HPLS®-2G

SDK Manual

VERSION 1.00

SEPTEMBER 2024

Table of Contents

1. SDK Overview 10
2. The Library 11
2.1. General Overview 11
2.1.1. Included Files 11
2.2. Basic Library Functions for Procedural Programming 11
2.2.1. Basic function – hplsSdk_CreateCHplsSdkObject 12
2.2.2. Basic function – hplsSdk_ReleaseCHplsSdkObject 12
2.2.3. Basic Callback Functions – Overview 13
2.2.4. Basic function – hplsSdk_WriteData 14
2.3. Library Functions to Simulate Data Message Reception 15
3. Data & Status Messages 16
3.1. Overview 16
3.1.1. The Status Message Template arguments 16
3.1.2. The Data Message Template Arguments 17
3.2. Data Flags 18
3.3. Data Message Structures 19
3.3.1. L1-GNSS Messages 19
3.3.2. Multi-Frequency Messages 22
4. API Commands 27
4.1. Overview 27
4.2. API SET commands 29
4.2.1. Enter/Exit Configuration Mode 29
4.2.2. Save Settings 30
4.2.3. Reset Device 30
4.2.4. Setting Device Communication Baud-Rate 30
4.2.5. Set Frame/Message Type 31
4.2.6. Set Frame Rate 31
4.2.7. Set Work-Mode 32
4.2.8. Settings Alarms 32
4.2.9. Set MSEP 32
4.2.10. Set Heading Offset 33
4.3. API GET Commands 33
4.3.1. ACK Reply 33
4.3.2. Get Configuration 34
4.3.3. Get Firmware Version 34

4.3.4. Get Board Type 35
4.3.5. Get MSEP 35
4.3.6. Get Heading Offset 35
4.3.7. Get Alarm Thresholds 35
5. Demo Application 37
5.1. Overview 37
5.2. Include headers 37
5.3. Static Library Loading 38
5.4. Declared Functions 38
5.4.1. HPLS2G Callback Functions 38
5.4.2. Serial Port Callback Function 38
5.5. Main function 39
5.5.1. Simulate Messages 39
5.5.2. Read Configuration 41
5.5.3. Write Configuration 42
5.5.4. Receive Messages (real-time) 43

LIST OF TABLES

Table 1. List of library Availability for Operation Systems 10
Table 2. Supported Languages for the Current SDK Library 10
Table 3. List of Included Files 11
Table 4. List of Basic Functions 11
Table 5. List of Instructions to Simulate Message Reception 15
Table 6. Optional Message IDs 17
Table 7. Message Status Flag Information 18
Table 8. M2 Message Extra Flag Bit Information 21
Table 9. M2 Message Extra Flag Bit Information 24
Table 10: List of Set Commands 29
Table 11. M2 Message Extra Flag Bit Information 30
Table 12. M2 Message Extra Flag Bit Information 31
Table 13. Optional Frame Rates 31
Table 14: List of Get Commands 33

LIST OF FIGURES

Figure 1. Antenna Mounting 33
Figure 2. List of Demo Options..........39
Figure 3. Simulate Messages..........40
Figure 4. Simulation Demo Execution 40
Figure 5. Read Configuration 41
Figure 6. Write Configuration..........42
Figure 7. Example of OnMessageRcvd 44

LIST OF CODES

Code 1. Callback Functions Template 16
Code 2. TeStatusCommand....16
Code 3. Header Information....17
Code 4. M1_MSG Structure....19
Code 5. M2_MSG Structure....20
Code 6. M3_MSG Structure....21
Code 7. GM_M1_ MSG Structure....22
Code 8. GM_M2_MSG Structure....23
Code 9. GM_M3_MSG Structure....24
Code 10. GM_M4_MSG Structure....26
Code 11. TeVerifyResult Structure....27
Code 12. TstCommandBuff Structure....27
Code 13. Config (Last GET Command) Structure....28
Code 14. HPLS2G – Header Files....37
Code 15. Serial Port Header File....37
Code 16. Static Library Loading....38
Code 17. HPLS2G – Callback Functions....38
Code 18. SerialPort – Callback Function....38
Code 19. OnSerialPortDataReady Implementation....39

ABBREVIATIONS

Abbreviation	Description

WARRANTY NOTICE

This Warranty covers products manufactured by Arazim, Ltd. (The "Company")

The Company warrants the products during the Warranty Period, under normal use and maintenance, to be free from defects in material and workmanship, and will substantially conform to the Company's specifications (attached) for the products. Yet the Company does not guarantee that the product will not endure any interruptions or faults. The product datasheet, specifications and additional instructions are included in the User Manual.

This Warranty shall apply to all parties. All other warranties or settlements are excluded.

In order to obtain a warranty service, the purchaser must provide the original purchase Invoice and serial number of the product.

The Purchaser is responsible for installing the product properly and to check and verify the functioning of the product. For any faults or claims the purchaser must contact and notify the Company within 14 days of the purchase of the product.

