ilmedia

TECHNISCHE UNIVERSITÄT
ILMENAU

Matthias Scheider

GREX – Software Guidance

Rev. 1.2009.11.19

Technische Universität Ilmenau
Ilmenau, 2011

URN: urn:nbn:de:gbv:ilm1-2011200360

Table of contents

1 Changelog

Version	Modifications
1.2009.11.19	VehicleID VID_SEAOTTER1 added
1.2009.10.21	PilotingData changed (ID of target added)
1.2009.09.08	MinSurfaceSpeed added to THAVehicleData wasIPC register update
1.2009.09.04	wasIPC update
1.2009.08.26	VehicleID VID_VORTEX1 added
1.2009.08.11	Initial version

2 Inter Process Communication

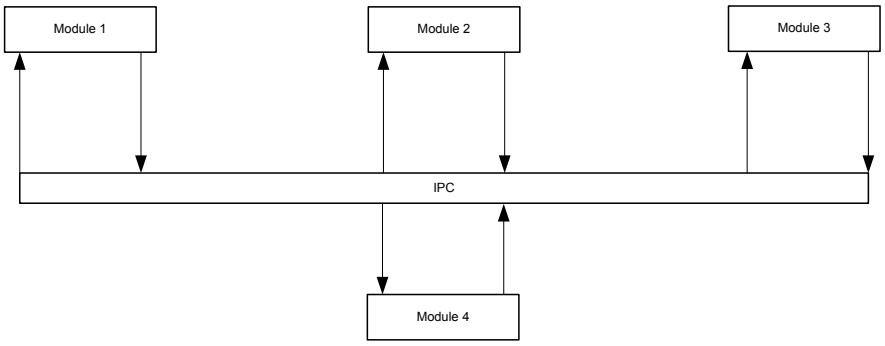

Figure 1 General IPC structure

The main goal of the IPC communication library is to enable different kinds of software modules to communicate with each other just by including the proposed **client** software library and by the use of a defined communication protocol, explained within this document. Hence the internal data structure of every module is completely independent and developer specific.

After running the IPC setup, you will find the following structure within the install folder:
{installFolder}\bin (IPC server executable and batch file)
{installFolder}\lib (static IPC client library package in release and debug)
{installFolder}\src (needed include files)

2.1 IPC Server

The pre-build (ready to run) IPC server allegorize the second part of the communication concept. Depending on the place of installation it has to be configured in the right way to ensure the correct network build-up. Due to the communication constraints and the static network structure every vehicle server has to have a **specified IP address** which depends on the current vehicle ID.

VehicleID's for GREX modules (enumeration VehicleID_E)

VID_SEAWOLF1 (0)
VID_ASTERIX1 (1)
VID_INFANTE1 (2)
VID_DELFIM1 (3)
VID_DELFIMX1 (4)
VID_SEABEE1 (5)
VID_VORTEX1 (6)
VID_SEAOTTER (7)
VID_SEETRACK_CONSOLE (9)
VID_AGUAS_VIVAS (10)

Build-up of VehicleIP for GREX servers

192.168.1.*'specified VehicleID_E'* (example for Seawulf: 192.168.1.0)

Running IPC server (start parameters)

Server.exe [server port] [vehicle id] [console logging]

2.2 IPC Client

The client library package has to be included by every module which wants to share data within the GREX network. This is done by using the proposed static IPC client libs. Thus any software module gets the basic functionality to send and receive data.

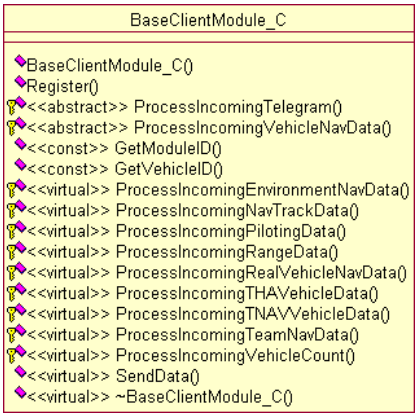

Figure 2 BaseClientModul

The BaseClientModule_C class provides the mentioned functionality and has to be implement by every client module.

Build a client module using provided libs

Within the wasIPC folder you will find two types of libs which can be used to realise a client module. Furthermore you have to include all header files of the src folder. Depending on your compiling options you can use dynamic linked libs (ClientLib.lib and ClientLibD.lib) or static linked libs (ClientLibMT.lib and ClientLibMTd.lib).

Build client module without using provided libs

Just include all files of the src folder into your client module project and build it on your own. This build-up is completely independent from the libs you will find within the lib folders.

Constructor method

BaseClientModule_C(bool printInfos=false)
printInfos – client console printout yes/no

Register method

Register(char* ipAddress, short port, unsigned char moduleID, unsigned int timeoutMS=1000, unsigned int retries=1)
ipAddress – string of server IP address
moduleID – module ID of client module
timeoutMS – register timeout in ms
retries – number of register retries which is handled internally

Public data methods

GetVehicleID()
returns the vehicle ID of the client module
GetModuleID()
returns the module ID of the client module

Send data method

SendData(const CGXParamsList &data, const CGXParamsList &callee)
data – data struct which should be send (build-up description in chapter 4)
callee – receiver address specification (build-up in chapter Sharing data using IPC network2.4)

Required receiver methods

ProcessIncomingTelegram(const CGXRemoteModule &modCaller, const CGXParamsList &params, CGXParamsList &result)
ProcessIncomingVehicleNavData(const CGXRemoteModule &modCaller, const CGXParamsList &params, CGXParamsList &result)
modCaller – includes vehicle and module ID of calling client
params – data which was transferred
result – result message (not used)

Optional receiver methods

ProcessIncomingTeamNavData()
ProcessIncomingTHAVehicleData()
ProcessIncomingTNAVVehicleData()
ProcessIncomingVehicleCount()
ProcessIncomingNavTrackData()
ProcessIncomingRangeData()
ProcessIncomingPilotingData()

Each module has to implement at least the required receiver methods. If it wants to receive other kinds of data, the related method must be implemented. The basic communication link build-up of the different modules should look like the following figure (Figure 3).

