trajectory are downloaded the first point of the following chunk shall be the last point of the already downloaded trajectory. It is the test objects responsibility to concatenate the trajectories chunks.

Sequence for sending data:

1) the first three ValueIDs: (`trajectoryID`, `trajectoryName`, `trajectoryInfo`) is only sent once per trajectory file (content i, ii, iii);
2) the succeeding ValueIDs: (content iv) - directly related to the positions and their dynamics- are repeated for each data point/line in the trajectory file (point/line content 1…11).

If the test object test mode is `Online planned trajectories` (dynamic trajectories), the test object shall respond with a GREM (chunk received) when data chunk is received.

Trajectory information can be described using trajectory text files. To enable interoperability, the text file format given in Annex A shall be used.

The TRAJ message is further detailed in Tables 10, 11 and 12.

Table 10 — ValueID table for TRAJ

ValueID name	ValueID value (Uint16)
[trajectoryID]	0x0101
[trajectoryName]	0x0102
[trajectoryInfo]	0x0104
[TrajPoint]	0x0001
[Line Info]	0x0053
[Vendor Specific] (range)	0xA000-0xAFFF

Table 11 — TRAJ

Content order number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
i	0x0101 [trajectoryID]	0x0002 [Uint16]	Trajectory Identifier <ID>	Identifier of the trajectory file. This allows the test object to distinguish between one or several saved/downloaded trajectories. Zero is not allowed,
ii	0x0102 [trajectoryName]	0x0040 [64 bytes String]	Trajectory name <Name>	The friendly/readable name of the trajectory file. It has only an information character.

Content order number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
				Length is 64 bytes. Characters in the text string are following an eight-bit character representation according to ISO 8859-1 (Latin-1). The text string shall be null terminated.

iii	0x0104	0x0001 [Uint8]	Trajectory Info	Representation:
				`TrajectoryInfo ::= INTEGER {`
				`RelativeToObject (1)`
				`RelativeToOsemOrigin (2)`
				`DeleteTrajectory (3)`
				`}`

iv	0x0001	0x001E [TRAJ point Struct]		[TRAJ Point struct] See Table 12.

v	0x0053	0x0001 [Uint8]	End of transmission	Representation:
				`ObjectState::=INTEGER {`
				`EOT(4)`
				`}`

Table 12 — TRAJ point struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
1	[Relative Time]	0x0004 [Uint32]	Time relative to starting time	Representation: `RelativeTime::=INTEGER {oneMilliSeconds(1)} (0..4294967295)` Definition: this value defines the relative time from start (start = 0). Unit: 0,001 s
2	[X- position]	0x0004 [Int32]	Position in the X-direction	Representation: `DeltaPosition::=INTEGER {oneMilliMeter(1)} (-2147483648..2147483647)` Definition: this value defines the distance between defined position and origin (OSEM) in X direction. Unit: 0,001 m
3	[Y- position]	0x0004 [Int32]	Position in the Y-direction	Representation: `DeltaPosition::=INTEGER {oneMilliMeter(1)} (-2147483648..2147483647)` Definition: this value defines the distance between defined position and origin (OSEM) in Y direction. Unit: 0,001 m

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
4	[Z- position]	0x0004 [Int32]	Position in the Z-direction	Representation: `DeltaPosition::=INTEGER {oneMilliMeter(1)} (-2147483648..2147483647)` Definition: this value defines the distance between defined position and origin (OSEM) in Z direction. NOTE Informative for non-flying test objects. Unit: 0,001 m
5	[Yaw]	0x0002 [Uint16]	Yaw of the Test Object	Representation: `YawValue::=INTEGER {positiveXdirection(0), positiveYdirection(9000), negativeXdirection(18000), negativeYdirection(27000), unavailable(65535)} (0..36001)` Definition: See 6.3.2. Unit: 0,01°
6	[Longitudinal Speed]	0x0002 [Int16]	Longitudinal speed (in forward direction)	Representation: `SpeedValue::=INTEGER {standstill(0), oneCentimeterPerSec(1) } (-32768..32767)` Definition: See 6.3.2, x-axis. Unit: 0,01 m/s
7	[Lateral Speed]	0x0002 [Int16]	Lateral speed (perpendicular to the forward direction)	Representation: `SpeedValue::=INTEGER {standstill(0), oneCentimeterPerSec(1) , unavailable(-32768) } (-32768..32767)` Definition: See 6.3.2, y-axis. Unit: 0,01 m/s
8	[Longitudinal Acceleration]	0x0002 [Int16]	Longitudinal acceleration (in forward direction)	Representation: `LongitudinalAccelerationValue::= INTEGER {milliMeterPerSecSquaredForward(1), milliMeterPerSecSquaredBackward(- 1), unavailable(-32768)} (-32000 .. 32001)` Definition: See 6.3.2, x-axis. Negative values indicate that the vehicle is braking. Positive values indicate that the vehicle is accelerating. Unit: 0,001 m/s² (1 mm/s²) Range: -32 m/s² to +32m/s²

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
9	[Lateral Acceleration]	0x0002 [Int16]	Lateral acceleration (perpendicular to the forward direction)	Representation: `LateralAccelerationValue::= INTEGER {milliMeterPerSecSquaredToLeft(1), milliMeterPerSecSquaredToRight(- 1), unavailable(-32768)} (-32000 .. 32000)` Definition: See 6.3.2, y-axis. Unit: 0,001 m/s² (1 mm/s²) Range: -32 m/s² to +32m/s²
10	[Curvature]	0x0004 [32-bit single precision float]	Curvature of the trajectory in this specific point. Curvature is expressed in inverse of meter [m^{-1}].	Representation: Definition: curvature, κ (kappa) is the multiplicative inverse (reciprocal) of the radius, r, of the osculating curve: $\kappa = 1/r$ $\kappa = 0$ equals a straight line; $\kappa < 0$ equals a clockwise curve (right); $\kappa > 0$ equals a counter-clockwise curve (left).

12.3.2 Object setting message – OSEM (MsgID 0x0002)

The test object setting message is used for transferring the origin coordinates from the server to the test object. This message shall be received to interpret where in the world the trajectory is placed.

It is also used for sending the local geofencing (maximum allowed longitudinal and lateral path deviation) information to the test object.

The DeviceID is received from the CC as part of the OSEM. The DeviceID shall be used as the TransmitterID in the header of all messages sent from the test object to the CC or other receiver.

All DeviceIDs in the ID-struct shall be unique numbers.

When associating foreign DeviceIDs and test object names the valueIDs of PairingCount, ParingID and PairingName shall be used. A foreign DeviceID shall not have the same DeviceID value as the intended value of the receiver of this message, see DeviceID in Table 14. This feature gives the opportunity to use ReceiverID in the message header for sending data to a specific test object.

This message shall only be sent during test object state INIT or DISARMED. When this message is received and before the data content of it is saved to the test object shall: already saved trajectories, settings for adaptive synchronization points, trigger and actions and geofences be deleted from the test object.

The OSEM is further detailed in Tables 13, 14, 15, 16, 17, 18 and 19.

Table 13 — ValueID table for OSEM, (M) is mandatory, (O) is optional

Adherence	ValueID (2 bytes)	Content length (2 bytes)	Data
M	0x0020	0x000C (12 bytes)	**[ID-struct]** See Table 14.
M	0x0021	0x0013 (19 bytes)	**[Origin-struct]** See Table 15.

Adherence	ValueID (2 bytes)	Content length (2 bytes)	Data
M	0x0022	0x000B (11 bytes)	**[DateTime-struct]** See Table 16.
M	0x0023	0x0012 (18 bytes)	**[AccReqMode-struct]** See Table 17.
O	0x0024	0x0006 (6 bytes)	**[TimeServer-struct]** See Table 18.
O	0x0025	0x0004 (4 bytes)	**PairingCount** Number of IDs to be associated with a test object name. See Table 19.
O	0x0026	0x0004 (4 bytes)	**ParingID** DeviceID to be associated with a test object name. See Table 19.
O	0x0027	ASCII bytes (Variable length)	**PairingName** Test object name to be associated with a DeviceID. See Table 19.