For any faults or deficiency on the purchaser's liability:

1) Return Merchandise Authorization (RMA) Report must be issued.

2) Non-Conformance Report including Product Serial Number must be attached.

3) Transport of the product to and from Arazim Ltd. offices.

The Warranty shall apply only in Israel and is subject to the availability of replacement parts. The Company will not change the design or functionality of the product in order to conform to restrictions of countries in which the product is restricted for legal or regulation issues.

This warranty shall comply with all governed regulations for the products including all Import and export regulations and inspections.

Warranty Period for the company products is for 12 months unless stated otherwise by the Company. The warranty Period applies from the purchase date indicated on the purchase invoice. For products carrying longer Warranty Periods, all of the foregoing liabilities stated above will apply during the second- and third-year warranty periods (excluding payment collections) In addition the Company will not be obligated to the repair lead times.

The Company may charge for repairs or replacements if required under any of the following conditions:

- Damage caused by accident or natural events such as fire/water; electrical discharges; Use of hardware or software, interfaces or devices not supplied by the Company and/or are not manufactured by Arazim, Ltd.
- Malice, negligence, abuse, misuse or improper installation of the products not complying with the user manual; expandable parts, external cover or color coatings; damage occurring in transport of the product by the Purchaser.
- Any changes, repairs or services performed on the product by any party other than Arazim, Ltd. including removal of the seal tag protecting the inner parts of the product from exposure.
- The Company is not obligated to adjust the product to any designation or purpose.
- The Company will not be liable for any claim by purchaser or any third party of direct or indirect damage including loss or deletion of applications or information, restoration expenses, loss of time and or profit, reputation damages including damages resulting using the product, any faults or services provided for the product.
- Any repair services for the products after the expiration of the warranty Period will bear charges.

The warranty and repair service will be executed in Israel and subject to the discretion of the Company whether to exchange parts, repair or replace the product. The product or parts may be reconditioned parts or used parts but will, in all cases, be functionally equivalent to the original parts or even improved.

This Warranty Notice will apply to all parties and in lieu of any other agreements or contradictions.

1. SDK Overview

The current Software Development Kit (SDK) includes a dynamic library that gathers all the functionality required to parse incoming messages from the HPLS device and to control the API commands sent to the HPLS device. Currently, the library is adjusted for the operating systems listed in Table 1.

In addition to the library, the SDK includes several program examples that show how to use library functions. Table 2 lists the current supported languages.

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 (Functions)	Windows 32\64 Linux 64bit	22/8/24		✓
C++	Windows 32\64 Linux 64bit	22/8/24		✓
Python	Windows 32\64	Soon		

This document describes library functionality with respect to:

- Data Messages.
- API Commands.

The SDK library assists programmers in integrating the HPLS platform within another system.

2. THE LIBRARY

2.1. General Overview

The dynamic link library (DLL) functions assist users in working with the HPLS platform through procedural programming or object-oriented programming.

The HPLS-2G API Commands document provides additional information.

2.1.1. Included Files

Table 3. List of Included Files

Index	File name	Overview
1	\x32\Hpls_sdk_dll_based.dll/lib	32bit DLL/LIB files for windows OS
2	\x64\Hpls_sdk_dll_based.dll/lib	64bit DLL/LIB files for windows OS
3	\h_files\Hpls_SDK_DataTypes.h	Structures and definitions for all data types.
4	\h_files\hpls2g_handler.h	Base class structure, for OOP
5	\h_files\hpls_sdk_dll_based.h	Exported C functions, for procedural programming
6	\h_files\dll_global.h	Exporting/Importing conventions
7	\h_files\DataPacket.hpp	Internal use

2.2. Basic Library Functions for Procedural Programming

Table 4 lists the basic functions required for procedural programming.

Table 4. List of Basic Functions

Index	Function Name
1	int8_t hplsSdk_CreateCHplsSdkObject(void);
2	bool hplsSdk_ReleaseCHplsSdkObject(int indx);
3	void hplsSdk_SetStatusRcvd_Callback(int indx, TOnStatusMsg statusRcvd);
4	void hplsSdk_SetMsgRcvd_Callback(int indx, TOnMessageRcvd msgRcvd);
5	void hplsSdk_SetApiReply_Callback(int indx, TOnApi_Reply apiReply);

Index	Function Name
6	uint8_t hplsSdk_WriteData(int indx, uint8_t* ucBuff, uint16_t usLen);
7	void hplsSdk_EnterConfig(int indx, TstCommandBuff* pstBuff);
8	void hplsSdk_ExitConfig(int indx, TstCommandBuff* pstBuff);
9	List of Configuration Set Commands (see: Table 10)
10	List of Configuration Get Commands (see: Table 14)

These basic functions connect the application to the HPLS2G driver so users can:

- initiate a handler
- connect callback functions to receive data messages and command reply data
- send commands to change the HPLS2G work mode settings
- read the current work mode settings of a connected device.

2.2.1. Basic function – hplsSdk_CreateCHplsSdkObject

First, you call Function #1: hplsSdk_CreateCHplsSdkObject, which creates the primary object handler (the object), responsible for all HPLS platform functionality.