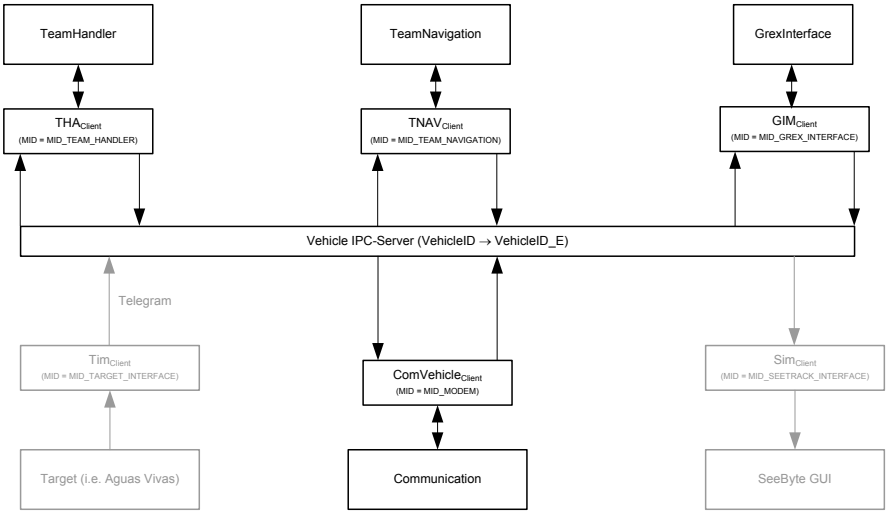

Figure 3 GREX Client-Server-Structure

To handle and identify different kinds of modules as well as to simplify the communication, every client has a fixed ModuleID. For now the following IDs are defined:

ModuleID's for GREX modules (enumeration ModuleID_E)

MID_TEAM_HANDLER
MID_TEAM_NAVIGATION
MID_GREX_INTERFACE
MID_MODEM
MID_SEETRACK_INTERFACE
MID_TARGET_INTERFACE

2.3 Communication data structures

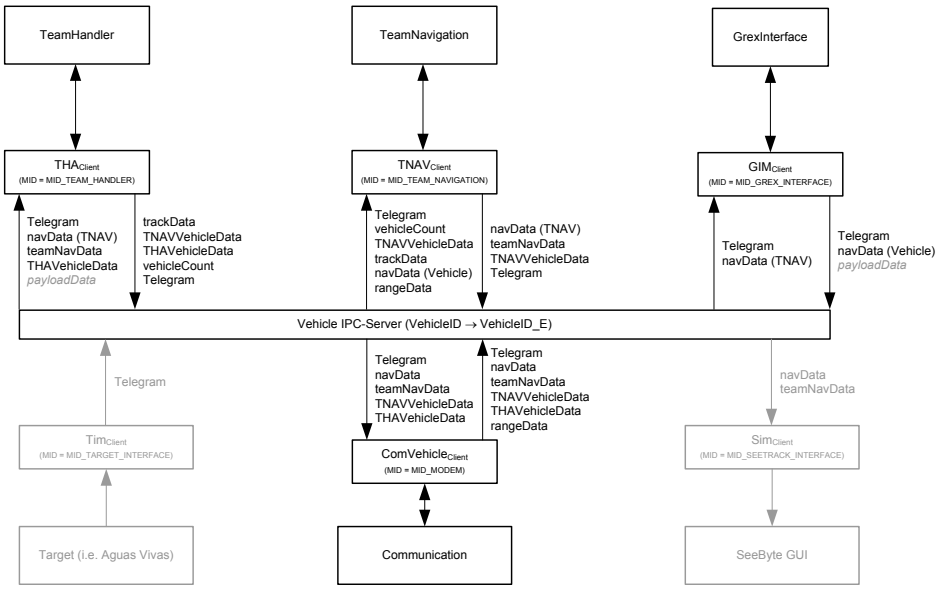

Figure 4 Data transfers within general GREX build-up

GREX modules are able to share predefined data structures like Figure 4 shows. Within this document the build-up of the different data structures used for communication between GREX modules and vehicles is shown. To send or read out data it is neccesary to use as well as understand the **CGXParamsList** data type! The following example shows the general way of use:

Set data

```
CGXParamsList paramsList;
paramsList.SetParamDouble( _T("Value"), double, true );
```

Get data

```
double value;
paramsList.GetParamDouble( _T("Value"), value );
```

Specials

Some data structs include more than one data package. To seperate the different data sets, every data set gets a sequential number, added to the value name of the *CGXParamsList* entry (e.g. 4.3). Use *GetParamsCount()* to calculate the amount of the data sets included in the *CGXParamsList*. The number of vehicles which take part in the mission is known. Some telegrams including a special count value to get the number of data in an easy way. This is commonly used within replanning telegrams.

2.4 Sharing data using IPC network

If you want to send data via IPC you need to define the callee (receiver module and data channel) using five parameters:

Build-up of callee

```
// set vehicle id (see VehicleID_E)
callee.SetParamUInt16( _T("VehicleID"), <VehicleID_E>, true );
// set destination module id (see ModuleID_E)
callee.SetParamUInt16( _T("ModuleID"), <ModuleID_E>, true );
// set destination object name (see object declaration)
callee.SetParamString( _T("IPC_Object"), <tstring>, true );
// set destination interface name (see object declaration)
callee.SetParamString( _T("IPC_Interface"), <tstring>, true );
// set destination methode name (see object declaration)
callee.SetParamString( _T("IPC_Method"), <tstring>, true );
```

Data channels

IPC_Object	IPC_Interface	IPC_Method	Data type
"ObjectInput"	"InterfaceNavigation"	"MethodVehicleNavigation"	vehicle navigation data (4.5)
		"MethodTeamNavigation"	team navigation data (4.6)
		"MethodPilotingNavigation"	piloting data (4.8)
	"InterfaceData"	"MethodTHAVehicleData"	THA vehicle data (4.1)
		"MethodTNAVVehicleData"	TNAV vehicle data (4.4)
		"MethodTrackData"	track data (4.3)
		"MethodVehicleCount"	vehicle count data (4.2)
		"MethodRangeData"	range data (4.7)
	"InterfaceCommands"	"MethodTelegram"	all kinds of telegram data (4.9)

3 GREX client modules realisation

To send or request data it is neccesary to use pre-defined abstract methods within the GREX IPC client module. These methods are the same on every module and have to be realised within every specific module development.

3.1 TeamHAndler

TeamHandler module performs the Multi System Control and is responsible for Mission Monitoring and Replanning. It will employ different algorithms according to the particular Multi Vehicle Primitive. The THA client module has to handle all kinds data as well as all kinds of telegrams which deal with (re-) planning of the mission.

3.1.1 THA Client communication methods

3.1.2 Supported data types

3.1.2.1 Incoming data

Data type	Definition
THAVehicleData	4.1
VehicleCount	4.2
TNAVVehicleData	4.4

3.1.2.2 Outgoing data

Data type	Definition
THAVehicleData	4.1
VehicleNavData	4.5
TeamNavData	4.6
PilotingData	4.8

3.1.3 Supported telegram types

3.1.3.1 Incoming Telegrams

All kinds…

3.1.3.2 Outgoing telegrams

All kinds…

3.2 TeamNAVigation

TNAV performs the position estimation of every vehicle in the current team based on several navigation data. The appropriate client module has to handle at least all kinds of navigation data.

3.2.1 TNAV Client communication methods

3.2.2 Supported data types

3.2.2.1 Incoming data

Data type	Definition
VehicleCount	4.2
NavTrackData	4.3
TNAVVehicleData	4.4

VehicleNavData	4.5
RangeData	4.7

3.2.2.2 Outgoing data

Data type	Definition
TNAVVehicleData	4.4
VehicleNavData	4.5
TeamNavData	4.6

3.2.3 Supported telegram types

3.2.3.1 Incoming Telegrams

Telegram type Expected answer telegram/data	Definition
TK_START_PREPARATION	4.9.13
TK_ABORT_MISSION	4.9.12
TK_OK	4.9.9
TK_FAIL	4.9.9
TK_REQUEST_DATA	4.9.1
Depending on the requested data	3.2.2.2

3.2.3.2 Outgoing telegrams

Telegram type Expected answer telegram/data	Definition
TK_OK	4.9.9
TK_FAIL	4.9.9
TK_REQUEST_DATA	4.9.1
Depending on the requested data	3.2.3.1

3.3 GrexInterfaceModule

This module represents the interface between the existing vehicle hardware and the GREX hard-/software. It has to handle (input/output) the navigation data structure (4.5). This data has to be send to the TNAV module periodically every 50ms till 500ms. If the vehicle is able/want to update the position based on the estimated value of the TNAV value, it can process the incoming VehicleNavData. The second duty of this module is to perform the (re-)planning commands of the THA module.

