# NAVIGATION SDK

# Manual

**VERSION 1.00**

**AUGUST 2025**

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF CODES

# ABBREVIATIONS

| Abbreviation | Description |
|---|---|
| AHRS | Attitude Heading Reference System |
| BIN Message | Binary Message |
| CEP | Circular Error Probability |
| CSEP | Calculated Separation |
| IMU | Inertial Measurement Unit |
| INS | Inertial Navigation System |
| MSEP | Measured Separation |
| N/A | Not Available |
| N/C | Not Connected |
| RMS | Root Mean Square |
| TBD | To Be Determined |
| UKF | Unscented Kalman Filter |
| VG | Vertical Gyro |

# 1. OVERVIEW

The current Software Development Kit (SDK) includes a library that gathers functionality to control arazim navigation products. Currently, the library works with the operating systems listed in Table 1.

The SDK also includes program examples that demonstrate how to use library functions. Table 2 lists the current languages supported.

**Table 1. List of Library Availability for Operation Systems**

| OS | Library | Examples |
|---|---|---|
| Windows | ✓ | ✓ |
| Linux | ✓ | ✓ |

**Table 2. Supported Languages for the current SDK Library**

| Language | OS | Date | Wrapper | Examples |
|---|---|---|---|---|
| C | Windows/Linux | 11/5/20 | ✓/✓ | ✓/✓ |
| C# | Windows | | --- | --- |
| Delphi | Windows | | --- | --- |
| Python | Windows/Linux | | --- | --- |

This document summarizes and explains the library functionality for:

- Configuration Instructions
- Calibration Instructions
- Data Messages

The SDK supports all navigation platforms, with both general and platform-specific functions and messages. This document specifies which platforms are relevant for each function and message.

The SDK library assists programmers in controlling the navigation device platform. When you use the navigation device platform in a system, you can use the library of functions tested with all navigation device platforms without having to write low level packet handlers or parsers (as long as the OS used is supported).

When using an unsupported OS or when the HOST is another MCU, you can implement the code, as described in {"NAV API Manual.pdf"}.

| Note | All explanations and examples in this SDK Manual are in the C language. |
|---|---|

# 2. LIBRARY FUNCTIONS

## 2.1. Overview

The dynamic link library (DLL) has several functions that allow you to work with any of the navigation devices supplied by arazim. The following collection of functions runs classes that handle query building, analyzing query responses, analyzing status and data messages, saving a binary file and exporting a binary file to ASCII files. In addition to these basic functions, the library includes functions that:

- simulate the receipt of any type of static data messages
- simulate dynamic three-plane tilt
- play back a recorded data file.

## 2.2. Basic Library Functions

Table 3 lists the basic functions used with all existing navigation platforms. You can use these functions to:

- build queries
- receive incoming data from the communication channel
- analyze and fire a relevant callback function:
    - command reply
    - acknowledge
    - data messages.

The information in this section refers to the function index numbers in Table 3.

**Table 3. List of Basic Functions**

| Index | Function Name |
|---|---|
| 1 | bool navSdk_CreateCNavSdkObject(void); |
| 2 | bool navSdk_ReleaseNavSdkObject(void); |
| 3 | uint8_t navSdk_WriteData(uint8_t* ucBuff, uint16_t usLen); |
| 4 | uint8_t navSdk_BuildCommand(TstCommand* pCmnd, TstCommandBuff* pstBuff); |
| 5 | void navSdk_SetOnApiAck_CallBack(TOnApi_Acknowledge cbOnApiAck); |
| 6 | void navSdk_SetOnApiReply_CallBack(TOnApi_Reply cbOnApiReply); |
| 7 | void navSdk_SetOnApiMsgRcvd_CallBack(TOnMessageRcvd cbOnApiMsgRcvd); |
| 8 | bool navSdk_Read_DllVersion(uint8_t& u8Major, uint8_t& u8SubMajor); |

### 2.2.1. Basic Function – navSdk_CreateCNavSdkObject

Call Basic Function 1 to create the main object (activate the constructor function of the class), responsible for all connectivity processed with the Navigation family of products.

The function returns “true” if the object creation succeeds or “false” if the object creation fails.

### 2.2.2. Basic Function – navSdk_ReleaseNavSdkObject

Call Basic Function 2 to release the object and the memory it occupies once the primary object process (responsible for managing the processes with the connected navigation system) is finished.

### 2.2.3. Basic Function – navSdk_WriteData

Call Basic Function 3 to deliver data bytes (collected by the host from the navigation device) to the main navigation object.

The function first argument consists of the collected data bytes, sent as an array of bytes, while the second argument consists of the array length.

The data bytes received from navigation device are analyzed (Detecting: Acknowledge, Command Reply, or Message Information). After that a meaningful block of data is created and directed to the relevant call back function.

The call back function includes the platform’s raw information, presented in a meaningful, ready-to-use data structure.

### 2.2.4. Basic Function – navSdk_BuildCommand

You can use this function to build all navigation instructions (GET/SET) to read and update configuration. Alternatively, you can use a set of dedicated functions to facilitate the process.

Call Basic Function 4 to build the following user instructions to the navigation device:

- change mode instructions (immediate group commands)
- inquiry instructions (GET)
- determination instructions (SET).

This command receives two arguments (input data structure and output data structure.

- Input data structure (TstCommand) is the first argument, and includes the instruction index, the device for which the instruction is intended, and the parameters required as additional arguments for the instruction ID.

- Output data structure (TstCommandBuff) is the second argument, which is actually the transformation of the input command structure into a raw command, given as array of bytes to be transmitted to the connected platform (Basic Function 3) The output structure consists of an array of bytes and the array size.

When this function succeeds, it returns "0".

Section 5 provides additional details on Basic Function 4 (navSdk_BuildCommand).

Instead of using a single function with huge structure as an argument, you can use dedicated functions that wrap the navSdk_BuildCommand and handle the large structure fields. Section 3 contains additional information.

| Note | • Chapter 5 contains information on the correct use of the low-level function navSdk_BuildCommand and the TstCommad structure.<br>• Chapter 3 contains a list of direct functions that wrap the navSdk_BuildCommand and the TstCommad structure. |
|---|---|

## 2.2.5. Basic Callback Functions – Overview

A callback function is a function that is passed as an argument to another function, and is intended to be executed after a specific event or task completes. It allows for flexible and asynchronous programming by letting you define custom behavior that runs at a later time.

First, define a function that meets the required template of the relevant callback function. Afterwards, pass a pointer to that function, to be called when the relevant process occurs.

Basic Functions 5-7 deliver raw data information (in a reply structure) from the navigation system to the user application. These functions link user functions to the main object process. Based on the relevant template, you can obtain the information sent in the call back functions from the navigation platform.

### 2.2.5.1. Basic Function – navSdk_SetOnApiAck_CallBack

By default, the navigation device replies to every SET instruction with an Acknowledge message, indicating that the last instruction was received successfully (without errors).

Create a function of type TOnApi_Acknowledge and send a pointer of that function as an argument to navSdk_SetOnApiAck_CallBack.

A user function of type TOnApi_Acknowldge is called for any SET command sent to, and received successfully by, the device.

**Code 1. TOnApi Acknowledge – Calback Function Template**

```
typedef void(*TOnApi_Acknowledge)(void* pObj, TstCommand stCmndReply);
```

### 2.2.5.2. Basic Function – navSdk_SetOnApiReply_CallBack

By default, the navigation device replies to every GET instruction (inquiry command) with a command reply, which contains the information you requested. Any API reply holds the same instruction index information with the requested information displayed as list of parameters.

Create a function of type TOnApi_Reply and send a pointer of that function as an argument to navSdk_SetOnApiReply_CallBack.

A user function of type TOnApi_Reply is called for any GET command sent to, and received successfully by, the device.

**Code 2. TOnApi_Reply – Callback Function Template**

```
typedef void(*TOnApi_Reply)(void* pObj, TstCommand stCmndReply);
```

### 2.2.5.3. Basic Function – navSdk_SetOnMsgRcvd_CallBack

By default, the navigation device transmits configured messages to the host controller. To receive data messages, create a function of type TOnMessageRcvd and send a pointer of that function as an argument to navSdk_SetOnMsgRcvd_CallBack.

The user function of type TOnMessageRcvd is called for any message or status information sent from the navigation device.

**Code 3. TOnMessageRcvd Calback Function Template**

```
typedef void(*TOnMessageRcvd)(void* pObj, TstSysTesting_Basic *MsgBase, void* pMsg);
```

## 2.2.6. Basic Function – navSdk_Read_DllVersion

Track compatibility by calling this function (the only function that is not part of the main object) to read the DLL version.

The command receives two arguments:

- The first argument, of type uint8_t, returns the major version value.
- The second argument, of type uint8_t, returns the sub-major value of the DLL version.
- Examples: 1.11 where major = 1, and sub-major = 11

## 2.3. Navigation Device States

A navigation device can be at any given time only in a single specific state:

- Boot – A pre-active condition in which the Host can program the device to update firmware.
- Initialization – A pre-active condition in which the device handles the BIT and waits for all conditions to be filled met before moving to "Real-Time" mode.
- Real-Time – The mode in which the device is fully working; it samples IMU and GPS and transmits selected messages to the Host controller.
- Configuration – The mode in which the device stops transmitting messages to the Host controller and listens to Host Read/Write commands. Use this mode to set the device's updated work mode.
- Calibration – The mode in which the device stops transmitting messages to the Host controller and listens to Host commands to handle dynamic calibrations, such as Bias-Estimation and Hard-Iron.

The Boot and Initialization modes are part of the device power up sequence and you cannot switch to these modes. However, you can send a soft reset command to restart the sequence.

Under the other three modes (Real-Time, Configuration & Calibration), the device has a list of commands under each group that you can use.

To use the commands for a specific mode, switch from the current mode to the required mode.

# 3. LIBRARY FUNCTIONS TO UPDATE SETTINGS

The API instructions control (read/write) the behavior of the navigation device connected to the Host computer (by default, via RS232 communication channel).

In general, you configure a device's settings at least once before using the device. After configuring those settings, you will receive incoming data messages.

## 3.1. General Information for Building API Commands

The SDK functions are wrappers for API commands that the user may decide to implement by himself (see: "NAV API Manual.pdf"). The API commands are divided into two sections:

- GET commands – commands that are used to read the current settings, meaning, commands that inquire for settings values and the connected navigation device replies with API Command Reply that includes the requested information.
- SET commands – commands that are used to write new setting values to the connected navigation device. Commands of type SET may enforce a process (reset, save to flash, etc…) or set new settings values. For any command of type SET, the connected device replies with API acknowledge reply, which indicates the user that the last command has been received OK.

As described earlier, the user can use a single function (**navSdk_BuildCommand**) to build all commands (GET/SET) by passing the `TstCommand` structure as argument which contains the information from which the **navSdk_BuildCommand** can build the required API command.

**Code 4. Build Command Argument – TstCommand Struct**

```
typedef struct packed{
        uint8_t DeviceType;      //0 = system\platform device, >=0x20 Specific Device
        uint8_t DeviceIndex;     //0 = all devices from DeviceID type, any
                                 //other value should indicate a specific device
        uint8_t CommandID;       //Command under DeviceType
        bool    bAcknowledge;
        TstAlgo_UKF              Algo;   //parameters values for UKF GET/SET
        TstCalibration           Calib;  //parameters values for Calibration GET/SET
        TstGlobalSystemSettings  Sys;    //parameters values for system GET/SET
        TstDeviceSettings        Device; //parameters values for specific device
                                         //GET/SET
}TstCommand;
```

The `TstCommand` structure fields are:

- DeviceType – represents the assignment group of the API command (Immediate, Configuration, ….)
- DeviceIndex – represents the specific device for which the command is intended.
- CommandID – defines the specific API command according to DeviceType index.
- bAcknowledge – is used as a return value in API command acknowledge reply (for SET commands).
- The rest of the fields are structures that bind fields according to the activity/responsibility.

When a build command succeeds, a command is returned as an array of bytes in `TstCommandBuff` structure.

### Code 5. TstCommandBuff Structure

```
typedef struct {
        uint8_t         ucaBuffer[250];
        uint8_t         ucBuffLen;
        uint8_t         eResult;
} TstCommandBuff;
```

The structure consists of:

- An array that holds the raw command bytes (ucaBuffer[250]) of API command. A general API command structure is exhibited below (see: Table 4).
- A byte that indicates how many bytes to use from the array, starting from byte 0 (ucBuffLen).
- A command build result byte (eResult), indicates whether creation of the raw command succeeded.
    - 0 – OK
    - 1 – ERROR
    - 2 – Buffer Full

### Table 4. General API Command structure

| | Header | Length | Group/D.T | Command | Device ID | Data | CkSum |
|---|---|---|---|---|---|---|---|
| **Bytes** | 2 | 1 | 1 | 1 | 1 | N | 2 |
| **Default** | [> | 5+N | < 0x80 | | | | |

As an alternative for the **navSdk_BuildCommand**, the user can use functions that wrap the **navSdk_BuildCommand** and supply the `TstCommand` structure as argument with the data fields updated accordingly.

Using the alternative functions, for most GET commands, the only argument is the returned TstCommandBuff structure that contains the raw bytes of the command. For the some of the GET commands and most of the SET commands, the user is required to send a value and receive the build command in TstCommandBuff.

All displayed API commands build the raw command as an array of bytes to be delivered, using the communication port on the Host controller to the navigation device. Therefore, all commands include an argument of type `TstCommandBuff` structure, which holds the returned raw command.

All API commands return a value of type uint8_t, as follows:

- 0 = command creation succeeded.
- 255 = The main object was not created and no command was built.
- Any other value = command creation failed.

The remainder of this chapter describes the direct functions that wrap the **navSdk_BuildCommand** to facilitate your work with the navigation device.

### 3.1.1. Expected Device Reply for SET Commands

Any SET command that is properly received by the navigation device will be responded to with API Acknowledge message. The DLL in response will call/activate the user template function for **TOnApi Acknowledge** with structure argument (`TstCommand`) filled with the relevant information.

For example: assuming stCmnd is argument of type `TstCommand`, the reply will be:
stCmnd.DeviceType = 'device type, as sent in last SET command'
stCmnd.DeviceIndex = 'device index, as sent in last SET command'
stCmnd.CommandID = 'command id, as sent in last SET command'
stCmnd.bAcknowledge = true/false

### 3.1.2. Expected Device Reply for GET Commands

Any GET command that is properly received by the navigation device will be responded to with API Reply message. The DLL in response will call/activate the user template function for TOnApi_Reply with structure argument (`TstCommand`) filled with relevant information.

For example: assuming stCmnd is argument of type `TstCommand`, the reply will be:
stCmnd.DeviceType = 'device type, as sent in last SET command'
stCmnd.DeviceIndex = 'device index, as sent in last SET command'
stCmnd.CommandID = 'command id, as sent in last SET command'

The information returned will be in the structured fields, according to the GET command.

- When inquiring configuration information (Read HW Version, FW Version, Baud-Rate….), the returned information will be in stCmnd.Sys.'relevant fields'.
- When inquiring calibration information (Read HI calibration coefficient…), the returned information will be in stCmnd.Calib.'relevant fields'.
- When inquiring algorithm information (Read IMU orientation…), the returned information will be in stCmnd.Algo.'relevant fields'.

## 3.2. Immediate Group Functions

The functions under Immediate Group can be delivered to the connected device when the connected device is in: "Real-Time" mode.

The Immediate group functions are mainly used to switch modes.

**Table 5. List of Immediate Functions**

| Index | Function Name | Description |
|---|---|---|
| 1 | Enter/Exit Configuration | Enter Configuration mode from Real-Time mode and Exit Configuration mode to return to Real-Time mode |
| | `uint8_t navSdk_EnterConfiguration(bool bEnterConfig, TstCommandBuff* pstBuff);` | |
| 2 | Enter/Exit Calibration | Enter Calibration mode from Real-Time mode and Exit Calibration mode to return to Real-Time mode |
| | `uint8_t navSdk_EnterCalibration(bool bEnterCalibration, TstCommandBuff* pstBuff);` | |
| 3 | Get Device Name | Read Device Name in Real-Time mode |
| | `uint8_t navSdk_GetDeviceName(TstCommandBuff* pstBuff);` | |
| 4 | Get Sample | Pull message information |
| | `uint8_t navSdk_GetSample(uint8_t u8MsgIndex, TstCommandBuff* pstBuff);` | |

### 3.2.1. Enter/Exit Configuration Mode

Use this command to change the current mode from Real-Time mode (default after proper power-up sequence) to Configuration mode. When the device enters Configuration mode, you can send the API instructions under the configuration group.

- You can only send Configuration commands when the device is in Configuration mode.
- If you send an Enter Configuration Mode command when the device is in Configuration mode, the device will remain in Configuration mode.
- If you send an Exit Configuration Mode command when the device is in Real-Time mode, the device will remain in Real-Time mode.

The command consists of two arguments:

- The first argument is Boolean. When True, it indicates a build command of Enter Configuration, when False it indicates a build command of Exit Configuration.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.
- Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully)

### 3.2.2. Enter/Exit Calibration Mode

Use this command to change the current mode from Real-Time mode (default after proper power-up sequence) to Calibration mode. When the device enters Calibration mode, you can send the API instructions under the Calibration mode.

- You can only send Calibration commands when the device is in Calibration mode.
- If you send an Enter Calibration Mode command when the device is in Calibration mode, the device will remain in Calibration mode.
- If you send an Exit Calibration Mode command when the device is in Real-Time mode, the device will remain in Real-Time mode.

The command receives two arguments:

- The first argument is Boolean. When True, it indicates a build command of Enter Calibration, when set to False it indicates a build command of Exit Calibration.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.
- Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully)

### 3.2.3. Get Device Name

This command was added to the immediate commands so you can verify that the correct device is connected before working with it.

This command is also under Configuration mode.

The command receives a single argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.
- Since this is a GET command, you can expect a command reply with the requested information.

### 3.2.4. Get Sample

Use this command if the navigation device was configured to work in Polling mode. While in Polling mode, the device does not transmit messages at a constant rate. Instead, you have to acquire a specific message by calling the Get Sample command.

The command receives two arguments:

- The first argument is a uint8_t, which holds the index of the required message type that you want to pull.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.
- Since this is a special GET command, you can expect a message reply from the device if the command was received successfully.

## 3.3. General Configuration Group Functions

The functions under Configuration Group can be used only after using the Immediate Enter Configuration command (`navSdk_EnterConfiguration`) and have received an Acknowledge reply.

These API commands let you obtain navigation device information and control the device's behavior for optimal use.