If the function succeeds in creating the object, the system returns a value between 0 and 9. If the function fails, the system returns a value of -1.

The value returned should be kept and used with all other functions.

2.2.2. Basic function – hplsSdk_ReleaseCHplsSdkObject

Call Function #2 only after you have finished working with the HPLS device in order to release the object and the memory it occupies.

Table 4. List of Basic Functions

Index	Function Name
1	int8_t hplsSdk_CreateCHplsSdkObject(void);
2	bool hplsSdk_ReleaseCHplsSdkObject(int indx);
3	void hplsSdk_SetStatusRcvd_Callback(int indx, TOnStatusMsg statusRcvd);
4	void hplsSdk_SetMsgRcvd_Callback(int indx, TOnMessageRcvd msgRcvd);
5	void hplsSdk_SetApiReply_Callback(int indx, TOnApi_Reply apiReply);

Index	Function Name
6	uint8_t hplsSdk_WriteData(int indx, uint8_t* ucBuff, uint16_t usLen);
7	void hplsSdk_EnterConfig(int indx, TstCommandBuff* pstBuff);
8	void hplsSdk_ExitConfig(int indx, TstCommandBuff* pstBuff);
9	List of Configuration Set Commands (see: Table 10)
10	List of Configuration Get Commands (see: Table 14)

The function receives a single argument, which is the object index provided after the object was created with Function #1.

2.2.3. Basic Callback Functions – Overview

A general function is a standard function that does not depend on other functions or events, can be called directly, executes its code independently and is used to perform specific tasks or calculations.

A callback function is a function, invoked by an outer function at a specific time or under certain conditions to provide custom behavior or actions within the context of the outer function.

As a developer, you use callback functions with a predefined structure, entering a code in the function to be executed when a specified condition has been fulfilled.

You must use the predefined structure in order to pass a pointer to the callback function (or code snippet), to be called when the relevant process takes place.

The HPLS2G library has three functions to set callback functions to handle status messages, data messages, and API replies.

Functions #3, Function #4, and Function #5 are used to link user functions to the main object process.

2.2.3.1. Basic Function – hplsSdk_SetStatusRcvd_Callback

The HPLS system includes status messages, delivered to the user via the callback function, regarding the progress of an internal process or the system status.

You can create a function of type `TOnStatusMsg` and send a pointer for that function as an argument to hplsSdk_SetStatusRcvd_CallBack to receive status information.

The function structure `TOnStatusMsg` is given in file:

Hpls_SDK_DataTypes.h

typedef void (*TOnStatusMsg)(void* pObj, TeStatusCommand status, unsigned char value);

2.2.3.2. Basic Function – hplsSdk_SetMsgRcvd_CallBack

In order to receive data messages from the HPLS platform, you must build a function according to function template `TOnMessageRcvd`, which will be called when a data message arrives. The function holds information on the message type so you know which type casting applies to the pointer holding the data structure.

The function structure `TOnMessageRcvd` is given in file:
Hpls_SDK_DataTypes.h

typedef void (*TOnMessageRcvd)(void* pObj, stHpls2g_Header *MsgBase, void* pMsg);

- stHpls2g_Header – the message type, with the structure shown in Code 3.
- void* pMsg –any message structures shown in Chapter 3.

2.2.3.3. Basic Function – hplsSdk_SetApiReply_Callback

To receive Reply data for API GET commands sent to the device, build a function according to function template TOnApi_Reply, which will be called for any successful API GET command. The function arguments include the command index, whose reply is (TeAPI_CmndsIDs) and a structure (stHPLS2G_Config) that holds the Reply data.

The function structure TOnApi_Reply is given in file:
Hpls_SDK_DataTypes.h

typedef void (*TOnApi_Reply)(void* pObj, TeAPI_CmndsIDs cmdIds, stHPLS2G_Config *AckReply);

- TeAPI_CmndsIDs –given in the Hpls_SDK_DataTypes.h file
- stHPLS2G_Config –structure shown in Code 13

2.2.4. Basic function – hplsSdk_WriteData

The HPLS system includes a serial communication channel (RS232/ RS422) that interacts with a Host controller (similar to a client task on a host computer). Since this communication channel is often converted to a USB or ETH channel the communication channel on the host computer is variable. Use thehplsSdk_WriteData function to deliver collected data from the communication channel to the main object.

The first argument is the object ID.

The second argument includes the collected data (sent as an array of bytes).

The third argument is the array length.

The information delivered to the main object through this function passes through processing functions that recognize the data (Status Messages, Data Messages, API Reply) and use it to build a meaningful block of data to be directed to one of the Callback functions.

2.3. Library Functions to Simulate Data Message Reception

The library includes a list of instructions to help you verify that the data message callback function works properly without having to connect it to an HPLS device. Each data message type has a matching simulation function that can be fired on request. The simulation function checks the entire route of a real message (without the physical communication channel).