3.3.1 Mission starting procedure

3.3.1.1 Important notes

a) In all GREX mission plans (TMP and SVMP), there will be two ways to express times: As **relative** time, expressed in seconds (meaning that a certain action shall happen after a certain period); and as **absolute** time, expressed in seconds after January 01st of 1970 (meaning that a certain action shall happen at a certain absolute time).
At the moment there is no other way two deal with time relevant problems: The fact that absolute times are not know during mission planning results in the need for relative times (e.g. the vehicle shall reach a certain point and then wait there for 20 seconds, because a sensor need to be employed for this time period). During mission execution, the delay of acoustic communication disables the possibilities for the use of relative times. As the communication delay is not predictable, it is not possible to command another vehicle to “start in 30 seconds”. The command must be like: “Start at time 1.151.496.000 (which should be a real time in 2008).
Of course it must be guaranteed that there is no confusion whether a time is absolute or relative. As both times are stored in the same format and it can be assumed, at least for the current situation, that a GREX mission will not take longer than one day (86.400 seconds), the following agreement is made:

Any time up to 86.400 seconds is a relative time.
Any time greater than 86.400 seconds is absolute time.

b) There are **two** ways to make the vehicles stop and wait: One possibility is the usage of a #POINT- manoeuvre in the SVMP, where the parameter *WayPointType* is set to 4 (loiter). This results in a stop of the vehicle. The waiting time (another parameter of #POINT) is expressed either in absolute or relative method (see above), meaning that the vehicle shall continue the movement after the time period is over (for relative times) or they shall continue at the specified time (for absolute times). Of course, this Waiting Time can be changed by a replanning command.
The second possibility to make the vehicles stop is a 'Stop and Keep Position'- command from the Team Handler (both own module or the module of another vehicle). This command telegram contains a time stamp which has to be ignored – the order must be obeyed immediately. The vehicle has to wait until it receives a 'Continue'- order telegram from a Team Handler. This telegram always contains an absolute time stamp, stating that the vehicle is allowed to continue at this time. If the time is already in the past, the vehicle may continue immediately.

3.3.1.2 Steps to start a GREX mission

At first, mission planning is performed, all times are expressed as relative times (because during planning, the absolute times are still unknown). The first real primitive should be a M_Init, usually followed by a M_GoToFormation to establish a formation. That means, the definition of the maximum mission time within the line #MISSIONCONSTRAINTS is relative! All keep position manoeuvres (#POINT primitive with *WayPointType* = 4) with unclear waiting times (e.g. after a M_GoToFormation or for waiting vehicles in M_HierarchicTriggeredActivities) include the defined maximum mission time as waiting time. Then, the plans are translated, checked and finally uploaded to the vehicles. After the vehicles are put into the water and are switched to automatic mode, the following actions happen:

GREX- HW	**Vehicle- HW**
1. TMP and SVMP are received from console 2. Telegram TK_START_PREPARATION (4.9.13) including the name of the valid TMP is sent from console to all THA. THA will forward this telegram to its local GIM, including the name of the current SVMP (note: this step will not happen if the mission plan was uploaded before THA starts)	• No action • Just wait and send VehicleNavData (4.5) to TNAV • Perform requested replannings • Send requested data to THA
3. Communication handshaking 4. THA replans first #POINT primitive (waiting manoeuvre due to *WaypointType* = *4*), the current placeholder position is replaced by real position 5. THA requests current heading and current hover center position of the vehicle (hover center should be the current waiting position)	
6. THA sends TK_START_MISSION (4.9.12) telegram to GIM	
7. Each THA determines whether it is leader according to the priority list of the following GoToFormation primitive in TMP. 8. Each slave THA sends TK_READY_FOR_ACTION telegram to leader THA until leader vehicle answered 9. Leader waits for all TK_READY_FOR_ACTION telegrams	• Execution of Mission Plan starts • Vehicle waits in first keep position-manoeuvre • perform requested replannings
10. Leading THA sends TK_START_MISSION telegram to all other THAs including the new mission starting time	
11. All vehicles replan all relative waiting times to absolute ones, based on the new mission starting time 12. Leading THA runs the algorithm for the following GoToFormation and gets a list of lines and arcs for each vehicle 13. Leading THA creates appropriate replanning commands and sends them to other THAs as well as (step by step) to its own own GIM using TK_CHANGE_PARAMETERS_OF_PRIMITIVE_SVMP (4.9.3)	
14. After leading THA got positive feedback from every THA, it defines a new short waiting time (depends on the amount of vehicles take part on the current mission) and send that value to the other THAs to start the mission simultaneously on every vehicle by reducing the waiting time of the current waiting manoeuvre	
the mission begins	

3.3.2 Supported data types

3.3.2.1 Incoming data

Data type	Definition
VehicleNavData	4.5

3.3.2.2 Outgoing data

Data type	Definition
VehicleNavData	4.5

3.3.3 Supported telegram types

3.3.3.1 Incoming Telegrams

Telegram type Expected answer telegram/data	Definition
TK_START_PREPARATION	4.9.13
TK_START_MISSION	4.9.12
TK_ABORT_MISSION	4.9.12
TK_OK	4.9.9
TK_FAIL	4.9.9
TK_CHANGE_PARAMETERS_OF_PRIMITIVE_SVMP	4.9.3
TK_CHANGE_PARAMETER_OF_PRIMITIVE_SVMP	4.9.2
TK_CHANGE_PARAMETER_OF_SUBMANOEUVRE_SVMP	4.9.2
TK_JUMP_AFTERWARDS_TO_PRIMITIVE_SVMP	4.9.4
TK_JUMP_IMMEDIATELY_TO_PRIMITIVE_SVMP	4.9.4
TK_STOP_AND_KEEP_POSITION	4.9.12
TK_CONTINUE	4.9.12
TK_MODIFICATION_OF_MISSIONPLAN_ACCOMPLISHED	4.9.6
TK_MODIFICATION_OF_MISSIONPLAN_DENIED	4.9.6
TK_GIVE_REPORT_ON_INTERNAL_STATE	4.9.13
TK_REPORT_ON_INTERNAL_STATE	4.9.8
TK_REQUEST_DATA	4.9.1
Depending on the requested data	3.3.2.2