Table 14 — ID-struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition

1	[DeviceID]	0x0004
		[Uint32]
			DeviceID to be used by the test object as TransmitterID in the header.	Representation:
				`DeviceID::=INTEGER { }`
				`(1..4294967295)`

2	[SubDeviceID]	0x0004
		[Uint32]
			SubDeviceID used locally by the test object.	Representation:
				`SubDeviceID::=INTEGER { }`
				`(0 when not used`
				`1..4294967295)`

3	[SystemControl CentreID]	0x0004
		[Uint32]
			SystemControlCentreID used locally by the test object.	Representation:
				`SystemControlCentreID::=INTEGER { }`
				`(1..4294967295)`

Table 15 — Origin-struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
1	[Latitude]	0x0006 [Int48]	Origin given as latitude. Absolute geographical latitude in the plate-drift-free coordinate system according to Table 2.	Representation: `Latitude ::= INTEGER {oneNanodegreeNorth (10), oneNanodegreeSouth (-10), unavailable(-140737488355328) } (-900000000000..9000000000001)` Positive values are used for latitude in north of the equator, negative values are used for latitude in south of the equator. Unit: 0,1 nanodegrees
2	[Longitude]	0x0006 [Int48]	Origin given as longitude. Absolute geographical longitude in the plate-drift-free coordinate system according to Table 2.	Representation: `Longitude ::= INTEGER {oneNanodegreeEast (10), oneNanodegreeWest (-10), unavailable(-140737488355328) } (-1800000000000..1800000000001)` Negative values are used for longitudes to the west, positive values are used for longitudes to the east. Unit: 0,1 nanodegrees
3	[Altitude]	0x0004 [Int32]	Origin given as altitude. Altitude in a plate-drift-free coordinate system according to Table 2.	Representation: `AltitudeValue ::= INTEGER {referenceSurface(0), oneCentimeter(1), unavailable(-2147483648) } (-100000..800001)` For altitude equal or greater than 8 000 m, the value shall be set to 800 000. For altitude equal or less than -1000 m, the value shall be set to -100 000. Unit: 0,01 m
4	[Rotation]	0x0002 [Uint16]	Rotation of PGRS compared to global coordinate system around the z-axis.	Representation: `Rotation::=INTEGER {eastYdirection(0), northYdirection(9000), westYdirection(18000), southYdirection(27000), unavailable(65535)} (0..36001)}` Definition: rotation is measured clockwise around the z-axis from geographical north to the PGRS y-axis. Unit: 0,01°
5	[CoordinateSystem]	0x0001 [Uint8]	Applied coordinate system	Representation: `BaseSystem::=INTEGER {ETRS89(0), NAD83(1), ITRF2000(2), WGS84(3), LocalCoordinate(4), unavailable(255)} (0..254)}`

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
				Definition: the test area coordinate system.

Table 16 — DateTime-struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
1	[DateISO8601]	0x0004 [Uint32]	Current date expressed in ISO 8601-format (YYYYMMDD)	Representation: `Date::=INTEGER{unavailable(0), july1st2018(20180701)}` `(0.. 99991231)` – yyymmdd Current GPS-time/date expressed as *year month day* as described in the ISO 8601 series. Range: YYYY (0000-9999) MM (01-12) DD (01-31)

2	[GPSWeek]	0x0002 [Uint16]	Current GPS week	Representation:
				`GPSWeek::=INTEGER{ firstGPSweekIn1980(0), oneGPSWeek(1), unavailable(65535)}`
				`(0..10001)`
				Definition: Number of elapsed weeks since GPS started on January 6th, 1980.
				EXAMPLE: The week beginning on Sunday 1st of July 2018 equals to GPSWeek 2008
				Unit: 1 week

3	[GPSSecondOfWeek]	0x0004 [Uint32]	Current time (from start of GPS week)	Representation:
				`GPSSecondOfWeek::=INTEGER { startOfWeek(0), oneMilliSecAfterStartOfWeek(4), endOfWeek(2419199999), unavailable(4294967295)}`
				`(0…2419200000)`
				Unit: 0,25 ms (0,00025 sec)

4	[LeapSeconds]	0x0001 [Uint8]	Current time difference / number of leap seconds between GPS-, and UTC-time.	Representation: `leapSeconds ::= INTEGER { (1), (2), (255)} (0..255))` Definition: GPS Leap Seconds, difference between GPS and UTC Time Unit: 1 s NOTE 1 UTCtime = GPStime – LeapSeconds.

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
				NOTE 2 GPS is currently 18 s ahead of UTC (as per date 2021-06-01).

Table 17 — Accuracy requirements mode struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
1	[MaxWayDeviation]	0x0002 [Uint16]	Maximum allowed longitudinal deviation	Representation: `deviation ::= INTEGER { oneMilliMeter(1), lastvalue(65534), unavailable(65535)} (0..65535))))` Definition: maximum allowed position error of the test object relative to the planned path. Unit: 0,001 m
2	[MaxLateralDeviation]	0x0002 [Uint16]	Maximum allowed lateral deviation	Representation: `deviation ::= INTEGER { oneMilliMeter(1), lastvalue(65534), unavailable(65535)} (0..65535))` Definition: maximum allowed position error of the test object relative to the planned path measured perpendicular from the planned path direction. Unit: 0,001 m
3	[MaxYawDeviation]	0x0002 [Uint16]	Maximum allowed Yaw (Ψ) deviation	Representation: `YawValue::=INTEGER {positiveXdirection(0), positiveYdirection(9000), negativeXdirection(18000), negativeYdirection(27000), unavailable(65535)} (0..36001)}` Definition: maximum allowed angular deviation of the test object relative to the planned trajectory. Measured in degrees. Unit: 0,01°
4	[MaxPosError]	0x0002 [Uint16]	Maximum positioning error to allow execution of a test	Representation: `MaxPosError ::= INTEGER { oneCentiMeter(1), lastvalue(65534), unavailable(65535)} (0..65535)}` Definition: Maximum position error from positioning system during the complete test, comparable to SPS- or WAAS-quality from a GPS receiver.

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
				Unit: 0,01 m

5	[Communication Timeout]	0x0002 [Uint16]	Communication Timeout of the HEAB Message	Representation:
				`communicationTimeout ::= UNSIGENDINTEGER {`
				`tenMilliseconds(1),`
				`twentyMilliseconds(2),`
				`(655535)} (0..65535))))`
				HEAB timeout
				Unit: 10 ms

6	[TestMode]	0x0001 [Uint8]	Type of test mode to run	Representation:
				`TestMode ::= INTEGER {`
				`Preplanned(0),`
				`Online(1),`
				`Scenario(2),`
				`unavailable(255)} (0..255)`
				Definition: test mode in which the test object shall act:
				0 = offline pre-planned trajectories,
				1 = online planned trajectories,
				2 = scenario-description file.

7	[MonrRate]	0x0001 [Uint8]	MONR message rate	Representation:
				`MonrRate ::= INTEGER {`
				`Rate(1),…,`
				`Rate(100))} (1..255)`
				Definition: the transmission rate of the MONR message.
				Unit: 1 Hz

8	[Monr2Rate]	0x0001 [Uint8]	MONR2 message rate	Representation:
				`Monr2Rate ::= INTEGER {`
				`Rate(1),…,`
				`Rate(100))} (1..255)`
				Definition: the transmission rate of the MONR2 message.
				Unit: 1 Hz

9	[HeabRate]	0x0001 [Uint8]	Heartbeat message rate	Representation:
				`HeabRate ::= INTEGER {`
				`Rate(10),…,`
				`Rate(100))} (10..255)`
				Definition: the transmission rate of the HEAB message.
				Unit: 1 Hz

10	[MaxMessageLength]	0x0004 [Uint32]	The maximum allowed length of a message sent by a test object	Representation:
				`MaxMessageLength::=INTEGER { }`
				`(0..4294967295)`

Table 18 — TimeServer-struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition

1	[Timeserver IP]	0x0004 [Uint32]	Timeserver IP	Representation
				`IPAddress::=INTEGER { }`
				`(0 when not used`
				`1..4294967295)`
				Definition: IP address of dedicated time server.