Table 6 contains a list of simulation instructions.

Table 5. List of Instructions to Simulate Message Reception

Index	Function Name
L1-GNSS	
1	bool SimulateMsgM1(int indx);
2	bool SimulateMsgM2(int indx);
3	bool SimulateMsgM3(int indx);
Multi-Frequency	
4	bool SimulateMsgGM_M1(int indx);
5	bool SimulateMsgGM_M2(int indx);
6	bool SimulateMsgGM_M3(int indx);
7	bool SimulateMsgGM_M4(int indx);

3. Data & Status Messages

3.1. Overview

This chapter describes the fields available on each data and status message available to the HPLS systems.

The SDK:

- recognizes packets
- verifies that messages are received correctly (checksum and length)
- converts the stream of bytes to a data structure that holds data fields that you can use.

First, you create adequate callback functions form templates: `TOnMessageRcvd` and `TOnStatusMsg.`

Next, send the function pointers to the main object using these functions: hplsSdk_SetMsgRcvd_Callback and hplsSdk_SetStatusRcvd_Callback.

Code 1. Callback Functions Template

```
typedef void(*TOnStatusMsg)(void* pObj, TeStatusCommand status, unsigned char value);
typedef void(*TOnMessageRcvd)(void* pObj, stHpls2g_Header *MsgBase, void* pMsg);
```

The callback function templates include the following three arguments:

- The first argument holds the object handler index.
- For TOnStatusMsg
    - The second argument is the command status type.
    - The third argument is a value that corresponds to the command status type.
- For TOnMessageRcvd
    - The second argument holds information on how to type-cast the third argument.
    - The third argument holds a data structure pointer that holds all of the message information.

3.1.1. The Status Message Template arguments

Code 2. TeStatusCommand

```
enum TeStatusCommand : int { scStart = 0xa0, scStop, scInform };
```

The `TeStatusCommand` holds the ID of three commands:

- The *scStart* command is sent when a new process begins.
    - For Power-up, scStop is sent to indicate that the BIT process has begun.
- The *scInform* command is sent while the process is running and the third argument indicates the status or time passed since the process began.
- The *scStop* command is sent when the process ends. The third argument indicates whether the process ended with or without errors.
    - The Power-up *scStop* 3rd argument may be:
        - 0 – Inclinometer BIT fail.
        - 1 – GPS communication fail.
        - 2 – BIT passed OK.

3.1.2. The Data Message Template Arguments

The HPLS-2G API Commands document provides additional information on handling message frame structure and data fields.

Code 3. Header Information

```
typedef struct {
        uint8_t cHdr[2];
        uint8_t ucDataSize;
        uint8_t ucPacketType;
        uint16_t        usFlags;
        uint16_t        usCheckSum;
}stHpls2g_Header;
```

The data message callback template function holds the message header information in the second argument. By testing the *ucPacketType* field, you can correctly type-cast the pointer message in the third argument. There are currently three messages for the L1-GNSS platform and four messages for the Multi-Frequency platform (Table 6).

Table 6. Optional Message IDs

Packet Type Name	Packet Type ID	Remarks
L1-GNSS		
HPLS_M1	17	Basic information
HPLS_M2	42	M1 + Extra Flags & MSEP Value

Packet Type Name	Packet Type ID	Remarks
L1-GNSS		
HPLS_M3	43	M2 + Date Information
Multi-Frequency		
HPLS_GM_M1	51	Basic information
HPLS_GM_M2	52	M1 + GM YAW
HPLS_GM_M3	53	M2 + Date Information
HPLS_GM_M4	54	M2 + IMU Information

3.2. Data Flags

All messages include a *usFlag* field in the message header that holds information on the power-up BIT and on continuous status flags.

Table 7. Message Status Flag Information

Bit Number	Description	Bit Number	Description
15	Primary Lock	7	Turn On Inclinometer BIT
14	Secondary Lock	6	PPS Valid
13	DGPS Lock	5	Temperature Valid
12	Diff Lock	4	Inclination Pitch Valid
11	HDT Lock	3	Inclination Roll Valid
10	UnUsed – L1-GNSS Static Status – Multi-Frequency	2	CSEP Valid
9	Unused – L1-GNSS Yaw Filter Valid – Multi-Frequency	1	HDT Valid
8	Turn On GPS BIT	0	GGA Valid

The flag categories are:

- Lock/Status bits [10-15] - System and GPS status information
- BIT bits [7-8] - Two bits (built-in-test results), set only at power-up.

- Valid bits [0-6, 9] - Indication of receipt of a new value

3.3. Data Message Structures

Each message type in the SDK includes a ready-made message structure for type-casting the message pointer.

Message structures can be found in file: *Hpls_SDK_DataTypes.h*.

3.3.1. L1-GNSS Messages

3.3.1.1. Message M1 Structure

The M1 message fields, supply GPS information (location, time, and statistics) and tilt information.