**Table 6. Configuration Group Functions**

| Index | Function Name | Description |
|---|---|---|
| General Control Commands | | |
| 1 | Reset | Software command to reboot the application |
| | `uint8_t navSdk_SoftReset(TstCommandBuff* pstBuff);` | |
| 2 | Reset To Factory | Change the device configuration to factory default settings |

| Index | Function Name | Description |
|---|---|---|
| | `uint8_t navSdk_DoResetToFactory(TstCommandBuff* pstBuff);` | |
| 3 | Save Settings | Save current configuration in local flash. |
| | `uint8_t navSdk_DoSaveAsDefault(TstCommandBuff* pstBuff);` | |
| Configuration GET Commands | | |
| 4 | Read Device Name | Read the device name |
| | `uint8_t navSdk_Read_DeviceName(TstCommandBuff* pstBuff);` | |
| 5 | Read FW Version | Read the device firmware version |
| | `uint8_t navSdk_Read_FWVersion(TstCommandBuff* pstBuff);` | |
| 6 | Read HW Version | Read the device hardware version |
| | `uint8_t navSdk_Read_HWVersion(TstCommandBuff* pstBuff);` | |
| 7 | Read Serial Number | Read the device serial number |
| | `uint8_t navSdk_Read_DeviceSN(TstCommandBuff* pstBuff);` | |
| 8 | Read Device Type | Read the device type |
| | `uint8_t navSdk_Read_DeviceType(TstCommandBuff* pstBuff);` | |
| 9 | Read Device Version | Read the device version |
| | `uint8_t navSdk_Read_DeviceVersion(TstCommandBuff* pstBuff);` | |
| 10 | Read Operation Mode | Read the device operation mode (Immediate/Calc Bias-estimation) |
| | `uint8_t navSdk_Read_OperationMode(TstCommandBuff* pstBuff);` | |
| 11 | Read Sample Mode | Read the device sample mode (continues/polling) |
| | `uint8_t navSdk_Read_SampleMode(TstCommandBuff* pstBuff);` | |
| 12 | Read Sample Rate | Read the IMU sensors sample rate |
| | `uint8_t navSdk_Read_SampleRate(TstCommandBuff* pstBuff);` | |
| 13 | Read Message Frame Rate | Read for each message the output frame rate |
| | `uint8_t navSdk_Read_MessageRate(TeMessages_SysTesting frameType, TstCommandBuff* pstBuff);` | |
| 14 | Read communication Baud-Rate | Read the device communication channel baud-rate |
| | `uint8_t navSdk_Read_BaudRate(TstCommandBuff* pstBuff);` | |
| 15 | Read Communication Channel Type | Read the device communication type (RS232/RS422) |
| | `uint8_t navSdk_Read_CommType(TstCommandBuff* pstBuff);` | |
| Configuration SET Commands | | |

| Index | Function Name | Description |
|---|---|---|
| 16 | Set Operation Mode | Sets the power up bias-estimation sequence |
| | `uint8_t navSdk_Set_OperationMode(uint8_t opMode, TstCommandBuff* pstBuff);` | |
| 17 | Set Sample Mode | Sets the device data transmission mode |
| | `uint8_t navSdk_Set_SampleMode(uint8_t sampleMode, TstCommandBuff* pstBuff);` | |
| 18 | Set Sample Rate | Sets the IMU raw data sampling rate |
| | `uint8_t navSdk_Set_SampleRate(uint8_t sampleRate, TstCommandBuff* pstBuff);` | |
| 19 | Set Message Frame Rate | Sets the active message types and their transmission rate |
| | `uint8_t navSdk_Set_MessageRate(TeMessages_SysTesting frameType, uint8_t rateIndex, TstCommandBuff* pstBuff);` | |
| 20 | Set Baud Rate | Sets the device communication channel baud-rate. |
| | `uint8_t navSdk_Set_BaudRate(uint32_t u32Baudrate, TstCommandBuff* pstBuff);` | |

### 3.3.1. Reset

Use this command to reset the system without having to perform a power cycle. The soft reset will be executed only after you exit the configuration mode. make sure to save any configuration changes before calling this function.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully)

### 3.3.2. Reset To Factory

Use this command to restore the device's default factory settings, which are in a different memory location that you cannot change.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully)

### 3.3.3. Save Settings

Use this command to save all updated settings to the local flash to be used as default settings on the next powerup. When configuration instructions determine system behavior, all changes are temporary and influence the active registers in memory. Therefore, the settings must be saved if you plan to use them in the next powerup.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully)

### 3.3.4. Read Device Name

Use this command to read the connected device name. There are several types of navigation devices, and reading the name and type of a device should be used to verify that you have connected the correct device.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

Table 7 lists optional reply values.

**Table 7. Navigation devices P/N and Names**

| Index | Device P/N | Device Name |
|---|---|---|
| 1 | EX100 | EXIGUO_AHRS |
| 2 | EX200 | EXIGUO_INS |
| 3 | EX300 | EXIGUO_INS2ANT |
| 4 | EXLP100 | EXIGUO_LP_AHRS |
| 5 | EXLP200 | EXIGUO_LP_INS |
| 6 | EXLP300 | EXIGUO_LP_INS2ANT |
| 7 | gX100 | GINS_AHRS |
| 8 | gX200 | GINS_INS |
| 9 | gX300 | GINS_INS2ANT |
| 10 | Arsys | ARSYS_INS2ANT_05 |

### 3.3.5. Read Firmware Version

Use this command to read the device firmware version. You can read the device firmware version to manage version tracking and to ensure that you are using the most updated version.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

### 3.3.6. Read Hardware Version

Use this command to read the device hardware version. The hardware version indicates changes in the internal PCB version. You can read the navigation device hardware version to track hardware compatibility with customer designs.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

### 3.3.7. Read Serial Number

Use this command to read the device serial number. Each navigation device has a unique serial-number, which should be used to track device life use and maintenance.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

### 3.3.8. Read Device Type

Use this command to read the device type.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

The navigation devices should reply with: "Navigation"

### 3.3.9. Read Device Version

Use this command to read the device version. In contrast to the hardware version, the device version points to a major update that may include major PCB changes and the replacement of sensors.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

### 3.3.10. Read Communication Channel Type

Use this command to read the navigation device communication channel. The navigation device communication channel may have been converted to USB, ETH or another communication port.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

**Table 8. Optional Communication Channel Types**

| Index | Channel Type |
|---|---|
| 0 | RS232 |
| 1 | RS422 |

### 3.3.11. Read Operation Mode

Use this command to read the current settings of the operation mode. The operation mode determines whether the system will wake up to and start the Bias-Estimation procedure or start immediately (using the last saved coefficients).

The factory default operation mode is to use Bias-Estimation on powerup.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

**Table 9. Optional Operation Modes**

| Return Value | Information |
|---|---|
| 0 | Start Immediate – use last save coefficients |
| 1 | Calculate Bias-Estimation on powerup |

### 3.3.12. Read Sample Mode

Use this command to read the current sample mode settings. The sample mode determines if the system continuously transmits messages at a constant rate to the Host controller or if the Host controller has to send a message query command.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

**Table 10. Optional Sample Modes**

| Return Value | Information |
|---|---|
| 0 | Continuous – messages are transmitted on constant rate |
| 1 | Polling – Host should send a message query command |

### 3.3.13. Read Sample Rate

Use this command to read the current sensors sample rate. You can configure the navigation sensors sampling rate and the navigation calculation messages rate. The navigation messages rate are limited to the sensors sampling rate.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

**Table 11. Optional Sample Rate Values**

| Index | Value [Hz] |
|---|---|
| 0x01 | 1 |
| 0x02 | 10 |
| 0x03 | 25 |
| 0x04 | 50 |
| 0x05 | 100 |
| 0x06 | 200 |
| 0x07 | 250 |
| 0x08 | 500 |
| 0x09 | 1000 |
| 0x0a | 2000 (gX platforms only) |

### 3.3.14. Read Message Frame Rate

Use this command to read the actual active rate of a selected message. Send a query for each message index to know the total message bandwidth.

The command consists of two arguments:

- The first argument is the frame type index, as listed in Table 12.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

**Table 12. Frame Types**

| Frame Index | Message Name | Platform |
|---|---|---|
| IMU Messages | | |
| 0x21 | IMU 2 – 6DOF | All Platforms |
| 0x22 | IMU 3 – 9DOF | All Platforms |
| 0x23 | IMU 4 – Misc Sensors | All Platforms |
| GPS Messages | | |
| 0x30 | GPS 1 – LLA | INS Platforms |
| 0x31 | GPS 2 – ECEF | INS Platforms |
| 0x32 | GPS 3 – Statistics | INS Platforms |
| 0x33 | GPS 4 – RTK | Kept for future |
| Tilt Messages | | |
| 0x40 | Tilt – [Roll, Pitch] | All Platforms |
| 0x41 | Tilt – [Roll, Pitch, Yaw] | All Platforms |
| 0x42 | Tile – [Quarter] | All Platforms |
| 0x43 | Tilt – [DCM] | All Platforms |
| Navigation Messages | | |
| 0x50 | INS 1 – Tilt & Pos | INS Platforms |
| 0x51 | INS 2 – INS 1 + Velocity | INS Platforms |
| 0x52 | INS 3 – INS 2 + Time | INS Platforms |
| 0x53 | INS 4 – Forward Lever ARM | EX300 Only |

**Table 13. Optional Frame Rates**

| Index | Rate [Hz] | Message Types |
|---|---|---|
| 0x00 | Disable | All Message Types |
| 0x01 | 1 | All Message Types |
| 0x02 | 2 | All Message Types |
| 0x03 | 5 | All Message Types |
| 0x04 | 10 | All Message Types |
| 0x05 | 25 | All Message Types |
| 0x06 | 50 | All Message Types |
| 0x07 | 100 | All Message Types |
| 0x08 | 200 | IMU Message Types |
| 0x09 | 250 | IMU Message Types |
| 0x0a | 500 | IMU Message Types |
| 0x0b | 1000 | IMU Message Types |
| 0x0c | 2000 | IMU Message Types for gX platforms only |

### 3.3.15. Read Communication Baud-Rate

Use this command to read the current communication channel baud-rate. In order to read this baud-rate, the Host controller and the navigation device should already be operating at the same baud-rate. This command is used for verification: after setting a new baud-rate value, you can read back the value. The new baud-rate will be used only after you exit the Configuration mode.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

**Table 14. Optional Baud-Rates**

| Index | Baud Rate Value |
|---|---|
| 1 | 115200 |
| 2 | 230400 |
| 3 | 460800 |
| 4 | 921600 |

### 3.3.16. Set Operation Mode

Use this command to set the required navigation device operation mode. The operation mode determines whether the system will wake up to and start the Bias-Estimation procedure or start immediately (using the last saved coefficients).

The factory default operation mode is to use Bias-Estimation on powerup.

The command consists of two arguments:

- The first argument is of type uint8_t and you should set operation mode according to values indicated in Table 9.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

### 3.3.17. Set Sample Mode

Use this command to set the current sampling mode. The sample mode determines if the system continuously transmits messages at a constant rate to the Host controller or if the Host controller has to send a message query command.

The factory default for sample mode is continuous transmission.

The command consists of two arguments:

- The first argument is of type uint8_t and you should set sample mode according to values indicated in Table 10.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

### 3.3.18. Set Sample Rate

Use this command to set the current sample rate for the IMU sensors. You can configure the navigation sensors sampling rate and the navigation calculation messages rate. The navigation messages rates are limited to the IMU sensors sampling rate.

The factory default for sample rate is 100Hz (index = 0x05).

The command consists of two arguments:

- The first argument is of type uint8_t and you should set IMU sensors sample rate according to the index values indicated in Table 11.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

### 3.3.19. Set Message Type and Rate

Use this command to set the navigation device messages types and their rates. The navigation device allows you to select several messages to be transmitted to the Host controller, setting for each message its own rate.

The command consists of three arguments:

- The first argument is of type uint8_t and you should set the frame type according to the index values indicated in Table 12.
- The second argument is of type uint8_t and you should set the frame rate according to the index values indicated in
- Table 13
- The third argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

### 3.3.20. Set Baud-Rate

Use this command to set the navigation device baud-rate. To be able to configure the navigation device baud-rate, the Host controller and the navigation device should already be operating at the same baud-rate. After setting the baud-rate, it is best to read it back to verify that correct baud-rate was set. When exiting configuration mode, the navigation device will use the new baud-rate settings. Be sure to change the baud-rate settings on the Host controller side.

The factory default baud-rate is 921600 (index = 0x04).

The command consists of two arguments:

- The first argument is of type uint32_t and you should set the baud-rate according to the index values indicated in Table 14.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

## 3.4. Device-Specific Configuration Group Functions

The API instructions for all navigation products include a set of instructions dedicated to navigation sub-devices.

Since the device-specific set of instructions is quite lengthy, this chapter covers only instructions relevant to user settings.

As with General Configuration instructions, the system must be in Configuration mode to configure any of the available device-specific groups.

### 3.4.1. UKF Device Group

You can configure several parameters in the UKF Device Group, based on the commands provided in Table 15.

**Table 15. UKF Configuration Functions**

| Index | Function Name | Description |
|---|---|---|
| UKF GET Commands | | |
| 1 | Read Device Orientation | Reads the last assembly orientation of the device with respect to the platform heading. |
| | `uint8_t navSdk_UKF_GetIMUOrientation(TstCommandBuff* pstBuff);` | |
| 2 | Read Number of Static Samples | Reads the number of static samples last defined by the user for bias-estimation calculation. |
| | `uint8_t navSdk_UKF_GetNumStaticSamples(TstCommandBuff* pstBuff);` | |
| 3 | Read IIR Filter Status | Reads the last activity status of the IIR filter |
| | `uint8_t navSdk_UKF_GetIIRFilterStatus(TstCommandBuff* pstBuff);` | |
| 4 | Read RPY Offsets | Reads the last offset values used for Roll, Pitch, and Yaw |
| | `uint8_t navSdk_UKF_GetRPYOffset(TstCommandBuff* pstBuff);` | |
| 5 | Read Lever-ARM Settings | Reads the relative position information of the Primary antenna to the Navigation device. |
| | `uint8_t navSdk_UKF_GetLeverArm(TstCommandBuff* pstBuff);` | |
| UKF SET Commands | | |
| 6 | Set Device Orientation | Updates the device's assembly orientation with respect to the platform heading. |
| | `uint8_t navSdk_UKF_SetIMUOrientation(uint8_t u8Orientation, TstCommandBuff* pstBuff);` | |

| Index | Function Name | Description |
|---|---|---|
| 7 | Set Number of Static Samples | Updates the number of static samples for bias-estimation calculation. |
| | `uint8_t navSdk_UKF_SetNumStaticSamples(uint16_t u16Samples, TstCommandBuff* pstBuff);` | |
| 8 | Set IIR Filter Activity Status | Updates the activity status of the IIR Filter |
| | `uint8_t navSdk_UKF_SetIIRFilterStatus(bool bEnable, TstCommandBuff* pstBuff);` | |
| 9 | Set RPY Offsets | Updates the offset values for Roll, Pitch, and Yaw |
| | `uint8_t navSdk_UKF_SetRPYOffset(bool bUseShiftValues, float fRoll, float fPitch, float fYaw, TstCommandBuff* pstBuff);` | |
| 10 | Set Lever-ARM Settings | Updates the relative position information of the Primary antenna to the Navigation device. |
| | `uint8_t navSdk_UKF_SetLeverArm(bool bUseLeverArm, float x, float y, float z, TstCommandBuff* pstBuff);` | |

#### 3.4.1.1. Read Device Orientation

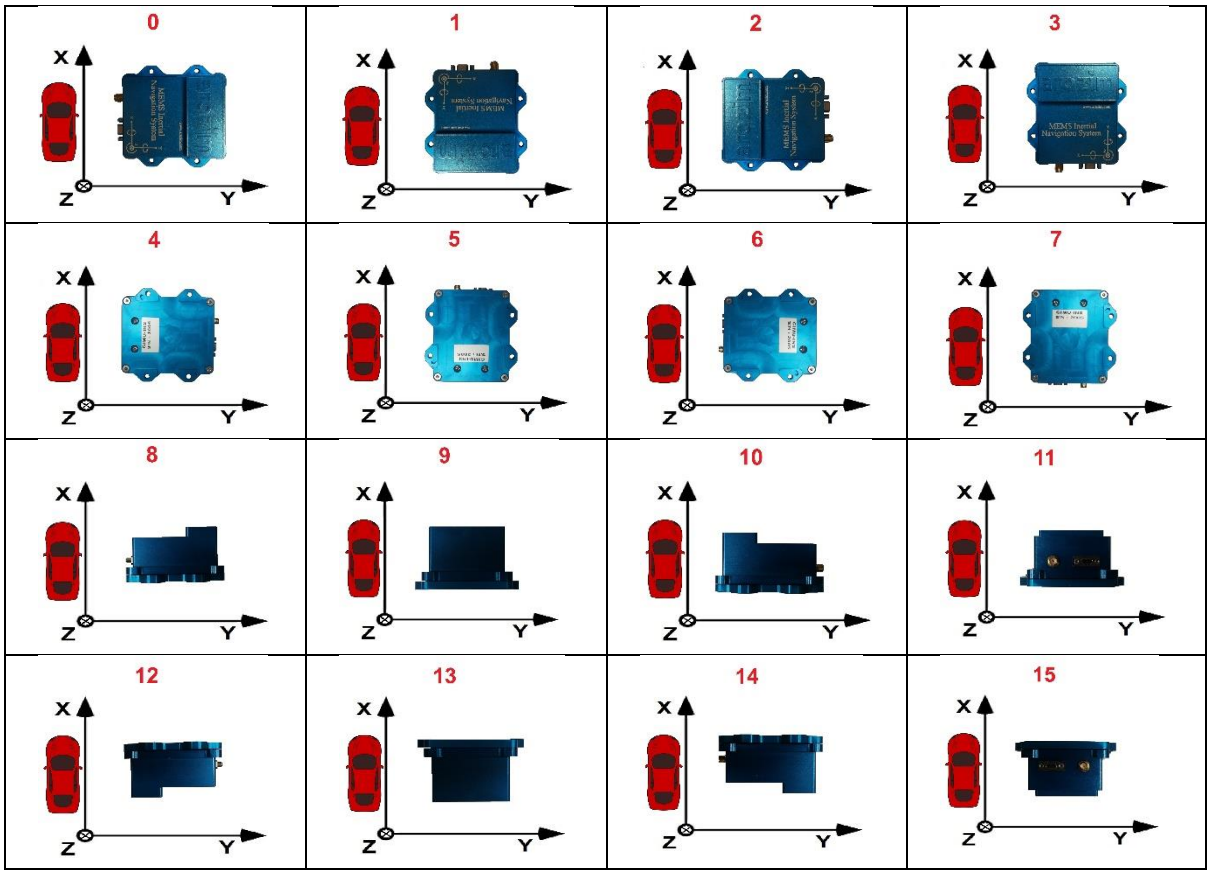
Use this command to read the last defined orientation settings, which determine how the navigation device is assembled with respect to the platform heading.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

**Table 16. Available Orientation Positions**




### 3.4.1.2. Read Number of Static Samples

Use this function to read the last number of static samples configured for bias-estimation calculation. This number can be any value between 1000 and 9000, which can be translated to time duration according to the sampling rate. For example, in a sampling rate of 100Hz, 2000 samples are equal to 20 seconds.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

### 3.4.1.3. Read IIR Filter Status

Use this function to read the last configuration activity status of the IIR Filter. The IIR filter is tuned for 10Hz with respect to the optional sampling rates. The user can activate or deactivate the use of the filter. By default, the IIR Filter is activated, during which a latency of three samples should be considered.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

#### 3.4.1.4. Read RPY Offsets

Use this command to read the last offset values determined for Roll, Pitch, and Yaw tilt results. The RPY offset values (which replace the device orientation settings) may be more specific, since the device orientation settings only support a rotation of 90°.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

#### 3.4.1.5. Read Lever-ARM settings

Use this command to read the last distance information of the primary antenna from the navigation device. When the antenna is not combined with the navigation device, you should provide the navigation device with information regarding the antenna position relative to the navigation device.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

Figure 1 shows additional information on the Lever-ARM.

#### 3.4.1.6. Set Device Orientation

Use this command to determine the actual orientation of the navigation device with respect to the heading of the platform on which the navigation device is assembled.

The orientation is critical for correct UKF results.