2	[TimeServer Port]	0x0002 [U16]	Timeserver port	Representation
				`IPAddress::=INTEGER { }`
				`(0 when not used`
				`1..65535)`
				Definition: port used by dedicated time server.

Table 19 — ID Association

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	data	Data definition

1	0x0025 [PairingCount(n)]	0x0004 [Uint32]	IDs to be associated with a test object name	Representation:
				`PairingCount::=INTEGER { }`
				`(0..4294967295)`
				Definition: the total number of IDs to be associated with a test object name.

i+1	0x0026 [PairingID]	0x0004 [Uint32]	DeviceID to be associated with a test object name	Representation:
				`DeviceID::=INTEGER { }`
				`(1..4294967295)`
				Definition: foreign DeviceID to be associated with a test object name. *i* is a counter from 1 to *n*.

i+2	0x0027 [PairingName]	Variable length	Test object name to be associated with DeviceID	Representation: `PairingName::= UTF8String` Definition: the friendly/readable name of the test object defined in DIDX-file. Characters in the text string are following an eight-bit character representation according to ISO 8859-1 (Latin-1).
i++				Increment counter *i*
i+2	Next 0x0026 [PairingID]	"	"	"
i+3	Next 0x0027 [PairingName]	"	"	"
…	…	…	…	…

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	data	Data definition
n				

12.3.3 Object state change request message – OSTM (MsgID 0x0003)

The object state message is used by the server to request a specific state change of the test objects. The message can, for example, be used to request the test object to go from disarmed state to armed, and vice versa.

The OSTM is further detailed in Tables 20 and 21.

Table 20 — ValueID table for OSTM

ValueID name	ValueID value (Uint16)
[StateChangeRequest]	0x0064
[Vendor Specific] (range)	0xA000-0xAFFF

Table 21 — OSTM

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0064 [StateChangeRequest]	0x0001 [Uint8]	Object state change request-value	Representation: `objStateChangeReq::=INTEGER { init(1), arm(2), disarm(3), remoteControl(6)} (0..10)` Definition: state change request from server to test object. The available change commands correspond to the test object state diagram: 0x01 = INIT 0x02 = ARM 0x03 = DISARM 0x06 = REMOTE_CONTROL
2	0x0101 [ScenarioID]	0x0002 [Uint16]	Scenario identifier	Representation: `ScenarioId::=INTEGER { } (1..65535)` Definition: identifier of the scenario which shall be executed.

12.3.4 Start message -STRT (MsgID 0x0004)

The start message is sent from the CC to a test object to invoke the test object to the RUNNING-state. It is only allowed for the CC to call for this this message when the test object is in ARMED state. Upon receiving a start message in the armed state, the test object shall go to RUNNING state.

The CC shall start transmitting its status - `CCStatus(running)` - in the HEAB-message as soon as the STRT-message is sent.

When current time (GPS second of week) becomes equal to received starting time from the CC, the test object will start to travel according to its trajectory.

The STRT message is further detailed in Tables 22 and 23.

Table 22 — ValueID table for STRT

ValueID name	ValueID value (Uint16)	Size
[STRT-struct]	0x0002	8 bytes, see Table 23
[Vendor Specific] (range)	0xA000-0xAFFF	

Table 23 — STRT-struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
1	GPSSecondOfWeek	0x0004 [Uint32]	The time (from start of current GPS week) when test shall start	Representation: `GPSSecondOfWeek::=INTEGER {startOfWeek(0), oneMilliSecAfterStartOfWeek(4), endOfWeek(2419199999), unavailable(4294967295)} (0..2419200000)` Unit: 0,25 ms (0,00025 s)
2	GPSWeek	0x0002 [Uint16]	Current GPS week	Representation: `GPSWeek::=INTEGER{ firstGPSweekIn1980(0), oneGPSWeek(1), unavailable(65535)} (0..10001)` Definition: .umber of elapsed weeks since GPS started on January 6th, 1980. NOTE Send GPS-week so it can be determined if message is old. EXAMPLE The week beginning on Sunday 1st of July 2018 equals to GPSWeek 2008. Unit: 1 week
3	TrajID	0x0002 [Uint16]	The trajectory identifier to start with	Representation: `TrajID::=INTEGER {ID(0), … , ID(65534), unavailable(65535)} (0..65535)`

12.3.5 Heartbeat message – HEAB (MsgID 0x0005)

The heartbeat message is continuously sent from the CC to the test objects with a frequency of normally 100 Hz. The purpose of this message is to report the status of the server and to request an emergency abort or a normal stop. The CCStatus information shall reflect what is described in Figure 15. The current time is also included in the message for the test objects to keep track of when a heartbeat message was sent from the server.

The CCStatus signal serves two purposes: 1. To inform about the current CC status (init, ready, running) and 2. To send command to test object to perform a normal/emergency stop. This is explained in Table 26.

To reduce size of the HEAB-message, the data are sent as a struct.

The ValueID and content length shall always be included when sending any message (including struct-based content like HEAB). See 12.2.4 for details.

Vendor Specific ValueID:s are not allowed in HEAB-message.

The HEAB message is further detailed in Tables 24, 25 and 26.

Table 24 — Content table for HEAB

ValueID name	ValueID value (Uint16)	Size
[HEAB-struct]	0x0090	5 bytes, see Table 25

Table 25 — HEAB-struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
1	GPSSecondOfWeek	0x0004 [Uint32]	The time when message was transmitted from control centre	Representation: `GPSSecondOfWeek::=INTEGER { startOfWeek(0), oneMilliSecAfterStartOfWeek(4), endOfWeek(2419199999), unavailable(4294967295) } (0…2419200000)` Unit: 0,25 ms (0,00025 s)
2	CCStatus	0x0001 [Uint8]	Current state of the control centre	Representation: `CCStatus::=ENUMEREATED { init(0), ready(1), abort(2), testRunning(3), testDone(4), normalStop(5), unavailable(255) }`

Table 26 — Detailed explanation of ControlCentreStatus (CCStatus)

CCStatus	Command or information	Detailed description
0x00, init	Info	Control centre information: control centre during initializing/startup.
0x01, ready	Info	Control centre information: control centre status OK / ready to send or receive messages and data.
0x02, abort	Command	Command to test object: emergency, stop the test! Perform an immediate abort of the test execution. Control centre shall continue to send `abort(2)` until all test objects are safely stopped.

CCStatus	Command or information	Detailed description
0x03, testRunning	Info	Control centre information: the control centre shall start transmitting `CCStatus(running)` in the HEAB-message as soon as the STRT-message is sent.
0x04, testDone	Info	State information: the control centre shall send `CCStatus(testDone)` when the test scenario is fully executed and all test objects have performed their actions. It is recommended to wait until this state is entered before uploading large data such as log files.
0x05, normalStop	Command	Command to test object: perform a normal stop of the test object.

12.3.6 Monitor message – MONR (MsgID 0x0006)

The monitor message is continuously sent from each test object as unicast to the CC with the purpose of reporting important data as well as maintaining the connection between the test object and the CC.

The monitor message also contains the status of the test object, so that the CC can determine what state the test object resides in.

If the server does not receive a monitor message within a given interval, the connection or the test object is considered as faulty. The CC can therefore use this information in order to take the necessary precautions.