Code 4. M1_MSG Structure

```
typedef struct {
        uint8_t         cHdr[2];
        uint8_t         ucDataSize;
        uint8_t         ucPacketType;
        uint16_t        usFlags;
        uint16_t        usCheckSum;
        uint16_t        usIndex;
        uint16_t        usSwVer;
        float           fCsep;
        double          dGpsUtc;
        double          dGpsLat;
        double          dGpsLon;
        float           fGpsAlt;
        uint8_t         ucQual;
        uint8_t         ucNumSat;
        uint16_t        usReserved;
        float           fTemp;
        float           fRoll;
        float           fPitch;
        float           fYaw;
}stHPLS2G_M1_MSG;
```

The *ucDataSize* field value for message M1 should be: 0x3D (61)

The *ucPacketType* field value for message M1 should be: 0x11 (17)

3.3.1.2. Message M2 structure

The M2 message fields are the same as M1 message fields with the following additional fields:

- An *Extra Flags* field that replaces the *usReserved* field
- A new field with information on the MSEP (manual antenna separation)

Code 5. M2_MSG Structure

```
typedef struct {
        uint8_t         cHdr[2];
        uint8_t         ucDataSize;
        uint8_t         ucPacketType;
        uint16_t        usFlags;
        uint16_t        usCheckSum;
        uint16_t        usIndex;
        uint16_t        usSwVer;
        float           fCsep;
        double          dGpsUtc;
        double          dGpsLat;
        double          dGpsLon;
        float           fGpsAlt;
        uint8_t         ucQual;
        uint8_t         ucNumSat;
        uint16_t        ExtFlags;
        float           fTemp;
        float           fRoll;
        float           fPitch;
        float           fYaw;
        float           fMsep;
}stHPLS2G_M2_MSG;
```

The *ucDataSize* field value for message M2 should be: 0x41 (65)

The *ucPacketType* field value for message M2 should be: 0x2a (42)

Table 8. M2 Message Extra Flag Bit Information

Bit Number	Description	Bit Number	Description
15	Temp Over Max T.H	7	N.U
14	Temp Below Min T.H	6	N.U
13	Inclination Above Max T.H	5	N.U
12	Inclination Below Min T.H	4	N.U
11	N.U	3	N.U
10	N.U	2	N.U
9	N.U	1	N.U
8	N.U	0	MSEP is Valid

3.3.1.3. Message M3 Structure

The M3 message structure provides the same information as the M2 message structure and features three additional fields to indicate the current date.

Code 6. M3_MSG Structure

```
typedef struct {
        uint8_t         cHdr[2];
        uint8_t         ucDataSize;
        uint8_t         ucPacketType;
        uint16_t        usFlags;
        uint16_t        usCheckSum;
        uint16_t        usIndex;
        uint16_t        usSwVer;
        float           fCsep;
        double          dGpsUtc;
        double          dGpsLat;
        double          dGpsLon;
        float           fGpsAlt;
        uint8_t         ucQual;
        uint8_t         ucNumSat;
        uint16_t        ExtFlags;
        float           fTemp;
        float           fRoll;
        float           fPitch;
```

```
        float       fYaw;
        float       fMsep;
        uint8_t     ucDay;
        uint8_t     ucMonth;
        uint16_t    usYear;
}stHPLS2G_M3_MSG;
```

The *ucDataSize* field value for message M3 should be: 0x45 (69)

The *ucPacketType* field value for message M3 should be: 0x2b (43)

3.3.2. Multi-Frequency Messages

3.3.2.1. Message GM-M1 Structure

The GM-M1 message fields provide GPS information (location, time, and statistics) and tilt information.

Code 7. GM_M1_ MSG Structure

```
typedef struct {
        uint8_t     cHdr[2];
        uint8_t     ucDataSize;
        uint8_t     ucPacketType;
        uint16_t    usFlags;
        uint16_t    usCheckSum;
        uint16_t    usIndex;
        uint16_t    usSwVer;
        float       fCsep;
        double      dGpsUtc;
        double      dGpsLat;
        double      dGpsLon;
        float       fGpsAlt;
        uint8_t     ucQual;
        uint8_t     ucNumSat;
        uint16_t    usReserved;
        float       fTemp;
        float       fRoll;
        float       fPitch;
        float       fYaw;
}stHPLS2G_GM_M1_MSG;
```

The *ucDataSize* field value for message GM-M1 should be: 0x3D (61)
The *ucPacketType* field value for message GM-M1 should be: 0x33 (51)

3.3.2.2. Message GM-M2 structure

The GM-M2 message fields are the same as the GM-M1 message fields with the following additional fields:

- An *Extra Flags* field that replaces the *usReserved* field
- A new field with information on GMYaw (filtered YAW)

Code 8. GM_M2_MSG Structure

```
typedef struct {
        uint8_t         cHdr[2];
        uint8_t         ucDataSize;
        uint8_t         ucPacketType;
        uint16_t        usFlags;
        uint16_t        usCheckSum;
        uint16_t        usIndex;
        uint16_t        usSwVer;
        float           fCsep;
        double          dGpsUtc;
        double          dGpsLat;
        double          dGpsLon;
        float           fGpsAlt;
        uint8_t         ucQual;
        uint8_t         ucNumSat;
        uint16_t        ExtFlags;
        float           fTemp;
        float           fRoll;
        float           fPitch;
        float           fYaw;
        float           fGMYaw;
}stHPLS2G_GM_M2_MSG;
```