The command consists of two arguments:

- The first argument u8Orientation is of type uint8_t and you should set the Orientation index according to Table 16.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

#### 3.4.1.7. Set Number of Static Samples

Use this command to set the number of samples to collect at static condition in order to calculate the bias-estimation, either at power-up or when dedicated calibration commands are sent. The value entered may be between 1000 to 9000 samples.

The command consists of two arguments:

- The first argument u16Samples is of type uint16_t and you should set a value between 1000 and 9000.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device if the command was received successfully.

Section 3.5.3 provides additional information on bias-estimation.

#### 3.4.1.8. Set IIR Filter Activity Status

Use this command to determine whether to enable or disable the IIR filter on the IMU raw data. When the IIR filter is disabled, the results are noisier; when enabled, the results are cleaner and there is a latency of three samples.

The command consists of two arguments:

- The first argument bEnable is of type bool. Set it to “true” to enable the IIR filter or set it to “false” to disable the IIR filter.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

#### 3.4.1.9. Set RPY Offsets

Use this command to manually set the orientation of the navigation device with respect to the heading of the platform on which the navigation device is assembled.

The command consists of five arguments:

- The first argument bUseShiftValues is of type bool. Set it to “true” if the navigation device will use the manual feed offset values for Roll, Pitch, and Yaw, or set it to “false” so the device does not use the manual feed offset values for Roll, Pitch, and Yaw.
- The second argument fRoll is of type float and you should feed the offset value, in degrees, for the Roll.
- The third argument fPitch is of type float and you should feed the offset value, in degrees, for the Pitch.
- The fourth argument fYaw is of type float and you should feed the offset value, in degrees, for the Yaw.

- The last argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

### 3.4.1.10. Set Lever-ARM Settings

Use this command to set the distance of the GPS primary antenna from the navigation device in three dimensions.

The command consists of five arguments:

- The first argument bUseLeverArm is of type bool. Set it to “true” if the navigation device will use the given manual feed lever-arm distance values from the primary antenna, or set it to “false” so that the navigation device does not use the distance values from the primary antenna (in which case, the distance values will be: 0,0,0).
- The second argument: x, is of type float, and you should feed the distance between the center of the navigation device to the center of the GPS primary antenna, according to the navigation x-axis direction.
- The third argument: y, is of type float, and you should feed the distance between the center of the navigation device to the center of the GPS primary antenna, according to the navigation y-axis direction.
- The fourth argument: z, is of type float, and you should feed the distance between the center of the navigation device to the center of the GPS primary antenna, according to the navigation z-axis direction.
- The last argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

| Note | If the antenna is in front of the navigation device x-axis, the X distance is positive.<br>If the antenna is in front of the navigation device y-axis, the Y distance is positive.<br>If the antenna is above the navigation device, the Z distance is negative. |
|---|---|

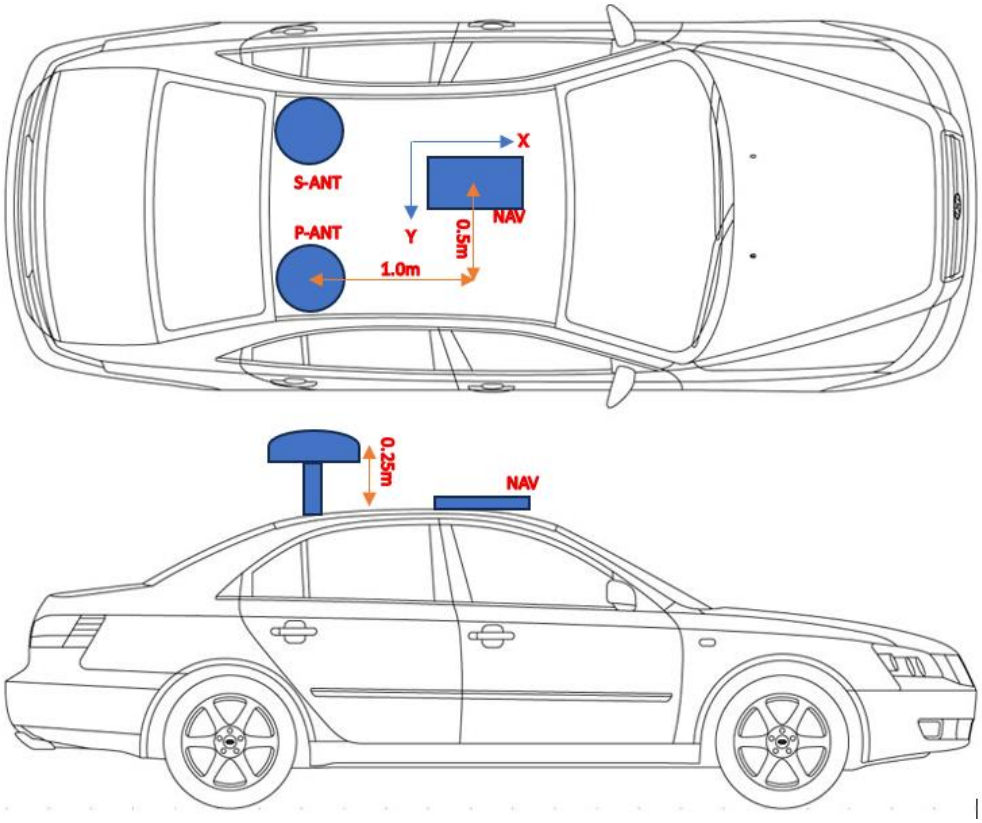



**Figure 1. Lever-ARM Sample Distance Measurement**

Figure 1 shows:

- The primary antenna is in the negative direction of the navigation x-axis, X distance = -1.0m.
- The navigation y-axis points to the primary antenna location, Y distance = +0.5m.
- The primary antenna is above the navigation platform, Z distance = -0.25m

## 3.5. Calibration Group Functions

Calibration group commands handle Bias-Estimation calibration (on all devices) and Hard-Iron calibration (on INS and AHRS devices).

You can only use Calibration group commands when you are in Calibration mode.

Table 17 lists the Calibration group commands.

**Table 17. Calibration Group Functions**

| Index | Function Name | Description |
|---|---|---|
| General Control Commands | | |
| 1 | Reset to Factory | Changes the device configuration to the factory default settings |
| | `uint8_t navSdk_Cal_ResetToFactory(TstCommandBuff* pstBuff);` | |
| 2 | Save Settings | Saves current configuration in local flash |
| | `uint8_t navSdk_Cal_SaveAsDefault(TstCommandBuff* pstBuff);` | |
| Bias-Estimation Calibration Commands | | |
| 3 | Start/Stop Bias-Estimation | Starts/Stops collecting data for bias estimation |
| | `uint8_t navSdk_Cal_StartBiasEstProc(bool bStartBiasProc, TstCommandBuff* pstBuff);` | |
| 4 | Get Bias-Estimation Calibration coefficients. | Reads the first block of bias estimation coefficients |
| | `uint8_t navSdk_Cal_Read_BiasEstCal(TstCommandBuff* pstBuff);` | |
| 5 | Set Bias-Estimation Calibration coefficients. | Updates the first block of bias estimation coefficients |
| | `uint8_t navSdk_Cal_SetBiasEstCal(double* ardBiasCal, TstCommandBuff* pstBuff);` | |
| 6 | Get Bias-Estimation Extra Calibration coefficients. | Reads the second block of bias estimation coefficients |
| | `uint8_t navSdk_Cal_Read_BiasEstExtraCal(TstCommandBuff* pstBuff);` | |
| 7 | Set Bias-Estimation Extra Calibration coefficients. | Updates the second block of bias estimation coeffcients. |
| | `uint8_t navSdk_Cal_SetBiasEstExtraCal(double* ardBiasExtCal, TstCommandBuff* pstBuff);` | |
| Hard-Iron Calibration Commands | | |

| Index | Function Name | Description |
|---|---|---|
| 8 | Start/Stop H.I. Calibration | Starts/Stops H.I. calibration |
| | `uint8_t navSdk_Cal_StartHIProc(bool bStartHIProc, TstCommandBuff* pstBuff);` | |
| 9 | Get H.I Calibration coefficients | Gets H.I. calibration coefficients |
| | `uint8_t navSdk_Cal_Read_HICal(TstCommandBuff* pstBuff);` | |
| 10 | Set H.I. Calibration coefficients | Sets H.I. calibration coefficients |
| | `uint8_t navSdk_Cal_SetHICal(double* ardHICal, TstCommandBuff* pstBuff);` | |

### 3.5.1. Reset to Factory

Use this command to restore the device's default factory settings, which are in a different memory location that you cannot change.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully)

### 3.5.2. Save Settings

Use this command to save all updated settings to the local flash to be used as default settings on the next powerup. When configuration instructions determine system behavior, all changes are temporary and influence the active registers in memory. Therefore, the settings must be saved if you plan to use them in the next powerup.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully)

### 3.5.3. Bias-Estimation Calibration Overview

A compensation algorithm compensates for sensor biases. By default, navigation systems perform the bias estimation calculation immediately after powerup in static conditions. The system retains the calibration in memory registers until the next power cycle.

If the bias estimation was not performed on power-up, you can still calculate it by sending the Start Bias Estimation instruction.

When a predefined number of samples have been collected (or when a Stop Bias Estimation command is sent to the system), the measurement collection process ends and the bias calculation begins.

If the Bias Estimation calculation succeeds, the Burn to Flash command should be executed to save the new calibration values.

You can read calibration values, using the Get Bias Estimation Calculation (two data structures). You can then save the results on the host controller and rewrite them if required, using the Set Bias Estimation Calculation and Set Bias Estimation Extra Calculation commands.

During the calibration process, the navigation system sends the following status messages (see more information on Status Message in Section: 7.1 )

1. smStartStaticCal - when the Start Bias Estimation Calculation command has been received
2. smInStaticCal – during data collection, the device sends the percentage of measurements processed
3. smDoneStaticCal - when the Bias Estimation calibration succeeds.
4. smStaticCalError - when the Bias Estimation calibration fails.

| Note | All platforms use bias estimation, which should always be run under static conditions.<br>It is best to save the new calibration values since, when the system exits the Calibration mode and reverts to Real-Time mode, it undergoes an automatic Soft Reset. |
|---|---|

The bias estimation modes are shown in Figure 2 (default) and Figure 3 (manual).

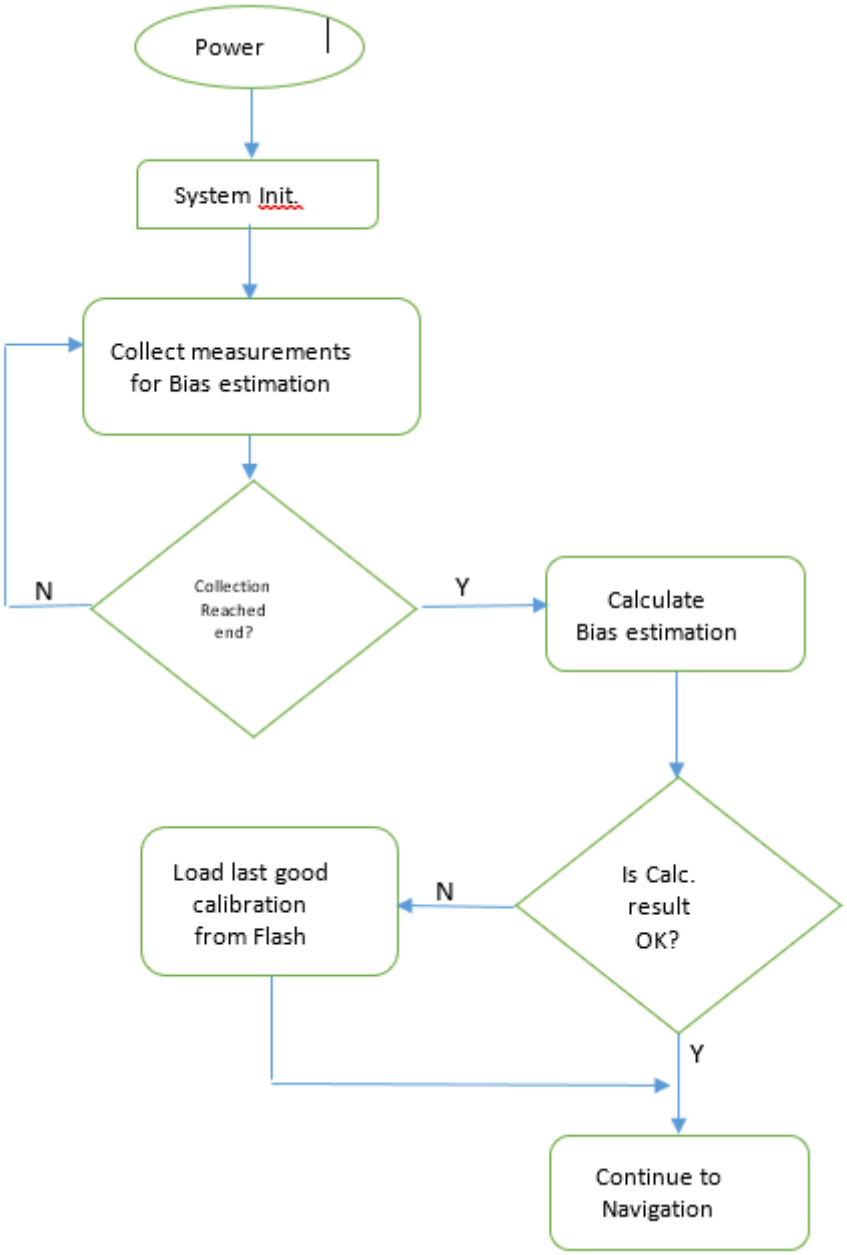



**Figure 2. Bias Estimation Default Power-up Flow**

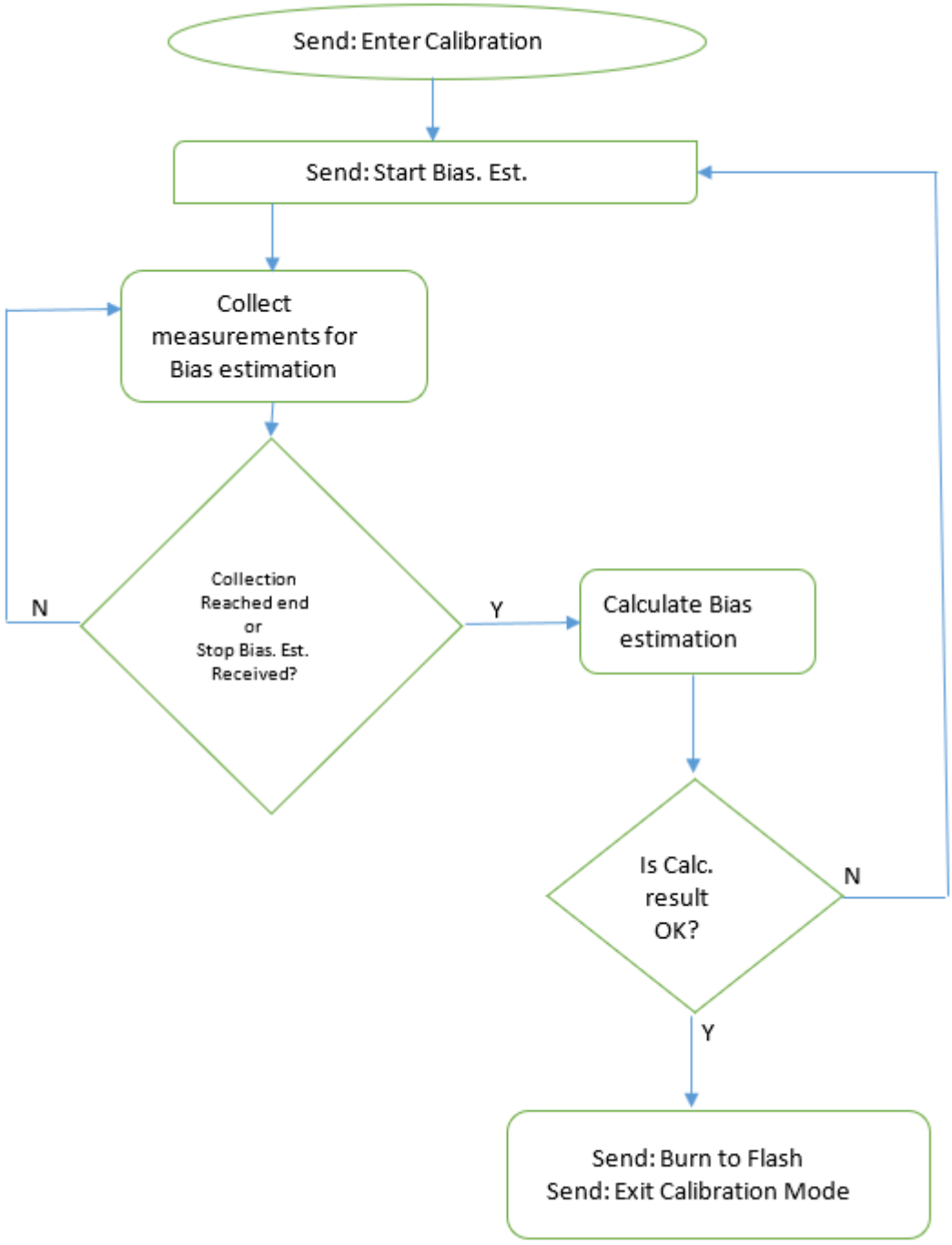



**Figure 3. Manual Bias Estimation Flow**

### 3.5.4. Start/Stop Bias-Estimation

Use this command to start or stop the execution of a bias estimation calibration when manual calibration is required.

The command consists of two arguments:

- The first argument is of type Boolean. Set to “true” to build a Start Bias-Estimation command or to “false” to build a Stop Bias-Estimation command.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

### 3.5.5. Get Bias-Estimation Calibration Coefficient

Use this command when the bias estimation process has finished to read the calibration results coefficients. You can use these coefficients later for post-processing or for manual reprogramming.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

### 3.5.6. Set Bias-Estimation Calibration Coefficient

Use this command to write a pre-loaded bias-estimation calibration coefficient to the navigation device.

The command consists of two arguments:

- The first argument consists of an array of 27 float values for VG/AHRS navigation devices and 28 float values for INS navigation devices.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

### 3.5.7. Get Bias-Estimation Extra Calibration Coefficients

Use this command, after a bias estimation procedure has ended, to read the calibration results coefficients. These values can be used later for post-processing or manual reprogramming.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

### 3.5.8. Set Bias-Estimation Extra Calibration coefficients

Use this command to write to the navigation device a pre-loaded bias-estimation extra calibration coefficient.

The command consists of two arguments:

- The first argument consists of an array of 19 float values for all navigation devices.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

### 3.5.9. Hard-Iron Calibration Overview

Navigation systems that use magnetometer sensors (AHRS, INS) must compensate for the magnetic environment surrounding the system, at least for the first installation.

The navigation system comes with instructions for performing, reading, and writing hard-iron calibration.

Hard-iron (H.I) calibration is a manual operation. In Calibration mode, send the Start H.I calibration command, rotate the navigation system platform at least 360° (recommended 720°) horizontally (make sure that the rotating rate is not too fast <=20deg/sec), and send the "Stop H.I calibration" command. When the calibration succeeds, send the Save Settings command to save the current calibration results.

During the calibration process, the connected system should send the following status messages (Section: 7.1 contains additional information on status messages):

1. smStartHICal – When the Start H.I Calculation command has been received
2. smInHiCal – during data collection, the percentage of measurements processed
3. smDoneCal – when the H.I calibration calculation succeeds
4. smHICalError – when the H.I calibration calculation fails

The Get Hard-Iron Calibration command reads the results to the host controller and lets you save them on a file. If required, the results can be rewritten to the device, using the complementary Set Hard-Iron Calibration command.

| Note | When you exit the Calibration mode, the connected navigation system performs a soft reset. |
|---|---|

### 3.5.10. Start/Stop Hard-Iron (H.I) Calibration

Use this command to start or stop the execution of a Hard-Iron calibration when manual calibration is required.