To reduce size of the MONR-message, the data are sent as a struct.

Vendor-specific ValueIDs are not allowed in a MONR-message.

The ValueID and content length shall always be included when sending any message (including struct-based content like MONR). See 12.2.4 for details.

The MONR message is further detailed in Tables 27 and 28.

Table 27 — Content table for MONR

ValueID name	ValueID value (Uint16)	Size
[MONR-struct]	0x0080	36 bytes, see Table 28

Table 28 — MONR-struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
1	GPSSecondOfWeek	0x0004 [Uint32]	GPS-time when the position was calculated in the test object	Representation: `GPSSecondOfWeek::=INTEGER {startOfWeek(0), oneMilliSecAfterStartOfWeek(4), endOfWeek(2419199999), unavailable(4294967295)} (0…2419199999)` Unit: 0,25 ms (0,00025 s)
2	X-Position	0x0004 [Int32]	Position in the X-direction	Representation: `DeltaPosition ::= INTEGER { oneMilliMeter (1),`

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition

			relative to the origin	`unavailable(-2147483648)}`
				`(-2147483648..2147483646)`
				Definition: this value defines the distance between position and origin in X-direction.
				Unit: 0,001 m

3	Y-Position	0x0004 [Int32]	Position in the Y-direction relative to the origin	Representation:
				`DeltaPosition ::= INTEGER { oneMilliMeter (1),`
				`unavailable(-2147483648)}`
				`(-2147483648..2147483646)`
				Definition: this value defines the distance between position and origin in Y-direction.
				Unit: 0,001 m

4	Z-Position	0x0004 [Int32]	Position in the Z-direction relative to the origin	Representation:
				`DeltaPosition ::= INTEGER { oneMilliMeter (1),`
				`unavailable(-2147483648)}`
				`(-2147483648..2147483646)`
				Definition: this value defines the distance between position and origin in Z-direction.
				Unit: 0,001 m

5	Yaw	0x0002 [Uint16]	Yaw of the test object	Representation:
				`YawValue::=INTEGER {positiveXdirection(0),`
				`positiveYdirection(9000),`
				`negativeXdirection(18000),`
				`negativeYdirection(27000),`
				`unavailable(65535)} (0..36001)`
				Definition: see 6.3.2.
				Unit: 0,01°

6	Pitch	0x0002 [Int16]	Pitch angle of the test object	Representation:
				`PitchValue::=INTEGER{`
				`up(-9000),`
				`horizontal(0),`
				`down(+9000),`
				`unavailable(-32768)}`
				`(-9000..9000)`
				Unit: 0,01°

7	Roll	0x0002 [Int16]	Roll angle of the test object	Representation:
				`RollValue::=INTEGER{`
				`rollLeft(-9000),`
				`horizontal(0),`
				`rollRight(+9000),`
				`unavailable(-32768)}`
				`(-18000..18000)`
				Unit: 0,01°

8	Longitudinal Speed	0x0002 [Int16]	Longitudinal speed (in forward direction)	Representation:
				`SpeedValue::=INTEGER {standstill(0),`
				`oneCentimeterPerSec(1),`
				`unavailable(-32768)`
				`} (-32768..32767)`

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
				Definition: see 6.3.2, x-axis. Unit: 0,01 m/s
9	Lateral Speed	0x0002 [Int16]	Lateral speed (perpendicular to the forward direction)	Representation: `SpeedValue::=INTEGER {standstill(0), oneCentimeterPerSec(1) , unavailable(-32768) } (-32768..32767)` Definition: see 6.3.2, y-axis. Unit: 0,01 m/s
10	Longitudinal Acceleration	0x0002 [Int16]	Longitudinal acceleration (in forward direction)	Representation: `LongitudinalAccelerationValue ::= INTEGER {milliMeterPerSecSquaredForward(1), milliMeterPerSecSquaredBackward(-1), unavailable(-32768)} (-32000 .. 32001)` Definition: see 6.3.2, x-axis. Unit: 0,001 m/s² (1 mm/s²) Range: -32 m/s² to +32m/s²
11	Lateral Acceleration	0x0002 [Int16]	Lateral acceleration (perpendicular to the forward direction)	Representation: `LateralAccelerationValue ::= INTEGER {milliMeterPerSecSquaredToRight(-1), milliMeterPerSecSquaredToLeft(1), unavailable(-32768)} (-32000 .. 32000)` Definition: see 6.3.2, y-axis. Unit: 0,001 m/s² (1 mm/s²) Range: -32 m/s² to +32m/s²
12	Drive Direction	0x0001 [Uint8]	Drive direction with respect to the forward direction	Representation: `DriveDirection ::= INTEGER {forward(0), backward (1), unavailable(255) }`
13	ObjectState	0x0001 [Uint8]	Current state of the test object	Representation: `ObjectState::= INTEGER { off(0), init(1), armed(2), disarmed(3), running(4), postrun(5), remoteControlled(6), aborting(7), unavailable(255) }` Definition: value reflects the current test object state as described in 9.2.1.
14	ReadyToArm	0x0001 [Uint8]	Ready to arm indicator	Representation: `ReadyToArm::= INTEGER { notReady(0),`

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
				readyToARM(1), notReadyNoTRAJ(2), notReadyNoOSEM(3), notReadyNotAtStartPos(4), unavailable(255) } Definition: flag to tell the CC if the test object is ready to enter armed state or not. Value readyToARM can only be sent when in DISMARMED-state. EXAMPLE: One pre-requisite can be that a trajectory file is present in test object.

15	ObjectErrorStatus	0x0001 [Uint8]	Error status of the test object, Bit field encoded	Representation:
				ObjectErrorStatus ::= SEQUENCE {
				abortRequest BOOLEAN (MSB)
				outSideGeoFence BOOLEAN
				poorPosAccuracy BOOLEAN
				lowBattery BOOLEAN
				errorActive BOOLEAN
				warning BOOLEAN
				fatalError BOOLEAN
				otherError BOOLEAN (LSB)
				}

16	ErrorCode	0x0002 [Uint16]	Test object error codes	Representation: ErrorCode::=INTEGER { } (0..65535) Definition: active error code on the test object as defined in the DIDX file.

12.3.7 Monitor message 2 – MONR2 (MsgID 0x0007)

The monitor 2 (MONR2) message is continuously sent from each test object to the CC with the purpose of reporting important data (including absolute position coordinates) as well as maintaining the connection between the test object and the CC.

The MONR2 message also contains the status of the test object, such that the CC can determine what state the test object resides in.

The MONR2 message is further detailed in Tables 29 and 30.

Table 29 — ValueID table for MONR2

ValueID name	ValueID value (Uint16)	Size
[MONR2 Struct]	0x0001	26 bytes, see Table 30
[Vendor Specific] (range)	0xA000-0xAFFF	

Table 30 — MONR2-struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
1	GPSSecondOfWeek	0x0004 [Uint32]	GPS-time when the position was calculated in the test object	Representation: `GPSSecondOfWeek::=INTEGER {startOfWeek(0), oneMilliSecAfterStartOfWeek(4), endOfWeek(2419199999), unavailable(4294967295)} (0…2419200000)` Unit: 0,25 ms (0,00025 s)
2	Latitude	0x0006 [Int48]	Absolute geographical latitude position of the test object.	Representation: `Latitude ::= INTEGER {oneNanodegreeNorth (10), oneNanodegreeSouth (-10), unavailable(-140737488355328) } (-900000000000..9000000000001)` Positive values are used for latitude in north of the equator, negative values are used for latitude in south of the equator. Unit: 0,1 nanodegrees
3	Longitude	0x0006 [Int48]	Absolute geographical longitude position of the test object	Representation: `Longitude ::= INTEGER {oneNanodegreeEast (10), oneNanodegreeWest (-10), unavailable(-140737488355328) } (-1800000000000..1800000000001)` Negative values are used for longitudes to the west, positive values are used for longitudes to the east. Unit: 0,1 nanodegrees
4	Altitude	0x0004 [Int32]	Absolute altitude	Representation: `AltitudeValue ::= INTEGER {referenceSurface(0), oneCentimeter(1), unavailable(-2147483648) } (-100000..800001)` For altitude equal or greater than 8 000 m, the value shall be set to 800 000. For altitude equal or less than -1000 m, the value shall be set to -100 000. Unit: 0,01 m
5	Heading	0x0002 [Uint16]	Heading of the test object	Representation: `HeadingValue::=INTEGER {northYdirection(0), eastYdirection(9000), southYdirection(18000), westYdirection(27000), unavailable(65535)} (0..36000)}` Unit: 0,01°
6	ObjectState	0x0001 [Uint8]	Test object state	Representation: `ObjectState::=ENUMERATED { off(0), init(1),`

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
				armed(2), disarmed(3), running(4), postrun(5), remoteControlled(6), aborting(7), unavailable(255) } Definition: value reflects the current test object state as described in 9.2.1.