The *ucDataSize* field value for message GM-M2 should be: 0x41 (65)
The *ucPacketType* field value for message GM-M2 should be: 0x34 (52)

Table 9. M2 Message Extra Flag Bit Information

Bit Number	Description	Bit Number	Description
15	Temp Over Max T.H	7	N.U
14	Temp Below Min T.H	6	N.U
13	Inclination Above Max T.H	5	N.U
12	Inclination Below Min T.H	4	N.U
11	N.U	3	N.U
10	N.U	2	N.U
9	N.U	1	N.U
8	N.U	0	MSEP is Valid

3.3.2.3. Message GM-M3 Structure

The GM-M3 message structure contains the same information as the GM-M2 message structure with three additional fields to indicate the current date.

Code 9. GM_M3_MSG Structure

```
typedef struct {
        uint8_t         cHdr[2];
        uint8_t         ucDataSize;
        uint8_t         ucPacketType;
        uint16_t        usFlags;
        uint16_t        usCheckSum;
        uint16_t        usIndex;
        uint16_t        usSwVer;
        float           fCsep;
        double          dGpsUtc;
        double          dGpsLat;
        double          dGpsLon;
        float           fGpsAlt;
        uint8_t         ucQual;
        uint8_t         ucNumSat;
        uint16_t        ExtFlags;
        float           fTemp;
        float           fRoll;
        float           fPitch;
```

```
        float       fYaw;
        float       fGMYaw;
        uint8_t     ucDay;
        uint8_t     ucMonth;
        uint16_t    usYear;
}stHPLS2G_GM_M3_MSG;
```

The *ucDataSize* field value for message GM-M3 should be: 0x45 (69)
The *ucPacketType* field value for message GM-M3 should be: 0x35 (53)

3.3.2.4. Message GM-M4 Structure

The GM-M4 message structure provides the same information as the GM-M2 message structure with six additional fields to indicate the IMU sensor values.

Code 10. GM_M4_MSG Structure

```
typedef struct {
        uint8_t         cHdr[2];
        uint8_t         ucDataSize;
        uint8_t         ucPacketType;
        uint16_t        usFlags;
        uint16_t        usCheckSum;
        uint16_t        usIndex;
        uint16_t        usSwVer;
        float           fCsep;
        double          dGpsUtc;
        double          dGpsLat;
        double          dGpsLon;
        float           fGpsAlt;
        uint8_t         ucQual;
        uint8_t         ucNumSat;
        uint16_t        ExtFlags;
        float           fTemp;
        float           fRoll;
        float           fPitch;
        float           fYaw;
        float           fGMYaw;
        float           fAccX;
        float           fAccY;
        float           fAccZ;
        float           fGyroX;
        float           fGyroY;
        float           fGyroZ;
}stHPLS2G_GM_M4_MSG;
```

The *ucDataSize* field value for message GM-M4 should be: 0x59 (89)

The *ucPacketType* field value for message GM-M4 should be: 0x36 (54)

4. API COMMANDS

4.1. Overview

The API commands control the HPLS2G device settings. The API commands are used to read the current setting of the device (GET Commands) or change the device settings (SET Commands).

The API command functions described below are actually builder commands. The result of calling for a specific command is an array of bytes that represents the command to be sent to the device.

Each API command includes several arguments, the first of which is always the object handler index (Section 2.2.1) while the last always has the following *TstCommandBuff* structure:

Code 11. TeVerifyResult Structure

```
typedef enum : uint8_t
{
	vrOK, vrError, vrBuffFull
} TeVerifyResult;
```

Code 12. TstCommandBuff Structure

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

The *ucaBuffer* field contains the command bytes.

The *ucBuffLen* field contains the total number of bytes in the array.

The *eResult* field indicates whether or not the build succeeded.

The arguments between the first and the last arguments are settings information.

Send the array of bytes (according to *ucBuffLen*) to the device through the communication channel in use (for example, RS232/RS422/USB/ETH) only if the *eResult* field value is *vrOK*.

The API commands are either SET commands or GET commands.

The HPLS2G replies with ACK command to successful SET commands, and replies with the requested information to successful GET commands.

The HPLS2G replies go through the callback function pointer sent to the driver, using command: "hplsSdk_SetApiReply_Callback" (Section 2.2.3.3).

```
typedef void (*TOnApi_Reply)(void* pObj, TeAPI_CmndsIDs cmdIds, stHPLS2G_Config *AckReply);
```

The second argument holds the command index, which indicates the field in the third argument that holds returned information. (*TeAPI_CmndsIDs* is located in unit "Hpls_SDK_DataTypes.h").

The third argument holds field structures that change according to the last GET command.