The command consists of two arguments:

- The first argument is of type Boolean. Set to “true” to build a Start Hard-Iron command or “false” to build a Stop Hard-Iron command.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

### 3.5.11. Get Hard-Iron (H.I.) Calibration

Use this command to read the current hard-iron calibration coefficient from the connected navigation device to the Host controller memory.

The command consists of one argument:

- The argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a GET command, you can expect a command reply from the device if the command was received successfully.

### 3.5.12. Set Hard-Iron (H.I) Calibration

Use this command to write a pre-loaded hard-iron calibration coefficient to the navigation device.

The command consists of two arguments:

- The first argument consists of an array of 12 float values for all navigation devices.
- The second argument is a returned structure that holds an array of bytes with the constructed command that you should send to the device.

Since this is a SET command, you can expect an acknowledgement reply from the device (if the command was received successfully).

# 4. MISCELLANEOUS LIBRARY FUNCTIONS

## 4.1. Library Functions to Save Data Messages to a File

In addition to the basic functionality described in Section 2.2, and the control function described in Section 3, the library contains instructions for saving incoming data and status messages to a binary file, which you can later export to a comma-delimited ASCII file.

**Table 18. List of Commands for Saving Data and Status Messages**

| Index | Function Name |
|---|---|
| 1 | uint8_t navSdk_CreateDestFile(char* fName); |
| 2 | uint8_t navSdk_StartSave(void); |
| 3 | uint8_t navSdk_PauseSave(void); |
| 4 | uint8_t navSdk_StopSave(void); |
| 5 | uint8_t navSdk_FinalizeDestFile(void) |

The information in this section refers to the function index numbers in Table 18. These functions save incoming data and status messages on the host computer.

Function 1 creates a destination file with a given file name in a requested path. This function delivers a full file name and its path as an argument. It returns a value of "0" if delivery succeeds, and one of the error messages listed in Table 19 if delivery fails.

**Table 19. Error Messages When the navSdk_CreateDestFle Function Fails**

| Error Result | Error Information |
|---|---|
| 1 | If the function was called twice without calling function number 5 to finalize the first call. |
| 2 | If the function could not create a file for writing at the requested directory. |
| 255 | If you tried to call this function before calling the basic function to create the handler object (see: **2.2.1 - Basic Function – navSdk_CreateCNavSdkObject**) |

Once a file has been created successfully, use Functions 2-4 to start, pause, and stop file recording. These functions do not get any arguments, and the navigation device replies with an Acknowledge response for each message.

Function 2 starts saving incoming data (any data written to the main object with the **navSdk_WriteData** command, analyzed as either a data message or a status message) to the file.

Function 3 pauses the file recording. Any information received after this command will not be added to the file. You can continue saving the file by recalling Function 2.

Function 4 stops the file recording, after which you will no longer be able to continue saving the file. After calling Function 4, call Function 5 to close the file correctly and perform any necessary cleanup.

## 4.2. Library Functions to Export Binary Files to ASCII File Format

Section 4 described the functions to create, save data, and save status messages to a binary file. This chapter describes the library functions that export the binary file to a readable ASCII file format that any application can use. Table 20 lists the commands used to export a binary file to ASCII file format.

**Table 20. List of Commands to Export Binary File to ASCII File Format**

| Index | Function Name |
|---|---|
| 1 | uint8_t navSdk_ExportFile(const char* srcFile, const char* dstInitials); |
| 2 | void navSdk_SetOnFileExportStatus_CallBack(TOnFileExport_Status cbOnFileExportStatus); |

The information in this section refers to the function index numbers in Table 20

Function 1, which exports the file, consists of the following arguments:

1. The name of the source binary file
2. The name of the destination ASCII file

Since you can configure the broadcast of several message types, the export function creates a different destination file extension for each data message (for example, *.GPS1 for a GPS1 message or *.IMU2 for an IMU2 message).

The entire export takes place in a dedicated Thread.

The function returns a value of “0” if the export succeeds, ??? if the export fails.

To check the status of the export process, create a function from the **TOnFileExport_Status** template and deliver a pointer from that function to the **navSdk_SetOnFileExportStatus_CallBack** function.

## 4.3. Library Functions to Simulate Receipt of Data Messages

The library also includes a list of instructions that can assist a programmer to verify that the data messages reception functions work well without connecting to a real navigation device. Each data message type has a matching simulation function that can be fired on request, which checks the entire route that a real message passes without the physical communication channel.

The list of instructions is given in Table 21.

**Table 21. List of Instructions to Simulate Message Reception**

| Index | Function Name |
|---|---|
| 1 | bool navSdk_SimulateGPS_1_Msg(void); |
| 2 | bool navSdk_SimulateGPS_2_Msg(void); |
| 3 | bool navSdk_SimulateGPS_3_Msg(void); |
| 4 | bool navSdk_SimulateGPS_4_Msg(void); |
| 5 | bool navSdk_SimulateIMU_1_Msg(void); |
| 6 | bool navSdk_SimulateIMU_2_Msg(void); |
| 7 | bool navSdk_SimulateIMU_3_Msg(void); |
| 8 | bool navSdk_SimulateIMU_4_Msg(void); |
| 9 | bool navSdk_SimulateVG_1_Msg(void); |
| 10 | bool navSdk_SimulateVG_2_Msg(void); |
| 11 | bool navSdk_SimulateVG_3_Msg(void); |
| 12 | bool navSdk_SimulateVG_4_Msg(void); |
| 13 | bool navSdk_SimulateINS_1_Msg(void); |
| 14 | bool navSdk_SimulateINS_2_Msg(void); |
| 15 | bool navSdk_SimulateINS_3_Msg(void); |

The simulation messages use constant values to make it easy to verify that the data was received with no errors. The tables below list the values that should be received from each message.

Section 7.3 contains information on the exact message structure.

**Table 22. Default Parameter Values for All Messages**

| Msg Fields | Value |
|---|---|
| Group | sysTesting = 0x80 |
| MsgType | Message unique index (see: Table 38) |
| Flag-1 | 0xA1 |
| Flag-2 | 0xA2 |
| Flag-3 | 0xA3 |
| MsgIndex | 1 |
| TimeTag | 1228 |
| Temperature | 35.4 |

### 4.3.1. For IMU Simulation Messages

**Table 23. Default Parameter Values Specific to IMU Messages**

| Msg Fields | IMU-1 | IMU-2 | IMU-3 | IMU-4 |
|---|---|---|---|---|
| MsgType | sstIMU1 | sstIMU2 | sstIMU3 | sstIMU4 |

For MsgType numeric value, see: Table 38.

The remaining floating-point fields will hold the value 0.1.
Other Temperature fields will hold the value 25.1.

### 4.3.2. GPS Simulation Messages

**Table 24. Default Parameter Values Specific to GPS Messages**

| Msg Fields | | GPS-1 | GPS-2 | GPS-3 | GPS-4 |
|---|---|---|---|---|---|
| MsgType | | sstGPS1 | sstGPS2 | sstGPS3 | sstGPS4 |
| Date | 20190506 | V | V | V | V |
| Time | 130430.30 | V | V | V | V |
| Lat\EcefX | 31.9 | V | V | --- | --- |
| Lon\EcefY | 34.8 | V | V | --- | --- |
| Alt\EcefZ | 80.0 | V | V | --- | --- |

For MsgType numeric value, see: Table 38

The remaining floating-point fields will hold the value 0.1.
All the rest of the integer fields will hold the value 1.

### 4.3.3. Tilt Simulation Messages

**Table 25. Default Parameter Values Specific to Tilt Messages**

| Msg Fields | Tilt-1 | Tilt-2 | Tilt-3 | Tilt-4 |
|---|---|---|---|---|
| MsgType | sstVG1 | sstVG2 | sstVG3 | sstVG4 |

For MsgType numeric value, see: Table 38

All other fields in each Tilt message will hold the value 0.1.

### 4.3.4. For INS Simulation Messages

**Table 26. Default Parameter Values Specific to INS Messages**

| Msg Fields | | INS-1 | INS-2 | INS-3 |
|---|---|---|---|---|
| MsgType | | sstINS1 | sstINS2 | sstINS3 |
| Lat | 31.9 | V | V | V |
| Lon | 34.8 | V | V | V |
| Alt | 80.0 | V | V | V |
| Date | 20190506 | --- | --- | V |
| Time | 130430.30 | --- | --- | V |

For MsgType numeric value, see: Table 38

The remaining floating-point fields will hold the value 0.1.

# 5. COMMUNICATION WITH THE NAVIGATION PLATFORM

The API instructions control (read or write) the behavior of the device connected to the host computer (by default, through the RS232 communication channel).

In general, you will configure the settings of a device at least once before using the device. Afterwards, you will just receive the incoming data messages.

As was indicated in the previous chapters, you can choose to use the alternative functions or a single function **(navSdk_BuildCommand)** with a dedicated data structure as an argument, to build all the device API (GET/SET) commands. This chapter focuses on the correct use of the **navSdk_BuildCommand** command**.**

## 5.1. Build Commands

The SDK functions are wrappers for API commands that you may decide to implement by yourself (see: "NAV API Manual.pdf").

The API commands may be either GET commands or SET commands, defined as follows:

- GET commands – commands used to read the current settings, i.e., commands that inquire settings values and connected navigation device replies, with the API Command Reply that includes the requested information.
- SET commands – commands used to write new setting values to the connected navigation device. Commands of type SET may enforce a process (reset, save to flash, and so on or set new settings values. For any command of type SET, the connected device replies with API acknowledge reply, which indicates that the last command was received.

The library file includes operational commands and a single generic command (**navSdk_BuildCommand**) that can be used to create all the system commands from type GET (reading current settings) and type SET (writing new settings). This command argument is a large structure of type `TstCommand` that should hold the command ID and the relevant parameters (if needed) to create the navigation API command. The navSdk_BuildCommand gets another argument of type `TstCommandBuff`, which holds the API command, created from the first argument. This buffer result should be sent to the navigation device.

The following codes show the build command function template and the structures used as arguments:

**Code 6. Build Command Function Template**

```
uint8_t navSdk_BuildCommand(TstCommand* pCmnd, TstCommandBuff* pstBuff);
```

### Code 7. Build Command – Input Argument Structure (TstCommand)

```
typedef struct {
        uint8_t         DeviceType;   //0 = system\platform device, >=0x20 Device
        uint8_t         DeviceIndex;  //0 = all devices from DeviceID type
        uint8_t         CommandID;
        bool            bAcknowledge;
        float           fRes;           //Reserved
        TstAlgo_UKF             Algo;
        TstCalibration          Calib;
        TstGlobalSystemSettings Sys;
        TstDeviceSettings       Device;
 } TstCommand;
```

### Code 8. Build Command – Output Argument Structure (TstCommandBuff)

```
typedef struct {
        uint8_t         ucaBuffer[250];
        uint8_t         ucBuffLen;
        uint8_t         eResult;        //(uint8_t)TeVerifyResult
} TstCommandBuff;
```

The first three fields in the input data structure (TstCommand), make up the API command identification, while the rest of the fields (four additional data structures Tst…) make up the optional command parameters, used according to the DeviceType parameter field and CommandId parameter field.

**The 1st field – Device Type or Group Name** represents the assignment group of the API commands. The assignment group of a command can be used for configuration, calibration, and other sub devices. **Code 9** contains a full list of optional device type values in the TeGroup.

**The 2nd field – Device Index** – represents a specific device when there is more than one device of the same type in the system (e.g., two cameras in the same system). If there is only one device or if the command is intended for all devices of that type, this field should be 0.

**The 3rd field – Command ID** – defines the specific API command according to the DeviceType index indicated in the first field.

Each DeviceType includes a specific enumeration for CommandId, adjusted to the task and the parameter structures.

For the DeviceType specific API Commands list see:

- API command enumeration for Immediate –**Code 10**.
- API command enumeration for Configuration –**Code 11**
- API command enumeration for Calibration –**Code 13**.
- API command enumeration for Device –**Code 15**

**The 4th field – bAcknowlegde** – is used as a return value in the API acknowledge reply command for any API command of type SET.

**Code 9. Device Type/Group Name – Enumeration List (TeGroups)**

```
typedef enum : uint8_t
{
        grpSystem = 0x00,
        grpImidiate = 0x01,
        grpConfiguration = 0x10,
        grpCalibration = 0x11,

        grpDevice_Gps = 0x20,           //not currently in use
        grpDevice_Odometer = 0x21,      //not currently in use
        grpDevice_Lidar = 0x22,         //not currently in use
        grpDevice_Camera = 0x23,        //not currently in use
        grpDevice_Acceleration = 0x24,
        grpDevice_Gyros = 0x25,
        grpDevice_Magnetometers = 0x26, //not currently in use
        grpDevice_Barometer = 0x27,     //not currently in use
        grpDevice_Temperature = 0x28,   //not currently in use
        grpDevice_Adc = 0x29,           //not currently in use
        grpDevice_Imu = 0x2A,

        grpAlgo_Ukf = 0x50,
        grpMsgs_Testing = 0x80,
        grpMsgs_sysExigoVG = 0x81,
        grpMsgs_sysExigoAHRS,
        grpMsgs_sysExigoINS,
        grpMsgs_sysExigoINS2Ant,
        grpMsgs_sysGinsVG = 0x85,
        grpMsgs_sysGinsAHRS,
        grpMsgs_sysGinsINS,
        grpMsgs_sysGinsINS2Ant,
        grpMsgs_sysArsys05 = 0x89,
        grpMsgs_sysMiniExiguoINS = 0x8A,

        grpMsgs_sysExiguoLPVG = 0x8B,
        grpMsgs_sysExiguoLPAHRS,
        grpMsgs_sysExiguoLPINS,
        grpMsgs_sysExiguoLPINS2Ant,
        grpMsgs_sysGIMU3 = 0xa0

 } TeGroups;
```

## 5.2. API Commands Group

The enumeration TeGroups represented in the DeviceType field (see: **Code 9**), defines sub-sections as follows:

- Section 1 - API commands, whose DeviceType field value is 0x01 to 0x1F are instructions to approach global system registers.
- Section 2 - API commands, whose DeviceType field value is 0x20 to 0x4f are instructions to approach specific sub-devices in a navigation device (although not many are implemented at this stage).
- Section 3 - API commands whose DeviceType field value is 0x50 to 0x5F are instructions to approach the navigation algorithm registers.

The first section DeviceType values relates to: grpImidiate, grpConfiguration and grpCalibration.

### 5.2.1. Group Section 1 - API Commands under grpImidiate

The API commands under grpImidiate (**Code 10**) are special since you can send any of them to the connected navigation device while the connected

device is working in real-time mode. The related commands in this group are mainly used to switch between system modes to allow control over the settings, calibration routines, and sub-device register values. When using API commands of any other group (DeviceType), the navigation device should be in Configuration mode or Calibration mode.

**Code 10. Device Type/Group Name – Enumeration List** (`TeGroups`)

```
typedef enum : uint8_t //Ver. 1.1
{
        imEnableConfig = 0x01,
        imEnableCalibration,
        imGetDeviceName,
        imGetSample,
        imEnterResetProtect,
        imExitResetProtect
} TeAPI_Imidiate;
```

### 5.2.2. Group Section 1 - API Commands Under grpConfiguration

The API commands under grpConfiguration (see: **Code 11**) are used for general device settings which are relevant to all navigation devices. The commands allow control of the active data messages and their rates, the communication channel baud-rate, and more.

This list of commands requires parameters that are supplied through the **Sys** field, which is of structure type `TstGlobalSystemSettings` (**Code 12**).

To use this list of commands, first switch from Real-Time mode to Configuration mode, using the Enter Configuration Mode command (grpImidiate -> imEnableConfig). When the navigation device enters configuration mode, it stops sending data messages, which results in a clean working communication channel.

**Code 11. grpConfiguration – API Command Enumeration List**

```
typedef enum : uint8_t
{
        ecReset2Factory = 0x01,
        ecSoftReset,
        ecBurn2Flash,
        ecGetHWVer = 0x04,
        ecGetFWVer,
        ecGetDeviceType,
        ecGetDeviceName,
        ecGetDeviceVersion,
        ecSetSerialNum = 0x09,                          //factory use only
        ecGetSerialNum,
        ecSetFrameTypeAndRate = 0x0b,   ecGetFrameTypeAndRate,
        ecSetBaudRate = 0x0d,           ecGetBaudRate,
        ecSetSampleMode = 0x0f,         ecGetSampleMode,
        ecSetOperationMode,             ecGetOperationMode,
        ecSetCommType = 0x13,                           //Only for Arsys

        ecGetCommType,
        ecSetCommCh = 0x15,                             //Only for Arsys
        ecGetCommCh,
        ecSetEnableAckReply = 0x17,     ecGetEnableAckReply,
        ecSetSampleRate = 0x19,         ecGetSampleRate,
        ecSetCommChAndBaud = 0x1B,      ecGetCommChAndBaud,
        ecSetTransparency = 0x30
        ecStartGPSMsgsTest,             ecGetGPSMsgsTestRes //factory use only
 } TeAPI_Configuration;
```

**Code 12. Global System Settings Structure**

```
typedef struct {
        bool            bUseAcknowledge;
        bool            bEnterConfiguration;
        bool            bEnterCalibration;
        uint8_t         eSystemStaus;
        TarInfo         HWVersion;
        TarInfo         FWVersion;
        TarInfo         SerialNum;
        TarInfo         DeviceType;
        TarInfo         DeviceName;
        TarInfo         DeviceVersion;
        uint8_t         FrameType;
        uint8_t         FrameRate;
        uint8_t         OperationMode;
        uint8_t         CommType;
        uint32_t        CommSpeed;
        uint8_t         SampleMode;
        uint8_t         CommCh;                 //irrelevant
        uint8_t         Res1, Res2;             //reserved
        uint16_t        SampleRate;             //
        uint8_t         Res3;                   //reserved
        uint8_t         GPS_Msgs_Ok;            //factory use only
        uint32_t        ui32Res5;               //reserved
} TstGlobalSystemSettings;
```

### 5.2.3. Group Section 1 - API Commands under grpCalibration

The API commands under grpCalibration (see: **Code 13**) are used for calibration purposes, depending on the platform, which may be:

- Bias Estimation calibration –relevant to all platforms.

- Hard-Iron calibration –relevant to navigation devices whose algorithm uses the magnetometers (AHRS, INS).

In this mode, the connected device can perform a specific calibration, use commands to read the current calibration coefficients, or write predefined calibration coefficients (taken earlier).

In order to use this list of commands, switch from Real-Time mode to Calibration mode by using the Enter Calibration Mode command (grpImidiate -> imEnableCalibration). When the navigation device enters Calibration mode, it stops sending data messages, which results in a clean working communication channel.

The parameters that this list of commands require are supplied through the Calib field, which is of structure type `TstCalibration` (See:Code 14).