3.3.3.2 Outgoing telegrams

Telegram type Expected answer telegram/data	Definition
TK_MODIFICATION_OF_MISSIONPLAN_ACCOMPLISHED	4.9.6
TK_MODIFICATION_OF_MISSIONPLAN_DENIED	4.9.6
TK_REPORT_ON_STARTING_OF_A_PRIMITIVE_SVMP	4.9.7
TK_REPORT_ON_FINISHING_OF_A_PRIMITIVE_SVMP	4.9.7
TK_REPORT_ON_STARTING_OF_A_SUBMANOEUVRE_SVMP	4.9.7
TK_REPORT_ON_ADVANCE_OF_A_PRIMITIVE_SVMP	4.9.8
TK_REPORT_ON_INTERNAL_STATE	4.9.8
TK_EMERGENCY_MESSAGE_MISSIONPLAN_SKIP	4.9.10
TK_EMERGENCY_MESSAGE_SENSOR_FAILURE	4.9.11
TK_EMERGENCY_MESSAGE_PRIMITIVE_SKIP	4.9.11
TK_OK	4.9.9
TK_FAIL	4.9.9
TK_REQUEST_DATA	4.9.1
Depending on the requested data	3.3.2.1

3.4 COmmunicationModule

This module is responsible for all kinds of inter vehicle communication using acoustics, radios or other kinds of hardware.

3.4.1 Supported data types

3.4.1.1 Incoming data

Data type	Definition
THAVehicleData	4.1
TNAVVehicleData	4.4
VehicleNavData	4.5
TeamNavData	4.6

3.4.1.2 Outgoing data

Data type	Definition
THAVehicleData	4.1
TNAVVehicleData	4.4
VehicleNavData	4.5
TeamNavData	4.6
RangeData	4.7

3.4.2 Supported telegram types

3.4.2.1 Incoming telegrams

All kinds…

3.4.2.2 Outgoing telegrams

All kinds…

3.5 SeabyteInterfaceModule

This module represents the interface between the vehicles and the graphical user interface running on a seperate SeeTrack computer. Due to the structure of communication this module will receive the navigation data of all vehicles take part on a mission periodically. Hence it is able to display the mission progress.

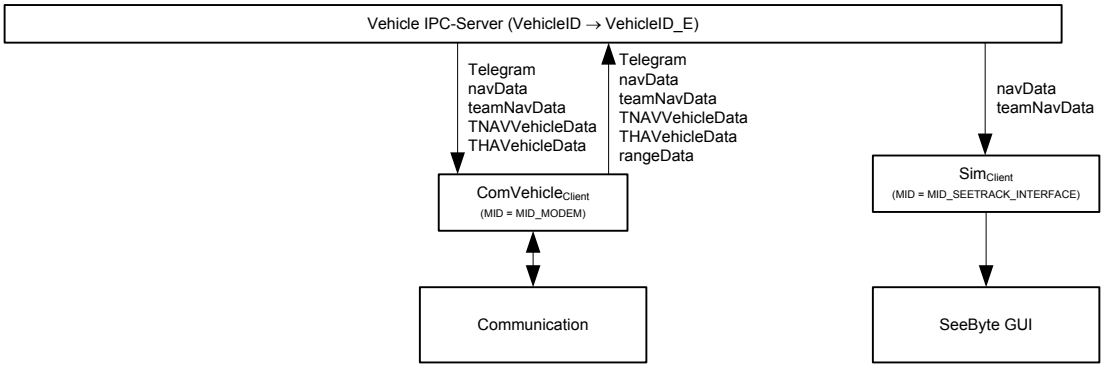

Figure 5 Structure of GREX console and interface

3.5.1 Supported data types

3.5.1.1 Incoming data

Data type	Definition
VehicleNavData	4.5
TeamNavData	4.6

3.5.1.2 Outgoing telegrams

None…

3.5.2 Supported telegram types

3.5.2.1 Incoming telegrams

None…

3.5.2.2 Outgoing telegrams

None…

3.6 TargetInterfaceModule

This module represents the interface for some kind of target, which the team should follow during the mission. It is not necessary that this module is able to receive any kind of GREX messages, it just send its possition periodically.

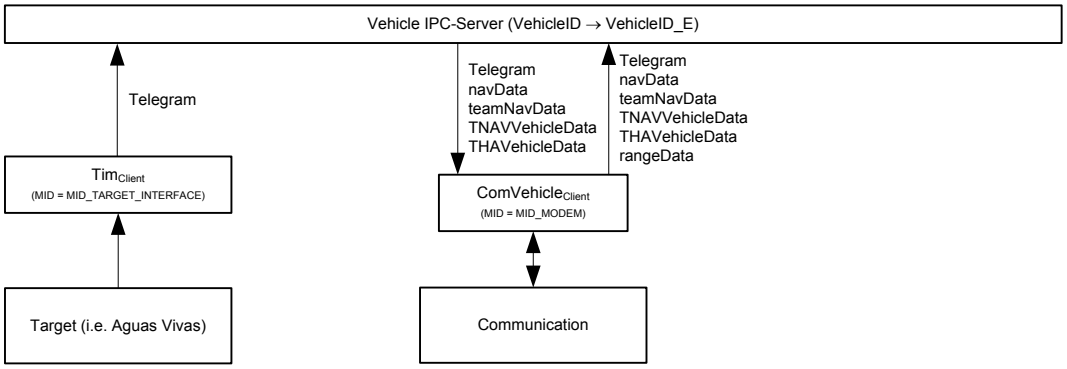

Figure 6 Target interface structure

3.6.1 Supported data types

3.6.1.1 Incoming data

None…

3.6.1.2 Outgoing telegrams

Data type	Definition
PilotingData	4.8

3.6.2 Supported telegram types

3.6.2.1 Incoming telegrams

None…

3.6.2.2 Outgoing telegrams

None…

4 Definition of communication language

In this chapter, the data structs, used for communication between the GREX modules, are defined. It is absolutely clear, that every module (different developers) can use its own data types, so the definitions are made in a general way – the way how to store the data for the different messages in a *CGXParamsList*, used for the communication via IPC. You will find small comments, the store command call and the data type for each single value.
Please pay attention to some values, that have to be send in a standardised way:

- Navigation data in **radian** with a depth in **meter**
- Time stamps are in the format **[seconds].[milliseconds]** since January 1st 1970
- The standard velocity unit is **m/s**
- Any depth or distance value is in **meter**

4.1 THAVehicleData

Description

Contains a parameter set of one vehicle, used by THA to execute mission replanning. This package will only be transferred if it is requested by another THA module and before the mission starts. The vehicle ID of sending vehicle can be obtained through IPC caller.