Code 13. Config (Last GET Command) Structure

```
typedef struct {
	bool		bLastCommandAck;
	uint8_t		ucBaudRate;
	uint8_t		ucFrameType;
	uint8_t		ucFrameRate;
	uint8_t		ucWorkMode;
	uint16_t	usFWVer;
	uint16_t	usFWBuild;
	float		fMsep;
	float		fHeadingOffset;
	int16_t		isTempMaxAlarm;
	int16_t		isTempMinAlarm;
	int16_t		isIncMaxAlarm;
	int16_t		isIncMinAlarm;
	uint8_t		ucIncType;
	uint8_t		ucIncRange;
	uint8_t		ucGpsType;
	uint8_t		ucCommType;
	uint8_t		ucBoardType;
	uint8_t		ucMsgsTypes;
	uint8_t		ucYawFilterMode;
	double		dAccTh;
	double		dGyroTh;

} stHPLS2G_Config;
```

At power-up, the HPLS2G device works in run-time mode, sending data messages to the user.

As a developer, you must enter the configuration mode in order to read or change any parameter value in the settings.

4.2. API SET commands

Table 10: List of Set Commands

Index	Function Name
1	void hplsSdk_EnterConfig(int indx, TstCommandBuff* pstBuff);
2	void hplsSdk_ExitConfig(int indx, TstCommandBuff* pstBuff);
3	void hplsSdk_SaveSettings(int indx, TstCommandBuff* pstBuff);
4	void hplsSdk_SW_Reset(int indx, TstCommandBuff* pstBuff);
5	void hplsSdk_Set_BaudByIndx(int indx, uint8_t BaudIndx, TstCommandBuff* pstBuff);
6	void hplsSdk_Set_BaudRate(int indx, int BaudRate, TstCommandBuff* pstBuff);
7	void hplsSdk_Set_FrameType(int indx, uint8_t FrameIndx, TstCommandBuff* pstBuff);
8	void hplsSdk_Set_FrameRateByIndex(int indx, uint8_t RateIndx, TstCommandBuff* pstBuff);
9	void hplsSdk_Set_FrameRate(int indx, uint8_t FrameRate, TstCommandBuff* pstBuff);
10	void hplsSdk_Set_WorkMode(int indx, uint8_t mode, TstCommandBuff* pstBuff);
11	void hplsSdk_Set_TempMaxAlarm(int indx, int16_t alarmVal, TstCommandBuff* pstBuff);
12	void hplsSdk_Set_TempMinAlarm(int indx, int16_t alarmVal, TstCommandBuff* pstBuff);
13	void hplsSdk_Set_IncMaxAlarm(int indx, int16_t alarmVal, TstCommandBuff* pstBuff);
14	void hplsSdk_Set_IncMinAlarm(int indx, int16_t alarmVal, TstCommandBuff* pstBuff);
15	void hplsSdk_Set_Msep(int indx, float fMsep, TstCommandBuff* pstBuff);
16	void hplsSdk_Set_HeadingOffset(int indx, float fHeadingOffset, TstCommandBuff* pstBuff);

4.2.1. Enter/Exit Configuration Mode

Call the following function to enter configuration mode:

void hplsSdk_EnterConfig(int indx, TstCommandBuff* pstBuff);

The function does not receive any settings values.

After finishing the work in configuration mode (reading and writing), call function *hplsSdk_ExitConfig* to exit configuration mode and return to run-time mode.

void hplsSdk_ExitConfig(int indx, TstCommandBuff* pstBuff);

Since the Enter/Exit API commands are SET commands, HPLS2G replies with ACK command to each of the messages.

4.2.2. Save Settings

To save settings changes as the default for next power cycle, send this function (which receives no settings values):

```
void hplsSdk_SaveSettings(int indx, TstCommandBuff* pstBuff);
```

Send this command in configuration mode.

4.2.3. Reset Device

Send the following command (which receives no settings values) to reset the HPLS2G without recycling the power:

```
void hplsSdk_SW_Reset(int indx, TstCommandBuff* pstBuff);
```

Send this command in configuration mode.

The device is reset only after the Exit Configuration command succeeds.

4.2.4. Setting Device Communication Baud-Rate

The default device baud-rate is 115200 bps. Alternatively, you can configure the device to the baud-rates in Table 11.

Table 11. Baud-Rate Options

Baud	Code	Baud	Code
19200	0x00	230400	0x04
38400	0x01	460800	0x05
57600	0x02	921600	0x06
115200 (factory default)	**0x03**		

Use the following two commands to set the device baud-rate:

In the first command, the second argument uses the baud-rate index 0-6 to set the required baud-rate.

```
void hplsSdk_Set_BaudByIndx(int indx, uint8_t BaudIndx, TstCommandBuff* pstBuff);
```

In the second command, the second argument uses the actual baud-rate value of 19200-921600.

```
void hplsSdk_Set_BaudRate(int indx, int BaudRate, TstCommandBuff* pstBuff);
```

Send this command in configuration mode.