**Code 13. grpCalibration – API Command Enumeration List**

```
typedef enum : uint8_t
{
        ccReset2Factory = 0x01,
        ccBurn2Flash,
        ccStartStaticCal = 0x03, ccStopStaticCal,
        ccGetStaticCalArray,     ccSetStaticCalArray,
        ccGetStaticExCalArray,   ccSetStaticExCalArray,
        ccStartHICal = 0x09,     ccStopHI,       ccGetHICal,     ccSetHICal
} TeAPI_Calibration;
```

**Code 14. Calibration Parameter Structure**

```
typedef struct {
        double  arCalBiasEstExt[CMAX_BiasEstExtCoef];
        float   arCalBiasEst[CMAX_BiasEstCoef];
        float   arCalHI[CMAX_HICoef];
} TstCalibration;
```

### 5.2.4. Group Section 2 – Configure Sub-Devices Settings

In this group section, it is possible to control many optional devices in the future through API commands. Such devices may be: GPS receiver, Odometer, Barometer, and others. This section gathers several basic commands that should work with all future devices.

contains a list of enumeration device commands.

To use this list of commands, switch from Real-Time mode to Configuration mode by using the Enter Configuration Mode command (grpImidiate -> imEnableConfiguration). When the navigation device enters Configuration mode, it stops sending data messages, which results in a clean working communication channel.

The parameters that this list of commands require are supplied through the Device field which is of structure type `TstDeviceSettings` (**Code 16**).

### Code 15. grpDevice – API Command Enumeration List

```
typedef enum : uint8_t
{
        dvGetNumDevices = 0x01,
        dvResetDevice,
        dvSetDeviceType = 0x03,                          //Factory use only
        dvGetDeviceType,
        dvSetDeviceName = 0x05,                          //Factory use only
        dvGetDeviceName,
        dvSetDeviceVer = 0x07,                           //Factory use only
        dvGetDeviceVer,
        dvSetEngUnitsCal = 0x09,dvGetEngUnitsCal,        //Factory use only
        dvSetMatrixCal = 0x0B,   dvGetMatrixCal,         //Factory use only
        dvSetAlarmVals = 0x0D,   dvGetAlarmVals,
        dvSetWorkingRange,       dvGetWorkingRange,
        dvSetSampleRate,         dvGetSampleRate,
        dvSetDeviceActiveState,  dvGetDeviceActiveState, //irrelevant
        dvSetAlarmsActiveState,  dvGetAlarmsActiveState,
        dvSetCommType,           dvGetCommType,          //irrelevant
        dvSetCommCh,             dvGetCommCh,            //irrelevant
        dvSetCommSpeed,          dvGetCommSpeed,         //irrelevant
        dvSetCommParam,          dvGetCommParam          //irrelevant for Exiguo
              } TeAPI_Device;
```

### Code 16. Device Configuration Structure

```
typedef struct {
        TarInfo         DeviceType;
        TarInfo         DeviceName;
        TarInfo         DeviceVersion;
        float           frefTemperature;
        double          dSlope;                          //irrelevant
        double          dOffset;                         //irrelevant
        TarMatrix3x4    Matrix3x4;                       //Factory use only
        TarMatrix3x3    Matrix3x3;                       //Factory use only
        TarMatrix1x3    Matrix1x3;                       //Factory use only
        double          dMaxAlarmValue;
        double          dMinAlarmValue;
        bool            bActiveAlarms;
        uint8_t         ucAlarmTypes;
        int16_t         WorkingRange;
        uint32_t        SampleRate;
        bool            DeviceEnabled;
        bool            MaxAlarmEnabled;
        bool            MinAlarmEnabled;
        TeCommType      eCommType;
        uint8_t         ucFrameType;
        uint8_t         ucMatrixType;                    //Factory use only
        uint16_t        usFrameRate;
        uint8_t         CommChannel;                     //irrelevant
        uint8_t         ucRes4, ucRes5, ucRes6;          //Reserved
        uint32_t        CommSpeed;
        void*           pDeviceParams;                   //irrelevant
} TstDeviceSettings;
```

### 5.2.5. Group Section 3 – Configure Algorithm Settings

The third section deals with the algorithm configuration. **Code 17** contains the API enumeration command list.

In order to use this list of commands, switch from Real-Time mode to Configuration mode by using the Enter Configuration Mode command (grpImidiate -> imEnableConfiguration). When the navigation device enters configuration mode, it stops sending data messages, which results in clean working communication channel.

The parameters that this list of commands require are supplied through the Device field which is of structure type `TstAlgo_UKF` (**Code 18**).

**Code 17. grpAlgo_Ukf – API Command Enumeration List**

```
typedef enum : uint8_t
{
        ukfGetAlgoName = 0x01,                          //irrelevant
        ukfGetAlgoVersion,                              //irrelevant
        ukfSetOrientation = 0x03,       ukfGetOrientation,
        ukfSetStaticSamples,            ukfGetStaticSamples,
        ukfSetIIRFilterStatus = 0x07,   ukfGetIIRFilterStatus,
        ukfSetUseMagneticStatus = 0x09, ukfGetUseMagneticStatus,
        ukfSetShiftValues = 0x0B,       ukfGetShiftValues,
        ukfSetLeverArmValues = 0x0D,    ukfGetLeverArmValues,
        ukfSetAntDistance = 0x0F,       ukfGetAntDistance,
        ukfSetStartPos = 0x11,          ukfGetStartPos,
        ukfSetOptimizationBlockIndex,   ukfGetOptimizationBlockIndex,
        ukfSetFwdLeverArmValues,        ukfGetFwdLeverArmValues,
        ukfSetInitialHeading,           ukfGetInitialHeading
} TeAPI_Ukf;
```

**Code 18. Algorithm Configuration Structure**

```
typedef struct {
        TarInfo         AlgoVersion;                    //irrelevant
        TarInfo         AlgoName;                       //irrelevant
        uint8_t         IMUOrientation;                 //0-15
        uint8_t         EnableIIRFilter;                //0=off, 1=on
        uint16_t        StaticSamples;                  //between 1000-9000
        uint8_t         UseMagneticInSolution;          //for INS 2xAnt
        uint8_t         UseBiasRemove;
        uint8_t         UseLeverArm;
        uint8_t         UseFwdLeverArm;                 //EX300 only
        uint8_t         UseAntDistance;                 //Arsys only
        uint8_t         u8Resx[7];                      //reserved
        float           RollShift;
        float           PitchShift;
        float           HeadingShift;
        float           LeverArmX;
        float           LeverArmY;
        float           LeverArmZ;
        float           FwdLeverArmX;
        float           FwdLeverArmY;
        float           FwdLeverArmZ;
        float           AntennasDistance;
        double          dLat;
        double          dLon;
        float           fAlt;
        float           fHeading;
        uint8_t         OptimizationBlockIndex;
        uint8_t         u8Res[3];                       //reserved
        float           arOptimizationBlock[CMAX_OPBLOCK_COEF];
} TstAlgo_UKF;
```

# 6. GRPCONFIGURATION COMMANDS

The navigation device should be in Configuration mode to work with the grpConfiguration commands.

To enter Configuration mode, use the code in **Code 19**.

**Code 19. Enter Configuration Mode Example Code**

```
TstCommand       stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff   stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpImidiate;//Set the group of commands to indicate
                                    //Immidiate
stICommnad.DeviceIndex = 0;         //Device index is always 0 for the system
stICommnad.CommandID = imEnableConfig;   //Set the command ID - EnableConfig
stICommnad.Sys.bEnterConfiguration = 1;  //Set the command Value: 1 to Enter
                                         //Configuration or 0 to Exit Configuration


BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

The example in **Code 19** shows how to build a correct command to enter Configuration mode from Real-Time mode. The command will be received in stBuff array. All the following sample codes have, more or less, the same structure which is:

- Define stICommand of type `TstCommand` structure to hold the command and the parameters.
- Define stBuff of type `TstCommandBuff` structure to hold the result of the BuildCommand instruction, which is the API command that must be sent to the navigation device.
- Fill the relevant DeviceType, DeviceIndex and CommandID as the instruction identification.
- Fill (if necessary) the parameters that should follow the command in use.

In our specific case, we use grpImidiate as DeviceType and imEnableConfig as CommandID. The Enter Configuration command needs a parameter that indicates whether to enter or exit configuration. The parameter value should reside in the Sys structure in the bEnterConfiguration field. Set the value to 1 to enter Configuration mode or set the value to 0 to enter Real-Time mode from Configuration mode.

When the BuildCommand function has finished processing, the programmer should verify that the process succeeded by testing field eResult in stBuff structure. If eResult holds the value 0, the command has succeeded.

Next, use the API command data built in array field: ucaBuffer to send it to the navigation device connected via the communication channel.

The Enter/Exit configuration is considered a SET command, and the system will reply with an ACK Reply message.

In order to receive the ACK Reply message, assign a callback function from template `TOnApi_Acknowledge` (**Code 20**), using the function navSdk_SetonApiAck_CallBack (Section 2.2.5.1).

**Code 20. TOnApi Acknowledge – Calback Function Template**

```
typedef void(*TOnApi_Acknowledge)(void* pObj, TstCommand stCmndReply);
```

# 6.1. SET Commands

## 6.1.1. Reset2Factory – Configuration Instruction

This command is used to reset the current settings back to the factory settings. Any settings that you changed will revert to their factory default values.

**Code 21. Reset2Factory – Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpConfiguration;//Set the group of commands to
                                 //Configuration
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ecReset2Factory;  //Set the command ID – Reset2Factory
                                         //No Data values
BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

## 6.1.2. SoftReset – Configuration Instruction

This command resets the navigation device by using a software command. The navigation device will behave as if a power cycle occurred.

This command has a delayed execution, since- the reset will only take effect after exiting the Configuration mode and entering Real-Time mode.

Use **Code 21** to replace the CommandID line with the command in **Code 22**:

**Code 22. Reset2Factory Configuration Instruction**

```
stICommnad.CommandID = ecSoftReset;
```

## 6.1.3. Burn2Flash – Configuration Instruction

This command saves the current active settings in RAM to the Flash area to be used as default for the next power-up cycle. When you use SET commands to change settings, the settings are only changed in RAM. To use these new settings in the future, use a Burn2Flash command.

Use **Code 21**, and replace the CommandID line with the command in **Code 23**:

**Code 23. Burn2Flash Configuration Instruction**

```
stICommnad.CommandID = ecBurn2Flash;
```

### 6.1.4. SetFrameTypeAndRate – Configuration Instruction

This command is used to determine which messages the system will send and the rate at which they are sent. Each device includes a different number of messages types that you can select from the frame types (and indexes) available on each platform (Table 27 and Table 28).

**Code 24. SetFrameTypeAnd Rate Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpConfiguration; //Setthe group of commands to Configuration
stICommnad.DeviceIndex = 0;               //Device index is always 0 for the system
stICommnad.CommandID = ecSetFrameTypeAndRate;    //Set the command ID –

stICommand.Sys.FrameType = 0x20;         //frame type IMU1 – Raw Data
stICommand.Sys.FrameRate = 0x06;         //frame rate of 50Hz
BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Table 27. Message Types**

| Frame Type Index | Message Type | Platforms |
|---|---|---|
| 0x21 | IMU 2 - 6Dof | All Platforms |
| 0x22 | IMU 3 - 9Dof | All Platforms |
| 0x23 | IMU 4 - Misc. Sensors | All Platforms |
| 0x30 | GPS 1 - LLA | INS |
| 0x31 | GPS 2 - ECEF | INS |
| 0x32 | GPS 3 - Statistic | INS |
| 0x33 | GPS 4 - RTK | INS |
| 0x34 | GPS 5 - Sat. Info. | Not implemented |
| 0x40 | Tilt 1 – [R, P] | All Platforms |
| 0x41 | Tilt 2 – [R, P, Y] | All Platforms |
| 0x42 | Tilt 3 – [Quarter] | All Platforms |
| 0x43 | Tilt 4 – [DCM] | All Platforms |
| 0x50 | INS 1 – Tilt & Pos | INS Only |
| 0x51 | INS 2 – Tilt, Pos & Vel | INS Only |
| 0x52 | INS 3 – Tilt, Pos, Vel & Time | INS Only |
| 0x54 | INS 4 – Fwd Lever Arm | EX300 only |

**Table 28. Frame Rates**

| Frame Rate index | Rate (Hz) |
|---|---|
| 0x00 | Disable |
| 0x01 | 1 |
| 0x02 | 2 |
| 0x03 | 5 |
| 0x04 | 10 |
| 0x05 | 25 |
| 0x06 | 50 |
| 0x07 | 100 |
| 0x08* | 200 |
| 0x09* | 250 |
| 0x0A* | 500 |
| 0x0B* | 1000 |

### 6.1.5. SetOperationMode – Configuration Instruction

This command defines the use of the bias estimation calibration at power-up. By default, the navigation device is configured to calculate the bias estimation immediately at power-up. You can use this command to change the default mode so that, on power-up, the system will use the last calibration values that were saved earlier in flash. (**Code 25**).

Table 29 lists the optional parameter values.

**Code 25. SetOperationMode Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpConfiguration;//Set the group of commands to Configuration
stICommnad.DeviceIndex = 0;               //Device index is always 0 for the system
stICommnad.CommandID = ecSetOperationMode;       //Set the command ID –

stICommand.Sys.OperationMode = 0x00;     //use Bias estimation calibration from flash

BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

**Table 29. Optional Parameter Values for SetOperationMode**

| Operation Value Index | Description |
|---|---|
| 0 | On power up (or after soft reset), use bias estimation calibration values from Flash memory. |
| 1 | On power up (or after soft reset), make sure the system is in static condition, the system will collect and calculate new bias estimation calibration values. |

## 6.1.6. SetBaudRate – Configuration Instruction

Use this command to change the current communication baud-rate to another communication baud-rate. Before using this command, be sure that the host controller is configured to the current active baud-rate of the connected navigation device.

**Code 26. Set Baud Rate Sample Code**

```
TstCommand              stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff          stBuff;         //Define a TstCommandBuff Command Result
                                        //Structure

stICommnad.DeviceType = grpConfiguration;//Set the group of commands to Configuration
stICommnad.DeviceIndex = 0;               //Device index is always 0 for the system
stICommnad.CommandID = ecSetOperationMode;       //Set the command ID –

stICommand.Sys.CommSpeed = 460800;

BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

**Table 30. Optional Parameter Values for SetBaudRate**

| | Optional Baud Rates | | | |
|---|---|---|---|---|
| **System Support** | **11520** | **230400** | **460800** | **921600** |
| EX100, EXLP100, gX100 | V | V | V | V |
| EX200, EXLP200, gX200 | V | V | V | V |
| EX300, EXLP300, gX300 | V | V | V | V |

### 6.1.7. SetSampleMode – Configuration Instruction

The navigation device is equipped with two optional methods of sending data messages to the host computer:

- Continuous data message transmission, according to a selected rate.
- Polling message transmission, using a sample on request.

Use this command to select the Sampling mode option.

**Code 27. Set Sample Mode Sample Code**

```
TstCommand              stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff          stBuff;         //Define a TstCommandBuff Command Result
Structure

stICommnad.DeviceType = grpConfiguration;//Set the group of commands to Configuration
stICommnad.DeviceIndex = 0;             //Device index is always 0 for the system
stICommnad.CommandID = ecSetSampleMode; //Set the command ID –

stICommand.Sys.SampleMode = 0;          //continuous mode

BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

**Table 31. List of Parameter Values for SetSampleMode**

| Operation Value Index | Description |
|---|---|
| 0 | Continues transmission |
| 1 | Polling transmission |

Use the immediate command imGetSample to inquire a sample when the system is in Real-Time mode (**Code 28**):

**Code 28. Get Sample Using Polling Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpImidate;     //Set the group of commands to Configuration
stICommnad.DeviceIndex = 0;             //Device index is always 0 for the system
stICommnad.CommandID = imGetSample;     //Set the command ID –

stICommand.Sys.FrameType = 0x20;        //See Table 27

BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

## 6.2. GET Commands

GET commands are queries to the navigation device regarding a specific parameter value or values. For every command of type SET that uses parameters, there is a complimentary command of type GET to read the parameters values. However, GET commands relating to system information are only read-only, and do not have a complementary SET command.

As mentioned in **2.2.5.2** in order to obtain the reply for a GET command, build a function of type `TOnApi_Reply` (**Code 2.**) and deliver a pointer of that function to the main object handler using command navSdk_SetOnApiReply_CallBack.

**Code 29. TOnApi_Reply – Callback Function Template**

```
typedef       void(*TOnApi_Reply)(void* pObj, TstCommand stCmndReply);

For example:
void My_OnApi_Reply(void* pObj, TstCommand stCmndReply)
{
//user code should go here to handle incoming information in stCmndReply structure
}
```

### 6.2.1. Get System Information – Configuration Instructions

The navigation devices include six registers that hold global and specific unit information. Each parameter has a corresponding GET command:

- Hardware Version – ecGetHWVer.
- Firmware Version – ecGetFWVer.
- Platform Type – ecGetDeviceType.
- Platform Name – ecGetDeviceName.
- Platform Version – ecGetDeviceVersion.
- Platform Serial Number – ecGetSerialNum.

Each of these commands has a complementary parameter that holds the result in the Sys field structure of type `TstGlobalSystemSettings` (**Code 12**):

- Hardware Version – Sys.HWVersion.
- Firmware Version – Sys.FWVersion.
- Platform Type – Sys.DeviceType.
- Platform Name – Sys.DeviceName.
- Platform Version – Sys.DeviceVersion.
- Platform Serial Number – Sys.SerialNum.

Use **Code 30** to read the Hardware version from the connected navigation device.

**Code 30. GetHWVer Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpConfiguration; //Set the group of commands to
                                          //Configuration
stICommnad.DeviceIndex = 0;               //Device index is always 0 for the system
stICommnad.CommandID = ecGetHWVer;        //Set the command ID –

BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

The command in **Code 30** does not use parameters, since the command ID indicates the data requested. The remaining five commands only require replacing the CommandID field value with the command enumeration requested.

The connected navigation device will reply with an adequate API_Reply message:

**Code 31. Command Reply for GetHWVer Sample Code**

```
stCmndReply.DeviceType          :=> grpConfiguration;
stCmndReply.DeviceIndex         :=> 0;
stCmndReply.CommandID           :=> ecGetHWVer;


stCmndReply.Sys.HWVersion       :=> array of up to 20 characers
```

The command reply returns the same command information (DeviceType, DeviceIndex, CommandID) as the requested command, with the additional parameter values requested in the adequate structure field. In our example, the relevant parameter is in Sys.HWVersion.

The remaining GET system information commands only require a change in the CommandID field and the Sys matching field.

### 6.2.2. GetFrameTypeAndRate – Configuration Instruction

Use this command to read the current frame rate of a specific data message type (Frame Type). The command must include a parameter value that indicates the Frame Type that corresponds to this query.

**Code 32. GetFrameTypeAndRate Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpConfiguration;//Set the group of commands to Configuration
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ecGetFrameTypeAndRate;    //Set the command ID –

stICommand.Sys.FrameType = 0x21;         //Frame type: IMU-2

BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

**Code 33. GetFrameTypeAndRate Sample Return Results**

```
stCmndReply.DeviceType          :=> grpConfiguration;
stCmndReply.DeviceIndex         :=> 0;
stCmndReply.CommandID           :=> ecGetFrameTypeAndRate;

stCmndReply.Sys.FrameType :=> 0x21;             //see:  Table 27
stCmndReply.Sys.FrameRate :=> 0x06              //see: Table 28
```

## 6.2.3. GetOperationMode – Configuration Instruction

Use this command to read the current active Operation mode (whether or not to use Bias-Estimation on power-up. This function is a complementary command to the SetOperationMode (**Code 25**), and does not use any parameters.

**Code 34. GetOperationMode Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad. DeviceType = grpConfiguration;//Set the group of commands to
                                          //configuration
stICommnad.DeviceIndex = 0;             //Device index is always 0 for the system
stICommnad.CommandID = ecGetOperationMode;      //Set the command ID –

BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

**Code 35. GetOperationMode Sample Return Results**