7	ErrorStatus	0x0001 [Uint8]	Error status of the test object. Bit field encoded.	Representation:
				ObjectErrorStatus::= SEQUENCE {
				abortRequest BOOLEAN (MSB)
				outSideGeoFence BOOLEAN
				poorPosAccuracy BOOLEAN
				engineFault BOOLEAN
				batteryFault BOOLEAN
				otherObjError BOOLEAN
				adaptSyPoEnded BOOLEAN
				vendorSpecific2 BOOLEAN (LSB)
				}

8	ErrorCode	0x0002 [Uint16]	Object error codes	Representation: ErrorCode::=INTEGER { } (0..65535) Definition: active error code on the test object as defined in the DIDX file.

12.3.8 GPS second of week roll over message – SOWM (MsgID 0x0008)

This is the message to inform about current GPS week and date, and to inform about when GPS second of week roll over happens. The CC shall transmit the week-roll-over message to the test objects if a new GPS-week is entered during execution of a test.

The SOWM is further detailed in Tables 31 and 32.

Table 31 — ValueID table for SOWM

ValueID name	ValueID value (Uint16)
[DateISO8601]	0x0004
[GPSWeek]	0x0003
[Vendor Specific] (range)	0xA000-0xAFFF

Table 32 — SOWM

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0004 [DateISO8601]	0x0004 [Uint32]	[Uint32] Current date expressed in ISO8601-	Representation: DateISO8601::=INTEGER{ unavailable(4294967295), july1st2018(20180701)} (0.. 99991231) –yyymmdd

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
			format (YYYYMMDD)	Definition: current GPS-time/date expressed as `Year Month Day` as described in the ISO 8601 series. Range: YYYY (0000-9999) MM (01-12) DD (01-31)
2	0x0003 [GPSWeek]	0x0002 [Uint16]	Current GPS week	Representation: `GPSWeek::=INTEGER{ firstGPSweekIn1980(0), oneGPSWeek(1), unavailable(65535)} (0..10001)` Definition: number of elapsed weeks since GPS started on January 6th 1980. Example: The week beginning on Sunday 1st of July 2018 equals to GPSWeek 2008. Unit: 1 week

NOTE GPS week counting is normally done with 11 bit while in this document it uses 16 bit and hence an overflow is not expected within the next several hundreds of years.

12.3.9 Synchronization point configuration message – SYPM (MsgID 0x000B)

This is the message for configuring synchronization point information to slave test object.

The SYPM is further detailed in Tables 33 and 34.

Table 33 — ValueID table for SYPM

ValueID name	ValueID value (Uint16)
[SyncPointTime]	0x0001
[FreezeTime]	0x0002
[Vendor Specific] (range)	0xA000-0xAFFF

Table 34 — SYPM

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0001 [SyncPointTime]	0x0004 [Uint32]	Slave synchronization point defined by time along trajectory.	Representation: `RelativeTime::=INTEGER {oneMilliSeconds(1)} (0..4294967295)` Definition: time between start and synchronization point along the slave trajectory.

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
				Unit: 0,001 s
2	0x0002 [FreezeTime]	0x0004 [Uint32]	Time to stop synchronization before defined synchronization point.	Representation: `RelativeTime::=INTEGER {oneMilliSeconds(1)} (0..4294967295)` Definition: relative time from received MTSP time. When MTSP time is less than current time, the test object shall stop trying synchronization. (MTSP(time) – Current time) < Freeze time => Stop sync Unit: 0,001 s

12.3.10 Master time to synchronization point message -MTSP (MsgID 0x000C)

This is the message for sending remaining time to synchronization point.

The MTSP message is further detailed in Tables 35 and 36.

Table 35 — ValueID table for MTSP

ValueID name	ValueID value (Uint16)
[EstSyncPointTime]	0x0001
[Vendor Specific] (range)	0xA000-0xAFFF

Table 36 — MTSP

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0001 [EstSyncPointTime]	0x0004 [Uint32]	Estimated (absolute) time when the master will reach the synchronization point.	Representation: `GPSSecondOfWeek::=INTEGER {startOfWeek(0), oneMilliSecAfterStartOfWeek(4), endOfWeek(2419199999), unavailable(4294967295)} (0..2419200000)` Unit: 0,25 ms (0,00025 s)

12.3.11 Remote-control manoeuvring message – RCMM (MsgID 0x000A)

This is themessage for manually controlling the test object during safe speed conditions.

Values can be transmitted as absolute values (unit: 0,01m/s, and 0,01°) or relative actuator values (in percent, %).

The control centre shall only send either the pair of absolute values, or the combination of relative actuator values.

The typical sequence for RCCM is as follows.

1) CC sends object state change message (OSTM) with `StateChangeRequest` set to `remoteControl`.
2) Test object responds with populating the monitor message (MONR) with `ObjectState` set to `remoteControlled`.
3) CC start sending RCMM continuously. The first message shall contain value 0 as first value.
4) CC continues to send RCMM with a frequency of ~10Hz during the complete maneuver. The test object responds with `ObjectState` equals to `remoteControlled`.
5) Last speed and steering values to send before stopping the remote controlling shall be zeroes (0).
6) When test object is at a standstill the CC may send a new object state change message (OSTM) with `StateChangeRequest` set to `disarm`.
7) After test object has fully stopped/standstill, it may leave the remote-controlled state and return to disarmed if requested by the CC.

NOTE 1 The RCMM is only intended for test objects that normally travel in the longitudinal direction. Flying test objects are not considered here but handled with the RCMM2.

NOTE 2 For any other manually controllable parameter in the test object, vendor-specific ValueIDs can be used, such as stationary test objects like traffic lights and weather generators, or other.

The RCMM is further detailed in Tables 37 and 38.