Build

```
// set arc ability
paramsList.SetParamBool( _T("ArcAbility"), <bool>, true );
// set hover mode (specified in HoverMode_E)
paramsList.SetParamUInt16( _T("HoverMode "), <unit16_t>, true );
// set arc radius for hover manoeuvre
paramsList.SetParamDouble( _T("HoverArcRadius"), <bool>, true );
// set surface max speed
paramsList.SetParamDouble( _T("SurfaceMaxSpeed"), <double>, true );
// set surface min speed
paramsList.SetParamDouble( _T("SurfaceMinSpeed"), <double>, true );
// set surface cruise speed
paramsList.SetParamDouble( _T("SurfaceCruiseSpeed"), <double>, true );
// set positive acceleration
paramsList.SetParamDouble( _T("SurfacePosAccel"), <double>, true );
// set surface negative acceleration
paramsList.SetParamDouble( _T("SurfaceNegAccel"), <double>, true );
// set surface heading rate
paramsList.SetParamDouble( _T("SurfaceHeadingRate"), <double>, true );
// set underwater maximun depth
paramsList.SetParamDouble( _T("UnderwaterMaxDepth"), <double>, true );
// set underwater maximum speed
paramsList.SetParamDouble( _T("UnderwaterMaxSpeed"), <double>, true );
// set underwater minimum speed
paramsList.SetParamDouble( _T("UnderwaterMinSpeed"), <double>, true );
// set underwater cruise speed
paramsList.SetParamDouble( _T("UnderwaterCruiseSpeed"), <double>, true );
// set underwater positive acceleration
paramsList.SetParamDouble( _T("UnderwaterPosAccel"), <double>, true );
// set underwater negative acceleration
paramsList.SetParamDouble( _T("UnderwaterNegAccel"), <double>, true );
// set underwater heading rate
paramsList.SetParamDouble( _T("UnderwaterHeadingRate"), <double>, true );
// set underwater surfacing speed
paramsList.SetParamDouble( _T("UnderwaterSurfSpeed"), <double>, true );
// set underwater descending speed
paramsList.SetParamDouble( _T("UnderwaterDescSpeed"), <double>, true );
```

```
// set underwater maximum positiv pitch
paramsList.SetParamDouble( _T("UnderwaterMaxPosPitch"), <double>, true );
// set underwater maximum negative pitch
paramsList.SetParamDouble( _T("UnderwaterMaxNegPitch"), <double>, true );
```

4.2 VehicleCount

Description

Contains the number of vehicles (*paramsList.GetParamsCount()*) and the IDs of all vehicles take part on the mission, separated by the value *tNumber*. This data is required for TNAV initialisation.

Build

```
// set vehicle id(s) (see VehicleID_E)
paramsList.SetParamUInt16( _T("VehicleID"+tNumber), <VehicleID_E>, true );
```

4.3 NavTrackData

Description

Contains the information of the current track, expected velocity and depth of each vehicle, separated by the value *tNumber*. This data will be transferred from THA to TNAV every time step (after THA has received a new team navigation data set). The value *tNumber* depends on the known number of vehicles take part on the current mission.

Build

```
// set time of nav data mesasurement [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"+tNumber), <double>, true );
// set vehicle id (see VehicleID_E)
paramsList.SetParamUInt16( _T("VehicleID"+tNumber), <VehicleID_E>, true );
// set radius [m] – positive value means clockwise direction, negative means counter clockwise
// this value is 0 if the track is a straight line
paramsList.SetParamDouble( _T("TrackRadius"+tNumber), <double>, true );
// set expected velocity in x direction [m/s] depending on current track
paramsList.SetParamDouble( _T("ExpectedVelX"+tNumber), <double>, true );
// set expected velocity in y direction [m/s] depending on current track
paramsList.SetParamDouble( _T("ExpectedVelY"+tNumber), <double>, true );
// set defined depth [m] of current mission element
paramsList.SetParamDouble( _T("ExpectedDepth"+tNumber), <double>, true );
```

4.4 TNAVVehicleData

Description

It contains specified data sets of each vehicle used by TNAV for initialisation. The Data can be separated by the value *tNumber*. The value *tNumber* depends on the known number of vehicles take part on the current mission.

Build

```
// set vehicle id (see VehicleID_E)
paramsList.SetParamUInt16( _T("VehicleID"+tNumber), <VehicleID_E>, true );
// set drift [m/s]
paramsList.SetParamDouble( _T("Drift"+tNumber), <double>, true );
// set uncertainty in horizontal position estimation
paramsList.SetParamDouble( _T("PosErrRMS"+tNumber), <double>, true );
// set uncertainty in depth estimation
paramsList.SetParamDouble( _T("DepthErrRMS"+tNumber), <double>, true );
// set uncertainty in velocity estimation
paramsList.SetParamDouble( _T("VelocityErrRMS"+tNumber), <double>, true );
// set uncertainty in range measurement
```

```
paramsList.SetParamDouble( _T("RangeErrRMS"+tNumber), <double>, true );
```

4.5 VehicleNavData

Description

Contains the navigation data of one vehicle. Vehicle ID of sending vehicle can be obtained through IPC caller.

Build

```
// set uncertainty value
paramsList.SetParamDouble( _T("UncertaintyValue"), <double>, true);
// set time of measurement [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
// set geodetic position [radian] and depth [m]
paramsList.SetParamDouble( _T("GeoPositionLat"), <double>, true);
paramsList.SetParamDouble( _T("GeoPositionLon"), <double>, true);
paramsList.SetParamDouble( _T("GeoPositionZ"), <double>, true);
// set velocity over ground (earth fixed) [m/s]
paramsList.SetParamDouble( _T("VelocityOGEfX"), <double>, true);
paramsList.SetParamDouble( _T("VelocityOGEfY"), <double>, true);
paramsList.SetParamDouble( _T("VelocityOGEfZ"), <double>, true);
// set current heading [RAD]
paramsList.SetParamDouble( _T("Heading"), <double>, true);
```

4.6 TeamNavData

Description

Contains the navigation data of the whole team. Same build-up like VehicleNavData (4.5) with numeration. The value *tNumber* depends on the known number of vehicles take part on the current mission.

Build

```
// set vehicle id (see VehicleID_E)
paramsList.SetParamUInt16( _T("VehicleID"+tNumber), <VehicleID_E>, true );
// set uncertainty value
paramsList.SetParamDouble( _T("UncertaintyValue"+tNumber), <double>, true);
// set time of vehicle nav measurement [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"+tNumber), <double>, true);
// set geodetic position [radian] and depth [m]
paramsList.SetParamDouble( _T("GeoPositionLat"+tNumber), <double>, true);
paramsList.SetParamDouble( _T("GeoPositionLon"+tNumber), <double>, true);
paramsList.SetParamDouble( _T("GeoPositionZ"+tNumber), <double>, true);
// set velocity over ground (earth fixed) [m/s]
paramsList.SetParamDouble( _T("VelocityOGEfX"+tNumber), <double>, true);
paramsList.SetParamDouble( _T("VelocityOGEfY"+tNumber), <double>, true);
paramsList.SetParamDouble( _T("VelocityOGEfZ"+tNumber), <double>, true);
// set current heading [RAD]
paramsList.SetParamDouble( _T("Heading"+tNumber), <double>, true);
```

4.7 RangeData

Description

Contains the range information between two vehicles. Vehicle ID of second vehicle can be obtained through IPC caller.

Build

```
// set time of measurement [seconds since January 1st 1970]
```

```
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
// set range [m]
paramsList.SetParamDouble( _T("Range"), <double>, true);
```

4.8 PilotingData

VehicleID's for GREX targets (enumeration VehicleID_E)

VID_AGUAS_VIVAS (9)
VID_GENERAL_TARGET (10)

Description

Contains a position data set of a target (vehicle) used by THA during coordinated target tracking.