```
stCmndReply.DeviceType   :=> grpConfiguration;
stCmndReply.DeviceIndex  :=> 0;
stCmndReply.CommandID    :=> ecGetOperationMode;

stCmndReply.Sys.OperationMode :=> 0x00;         //see: Table 9
```

## 6.2.4. GetBaudRate – Configuration Instruction

Use this command to read the current communication Baud-Rate. This function is a complementary command to SetBaudRate command (Section 6.1.6 and does not use any parameters.

**Code 36. GetBaudRate Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpConfiguration;//Set the group of commands to Configuration
stICommnad.DeviceIndex = 0;             //Device index is always 0 for the system
stICommnad.CommandID = ecGetBaudRate;   //Set the command ID –

BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

**Code 37. GetBaudRate Sample Return Results**

```
stCmndReply.DeviceType   :=> grpConfiguration;
stCmndReply.DeviceIndex  :=> 0;
stCmndReply.CommandID            :=> ecGetBaudRate;
stCmndReply.Sys.CommSpeed        :=> 115200;            //see: Table 14
```

### 6.2.5. GetSampleMode – Configuration Instruction

Use this command to read the current method used to transmit data messages from the navigation device to the host controller (Continuous or Polling). This function is a complementary command to the SetSampleMode command (**Code 27**) and does not use any parameters.

**Code 38. GetSampleMode Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpConfiguration;//Set the group of commands to Configuration
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ecGetSampleMode;  //Set the command ID –

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 39. GetSampleMode Sample Return Results**

```
stCmndReply.DeviceType           :=> grpConfiguration;
stCmndReply.DeviceIndex          :=> 0;
stCmndReply.CommandID            :=> ecGetSampleMode;

stCmndReply.Sys.SampleMode       :=> 0;          //see: Table 31
```

## 6.3. grpDevice Commands

The instructions under this group are used to control sub-devices of the navigation device. The list of devices in `TeGroups`, includes a large set of enumeration devices, most of which are not implemented (**Code 9**). This chapter describes the relevant commands to the current supported devices.

When working with device commands, be sure the navigation device is in Configuration mode (**Code 19**).

### 6.3.1. SET Commands

#### 6.3.1.1. SetSampleRate – Device Configuration Instruction

This command defines the IMU frame rate messages, which is the rate at which the internal IMU supplies calibrated raw data messages to the navigation algorithm. This command should use the IMU device as DeviceType.

**Code 40. SetSampleRate Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpDevice_Imu;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = dvSetSampleRate;  //Set the command ID –

stICommand.Device.SampleRate = 100;      //see optional values:Table 32

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Table 32. Optional Parameter Values for Sample Rate**

| Index | Sample Rate (Hz) |
|---|---|
| 0 | 1 |
| 1 | 10 |
| 2 | 25 |
| 3 | 50 |
| 4 | 100 |
| 5 | 200 |
| 6 | 250 |
| 7 | 500 |
| 8 | 1000 |

## 6.3.2. GET Commands

### 6.3.2.1. GetSampleRate – Device Configuration Instruction

Use this command to read the sub-device message rate, which is only relevant to device grpDevice_Imu. It reads the current IMU raw data message frame out, complements the SetSampleRate command described in Section **6.3.1.1**, and does not use any parameters.

**Code 41. GetSampleRate Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpDevice_Imu;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = dvGetSampleRate;//Set the command ID –

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 42. GetSampleRate Sample Return Results**

```
stCmndReply.DeviceType   :=> grpDevice_Imu;
stCmndReply.DeviceIndex  :=> 0;
stCmndReply.CommandID            :=> dvGetSampleRate;//Set the command ID –

//returned information values
stCmndReply.Device.SampleRate :=> 100;  //see optional values: Table 32
```

# 6.4. grpAlgo_UKF Commands

The instructions under this group control the algorithm behavior in the navigation device. Section describes API command enumeration and the list of optional parameters in field Algo of type `TstAlgo_UKF` structure.

To work with algorithm commands, be sure the navigation device is in Configuration mode (Section 6.36).

## 6.4.1. SET Commands

### 6.4.1.1. SetOrientation – Algorithm Configuration Instruction

Under certain circumstances, you may not want to install the navigation device with the default orientation. (In this orientation, the unit is horizontal and the X axis indicates the direction of moving forward).

Under these circumstances, use the Set Orientation command with a suitable installation orientation value as a parameter in the system, so the algorithm compensates for the new installation.

Table 16 shows the 16 different orientation positions that the platform supports.

**Code 43. SetOrientation Algorithm Configuration Structure**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfSetOrientatoin;//Set the command ID –

stICommand.Algo.IMUOrientation = 0;     //see optional values:
BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

### 6.4.1.2. SetStaticSamples – Algorithm Configuration Instruction

Section 3.3.16 and Section 6.1.5 described the SET Operation mode. To calculate bias estimation, the navigation platform must first know how many samples to collect. This command sets the number of samples to collect, which can vary from 1000 to 9000 samples.

**Code 44. SetStaticSamples Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfSetStaticSamples;//Set the command ID –

stICommand.Algo.StaticSamples = 2000;    //optional values: 1000 - 9000

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

### 6.4.1.3. SetIIRFilterStatus – Algorithm Configuration Instruction

The algorithm can filter the incoming raw data from the IMU before entering the UKF algorithm. When the IIR filter is enabled, the results are delayed by three samples.

**Code 45. SetIIRFilterStatus Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfSetIIRFilterStatus;//Set the command ID –

stICommand.Algo.EnableIIRFilter = 1;     //see optional values:

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Table 33. Optional Parameters Values for SetIIRFilterStatus**

| Value | Description |
|---|---|
| 0 | Filter is Disabled |
| 1 | Filter is enabled, there is a delay of 3 samples |

### 6.4.1.4. SetShiftValues – Algorithm Configuration Instruction

You can reset the assembly inclination offset by entering the opposite values in the ROLL, PITCH and YAW fields, which the system uses before sending messages to the host controller. These values do not affect the algorithm. Use these shift values instead of using the SetOrientation optional states (orientation = 0).

**Code 46. SetShiftValues Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfSetShiftValues;//Set the command ID –

stICommand.Algo.RollShift = -1.24;
stICommand.Algo.PitchShift = -0.56;
stICommand.Algo.HeadingShift = 0.00;
stICommand.Algo.UseBiasRemove = 1;//optional Values: 0=Disable use, 1=Enable use

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Table 34. Optional Parameter Values for UseBiaseRemove Field for SetShiftValues Instruction**

| Value | Description |
|---|---|
| 0 | Disable the use of shift values |
| 1 | Enable the use of shift values |

### 6.4.1.5. SetLeverArmValues – Algorithm Configuration Instruction

When the GNSS primary antenna is assembled far from the navigation device, the algorithm calculations will be more accurate if you enter the antenna position relative to the navigation device location. These relative antenna locations affect the algorithm results.

The position of the antenna should be set according to the navigation device acceleration axes as follows (with distance entered in meters):

- If the antenna is ahead of the navigation device AccX arrow, enter position X as a positive value; otherwise, use a negative value.
- If the antenna is ahead of the navigation device AccY arrow, enter position Y as a positive value; otherwise, use a negative value.
- If the antenna is above the navigation device (AccZ) enter position Z as a negative value; otherwise, use a positive value.

Figure 1 shows a visual example of the antenna position.

### Code 47. SetLeverArmValues Sample Code

```
TstCommand              stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff   stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfSetShiftValues;//Set the command ID –

stICommand.Algo.LeverArmX = 1.10;
stICommand.Algo.LeverArmY = -0.56;
stICommand.Algo.LeverArmZ = -0.50;
stICommand.Algo.UseLeverArm = 1; //optional Values: 0=Disable use, 1=Enable use

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

### Table 35. Optional Parameter Values for UseLeverArm Field for SetLeverArm Instruction

| Value | Description |
|---|---|
| 0 | Disable the use of LeverArm in calculation |
| 1 | Enable the use of LeverArm in calculation |

#### 6.4.1.6. SetAntDistance – Algorithm Configuration Instruction

This command is implemented but will be relevant only for the Arsys platform.

If you enter the separation distance (in meters, with millimeter accuracy) between two GPS antennas, the GPS algorithm may converge faster and bind the startup distance error.

### Code 48. SetAntDistance Sample Code

```
TstCommand       stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff   stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfSetShiftValues;//Set the command ID –

stICommand.Algo.AntennaDistance = 1.114;
stICommand.Algo.UseAntDistance = 1; //optional Values: 0=Disable use, 1=Enable use


BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Table 36. Optional Parameter Values for UseAntDistance Field for SetAntDistance Instruction**

| Value | Description |
|---|---|
| 0 | Disable the use AntennaDistance value |
| 1 | Enable the use of AntennaDistance value |

#### 6.4.1.7. SetStartPos – Algorithm Configuration Instruction

These values indicates the GPS position to be used when a fast startup is needed without having to wait for a GPS lock.

Be sure to configure the navigation device with Start Immediate (Section 6.1.5).

**Code 48. SetStartPos – Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfSetShiftValues;//Set the command ID –

stICommand.Algo.dLat = 34.58;    //it is advised to use higher accuracy values
stICommand.Algo.dLon = 31.56;
stICommand.Algo.fAlt = 60.25;

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

### 6.4.2. GET Commands

#### 6.4.2.1. GetOrientation – Algorithm Configuration Instruction

This command complements the SetOrientation command described in Section 6.4.1.1. Use this command to read a device's current orientation. Table 16 shows available orientation positions.

**Code 49. GetOrientation Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfGetOrientation;//Set the command ID –

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 50. GetOrientation Sample Return Results**

```
stCmndReply.DeviceType   :=> grpAlgo_Ukf;
stCmndReply.DeviceIndex  :=> 0;
stCmndReply.CommandID    :=> ukfGetOrientation;//Set the command ID –
//returned information values
stCmndReply.Algo.IMUOrientation :=> 0; //see optional values: Table 16
```

### 6.4.2.2. GetStaticSamples – Algorithm Configuration Instruction

This command complements the SetStaticSamples command. Use this command to read the number of samples needed before starting the bias estimation calculation. This command does not use any parameters.

**Code 51. GetStaticSamples Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfGetStaticSamples;//Set the command ID –

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 52. GetStaticSamples Sample Return Results**

```
stCmndReply.DeviceType           :=> grpAlgo_Ukf;
stCmndReply.DeviceIndex          :=> 0;
stCmndReply.CommandID            :=> ukfGetStaticSamples;//Set the command ID –
//returned information values
stCmndReply.Algo.StaticSamples   :=> 2000;       // optional values: 1000 - 9000
```

### 6.4.2.3. GetIIRFilterStatus – Algorithm Configuration Instruction

This command complements the SetIIRFilterStatus command. Use this command to read the current status of the IIR Filter: Enabled or Disabled. This command does not use any parameters.

**Code 53. GetIIRFilterStatus Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfGetIIRFilterStatus;//Set the command ID –
BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 54. GetIIRFilterStatus Sample Return Results**

```
stCmndReply.DeviceType   :=> grpAlgo_Ukf;
stCmndReply.DeviceIndex  :=> 0;
stCmndReply.CommandID            :=> ukfGetIIRFilterStatus;//Set the command ID –
//returned information values
stCmndReply.Algo.EnableIIRFilter :=> 0;  // see optional values: Table 33
```

### 6.4.2.4. GetShiftValues – Algorithm Configuration Instruction

This command complements the SetShiftValues command. Use this command to read the last shift values entered and to see whether they are in use or not. This command does not use any parameters.

**Code 55. GetShiftValues Sample Code**

```
TstCommand              stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfGetShiftValues;//Set the command ID –

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 56. GetShiftValues Sample Return Results**

```
stCmndReply.DeviceType   :=> grpAlgo_Ukf;
stCmndReply.DeviceIndex  :=> 0;
stCmndReply.CommandID            :=> ukfGetShiftValues;//Set the command ID –
//returned information values
stCmndReply.Algo.UseBiasRemove :=> 1;   // see optional values: Table 34
stCmndReply.Algo.RollShift      :=> -1.24;
stCmndReply.Algo.PitchShift     :=> -0.56;
stCmndReply.Algo.HeadingShift   :=> 0;
```

### 6.4.2.5. GetLeverArmValues – Algorithm Configuration Instruction

This command complements the SetLeverArmValues command. Use this command to read the last LeverArm values entered and to see whether they are in use or not. This command does not use any parameters.

**Code 57. GetLeverArmValues Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfGetLeverArmValues;//Set the command ID –

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 58. GetLeverArmValues Sample Return Results**

```
stCmndReply.DeviceType   :=> grpAlgo_Ukf;
stCmndReply.DeviceIndex  :=> 0;
stCmndReply.CommandID            :=> ukfGetLeverArmValues;//Set the command ID –
//returned information values
stCmndReply.Algo.UseLeverArm     :=> 1;  // see optional values: Table 35
stCmndReply.Algo.LeverArmX       :=> 1.1;
stCmndReply.Algo.LeverArmY       :=> -0.56;
stCmndReply.Algo.LeverArmZ       :=> -0.5;
```

### 6.4.2.6. GetAntDistance – Algorithm Configuration Instruction

This command complements the SetAntDistance command. Use this command to read the last antenna separation distance value entered and to see whether this value is in use or not.

While this command is active, it was implemented for future use and does not currently affect the system. This command does not use any parameters.

**Code 59. GetAntDistance Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfGetAntDistance;//Set the command ID –

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 60. GetAntDistance Sample Return Results**

```
stCmndReply.DeviceType           :=> grpAlgo_Ukf;
stCmndReply.DeviceIndex          :=> 0;
stCmndReply.CommandID            :=> ukfGetAntDistance;//Set the command ID –

stCmndReply.Algo.UseAntDistance :=> 1;  // see optional values: Table 36
```

### 6.4.2.7. GetStartPos – Algorithm Configuration Instruction

This command complements the SetStartPos command. Use this command to read the last GPS position you set for power-up with a GPS outage condition.

This command does not use any parameters.

**Code 61. GetStartPos Sample Code**

```
TstCommand              stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff          stBuff;         //Define a TstCommandBuff Command Result
Structure

stICommnad.DeviceType = grpAlgo_Ukf;
stICommnad.DeviceIndex = 0;
stICommnad.CommandID = ukfGetStartPos;//Set the command ID –

BuildCommand(&stICommnad, &stBuff);     //Build the Command into stBuff
```

**Code 62. GetStartPos Sample Return Results**

```
stCmndReply.DeviceType          :=> grpAlgo_Ukf;
stCmndReply.DeviceIndex         :=> 0;
stCmndReply.CommandID           :=> ukfGetStartPos;//Set the command ID –
//returned information values
stCmndReply.Algo.dLat           :=> 34.58;
stCmndReply.Algo.dLon           :=> 31.56;
stCmndReply.Alto.fAlt           :=> 60.25;
```

## 6.5. grpCalibration Commands

The calibration instructions group holds all the commands needed to handle Bias Estimation calibration and Hard-Iron calibration. **Code 13** lists the calibration enumeration commands (`TeAPI_Calibration`).

You must be in Calibration mode to use the Calibration group instructions.

**Code 63. Enter Calibration Mode Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpImidiate;     //Set the group of commands to indicate
Immidiate
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = imEnableCalibration;      //Set the command ID

stICommnad.Sys.bEnterCalibration = 1;    //Set the command Value: 1 to Enter
                                         //calibration or 0 to Exit Calibration

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

### 6.5.1. SET Commands

#### 6.5.1.1. Reset2Factory – Calibration Instruction

Use this command to revert all calibration settings to their factory default values. This command is of type SET and does not use any parameters.

**Code 64. Reset2Factory Sample Code**

```
TstCommand       stICommnad;      //Define a TstCommand Command Structure
TstCommandBuff   stBuff;          //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands to indicate
Immidiate
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccReset2Factory;  //Set the command ID


BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

#### 6.5.1.2. Burn2Flash – Calibration Instruction

When using calibration instructions, all changes are temporarily retained in the memory registers and take effect immediately. To use the new calibration values as the default calibration values for the next power-up, save those values, using the "Burn to Flash" command. This command is of type SET and does not use any parameters.

**Code 65. Burn2Flash Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands to indicate
                                         //Immediate
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccBurn2Flash;     //Set the command ID

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

### 6.5.1.3. Bias Estimation Calibration

| Note | Sections 3.5.3 to 3.5.8 (inclusive) contain information on bias estimation calibration. |
|---|---|

#### 6.5.1.3.1. StartStaticCal – Calibration Instruction

You may perform Bias Estimation calibration manually at any time by entering Calibration mode and sending the StartStaticCal command. This command is of type SET and does not use any parameters.

**Code 66. StartStaticCal Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccStartStaticCal; //Set the command ID


BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

When starting a bias estimation procedure, the navigation device will inform start state and progress state using information messages.

#### 6.5.1.3.2. StopStaticCal – Calibration Instruction

By default, the bias estimation mechanism register indicates how many measurements (from 1000 to 9000 samples) to collect before performing the calculation.

You can send the Stop Bias Estimation command to collect fewer samples than were already defined. This command is of type SET and does not use any parameters.

**Code 67. StopStaticCal Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff; //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccStopStaticCal;  //Set the command ID


BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

When the Bias estimation process ends, the platform sends a status message to indicate whether the process succeeded (smDoneStaticCal) or failed (smStaticCalError). You must repeat this process if it fails.

#### 6.5.1.3.3. SetStaticCalArray – Calibration Instruction

You can read, save (to the host computer), and write bias estimation calibration values. Use two commands to load the calibration values and two other commands to write those values.

The first write command is SetStaticCalArray.

| Note | The CMAX_BiasEstCoef value is platform-dependent.<br>For VG&AHRS = 27 fields<br>For INS single antenna/INS dual antenna = 28 fields |
|---|---|

**Code 68. SetStaticCalArray Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccSetStaticCalArray;      //Set the command ID

//Fill Static Calibration Coefficients
for(int i=0; i < CMAX_BiasEstCoef; i++)          //CMAX_BiasEstCoef = 27\28
        stICommand.Calib.arCalBiasEst[i] = staticCal[i];

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

### 6.5.1.4. SetStaticExtCalArray – Calibration Instruction

The second write command is SetStaticExtCalArray. The array length is 19.

**Code 69. SetStaticExtCalArray Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccSetStaticCalArray;      //Set the command ID

//Fill Static Calibration Coefficients
for(int i=0; i < CMAX_BiasEstExtCoef; i++)               // CMAX_BiasEstExtCoef = 19
        stICommand.Calib.arCalBiasEstExt[i] = staticCal[i];

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

### 6.5.1.5. Hard-Iron Calibration

| Note | Sections 3.5.9 to 3.5.12 (inclusive) contain information on hard iron calibration. |
|---|---|

#### 6.5.1.5.1. StartHICal – Calibration Instruction

You may perform Hardo-Iron Calibration manually at any time by entering Calibration mode and sending the StartHICal command. This command is of type SET and does not use any parameters.

**Code 70. StartHICal – Example Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccStartHICal;     //Set the command ID

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

#### 6.5.1.5.2. StopHICal – Calibration Instruction

To finalize a Hard-Iron procedure, you should send the "Stop Hard-Iron Calibration" command.

**Code 71. StopHICal – Example Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccStopHICal;      //Set the command ID

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

#### 6.5.1.5.3. SetHICal – Calibration Instruction

You can read, save (to the host computer), and write Hard-Iron calibration values. To write Hard-Iron calibration values from the host computer, use command SetHICal.

**Code 72. SetHICal – Example code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccSetHICal;       //Set the command ID

for(int i=0; i < CMAX_HICoef; i++)       // CMAX_HICoef = 12
        stICommand.Calib.arCalHI[i] = staticCal[i];
BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

## 6.5.2. GET Commands

### 6.5.2.1. Read Bias Estimation Calibration

#### 6.5.2.1.1. GetStaticCalArray – Calibration Instruction

The first read command is GetStaticCalArray. This command does not use any parameters.

**Code 73. SetStaticCalArray Sample Code**

```
TstCommand              stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccGetStaticCalArray;      //Set the command ID

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 74. SetStaticCalArray Sample Return Results**

```
stCmndReply.DeviceType   :=> grpCalibration;
stCmndReply.DeviceIndex  :=> 0;
stCmndReply.CommandID            :=> ccGetStaticCalArray;//Set the command ID –


//Fill Static Calibration Coefficients
for(int i=0; i < CMAX_BiasEstCoef; i++)          //CMAX_BiasEstCoef = 27\28
        staticCal[i] = stCmndReply.Calib.arCalBiasEst[i];
```

| Note | The CMAX_BiasEstCoef value is platform-dependent:<br>For VG&AHRS = 27 fields<br>For INS single antenna/INS dual antenna = 28 fields |
|---|---|

#### 6.5.2.1.2. GetStaticExCalArray – Calibration Instruction

The second read command is GetStaticExCalArray. This command does not use any parameters.