Table 37 — ValueID table for RCMM

ValueID name	ValueID value (Uint16)
[Speed_absolute] (cm/sec)	0x0011
[Steering_absolute] (degree)	0x0012
[Throttle] (%)	0x0031
[Brake_friction] (%)	0x0032
[Longitudinal_direction]	0x0033
[Steering_relative] (%)	0x0034
[Vendor Specific] (range)	0xA000-0xAFFF

Table 38 — RCMM

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0011 [SpeedAbsolute]	0x0002 [Int16]	The requested speed value in longitudinal (x) direction of the test object	Representation: `SpeedValue::=INTEGER {standstill(0), oneCentimeterPerSec(1), unavailable(-32768) } (-32768..32767)` Absolute values Unit: 0,01 m/s
2	0x0012 [SteeringAbsolute]	0x0002 [Int16]	Requested steering/yaw value	Representation `SteeringYaw::=INTEGER{ right(-9000), straight(0), left(+9000),`

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition

				`unavailable(-32768)}`
				`(-9000..9000)`
				Definition: the requested value for the yaw steering angle between -90° and +90° around the z-axis.
				Absolute values
				Unit: 0,01°

3	0x0031 [ThrottleRelative]	0x0002 [UInt16]	The relative throttle position (%) as a power request to the propulsion system for longitudinal movement	Representation:
				`ThrottleRelative::=INTEGER{`
				`noRequest(0),`
				`maxThrottle(100),`
				`unavailable(65535)}`
				`(0..100)`
				Definition: direction of movement is achieved by combining throttle and longitudinal direction.
				Relative values
				Unit: Percent

4	0x0032 [BrakeRelative]	0x0002 [UInt16]	Requested relative brake force (%)	Representation:
				`BrakeRelative::=INTEGER{`
				`noRequest(0),`
				`maxBrake (100),`
				`unavailable(65535)}`
				`(0..100)`
				Definition: brake request has priority over throttle request, i.e. throttle requests shall be ignored when brake request is not equal to zero.
				Relative values
				Unit: Percent

5	0x0033 [DirectionRelative]	0x0001
		[Uint8]
			Requested direction for longitudinal movement (forward/reverse)	Representation:
				`DirectionRelative::=ENUMERATED`
				`{`
				`forward(0),`
				`reverse(1),`
				`unavailable(255)`
				`}`
				Definition: see 6.3.2, x-axis.

6	0x0034 [SteeringRelative]	0x0002 [Int16]	Requested steering/yaw value	Representation:
				`SteeringRelative::=INTEGER{`
				`maxRight(-100),`
				`straight(0),`
				`maxLeft(+100),`
				`unavailable(-32768)}`
				`(-100..100)`
				Definition: the requested steering value between maximum right value (-100 %) and maximum left +100 % .

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
				Relative values Unit: Percent

12.3.12 Remote-control manoeuvring message 2 – RCMM2 (MsgID 0x0016)

The difference between RCMM and RCMM2 is that the later one is designed to being able to manually control flying test objects and/or test objects with tank steering. The values will apply to the flying object attitude/posture. Figure 35 illustrates the possible remote-control commands for dimensional movement.

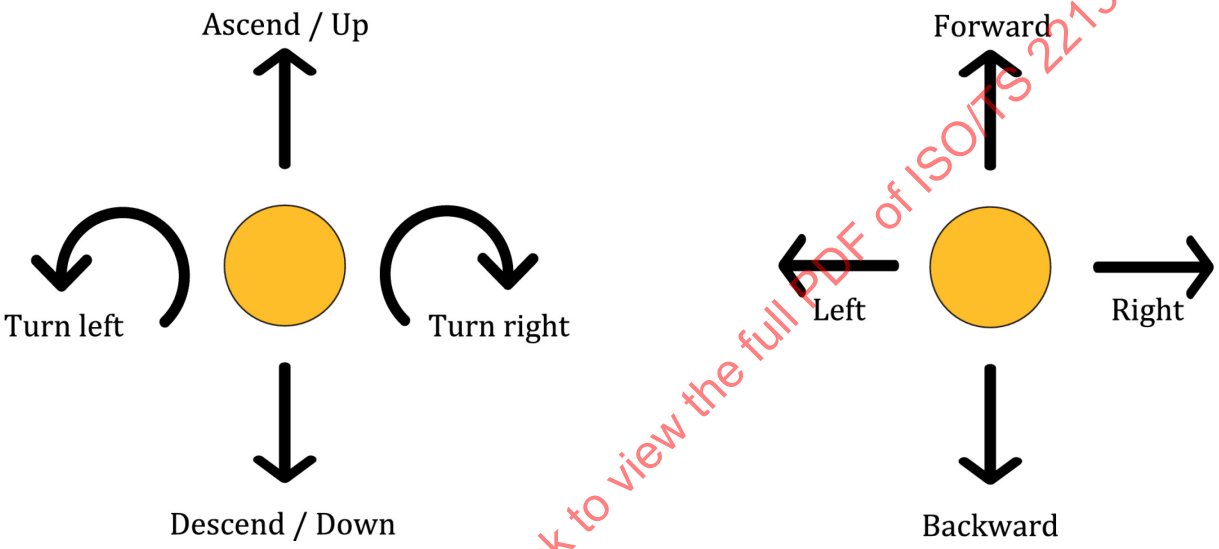


Figure 35 — Possible remote-control commands (in three dimensions)

The RCMM2 is further detailed in Tables 39, 40 and 41.

Table 39 — ValueID table for RCMM2

ValueID name	ValueID value (Uint16)	Size
Joystic_relative-struct	0x0001	8 bytes, see Table 40
Joystic_absolute-struct	0x0002	12 bytes, see Table 41
[Vendor Specific] (range)	0xA000-0xAFFF	

Table 40 — Joystick_relative-struct

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
1	fwdbwd	0x0002 [Int16]	The requested level for controlling the test object movement backward or forward	Representation: `fwdbwd::=INTEGER{ maxBackward(-100), standstill(0), maxForward(+100), unavailable(-32768)} (-100..100)`

Struct order number	Struct data name	Struct data length [and type]	Data	Data definition
				Unit: Percent
2	leftright	0x0002 [Int16]	The requested level for moving to the right and left	Representation: `leftright::=INTEGER{ maxRight(-100), standstill(0), maxLeft(+100), unavailable(-32768)} (-100..100)` Unit: Percent
3	ascdesc	0x0002 [Int16]	The requested level for ascending or descending	Representation: `ascdesc::=INTEGER{ maxDescend(-100), standstill(0), maxAscend(+100), unavailable(-32768)} (-100..100)` Unit: Percent
4	turn	0x0002 [Int16]	The requested level for turning/rotating to the right and left	Representation: `turn::=INTEGER{ maxTurnRight(-100), standstill(0), maxTurnLeft(+100), unavailable(-32768)} (-100..100)` Unit: Percent

Table 41 — Joystick_absolute-struct

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	[LongSpeed]	0x0002 [Int16]	The requested speed value in longitudinal (x) direction of the test object	Representation: `SpeedValue::=INTEGER {standstill(0), oneCentimeterPerSec(1), unavailable(-32768) } (-32768..32767)` Absolute values Unit: 0,01 m/s
2	[LatSpeed]	0x0002 [Int16]	The requested speed value in lateral (y) direction of the test object	Representation: `SpeedValue::=INTEGER {standstill(0), oneCentimeterPerSec(1), unavailable(-32768) } (-32768..32767)` Absolute values Unit: 0,01 m/s
3	[AltSpeed]	0x0002 [Int16]	The requested speed value in altitude (z) direction of the test object	Representation: `SpeedValue::=INTEGER {standstill(0),`

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition

				`oneCentimeterPerSec(1),`
				`unavailable(-32768)`
				`} (-32768..32767)`
				Absolute values
				Unit: 0,01 m/s

4	[Yaw]	0x0002 [Int16]	Requested steering/yaw value	Representation:
				`SteeringYaw::=INTEGER{`
				`right(-9000),`
				`straight(0),`
				`left(+9000),`
				`unavailable(-32768)}`
				`(-18000..18000)`
				Definition: the requested value for the yaw steering angle between -180° and +180° around the z-axis.
				Absolute values
				Unit: 0,01°

5	[Roll]	0x0002 [Int16]	Requested steering/roll value	Representation:
				`SteeringYaw::=INTEGER{`
				`right(-9000),`
				`straight(0),`
				`left(+9000),`
				`unavailable(-32768)}`
				`(-18000..18000)`
				Definition: the requested value for the yaw steering angle between -180° and +180° around the x-axis.
				Absolute values
				Unit: 0,01°

6	[Pitch]	0x0002 [Int16]	Requested steering/pitch value	Representation:
				`SteeringYaw::=INTEGER{`
				`right(-9000),`
				`straight(0),`
				`left(+9000),`
				`unavailable(-32768)}`
				`(-18000..18000)`
				Definition: The requested value for the yaw steering angle between -180° and +180° around the y-axis.
				Absolute values
				Unit: 0,01°

12.3.13 Trigger configuration message – TRCM (MsgID 0x0021)

This is the message to enable and configure the TriggerEvent in the trigger reporting test object.