Build

```
// set target's id (see VehicleID_E)
paramsList.SetParamUInt16( _T("TargetID"), <VehicleID_E>, true );
// set time stamp [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
// set geodetic position [radian]
paramsList.SetParamDouble( _T("GeoPositionLat"), <double>, true);
paramsList.SetParamDouble( _T("GeoPositionLon"), <double>, true);
```

4.9 Telegrams

To realise a communication between several modules of the GREX software, it is neseccary to define some standard messages - telegrams. Each GREX module has to unterstand at least some of these telegrams (see the module specifications). To reduce the complexity of this problem, we use enumerations to specify the telegram classes and some data.

Enumeration TelegramKind_E

```
// receiver should send specified data to sender
TK_REQUEST_DATA
// change specified parameters of a primitive in the single vehicle mission plan
TK_CHANGE_PARAMETER_OF_PRIMITIVE_SVMP
// change a complete SVP of the SVMP
TK_CHANGE_PARAMETERS_OF_PRIMITIVE_SVMP
// change specified parameters of a submanoeuvre in the single vehicle mission plan
TK_CHANGE_PARAMETER_OF_SUBMANOEUVRE_SVMP
// change specified parameters of a primitive in the team mission plan
TK_CHANGE_PARAMETER_OF_PRIMITIVE_TMP
// change specified parameters of a submanoeuvre in the team mission plan
TK_CHANGE_PARAMETER_OF_SUBMANOEUVRE_TMP
// includes a complete path for one GoToFormation (THA communication)
TK_REPLAN_PATH_GOTOFORMATION_TMP
// includes one or more new parts of the path of a CoordinatedTargetTracking (THA communication)
TK_ADD_PATH_COORDINATEDTARGETTRACKING_TMP
// jump to the defined primitive after the termination of the current one
TK_JUMP_AFTERWARDS_TO_PRIMITIVE_SVMP
// jump to the defined primitive immediately
TK_JUMP_IMMEDIATELY_TO_PRIMITIVE_SVMP
// jump to the defined primitive after the current one
TK_JUMP_AFTERWARDS_TO_PRIMITIVE_TMP
// jump to the defined primitive immediately
TK_JUMP_IMMEDIATELY_TO_PRIMITIVE_TMP
// if there was set a new team leader during the mission, every team member has to confirm this
// change this telegram ratifies the new leader
TK_LEADERSHIP_CHANGE_ACCEPTED
// if there was set a new team leader during the mission, every team member has to confirm this
// change this telegram declines the new leader
```

TK_LEADERSHIP_CHANGE_DENIED
// every modification of the mission plan has to be confirmed by the GIM
// this telegram confirms the modification(s)
TK_MODIFICATION_OF_MISSIONPLAN_ACCOMPLISHED
// every modification of the mission plan has to be confirmed by the GIM
// this telegram declines the modification(s)
TK_MODIFICATION_OF_MISSIONPLAN_DENIED
// every vehicle that performs an arc by use of the non # lines has to send this telegram to team
// handler every time it starts a new line (without a request!)
TK_REPORT_ON_STARTING_OF_A_SUBMANOEUVRE_SVMP
// every time a vehicle starts with a new primitive in the mission plan, this telegram has to be send to
// the team handler (without a request!)
TK_REPORT_ON_STARTING_OF_A_PRIMITIVE_SVMP
// every time a vehicle finishes a primitive in the mission plan, this telegram has to be send to the
// team handler (without a request!)
TK_REPORT_ON_FINISHING_OF_A_PRIMITIVE_SVMP
// request for the status report of the current execution level of the active primitive
TK_GIVE_REPORT_ON_CURRENT_EXECUTION_LEVEL_PRIMITIVE
// status report of the current execution level of the active primitive [%]
TK_REPORT_ON_ADVANCE_OF_A_PRIMITIVE_SVMP
// complete mission plan was skipped by the sender
TK_EMERGENCY_MESSAGE_MISSIONPLAN_SKIP
// defined sensor(s) is/are damaged
TK_EMERGENCY_MESSAGE_SENSOR_FAILURE
// defined primitive(s) was/were skipped by the sender
TK_EMERGENCY_MESSAGE_PRIMITIVE_SKIP
// with this message the vehicle stops the execution of the mission plan immediately and keeps
// the position
TK_STOP_AND_KEEP_POSITION
// this telegram deactivates the “stop and keep position” telegram, the vehicle has to continue
// the mission plan to a specified time if the time stamp includes a future time, otherwise immediately
TK_CONTINUE
// this message will be send from the team handler to the local GIM of the current vehicle to
// start execute the mission plan and to define the mission start time (just between team handlers)
TK_START_MISSION
// this message will be send from every team handler to the team handler of the leading vehicle
// after the vehicles initialisation is done
TK_READY_FOR_ACTION
// if it is neseccary, a new leader can be set (must be confirmed by every team member)
TK_TAKEOVER_OF_LEADERSHIP
// to improve the mission (re-)planning, team handler can request some internal informations
// from the vehicle (battery level, …)
TK_GIVE_REPORT_ON_INTERNAL_STATE
// answer for “give report on internal state” telegram
TK_REPORT_ON_INTERNAL_STATE
// general sync message used by team handler
TK_SYNC_PULSE
// telegram send by THA for internal use to handle target pursuit mission parts
TK_PILOTING_CONTINUE
// telegram send by THA for internal use to handle target pursuit mission parts
TK_PILOTING_STOP
// telegram send by GREX console to start the init routines of team handler and all the other modules
// which need to load vehicle mission plans and mission specific files that will be uploaded to the
// vehicles GREX computer just a short time before the mission starts (can’t do offline)
// this telegram includes at least the file name of the team mission plan
TK_START_PREPARATION
// this telegram shows the complete opposite of “start preparation”, if a mission should be aborted, this
// telegram will be send from the GREX console to leading vehicle
// after executing this telegram, all modules are ready for re-initialisation, done by “start preparation”
TK_ABORT_MISSION
// general “ok” message
TK_OK

// general “fail” message
TK_FAIL
// contains a complete path for a MVP_GoToFormation (just for THA communication)
TK_REPLAN_PATH_GOTOFORMATION_TMP
// contains a complete path for a MVP_CoordinatedTargetTracking (just for THA communication)
TK_ADD_PATH_COORDINATEDTARGETTRACKING_TMP

Enumeration RequestDataType_E

// defines vehicle count vector (4.2)
RDT_VEHICLE_COUNT
// defines range data (4.7)
RDT_RANGE_DATA
// defines navigation track data (4.3)
RDT_NAV_TRACK_DATA
// defines THA vehicle data (4.1)
RDT_THA_VEHICLE_DATA
// defines TNAV vehicle data (4.4)
RDT_TNAV_VEHICLE_DATA
// defines vehicle navigation data (4.5)
RDT_VEHICLE_NAV_DATA
// defines team navigation data vector (4.6)
RDT_TEAM_NAV_DATA
// defines the center pos of the hovering manoeuvre (4.8)
RDT_HOVERING_CENTER_POS

4.9.1 RequestData Telegram

Description

Contains the data type of the requested data. It should be possible that another module/vehicle can request every data a module provides. The callee gets the information about the caller over IPC.