**Code 75. GetStaticExCalArray Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccGetStaticExCalArray;    //Set the command ID

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 76. GetStaticExCalArray Sample Return Results**

```
stCmndReply.DeviceType   :=> grpCalibration;
stCmndReply.DeviceIndex  :=> 0;
stCmndReply.CommandID            :=> ccGetStaticExCalArray;//Set the command ID –

//Fill Static Calibration Coefficients
for(int i=0; i < CMAX_BiasEstExtCoef; i++)              //CMAX_BiasEstExtCoef = 19
        staticCal[i] = stCmndReply.Calib.arCalBiasEstExt[i];
```

### 6.5.2.2. Read Hard-Iron Calibration

#### 6.5.2.2.1. GetHICal – Calibration Instruction

Use this command to read the hard-iron calibration values from the connected navigation device. This command does not use any parameters.

**Code 77. GetHICal Sample Code**

```
TstCommand      stICommnad;     //Define a TstCommand Command Structure
TstCommandBuff  stBuff;         //Define a TstCommandBuff Command Result Structure

stICommnad.DeviceType = grpCalibration;  //Set the group of commands
stICommnad.DeviceIndex = 0;              //Device index is always 0 for the system
stICommnad.CommandID = ccGetHICal;       //Set the command ID

BuildCommand(&stICommnad, &stBuff);      //Build the Command into stBuff
```

**Code 78. GetHICal Sample Return Results**

```
stCmndReply.DeviceType   :=> grpCalibration;
stCmndReply.DeviceIndex  :=> 0;
stCmndReply.CommandID            :=> ccGetHICal; //Set the command ID –


//Fill Static Calibration Coefficients
for(int i=0; i < CMAX_HICoef; i++)              // CMAX_HICoef = 12
        hiCal[i] = stCmndReply.Calib.arCalHI[i];
```

# 7. DATA AND STATUS MESSAGES

## 7.1. Overview

This chapter summarizes the list of messages available for navigation devices. Each data message will indicate whether it is global (present in all navigation devices) or system-specific.

Using the configuration instructions described in the previous chapters, you can select and use several messages with the same or different frame rates, as long as the total bandwidth requested does not exceed the communication link baud-rate in use.

The SDK handles packet recognition, verification of proper receipt of messages, and conversion of a stream of bytes into a friendly data structure that holds data fields that you can use.

To receive the data and status messages, create a callback function form template `TOnMessageRcvd` and send the function pointer through navSdk_SetOnApiMsgRcvdCallBack (Section **2.2.5.3**) to the main object that handles all communication with the navigation devices.

**Code 79. TOnMessageRcvd Calback Function Template**

```
typedef void(*TOnMessageRcvd)(void* pObj, TstSysTesting_Basic *MsgBase, void* pMsg);
```

The template callback function arguments are :

- A pointer to the class owner when using C++ code.
- The basic fields common to all messages, used to analyze the correct typecasting needed for the third argument, which holds the specific message data fields
- A pointer to the data message structure. Adequate typecasting is required to use data in a structure with data fields.

**Code 80. TstSysTesting Common Data/Status Message Fields**

```
typedef struct {
	uint8_t ucHeader1;
	uint8_t ucHeader2;
	uint8_t ucLength;
	uint8_t ucSysType;
	uint8_t ucMsgType;
	uint8_t ucFlag1;
	uint8_t ucFlag2;
	uint8_t ucFlag3;
} TstSysTesting_Basic;
```

**Code 80** exhibits the common message fields returned in the second argument of the `TOnMessageRcvd` callback function.

The structure includes:

- Fields relating to data packet recognition (Header and Length)
- Field ucSysType – holds the current navigation device unique identifier (**Code 9** for TeGroups exhibiting grpMsgs_sys…. Or Table 37 )
- Field ucMsgType – holds the specific message identifier (Table 27. Message Types or Table 38 below)
- Fields ucFlag1 to ucFlag3 – holds Status and Bit flags (see: Section 7.2).

### Table 37. List of Available Platforms

| Platform Index | Platform Name |
|---|---|
| 0x82 | EX100 (Exiguo AHRS) |
| 0x83 | EX200 (Exiguo INS) |
| 0x84 | EX300 (Exiguo INS2ANT) |
| 0x86 | gX100 (gINS AHRS) |
| 0x87 | gX200 (GINS INS) |
| 0x88 | gX300 (GINS INS2ANT) |
| 0x89 | Arsys |
| 0x8C | EXLP100 (ExiguoLP AHRS) |
| 0x8D | EXLP200 (ExiguoLP INS) |
| 0x8E | EXLP300 (ExiguoLP INS2ANT) |

### Table 38. List of Message IDs in Relation to Platforms

| Message Index | Message Name | Platforms |
|---|---|---|
| 0x21 | IMU2-6DOF | All Platforms |
| 0x22 | IMU3-9DOF | All Platforms |
| 0x23 | IMU4-Misc. | All Platforms |
| 0x30 | GPS1-LLA | AHRS, INS, INS2ANT |
| 0x31 | GPS2-ECEF | AHRS, INS, INS2ANT |
| 0x32 | GPS3-Statistic | AHRS, INS, INS2ANT |
| 0x33 | GPS4-RTK | To be implemented soon |
| 0x34 | GPS5-Sat. Info | To be implemented soon |
| 0x40 | Tilt1 – [R, P] | All Platforms |
| 0x41 | Tilt2 – [R, P, Y] | All Platforms |

| Message Index | Message Name | Platforms |
|---|---|---|
| 0x42 | Tilt3 – [Quarter] | All Platforms |
| 0x43 | Tilt4 – [DCM] | All Platforms |
| 0x50 | INS1 – Tilt & Pos. | INS, INS2ANT |
| 0x51 | INS2 – Tilt, Pos. & Vel. | INS, INS2ANT |
| 0x53 | INS3 – Tilt, Pos., Vel. & Time | INS, INS2ANT |

In addition to data messages (Table 38), the navigation device also sends status messages (for example, the current mode when it is changed) or information messages when a specific condition is to be fulfilled (for example, waiting for GPS lock). Alarm status messages provide information about deviations from system-defined working boundaries (for example, above or below temperature range, above or below power supply ranges). Table 39 lists the current available Status messages per platform.

### Table 39. List of Information/Status Message IDs per Platform

| Message Index | Message Name | Platforms |
|---|---|---|
| 0x00 | Done MCU Init | All Platforms |
| 0x01 | Start Static Cal. | All Platforms |
| 0x02 | In Static Cal. | All Platforms |
| 0x03 | Done Static Cal. | All Platforms |
| 0x04 | Static Cal. Error | All Platforms |
| 0x05 | Start HI Cal. | AHRS, INS, INS2ANT |
| 0x06 | In HI Cal. | AHRS, INS, INS2ANT |
| 0x07 | Done HI Cal. | AHRS, INS, INS2ANT |
| 0x08 | HI Cal. Error | AHRS, INS, INS2ANT |
| 0x09 | Wait for GPS Loc. | INS, INS2ANT |
| 0x0A | GPS Loc. Exist | INS, INS2ANT |
| 0x0B | Wait for GPS Heading | INS2ANT |
| 0x0C | GPS Heading Exist | INS2ANT |
| 0x0D | Inform Alarm | All Platforms |
| 0x0E | In Calibration | All Platforms |
| 0x0F | In Configuration | All Platforms |

| Message Index | Message Name | Platforms |
|---|---|---|
| 0x10 | In Real-Time | All Platforms |

## 7.2. Data Flags

The data flags include three registers of type uint8_t (unsigned char). Each register holds specific information on all platforms, although the information may not be relevant for every platform. The following tables list the information per platform.

The first flag register (ucFlag1) holds general information relevant to all platforms.

**Table 40. Data Flag-1 – Bit Structure Information**

| Bit | Flag Name | Description |
|---|---|---|
| 0 | MCU OK | Power up – BIT – Set to 1 when all HW finished initialization OK |
| 1 | Flash CkSum OK | Power up – BIT – Set to 1 when flash check sum is OK |
| 2 | GPS connected OK | Power up – BIT – Set to 1 when Bi-directional communication with GPS Tested OK |
| 3 | IMU Connected OK | Power up – BIT – Set to 1 when Bi-directional communication with IMU Tested OK |
| 4 | Temp OK | Continues – BIT – Set to 1 when Temperature is in defined borders. |
| 5 | Supply In OK | Continues – BIT – Set to 1 when Supply in is in defined borders. |
| 6 | Work Supply OK | Continues – BIT – Set to 1 when working supply is in defined borders |
| 7 | Reserved | |

The second flag register (ucFlag2) holds information only relevant to platforms with a GPS receiver.

**Table 41. Data Flag-2 – Bit Structure Information**

| Bit | Flag Name | Description |
|---|---|---|
| 0 | GPS PPS | Software IO indication for PPS signal – should blink |
| 1 | GPS ANT1 SHORT | Continues – BIT – (EX200/EX300 only)<br>Set to 1 when short detected on Antenna – 1 |

| Bit | Flag Name | Description |
|---|---|---|
| 2 | GPS ANT1 OPEN | Continues – BIT – (EX200/EX300 only)<br>Set to 1 when Antenna load is detected |
| 3 | GPS ANT2 SHORT | Continues – BIT – (EX200/EX300 only)<br>Set to 1 when short detected on Antenna – 2 |
| 4 | GPS ANT2 OPEN | Continues – BIT – (EX200/EX300 only)<br>Set to 1 when Antenna load is detected |
| 5 | GPS Position Lock | Set to 1 when 2D or 3D Fix is available |
| 6 | GPS Heading Lock | Set to 1 when Heading is valid |
| 7 | Reserved | |

The third flag register (ucFlag3) holds platform-specific values, regarding the Static or Dynamic recognition of the platform

**Table 42. Data Flag-3 – Bit Structure Information for VG Platform**

| Value | Value Indication | Description |
|---|---|---|
| 0 | Non-Moving | System detected static condition |
| 1 | Moving | System has detected dynamic condition |

**Table 43. Data Flag-3 – Bit Structure Information for AHRS Platform**

| Value | Value Indication | Description |
|---|---|---|
| 0 | Non-Moving | System has detected static condition (magnetometers = active) |
| 1 | Non-Moving | System has detected static condition (magnetometers = not active) |
| 2 | Moving | System has detected dynamic condition (magnetometers = active) |
| 3 | Moving | System has detected dynamic condition (magnetometers = not active) |

**Table 44. Data Flag-3 – Bit Structure Information for INS Platform**

| Value | Value Indication | Description |
|---|---|---|
| 0 | Non-Moving | System has detected static condition (PVT = active, VelTH = 0, Magnetometers = active) |
| 1 | Non-Moving | System has detected static condition (PVT = active, VelTH = 0, Magnetometers = not active) |
| 2 | Non-Moving | System has detected static condition (PVT = not active, VelTH = 0, Magnetometers = active) |
| 3 | Non-Moving | System has detected static condition (PVT = not active, VelTH = 0, Magnetometers = not active) |
| 4 | Moving | System has detected dynamic condition (PVT = active, VelTH = (+) Sog > Th, Magnetometers = active) |
| 5 | Moving | System has detected dynamic condition (PVT = active, VelTH = (+), Magnetometers = not active) |
| 6 | Moving | System has detected dynamic condition (PVT = active, VelTH = (+) Sog <= Th, Magnetometers = active) |
| 7 | Moving | System has detected dynamic condition (PVT = active, VelTH = (-), Magnetometers = not active) |
| 8 | Moving | System has detected dynamic condition (PVT = not active, VelTH = 0, Magnetometers = active) |
| 9 | Moving | System has detected dynamic condition (PVT = not active, VelTH = 0, Magnetometers = not active) |

**Table 45. Data Flag-3 – Bit Structure Information for INS2ANT Platform**

| Value | Value Indication | Description |
|---|---|---|
| 0 | Non-Moving | System has detected static condition (wPVT, wHDT, wMag, woVel) |
| 1 | Non-Moving | System has detected static condition (wPVT, wHDT, woMag, woVel) |
| 2 | Non-Moving | System has detected static condition (wPVT, woHDT, wMag, woVel) |
| 3 | Non-Moving | System has detected static condition (wPVT, woHDT, woMag, woVel) |

| Value | Value Indication | Description |
|---|---|---|
| 4 | Non-Moving | System has detected static condition (woPVT, woHDT, wMag, woVel) |
| 5 | Non-Moving | System has detected static condition (woPVT, woHDT, woMag, woVel) |
| 6 | Moving | System has detected dynamic condition (wPVT, wHDT, wMag, Vel>Th) |
| 7 | Moving | System has detected dynamic condition (wPVT, wHDT, woMag, wVel) |
| 8 | Moving | System has detected dynamic condition (wPVT, woHDT, wMag, wVel) |
| 9 | Moving | System has detected dynamic condition (wPVT, woHDT, woMag, wVel) |
| 10 | Moving | System has detected dynamic condition (wPVT, wHDT, wMag, Vel<Th) |
| 11 | Moving | System has detected dynamic condition (wPVT, wHDT, woMag, woVel) |
| 12 | Moving | System has detected dynamic condition (wPVT, woHDT, wMag, woVel) |
| 13 | Moving | System has detected dynamic condition (wPVT, woHDT, woMag, woVel) |
| 14 | Moving | System has detected dynamic condition (woPVT, woHDT, wMag, woVel) |
| 15 | Moving | System has detected dynamic condition (woPVT, woHDT, woMag, woVel) |

# 7.3. Data Message Structures

The SDK includes a ready-made structure type for each message type, which you can use for type-casting the message pointer. This chapter describes the structures and fields of each structure.

## 7.3.1. IMU-2 – Message structure

This message includes calibrated IMU data. The 6DOF indicates six degrees of freedom which include three Accelerometers and three Rate Gyros.

**Code 81. Data Message Structure for IMU-2 – 6DOF**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        float           fAccX;
        float           fAccY;
        float           fAccZ;
        float           fGyroX;
        float           fGyroY;
        float           fGyroZ;
} TstSysTesting_IMU_2;
```

**Table 46. IMU-2 – 6DOF Description**

| Message Name: | IMU_MSG_2 | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x21 | | | |
| **Field Name** | **Field Type** | **Source Message** | **Description** | |
| Message Index | Uint32 | System | Message Index | |
| Time Tag | Uint32 | System | Time Tag | |
| Temperature | Float | System | Temperature | |
| ACC. X | Float | IMU | Scaled to g units | |
| ACC. Y | Float | IMU | Scaled to g units | |
| ACC. Z | Float | IMU | Scaled to g units | |
| Gyro X | Float | IMU | Scaled to deg/sec | |
| Gyro Y | Float | IMU | Scaled to deg/sec | |
| Gyro Z | Float | IMU | Scaled to deg/sec | |

## 7.3.2. IMU-3 – Message Structure

This message includes calibrated IMU data. The 9DOF indicates nine degrees of freedom, which include three Accelerometers, three Rate Gyros and three Magnetometers.

**Code 82. Data Message structure for IMU-3 – 9DOF**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        float           fAccX;
        float           fAccY;
        float           fAccZ;
        float           fGyroX;
        float           fGyroY;
        float           fGyroZ;
        float           fMagX;
        float           fMagY;
        float           fMagZ;
} TstSysTesting_IMU_3;
```

**Table 47. IMU-3 – 9DOF Description**

| Message Name: | IMU_MSG_3 | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x22 | | | |
| Field Name | Field Type | Source Message | Description | |
| Message Index | Uint32 | System | Message Index | |
| Time Tag | Uint32 | System | Time Tag | |
| Temperature | Float | System | Temperature | |
| ACC. X | Float | IMU | Scaled to g units | |
| ACC. Y | Float | IMU | | |
| ACC. Z | Float | IMU | | |
| Gyro X | Float | IMU | Scaled to deg/sec | |
| Gyro Y | Float | IMU | | |
| Gyro Z | Float | IMU | | |
| Mag. X | Float | IMU | Scaled to Gauss | |
| Mag. Y | Float | IMU | | |
| Mag. Z | Float | IMU | | |

### 7.3.3. IMU-4 – Message structure

This message, while not currently in use, lists additional sensors for future use in the navigation devices.

**Code 83. Data Message Structure for IMU-4 – Misc. Sensors**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        float           fBaro;
        float           fOdo;
        float           fPito;
        float           fHumidity;
} TstSysTesting_IMU_4;
```

### Table 48. IMU-4 – Miscellaneous Sensors Description

| Message Name: | IMU_MSG_4 | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x23 | | | |
| **Field Name** | **Field Type** | **Source Message** | **Description** | |
| Message Index | Uint32 | System | Message Index | |
| Time Tag | Uint32 | System | Time Tag | |
| Temperature | Float | System | Temperature | |
| Baro | Float | IMU | If available | |
| Odo | Float | IMU | If available | |
| Pito | Float | IMU | If available | |
| Humidity | Float | IMU | If available | |

### 7.3.4. GPS-1 – Message Structure

This message includes the GPS position (in degrees) with information on position quality and velocity quality.

**Code 84. GPS-1 Message Structure**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        uint32_t        uiDate;         //yyyymmdd
        double          dTime;          //hhmmss.ss
        double          dLat;
        double          dLon;
        float           fAlt;
        float           fLatSigma;
        float           fLonSigma;
        float           fAltSigma;
        float           fVelN;
        float           fVelE;
        float           fVelU;
        float           fVelNSigma;
        float           fVelESigma;
        float           fVelUSigma;
        float           fHeading;
        float           fCsep;
        uint8_t         ucHeadingStatus;
        uint8_t         ucPosQual;
        uint8_t         ucFixed;
        uint8_t         ucRes2;
} TstSysTesting_GPS_1;
```

**Table 49. GPS-1 – LLA Description**

| Message Name: | GPS_MSG_1 - LLA | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x30 | | | |
| Field Name | Field Type | Source Message | Description | |
| Message Index | Uint32 | System | | |
| Time Tag | Uint32 | System | | |
| Temperature | Float | System | | |
| Date | Uint32 | GPS | YYYYMMDD (fields: 4,5,6) | |
| Time | Double | GPS | HHMMSS.SS (fields: 7,8,9) | |
| Latitude | Double | GPS | | |
| Longitude | Double | GPS | | |
| Altitude | Float | GPS | | |
| Lat_Sigma | Float | GPS | | |
| Lon_Sigma | Float | GPS | | |
| Alt_Sigma | Float | GPS | | |
| Velocity_N | Float | GPS | | |
| Velocity_E | Float | GPS | | |
| Velocity_U | Float | GPS | | |
| Vel_N_Sigma | Float | GPS | | |
| Vel_E_Sigma | Float | GPS | | |

| Message Name: | GPS_MSG_1 - LLA | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x30 | | | |
| Field Name | Field Type | Source Message | Description | |
| Vel_U_Sigma | Float | GPS | | |
| Heading | Float | GPS | | |
| CSEP | Float | GPS | | |
| Heading_Status | Uint8 | GPS | 0: ineffective, 1: Fix solution, 2: float solution (field: 11) | |
| Position Qual. | Uint8 | GPS | 0: ineffective, 1: Single Point, 2: Pseudo-Range, 4: Fix Solution, 5: Float Solution (field: 10) | |
| GPS Fix | Uint8 | GPS | 1: No Fix, 2: 2D Fix, 3: 3D Fix | |
| Reserved_2 | Uint8 | | | |

## 7.3.5. GPS-2 Message Structure

This message includes the GPS position given in ECEF (Earth-Centered, Earth-Fixed), with information on position quality and velocity quality.