The TRCM contains a number of ValueID:s that are used for setting up the trigger condition. If the TriggerType and the TriggerTypeParameters are not sufficient for configuring the trigger event, then it is appropriate to use vendor specific ValueID:s.

The TriggerID is an identifier for this TriggerEvent. This TriggerID is normally related to a ActionID in the control centre or in a test object. When the conditions are fulfilled is a trigger event occurred message (TREO) sent to the CC where it is handled.

The TRCM is further detailed in Tables 42 and 43.

Table 42 — ValueID table for TRCM

ValueID name	ValueID value (Uint16)
[TriggerID]	0x0001
[TriggerType]	0x0002
[TREOSendCount]	0x0003
[TREOSendInterval]	0x0004
[TriggerComparisonType]	0x0011
[TriggerParamSign]	0x0012
[TriggerParam1]	0x0013
[TriggerParam2]	0x0014
[TriggerParam3]	0x0015
[TriggerParam4]	0x0016
[TriggerParam5]	0x0017
[Vendor Specific] (range)	0xA000-0xAFFF

Table 43 — TRCM

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0001 [TriggerID]	0x0002 [Uint16]	Unique ID of the TriggerEvent	Representation: `TriggerID::=INTEGER{ unavailable(65535)} (0..65535)`
2	0x0002 [TriggerType]	0x0002 [Uint16]	The type of trigger	Representation: `TriggerType::=INTEGER{ unavailable(65535)} (0..65535)` See defined triggers in Table 45.
3	0x0003 [TREOSendCount]	0x0002 [Uint16]	The number of times to send TREO if the conditions are true	Representation: `TreoSendCount::=INTEGER{ once(1), }(1..65535)`
4	0x0004 [TREOSendInterval]	0x0002 [Uint16]	The time between each sending of TREO (ms)	Representation: `TreoSendInterval::=INTEGER{ unavailable(65535)} (1..65535)` Unit: ms
5	0x0011 [TriggerComparisonType]	0x0001 [Uint8]	Comparison method	See comparison Table 44.

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
6	0x0012 [TriggerParamSign]	0x0001 [Uint8]	The sign of the parameters	Representation: `TriggerParamSign::=INTEGER{ unsigned(1), signed(2), }(1..2)`

7	0x0013 [TriggerParam1]	0x0004 [Uint32/ Int32]	Trigger type parameter for the specific trigger type.	See Table 45 for definitions.
				Representation:
				`TriggerTypeParam::=UINT32_INTEGER{ unavailable(4294967295)} (0.. 4294967295)`
				`TriggerTypeParam::=INT32_INTEGER{ unavailable(2147483647)} (0.. +/- 2147483646)`
				Definition: Trigger Parameter (in the context of the specific TriggerType)

8	0x0014 [TriggerParam2]	0x0004 [Uint32/ Int32]	Trigger type parameter for the specific trigger type.	Representation: See content number 5.
9	0x0015 [TriggerParam3]	0x0004 [Uint32/ Int32]	Trigger type parameter for the specific trigger type.	Representation: See content number 5.
10	0x0016 [TriggerParam4]	0x0004 [Uint32/ Int32]	Trigger type parameter for the specific trigger type.	Representation: See content number 5
11	0x0017 [TriggerParam5]	0x0004 [Uint32/ Int32]	Trigger type parameter for the specific trigger type.	Representation: See content number 5.

Table 44 — Comparison codes

Name	==	>	>=	<	<=	!=	No comp
Value	0x60	0x61	0x62	0x63	0x64	0x65	0x66

Table 45 — Defined triggers

Name	Trigger type	Sign	Comparison	Param1 [a]	Param2	Param3	Param4	Param5
Speed threshold	0x0001	2	0x60-0x65	DeviceId	Speed value [cm/s]	True time [b] [ms]	Not used	Not used

Name	Trigger type	Sign	Comparison	Param1 [a]	Param2	Param3	Param4	Param5
Relative distance [c]	0x0002	2	0x60-0x65	DeviceId 1	DeviceId 2	Longitudinal distance [mm]	Lateral distance [mm]	True time [b] [ms]
Position reached	0x0003	2	0x60-0x64	DeviceId	X[mm]	Y[mm]	Z[mm]	Radius [mm]
Brake pressed	0x0004	1	0x60-0x65	DeviceId	Level	True time [b] [ms]	Not used	Not used
Acceleration occurred	0x0005	2	0x60-0x65	DeviceId	Level [cm/s^2]	True time [b] [ms]	Not used	Not used
Digital input event	0x0006	1	0x60	DeviceId	Digital pin number	Level (high/low)	True time [b] [ms]	Not used
Analog input level	0x0007	2	0x60-0x65	DeviceId	Input channel number	Level (mV)	True time [b] [ms]	Not used
Absolute time passed	0x0008	1	0x60-0x62	DeviceId	Time [gpssow]	Week [GpsWeek]	Not used	Not used
End of trajectory	0x0009	1	0x66	DeviceId	TrajectoryID	Not used	Not used	Not used
Time to collision[c]	0x000A	1	0x60-0x65	DeviceId1 (Master)	DeviceId2 (Slave)	Time threshold [ms]	Not used	Not used
Vendor specific	0x0100-0x0200	-	-	-	-	-	-	-

[a] DeviceId is used for an external process to check the trigger condition of a test object, for example, the CC checking triggers for the test object(s).

[b] The true time is the minimum elapsed time during when the comparison is true before TREO is fired.

[c] DeviceId1 is to relative DeviceId2, for example, the longitudinal distance between |DeviceId1 – DeviceId2| is ≤ Param3 value. If this trigger is used by a test object message gating shall be implemented and used by the test object and the CC.

12.3.14 Action configuration message – ACCM (MsgID 0x0022)

This is a message to enable and configure the ActionEvent in the action object. A TriggerID can result in multiple actions (ActionID) as long there is a link between the IDs created by this message.

The ACCM is further detailed in Tables 46 and 47.

Table 46 — ValueID table for ACCM

ValueID name	ValueID value (Uint16)
[TriggerID]	0x0002
[ActionID]	0x0003
[ActionType]	0x0004
[ActionParamSign]	0x0005
[ActionParam1]	0x00A1
[ActionParam2]	0x00A2

ValueID name	ValueID value (Uint16)
[ActionParam3]	0x00A3
[ActionParam4]	0x00A5
[ActionParam5]	0x00A6
[Vendor Specific] (range)	0xA000-0xAFFF

Table 47 — ACCM

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0002 [TriggerID]	0x0002 [Uint16]	Unique ID of the TriggerID to perform action when received in TREO	Representation: `ActionID::=INTEGER{ unavailable(65535)} (0..65535)`
2	0x0003 [ActionID]	0x0002 [Uint16]	Unique ID of the ActionEvent	Representation: `TriggerID::=INTEGER{ unavailable(65535)} (0..65535)`
3	0x0004 [ActionType]	0x0002 [Uint16]	The type of action	Representation: `ActionType::=INTEGER{ unavailable(65535)} (0..65535)` See defined actions in Table 48.
4	0x0005 [Action ParamSign]	0x0001 [Uint8]	The sign of the parameters	Representation: `ActionParamSign::=INTEGER{ unsigned(1), signed(2), }(1..2)`

5	0x00A1 [ActionParam1]	0x0004 [Uint32/ Int32]	Action parameter 1	Representation:
				`TriggerTypeParam::=UINT32_INTEGER{ unavailable(4294967295)} (0.. 4294967295)`
				
				`TriggerTypeParam::=INT32_INTEGER{ unavailable(2147483647)} (0.. +/- 2147483646)`

6	0x00A2 [ActionParam2]	0x0004 [Uint32/ Int32]	Action parameter 2	See content number 5.
7	0x00A3 [ActionParam3]	0x0004 [Uint32/ Int32]	Action parameter 3	See content number 5.
8	0x00A4 [ActionParam4]	0x0004 [Uint32/ Int32]	Action parameter 4	See content number 5.
9	0x00A5 [ActionParam5]	0x0004 [Uint32/ Int32]	Action parameter 5	See content number 5.