Data types (enumeration TelegramKind_E)

TK_REQUEST_DATA

Data types (enumeration RequestDataType_E)

RDT_VEHICLE_COUNT
RDT_RANGE_DATA
RDT_NAV_TRACK_DATA
RDT_THA_VEHICLE_DATA
RDT_TNAV_VEHICLE_DATA
RDT_VEHICLE_NAV_DATA
RDT_TEAM_NAV_DATA

Build

```
// set telegram kind (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set requested data type (see RequestDataType_E)
paramsList.SetParamUInt16( _T("DataType"), <RequestDataType_E>, true);
```

4.9.2 Modification Telegrams – modify manoeuvre

Description

Contains the information for one replanning command. This modification can include more than one parameter but only for one (sub-)manoeuvre.

Data types (enumeration TelegramKind_E)

TK_CHANGE_PARAMETER_OF_PRIMITIVE_SVMP

TK_CHANGE_PARAMETER_OF_SUBMANOEUVRE_SVMP
TK_CHANGE_PARAMETER_OF_PRIMITIVE_TMP
TK_CHANGE_PARAMETER_OF_SUBMANOEUVRE_TMP

Data types (enumeration PlanParameters_E)

PP_LATITUDE
PP_LONGITUDE
PP_DEPTH_MODE
PP_DEPTH
PP_VELOCITY_REL_TO
PP_VELOCITY
PP_VEL_UNITS
PP_WAYPOINT_TYPE
PP_WAIT_TIME
PP_END_TIME
PP_MIN_DEPTH
PP_MAX_DEPTH
PP_SENSOR_STRING
PP_EPSILON
PP_START_LATITUDE
PP_START_LONGITUDE
PP_END_LATITUDE
PP_END_LONGITUDE
PP_CENTER_LATITUDE
PP_CENTER_LONGITUDE
PP_DIRECTION

Build

```
// set telegram kind (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set primtive id [PointNo] or subprimitive id [PointNumber]
paramsList.SetParamUInt16( _T("ManoeuvreID"), <uint16_t>, true);
// set amount of modifications (value equals max tNumber+1)
paramsList.SetParamUInt16( _T("ModificationCount"), <uint16_t>, true);
// set time of replanning command (“age” of command) [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
// set kind of parameter should be changed (see PlanParameters_E)
paramsList.SetParamUInt16( _T("ParameterKind"+tNumber), <PlanParameters_E>, true);
// set new parameter value [according to PlanParameter]
paramsList.SetParamDouble( _T("Value"+tNumber), <double>, true);
```

4.9.3 Modification Telegrams – modify a complete manoeuvre

Description

Contains the information to replan a complete SingleVehiclePrimitive. This telegram is used to replan SVP’s during a CoordinatedTargetTracking or GoToFormation manoeuvre. After a replanning a vehicle is able to validate the new data of the modified primitive. Due to the values *tNumber* and *Number* the replanning package can be devided into the several parts. The maximum values for the both numerations can be found in _T("ManoeuvreCount") for *tNumber* and _T("ModificationCount"+tNumber) for *pNumber*.

Data types (enumeration TelegramKind_E)

TK_CHANGE_PARAMETERS_OF_PRIMITIVE_SVMP

Build

```
// set main telegram kind (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
```

```
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set amount of manoeuvre modifications (value equals max tNumber+1)
paramsList.SetParamUInt16( _T("ManoeuvreCount"), <uint16_t>, true);
// set time of replanning command (“age” of command) [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
// set sub telegram kind (TK_CHANGE_PARAMETER_OF_PRIMITIVE_SVMP or
// TK_CHANGE_PARAMETER_OF_SUBMANOEUVRE_SVMP)
paramsList.SetParamUInt16( _T("TelegramKind"+tNumber), <TelegramKind_E>, true);
// set primtive id [PointNo] or subprimitive id [PointNumber]
paramsList.SetParamUInt16( _T("ManoeuvreID"+tNumber), <uint16_t>, true);
// set amount of value modifications (value equals max pNumber+1)
paramsList.SetParamUInt16( _T("ModificationCount"+tNumber), <uint16_t>, true);
// set kind of parameter should be changed (see PlanParameters_E)
paramsList.SetParamUInt16( _T("ParameterKind"+tNumber+"_"+pNumber), <PlanParameters_E>, true);
// set new parameter value [according to PlanParameter]
paramsList.SetParamDouble( _T("Value"+tNumber+"_"+pNumber), <double>, true);
```

4.9.4 Modification Telegrams – change manoeuvre

Description

Contains the information for one replanning command.

Data types (enumeration TelegramKind_E)

TK_JUMP_AFTERWARDS_TO_PRIMITIVE_SVMP
TK_JUMP_IMMEDIATELY_TO_PRIMITIVE_SVMP
TK_JUMP_AFTERWARDS_TO_PRIMITIVE_TMP
TK_JUMP_IMMEDIATELY_TO_PRIMITIVE_TMP

Build

```
// set telegram kind (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set primtive id [PointNo]
paramsList.SetParamUInt16( _T("ManoeuvreID"), <uint16_t>, true);
// set time of command (“age” of command) [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

4.9.5 Replanning Telegrams – GoToFormation/CoordinatedTargetTracking

Description

Contains the information for one complete replanning of a GoToFormation primitive or new tracks/arcs for a CoordinatedTargetTracking primitive. The different parts of the new path can be distinguished by the value *tNumber*. This telegram is used for TeamHandler communication to shore the results of the named algorithms. For the replanning of a SingleVehiclePrimitive, THA-GIM-Communication, the telegram shown in 4.9.3 will be used.

Data types (enumeration TelegramKind_E)

TK_REPLAN_PATH_GOTOFORMATION_TMP
TK_ADD_PATH_COORDINATEDTARGETTRACKING_TMP

Build

```
// set telegram kind (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
```

```
// set time of command (“age” of command) [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
// set amount of manoeuvre modifications (value equals max tNumber+1)
paramsList.SetParamUInt16( _T("ManoeuvreCount"), <uint16_t>, true);
// set primtive id [PointNo]
paramsList.SetParamUInt16( _T("ManoeuvreID"+tNumber), <uint16_t>, true);
// set geodetic start position [radian] of the arc or track
paramsList.SetParamDouble( _T("StartPositionLat"+tNumber), <double>, true);
paramsList.SetParamDouble( _T("StartPositionLon"+tNumber), <double>, true);
// set geodetic end position [radian] of the arc or track
paramsList.SetParamDouble( _T("EndPositionLat"+tNumber), <double>, true);
paramsList.SetParamDouble( _T("EndPositionLon"+tNumber), <double>, true);
// set geodetic center position [radian] – if that position eqals PLACEHOLDER, new part is a track
// otherwise it is an arc
paramsList.SetParamDouble( _T("CenterPositionLat"+tNumber), <double>, true);
paramsList.SetParamDouble( _T("CenterPositionLon"+tNumber), <double>, true);
// set arc direction
paramsList.SetParamUInt16( _T("ArcDirection"+tNumber), <uint16_t>, true);
// set velocity
paramsList.SetParamDouble( _T("Velocity"+tNumber), <double>, true);
```

4.9.6 Modification Report Telegrams

Description

Contains the answer of a modification command.

Data types (enumeration TelegramKind_E)

TK_LEADERSHIP_CHANGE_ACCEPTED
TK_LEADERSHIP_CHANGE_DENIED
TK_MODIFICATION_OF_MISSIONPLAN_ACCOMPLISHED
TK_MODIFICATION_OF_MISSIONPLAN_DENIED

Build

```
// set command id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set time of report [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

4.9.7 Status Reports without data

Description

Contains a simple status report.