**Code 85. Data Message Structure for GPS-2 - ECEF**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        uint32_t        uiDate;                 //yyyymmdd
        double          dTime;                  //hhmmss.ss
        double          dEcef_X;
        double          dEcef_Y;
        double          dEcef_Z;
        float           dEcef_XSigma;
        float           dEcef_YSigma;
        float           dEcef_ZSigma;
        float           fVelN;
        float           fVelE;
        float           fVelU;
        float           fVelNSigma;
        float           fVelESigma;
        float           fVelUSigma;
        float           fHeading;
        float           fCsep;
        uint8_t         ucHeadingStatus;
        uint8_t         ucPosQual;
        uint8_t         ucFixed;
        uint8_t         ucRes2;
} TstSysTesting_GPS_2;
```

### Table 50. GPS-2 – ECEF Description

| Message Name: | GPS_MSG_2 - ECEF | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x31 | | | |
| **Field Name** | **Field Type** | **Source Message** | **Description** | |
| Message Index | Uint32 | System | | |
| Time Tag | Uint32 | System | | |
| Temperature | Float | System | | |
| Date | Uint32 | GPS | YYYYMMDD (fields: 4,5,6) | |
| Time | Double | GPS | HHMMSS.SS (fields: 7,8,9) | |
| ECEF_X | Double | GPS | | |
| ECEF_Y | Double | GPS | | |
| ECEF_Z | Double | GPS | | |
| ECEF_X_Sigma | Float | GPS | | |
| ECEF_Y_Sigma | Float | GPS | | |
| ECEF_Z_Sigma | Float | GPS | | |
| Velocity_N | Float | GPS | | |
| Velocity_E | Float | GPS | | |
| Velocity_U | Float | GPS | | |
| Vel_N_Sigma | Float | GPS | | |
| Vel_E_Sigma | Float | GPS | | |
| Vel_U_Sigma | Float | GPS | | |
| Heading | Float | GPS | | |
| CSEP | Float | GPS | | |
| Heading_Status | Uint8 | GPS | 0: ineffective, 1: Fix solution, 2: float solution (field: 11) | |
| Position Qual. | Uint8 | GPS | 0: ineffective, 1: Single Point, 2: Pseudo-Range, 4: Fix Solution, 5: Float Solution (field: 10) | |
| GPS Fix | Uint8 | GPS | 1: No Fix, 2: 2D Fix, 3: 3D Fix | |
| Reserved_2 | Uint8 | | | |

## 7.3.6. GPS-3 – Message Structure

This message includes statistical information on the satellites observed, the number of satellites, and more.

**Code 86. Data Message Structure for GPS-3 - Statistic**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        uint32_t        uiDate;                 //yyyymmdd
        double          dTime;                  //hhmmss.ss
        uint8_t         ucNumGpsSat;
        uint8_t         ucNumGloSat;
        uint8_t         ucNumBdsSat;
        uint8_t         ucNumGalSat;
        uint8_t         ucNumSatTracked;
        uint8_t         ucNumSatInUse;
        uint8_t         ucNumSatAboveElev;
        uint8_t         ucNumSatAboveElevL2;
        uint8_t         ucSolutionStatus;
        uint8_t         ucExtSolutionStatus;
        uint8_t         ucSigMask;
        uint8_t         ucRes1;
} TstSysTesting_GPS_3;
```

**Table 51. GPS-3 – Statistic Description**

| **Message Name:** | **GPS_MSG_3 - Statistic** | | **Ver: 1.0** | **Date: 4/9/19** |
|---|---|---|---|---|
| **Msg. Type ID:** | **0x32** | | | |
| **Field Name** | **Field Type** | **Source Message** | **Description** | |
| Message Index | Uint32 | System | | |
| Time Tag | Uint32 | System | | |
| Temperature | Float | System | | |
| Date | Uint32 | GPS | YYYYMMDD (fields: 4,5,6) | |
| Time | Double | GPS | HHMMSS.SS (fields: 7,8,9) | |
| Num. GPS Sat. | Uint8 | GPS | | |
| Num. GLO Sat. | Uint8 | GPS | | |
| Num. BDS Sat. | Uint8 | GPS | | |
| Num. GAL Sat. | Uint8 | GPS | | |
| Num Sat. Tracked | Uint8 | GPS | | |
| Num. Sat. in Use | Uint8 | GPS | | |
| Num. Sat. Above | Uint8 | GPS | | |

| Message Name: | GPS_MSG_3 - Statistic | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x32 | | | |
| Field Name | Field Type | Source Message | Description | |
| elevation mask | | | | |
| Num. Sat Above elevation mask L2 | Uint8 | GPS | | |
| Solution Status | Uint8 | GPS | 0: solution computed, 1: Insufficient observation, 2: No Convergence, 4: Covariance trace exceeds maximum (field: 2) | |
| Ext. Solution Status | Uint8 | GPS | Bit 0: 0 = not verified, 1= verified<br>Bit 1-3: 0=unkown, 1=Broadcase, 2=SBAS Broadcase, 3=Multi Freq Correction, 4=PSRDiff Correction (field: 16) | |
| Sig mask | Uint8 | GPS | Bit 0: Gps L1 used in solution<br>Bit 1: Gps L2 used in solution<br>Bit 2: Gps L5 used in solution<br>Bit 3: BD2 B3 used in solution<br>Bit 4: Glo L1 used in solution<br>Bit 5: Glo L2 used in solution<br>Bit 6: BD2 B1 used in solution<br>Bit 7: BD2 B2 used in solution<br>(field: 18) | |
| Reserved_1 | Uint8 | | | |

## 7.3.7. GPS-4 Message Structure

This message includes RTK information about the base unit location and information regarding the rover-to-base position.

**Code 87. Data Message Structure for GPS-4 - RTK**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        uint32_t        uiDate;                 //yyyymmdd
        double          dTime;                  //hhmmss.ss
        float           fBaseLineN;             //from base to rover
        float           fBaseLineE;
        float           fBaseLineU;
        float           fBaseLineN_std;
        float           fBaseLineE_std;
        float           fBaseLineU_std;
        double          dBaseLat;               //base unit position
        double          dBaseLon;
        float           fBaseAlt;
        uint8_t         ucRTKStatus;    //1 single point, 4 fix, 5 float solution
        uint8_t         ucHdtStatus;    //4 fix solution, 5 float solution
} TstSysTesting_GPS_4;
```

**Table 52. GPS-4 – RTK Description**

| Message Name: | GPS_MSG_4 - RTK | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x33 | | | |
| **Field Name** | **Field Type** | **Source Message** | **Description** | |
| Message Index | Uint32 | System | Message Index | |
| Time Tag | Uint32 | System | Time Tag | |
| Temperature | Float | System | Temperature | |
| Date | Uint32 | GPS | YYYYMMDD (fields: 4,5,6) | |
| Time | Double | GPS | HHMMSS.SS (fields: 7,8,9) | |
| RTK Status | Uint8 | GPS | 1=Single point, 4=Fix solution, 5=float solution | |
| Heading Status | Uint8 | GPS | 4=Fix solution, 5=Float solution | |
| Baseline_N | Float | GPS | From base to rover | |
| Baseline_E | Float | GPS | From base To rover | |
| Baseline_U | Float | GPS | From base To rover | |
| Baseline_N_std | Float | GPS | | |
| Baseline_E_std | Float | GPS | | |
| Baseline_U_std | Float | GPS | | |
| Base Lat. | Double | GPS | | |
| Base Lon. | Double | GPS | | |
| Base Alt. | Float | GPS | | |

## 7.3.8. Tilt-1 Message Structure

This message holds calculated information, resulting from the UKF algorithm running in the navigation device.

**Code 88. Data Message Structure for Tilt-1 – [R, P]**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        float           fRoll;
        float           fPitch;
} TstSysTesting_VG_1;
```

**Table 53. Tilt-1 – [R, P] Description**

| Message Name: | Tilt_MSG_1 | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x40 | | | |
| Field Name | Field Type | Source Message | Description | |
| Message Index | Uint32 | System | Message Index | |
| Time Tag | Uint32 | System | Time Tag | |
| Temperature | Float | System | Temperature | |
| Roll | Float | Calculated | | |
| Pitch | Float | Calculated | | |

## 7.3.9. Tilt-2 – Message Structure

This message holds calculated information resulting from the UKF algorithm running in the navigation device.

**Code 89. Data Message Structure for Tilt-2 – [R, P, Y]**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        float           fRoll;
        float           fPitch;
        float           fYaw;
} TstSysTesting_VG_2;
```

**Table 54. Tilt-2 – [R, P, Y] Description**

| Message Name: | Tilt_MSG_1 | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x40 | | | |
| Field Name | Field Type | Source Message | Description | |
| Message Index | Uint32 | System | Message Index | |
| Time Tag | Uint32 | System | Time Tag | |
| Temperature | Float | System | Temperature | |
| Roll | Float | Calculated | | |
| Pitch | Float | Calculated | | |
| Yaw | Float | Calculated | | |

## 7.3.10. Tilt-3 Message Structure

This message holds the Quaternions matrix values.

**Code 90. Data Message Structure for Tilt-3 – [Quarter]**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        float           fQu_1;
        float           fQu_2;
        float           fQu_3;
        float           fQu_4;
} TstSysTesting_Quater;
```

**Table 55. Tilt-3 – [Quarter]**

| Message Name: | Tilt_MSG_3 | | Ver: 1.3 | Date: 17/6/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x42 | | | |
| Field Name | Field Type | Source Message | Description | |
| Message Index | Uint32 | 4 | Message Index | |
| Time Tag | Uint32 | 4 | Time Tag | |
| Temperature | Float | 4 | Temperature | |
| Qu_1 | Float | | | |
| Qu_2 | Float | | | |
| Qu_3 | Float | | | |
| Qu_4 | Float | | | |

## 7.3.11. Tilt-4 Message Structure

This message holds the DCM matrix values.

**Code 91. Data Message Structure for Tilt-4 – [DCM]**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        float           fDcm_1;
        float           fDcm_2;
        float           fDcm_3;
        float           fDcm_4;
        float           fDcm_5;
        float           fDcm_6;
        float           fDcm_7;
        float           fDcm_8;
        float           fDcm_9;
 } TstSysTesting_DCM;
```

**Table 56. Tilt-4 – [DCM] Description**

| **Message Name:** | **Tilt_MSG_4** | | **Ver: 1.0** | **Date: 4/9/19** |
|---|---|---|---|---|
| **Msg. Type ID:** | **0x43** | | | |
| **Field Name** | **Field Type** | **Source Message** | **Description** | |
| **Message Index** | **Uint32** | **System** | **Message Index** | |
| Time Tag | Uint32 | System | Time Tag | |
| Temperature | Float | System | Temperature | |
| DCM_1 | Float | Calculated | | |
| DCM_2 | Float | Calculated | | |
| DCM_3 | Float | Calculated | | |
| DCM_4 | Float | Calculated | | |
| DCM_5 | Float | Calculated | | |
| DCM_6 | Float | Calculated | | |
| DCM_7 | Float | Calculated | | |
| DCM_8 | Float | Calculated | | |
| DCM_9 | Float | Calculated | | |

## 7.3.12. INS-1 Message Structure

This message includes calculated tilt and position information.

**Code 92. Data message Structure for INS-1 – Tilt & Pos**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        float           fRoll;
        float           fPitch;
        float           fYaw;
        double          dLat;
        double          dLon;
        float           fAlt;
} TstSysTesting_INS_1;
```

**Table 57. INS-1 – Tilt & Pos Description**

| Message Name: | INS_MSG_1 | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x50 | | | |
| **Field Name** | **Field Type** | **Source Message** | **Description** | |
| Message Index | Uint32 | System | Message Index | |
| Time Tag | Uint32 | System | Time Tag | |
| Temperature | Float | System | Temperature | |
| Roll | Float | Calculated | | |
| Pitch | Float | Calculated | | |
| Yaw | Float | Calculated | | |
| Lat. | Double | Calculated | | |
| Lon. | Double | Calculated | | |
| Alt. | Float | Calculated | | |

## 7.3.13. INS-2 Message Structure

This message includes calculated Tilt, Position and Velocity information.

**Code 93. Data Message Structure for INS-2 – Tilt, Pos & Vel**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        float           fRoll;
        float           fPitch;
        float           fYaw;
        double          dLat;
        double          dLon;
        float           fAlt;
        float           fVelN;
        float           fVelE;
        float           fVelU;
} TstSysTesting_INS_2;
```

**Table 58. INS-2 – Tilt, Pos & Vel Description**

| Message Name: | INS_MSG_2 | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x51 | | | |
| **Field Name** | **Field Type** | **Source Message** | **Description** | |
| **Message Index** | **Uint32** | **System** | **Message Index** | |
| Time Tag | Uint32 | System | Time Tag | |
| Temperature | Float | System | Temperature | |
| Roll | Float | Calculated | | |
| Pitch | Float | Calculated | | |
| Yaw | Float | Calculated | | |
| Lat. | Double | Calculated | | |
| Lon. | Double | Calculated | | |
| Alt. | Float | Calculated | | |
| Vel. N | Float | Calculated | | |
| Vel. E | Float | Calculated | | |
| Vel. U | Float | Calculated | | |

## 7.3.14. INS-3 Message Structure

This message includes calculated Tilt, Position, Velocity and Time information.

**Code 94. Data Message structure for INS-3 – Tilt, Position, Velocity, Time**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        float           fRoll;
        float           fPitch;
        float           fYaw;
        double          dLat;
        double          dLon;
        float           fAlt;
        float           fVelN;
        float           fVelE;
        float           fVelU;
        double          dTime;
        uint32_t        uiDate;
}TstSysTesting_INS_3;
```

**Table 59. INS-3 – Tilt, Pos, Vel and Time Description**

| Message Name: | INS_MSG_3 | | Ver: 1.0 | Date: 4/9/19 |
|---|---|---|---|---|
| Msg. Type ID: | 0x52 | | | |
| **Field Name** | **Field Type** | **Source Message** | **Description** | |
| Message Index | Uint32 | System | Message Index | |
| Time Tag | Uint32 | System | Time Tag | |
| Temperature | Float | System | Temperature | |
| Roll | Float | Calculated | | |
| Pitch | Float | Calculated | | |
| Yaw | Float | Calculated | | |
| Lat. | Double | Calculated | | |
| Lon. | Double | Calculated | | |
| Alt. | Float | Calculated | | |
| Vel. N | Float | Calculated | | |
| Vel. E | Float | Calculated | | |
| Vel. U | Float | Calculated | | |
| Time | Double | GPS\Calculated | | |
| Date | Uint32 | GPS | | |

## 7.4. Status Message Description

The navigation device sends status messages to the host controller regarding the current system status or current active mode. The Status Messages are of the same message structure as general data messages, although the Data element of the Status messages is fixed at four bytes long, even if some of those bytes are not used for all status messages.

**Code 95** shows the Status message structure for typecasting:

**Code 95. Status Message Structure**

```
typedef struct {
        uint32_t        uiMsgIndex;
        uint32_t        uiTimeTag;
        float           fTemperature;
        uint32_t        uiStatusInformation;
} TstSysStatus_Info;
```

### 7.4.1. Status Messages without Parameters

The index of Status messages without valid arguments represents status information, and the four bytes of data information are sent with 0 values.

**Table 60. List of Status Messages without Parameters**

| Message Index | Message Name |
|---|---|
| 0x00 | Done MCU Init |
| 0x01 | Start Static Cal. |
| 0x03 | Done Static Cal. |
| 0x04 | Static Cal. Error |
| 0x05 | Start HI Cal. |
| 0x07 | Done HI Cal. |
| 0x08 | HI Cal. Error |
| 0x09 | Wait for GPS Loc. |
| 0x0A | GPS Loc. Exist |
| 0x0B | Wait for GPS Heading |
| 0x0C | GPS Heading Exist |
| 0x0E | In Calibration |
| 0x0F | In Configuration |
| 0x10 | In Real-Time |

### 7.4.2. Status Messages with Parameters

The section describes the Data element for every message that contains data. Table 61 lists status messages with data information.

**Table 61. List of Status Messages with Parameters**

| Message Index | Message Name |
|---|---|
| 0x02 | In Static Cal. |
| 0x06 | In HI Cal. |
| 0x0D | Inform Alarm |

#### 7.4.2.1. Status Message – In Static Calibration

The "In Static Calibration" status message indicates the percentage of measurements collected up to the time the message was sent. The calculation relates to the register with the number of measurements to collect before the Bias Estimation is calculated.

The information is sent once a second while the process is running.

**Table 62. Status Message – In Static Calibration**

| Command Index | 0x02 | | |
|---|---|---|---|
| **List of Arguments** | | | |
| **Field Index** | **Field Name** | **Field Type** | **Field Values** |
| 1 | Percent | Uint32_t | 0 - 100 |

#### 7.4.2.2. Status Message – In Hard-Iron Calibration

The "In Hard-Iron Calibration" status message indicates the number of measurements collected (as a simple counter number).

The information is sent once a second while the process runs.

**Table 63. Status Message – In Hard-Iron Calibration**

| Command Index | 0x06 | | |
|---|---|---|---|
| List of Arguments | | | |
| Field Index | Field Name | Field Type | Field Values |
| 1 | Amount | Uint32_t | |

#### 7.4.2.3. Status Message – Inform Alarm

The "Inform Alarm" status message indicates when an environmental condition exceeds the boundaries defined for the working temperature, the Supply-in, and the working supply.

The information is sent once a second when the samples exceed the boundaries.

**Table 64. Status Message – Inform Alarm**

| Command Index | 0x0D | | |
|---|---|---|---|
| List of Arguments | | | |
| Field Index | Field Name | Field Type | Field Values |
| 1 | Alarm Type | Uint8_t | See: Table 65 |
| 2 | Hi Range Exceeded | Uint8_t | 0 - did not exceed<br>1 – Exceed |
| 3 | Lo Range Exceeded | Uint8_t | 0 - did not exceed<br>1 – Exceed |

**Table 65. List of Optional Alarm Types**

| Index | Alarm Type |
|---|---|
| 0 | Temperature |
| 1 | Vin |
| 3 | VDD |

# TABLE OF REVISIONS

| Ver. # | Description | Author/s | Date |
|---|---|---|---|
| <1.00> | SDK First Version for Customer | | 24/8/2025 |
| | | | |
| | | | |