Table 48 — Defined actions

Action name	Action type	Sign	Param1	Param2	Param3	Param4	Param5
Digital output	0x0101	1	Output pin	Active level (high/low)	Active time [a] [ms]	Inactive level	Not used
Analog output	0x0102	2	Output channel	Active analog output level [mV]	Active time [a] [ms]	Inactive analog output level [mV]	Not used
Path offset	0x0103	2	X [mm]	Y [mm]	Z [mm]	Not used	Not used
Speed offset	0x0104	2	Longitudinal speed [cm/s]	Longitudinal acceleration [cm/s^2]	Absolute(1) /relative(2)	Not used	Not used
Change trajectory	0x0105	1	TrajectoryId to change to	Not used	Not used	Not used	Not used
Send STRT (action only allowed by the control centre)	0x0106	1	Time to start [gpssow]	Not used	Not used	Not used	Not used
Alter steering [b]	0x0107	1	Enable(1) /disable(2) value	Active time [a] [ms]	Enable(1) /disable(2) value	Not used	Not used
Alter velocity lock [c]	0x0108	1	Lock(2) /unlock(1) value	Active time [a] [ms]	Lock(2) /unlock(1) value	Not used	Not used
General action (vendor specific)	0x0200-0x0300	-	-	-	-	-	-

[a] Active time is the time duration of for how long the value of Param1 shall be used before the value of Param2 becomes used.

[b] Activates or deactivates the steering actuator of the steering system.

[c] Activates or deactivates the speed control of the steering system.

12.3.15 Trigger event occured message – TREO (MsgID 0x0023)

This is the message to inform that the trigger occurred. The TREO message is further detailed in Tables 49 and 50.

Table 49 — ValueID table for TREO

ValueID name	ValueID value (Uint16)
[TriggerID]	0x0001
[TriggerTimestamp]	0x0002
[Vendor Specific] (range)	0xA000-0xAFFF

Table 50 — TREO

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0001 [TriggerID]	0x0002 [Uint16]	ID of the TriggerEvent	Representation `TriggerID::=INTEGER{ unavailable(65535)} (0..65535)`
2	0x0002 [TriggerTimeStamp]	0x0004 [Uint32]	Time stamp when trigger condition was fulfilled	Representation: `GPSSecondOfWeek::=INTEGER {startOfWeek(0), oneMilliSecAfterStartOfWeek(4), endOfWeek(2419199999), unavailable(4294967295)} (0..2419200000)` Unit: 0,25 ms (0,00025 s)

12.3.16 Execute action message – EXAC (MsgID 0x0024)

This is the message to inform the action executor to execute the action defined in the ACCM.

The EXAC message is further detailed in Tables 51 and 52.

Table 51 — ValueID table for EXAC

ValueID name	ValueID value (Uint16)
[ActionID]	0x0002
[ExecuteTime]	0x0003
[Vendor Specific] (range)	0xA000-0xAFFF

Table 52 — EXAC

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0002 [ActionID]	0x0002 [Uint16]	ID of the *action.*	Representation `ActionID::=INTEGER{ unavailable(65535)} (0..65535)`
2	0x0003 [ExecuteTime]	0x0004 [Uint32]	Absolute time when action shall be executed (if delay wanted)	Representation: `GPSSecondOfWeek::=INTEGER {startOfWeek(0), oneMilliSecAfterStartOfWeek(4), endOfWeek(2419199999), unavailable(4294967295)} (0..2419200000)` Unit: 0,25 ms (0,00025 s) NOTE If ExecuteTime equals to 0 or `unavailable` then immediate execution of the action shall be performed.

12.3.17 Cancel or delete trigger and action – CADE (MsgID 0x0025)

This is the message to the involved test objects to cancel/abort or delete the trig and action function. Cancel/abort can be sent during ongoing test. Deletion of a trigger and action setting can only be done prior ongoing test.

The CADE message is further detailed in Tables 53 and 54.

Table 53 — ValueID table for CADE

ValueID name	ValueID value (Uint16)
[ActionCode]	0x0001
[TriggerID]	0x0002
[ActionID]	0x0003
[Vendor Specific] (range)	0xA000-0xAFFF

Table 54 — CADE

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0001 [ActionCode]	0x0001 [Uint8]	Cancel or deletion of trigger and action	Representation: `ActionCode::=INTEGER{ cancel(1), delete(2)} (0..255)`
2	0x0002 [TriggerID]	0x0002 [Uint16]	ID of the trigger	Representation: `TriggerID::=INTEGER{ unavailable(65535)} (0..65535)`
3	0x0003 [ActionID]	0x0002 [Uint16]	ID of the action	Representation: `ActionID::=INTEGER{ unavailable(65535)} (0..65535)`

12.3.18 Action performed message – APEM (MsgID 0x0026)

This is the message to inform that an action was executed.

The APEM is further detailed in Tables 55 and 56.

Table 55 — ValueID table for APEM

ValueID name	ValueID value (Uint16)
[ActionID]	0x0010
[ActionType]	0x0011
[ExecutionTime]	0x0012
[TriggerID]	0x0013
[Vendor Specific] (range)	0xA000-0xAFFF

Table 56 — APEM

Content number (Content *n*)	ValueID (2 bytes)	Content length (2 bytes)	Data	Data definition
1	0x0010 [ActionID]	0x0002 [Uint16]	ID of the action	Representation: `ActionID::=INTEGER{ unavailable(65535)} (0..65535)`
2	0x0011 [ActionType]	0x0002 [Uint16]	The type of action	Representation `ActionType::=INTEGER{ unavailable(65535)} (0..65535)`
3	0x0012 [ExecutionTime]	0x0004 [Uint32]	Absolute time when action was executed	Representation: `GPSSecondOfWeek::=INTEGER {startOfWeek(0), oneMilliSecAfterStartOfWeek(4), endOfWeek(2419199999), unavailable(4294967295)} (0..2419200000)` Unit: 0,25 ms (0,00025 s)
4	0x0013 [TriggerID]	0x0002 [Uint16]	The TriggerID that triggered the action	Representation: `TriggerID::=INTEGER{ unavailable(65535)} (0..65535)`

12.3.19 Discovery request DREQ (MsgID 0x0010)

The discovery request message shall be sent as UDP from the CC as broadcast or unicast on port 53242. This message contains no data. The message header contains the transmitter ID and receiver ID set to 0xFFFFFFFF respectively.

12.3.20 Discovery response DRES (MsgID 0x0011)

The discovery response message shall be sent as UDP from the test object to the CC when the test object receives a discovery request message. It shall be sent from port 53242 to the same IP address where the discovery request message came from (unicast).

When a test object enters initialization state, it shall continuously send a discovery response message as broadcast to the network on port 53242. The discovery response message shall be sent with a time interval of 30 s until connection is established with the control centre.

The encoding of all string in this message shall be ascii.

The DRES message is further detailed in Tables 57 and 58.

Table 57 — ValueID Table for DRES

ValueID name	ValueID value (Uint16)
Vendor	0x0090
Product Name	0x0091
FW Version	0x0092
Test Object name	0x0093
Test Object Type	0x0094
SubDevice ID	0x0095