Data types (enumeration TelegramKind_E)

TK_REPORT_ON_STARTING_OF_A_PRIMITIVE_SVMP
TK_REPORT_ON_STARTING_OF_A_SUBMANOEUVRE_SVMP
TK_REPORT_ON_FINISHING_OF_A_PRIMITIVE_SVMP

Build

```
// set report id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set primitive id [PointNo] for primitive and [PointNumber] for submanoeuvre
paramsList.SetParamUInt16( _T("ID“), <uint16_t>, true);
// set time of report [seconds since January 1st 1970]
```

```
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

4.9.8 Status Reports with data

Description

Contains a status report with separate data.

Data types (enumeration TelegramKind_E)

TK_REPORT_ON_ADVANCE_OF_A_PRIMITIVE_SVMP
TK_REPORT_ON_INTERNAL_STATE

Data types including just one information (enumeration StateKind_E)

SK_CURRENT_HEADING

Data types including more than one information (enumeration StateKind_E)

SK_CURRENT_HOVER_CENTER (latitude and longitude)

Build

```
// set report id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set primitive id [PointNo] or id of internal state [StateKind_E]
paramsList.SetParamUInt16( _T("ID“), <uint16_t>, true);
// set data [%,…]
paramsList.SetParamDouble( _T("Data”+tNumber), <double>, true);
// set time of report [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

4.9.9 General Status Reports

Description

Contains a general status report.

Data types (enumeration TelegramKind_E)

TK_READY_FOR_ACTION
TK_OK
TK_FAIL

Build

```
// set report id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set time of report [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

4.9.10 Emergerncy Messages without data

Description

Contains an emergency message.

Data types (enumeration TelegramKind_E)

TK_EMERGENCY_MESSAGE_MISSIONPLAN_SKIP

Build

```
// set message id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set time of message [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

4.9.11 Emergerncy Messages with data

Description

Contains an emergency message with separate data.

Data types (enumeration TelegramKind_E)

TK_EMERGENCY_MESSAGE_SENSOR_FAILURE
TK_EMERGENCY_MESSAGE_PRIMITIVE_SKIP

Build

```
// set message id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set amount of id's (value equals max tNumber+1)
paramsList.SetParamUInt16( _T("IDCount"), <uint16_t>, true);
// set sensor id's [to be defined…] or primitive id's [PointNo] of skipped MVPs
paramsList.SetParamUInt16( _T("ID"+tNumber), <uint16_t>, true);
// set time of message [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

4.9.12 General Commands without data

Description

Contains a simple command.

Data types (enumeration TelegramKind_E)

TK_GIVE_REPORT_ON_CURRENT_EXECUTION_LEVEL_PRIMITIVE
TK_START_MISSION
TK_ABORT_MISSION
TK_STOP_AND_KEEP_POSITION
TK_CONTINUE
TK_PILOTING_CONTINUE
TK_PILOTING_STOP

Build

```
// set command id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), <TelegramKind_E>, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set time of execution (if smaller than current time do it immediately) [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

4.9.13 General Commands

Description

Contains a command with specified data.

Data types (enumeration TelegramKind_E)

TK_TAKEOVER_OF_LEADERSHIP

TK_GIVE_REPORT_ON_INTERNAL_STATE
TK_SYNC_PULSE
TK_START_PREPARATION

Build for TK_TAKEOVER_OF_LEADERSHIP

```
// set command id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), TK_TAKEOVER_OF_LEADERSHIP, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set amount of id's (value equals max tNumber+1)
paramsList.SetParamUInt16( _T("IDCount"), <uint16_t>, true);
// set list of subordinated vehicles (specified via VehicleID_E)
paramsList.SetParamUInt16( _T("VehicleID"+tNumber), <VehicleID_E>, true);
// set time of command [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

Build for TK_GIVE_REPORT_ON_INTERNAL_STATE

```
// set command id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), TK_GIVE_REPORT_ON_INTERNAL_STATE, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set statekind (see StateKind_E)
paramsList.SetParamUInt16( _T("StateKind"), <StateKind_E>, true);
// set time of command [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

Build for TK_SYNC_PULSE

```
// set command id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), TK_SYNC_PULSE, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set sync number
paramsList.SetParamUInt16( _T("SyncNumber"), <uint16_t>, true);
// set time stamp [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

Build for TK_START_PREPARATION

```
// set command id (see TelegramKind_E)
paramsList.SetParamUInt16( _T("TelegramKind"), TK_START_PREPARATION, true);
// set unique telegram id
paramsList.SetParamUInt16( _T("TelegramID"), <uint16_t>, true);
// set string of mission name
paramsList.SetParamString( _T("MissionFileName"), <tstring>, true);
// set time stamp [seconds since January 1st 1970]
paramsList.SetParamDouble( _T("TimeStamp"), <double>, true);
```

5 Data structs for communication in TUI simulation

Just for simulation use (TUI).

5.1 EnvironmentNavData

Description

Contains the current speed of one specified position in x, y and z direction.

Build

```
// set height over ground
paramsList.SetParamDouble( _T("HeightOG"), <double>, true);
// set current earth fixed in x direction
paramsList.SetParamDouble( _T("CurrentEfX"), <double>, true);
// set current earth fixed in y direction
paramsList.SetParamDouble( _T("CurrentEfY"), <double>, true);
// set current earth fixed in z direction
paramsList.SetParamDouble( _T("CurrentEfZ"), <double>, true);
```

5.2 RealVehicleNavData

Description

Contains the real position (in cartesian and geodetic coordinates) of the vehicle. This data is required to get correct *PayloadData* and *EnvironmentNavData* from the EnvironmentModule.

Build

```
// set latitude
paramsList.SetParamDouble( _T("GeoPositionLat"), <double>, true);
// set longitude
paramsList.SetParamDouble( _T("GeoPositionLon"), <double>, true);
// set depth
paramsList.SetParamDouble( _T("GeoPositionZ"), <double>, true);
// set x
paramsList.SetParamDouble( _T("PositionX"), <double>, true);
// set y
paramsList.SetParamDouble( _T("PositionY"), <double>, true);
// set z
paramsList.SetParamDouble( _T("PositionZ"), <double>, true);
// set attitude (phi)
paramsList.SetParamDouble( _T("EulerAnglePhi"), <double>, true);
// set attitude (theta)
paramsList.SetParamDouble( _T("EulerAngleTheta"), <double>, true);
// set attitude (psi)
paramsList.SetParamDouble( _T("EulerAnglePsi"), <double>, true);
```