Note: The baud-rate changes only after you exit the Configuration mode.

4.2.5. Set Frame/Message Type

The HPLS2G can be configured to send a data message.

```
void hplsSdk_Set_FrameType(int indx, uint8_t FrameIndx, TstCommandBuff* pstBuff);
```

Table 12 lists the frame index that the second argument holds.

Table 12. Frame Type Options

Code	Packet Type	Packet Type Name
0x00	Basic altitude and position information	**HPLS_M1 (factory default)**
0x01	M1 + MSEP (L1 GNSS) M1 + GMYAW (Multi Freq)	HPLS_M2
0x02	M2 + Date	HPLS_M3
0x03	M2 + IMU (Multi Freq Only)	HPLS_M4

4.2.6. Set Frame Rate

After configuring the frame type, you can configure the HPLS2G frame type transmission rate.

Table 13. Optional Frame Rates

Code	Frame Rate
0x00	1
0x01	2
0x02	5
0x03	10 (factory default)
0x04	20

Use two commands to set the message frame rate:

In the first command, the second argument uses the index of the frame rate: 0-4.

```
void hplsSdk_Set_FrameRateByIndex(int indx, uint8_t RateIndx,
TstCommandBuff* pstBuff);
```

In the second command, the second argument uses the actual frame rate value: 1-20

```
void hplsSdk_Set_FrameRate(int indx, uint8_t FrameRate,
TstCommandBuff* pstBuff);
```

4.2.7. Set Work-Mode

The HPLS2G can work in either the **Legacy** or **1mRad** real-run mode.

To set the work real-run mode, send the command:

" hplsSdk_Set_WorkMode ".

```
void hplsSdk_Set_WorkMode(int indx, uint8_t mode, TstCommandBuff* pstBuff);
```

The second argument should hold the selected mode, either:

- 0 = Legacy or
- 1 = 1mRad

4.2.8. Settings Alarms

The HPLS2G can notify you through the extra flag field of messages M2 (see Table 12) to M4 when the temperature or inclination level has passed the minimum or maximum threshold values you entered.

Use the functions below to change the minimum and maximum thresholds:

```
void hplsSdk_Set_TempMaxAlarm(int indx, int16_t alarmVal, TstCommandBuff* pstBuff);
void hplsSdk_Set_TempMinAlarm(int indx, int16_t alarmVal, TstCommandBuff* pstBuff);
void hplsSdk_Set_IncMaxAlarm(int indx, int16_t alarmVal, TstCommandBuff* pstBuff);
void hplsSdk_Set_IncMinAlarm(int indx, int16_t alarmVal, TstCommandBuff* pstBuff);
```

You can use the second argument of each function to set the threshold value.

4.2.9. Set MSEP

You can call the following function to set the antenna separation distance directly to the HPLS2G:

```
void hplsSdk_Set_Msep(int indx, float fMsep, TstCommandBuff *pstBuff);
```

The second argument is used to enter the distance, in meters, between the antennas.

4.2.10. Set Heading Offset

The antennas should be mounted along the vehicle in the direction of travel, so that the primary antenna is at the rear and the secondary antenna is in the front. If the antennas are mounted differently (for example, the primary on the right and secondary on the left) you can compensate by offsetting the heading by 90°.

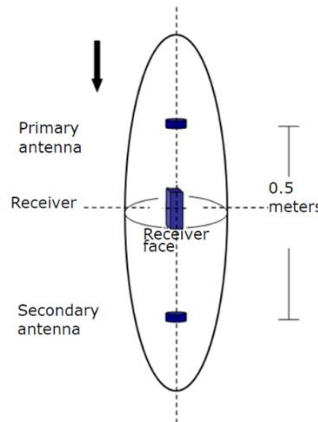


Figure 1. Antenna Mounting

void hplsSdk_Set_HeadingOffset(int indx, float fHeadingOffset, TstCommandBuff *pstBuff);

The second argument is used to enter the heading compensation value.

4.3. API GET Commands

Table 14: List of Get Commands

Index	Function Name
1	`void hplsSdk_Get_Config(int indx, TstCommandBuff* pstBuff);`
2	`void hplsSdk_Get_FWVer(int indx, TstCommandBuff* pstBuff);`
3	`void hplsSdk_Get_BoardType(int indx, TstCommandBuff* pstBuff);`
4	`void hplsSdk_Get_Msep(int indx, TstCommandBuff * pstBuff);`
5	`void hplsSdk_Get_HeadingOffset(int indx, TstCommandBuff* pstBuff);`
6	`void hplsSdk_Get_TempMaxAlaram(int indx, TstCommandBuff* pstBuff);`
7	`void hplsSdk_Get_TempMinAlaram(int indx, TstCommandBuff* pstBuff);`
8	`void hplsSdk_Get_IncMaxAlaram(int indx, TstCommandBuff* pstBuff);`
9	`void hplsSdk_Get_IncMinAlaram(int indx, TstCommandBuff* pstBuff);`

4.3.1. ACK Reply

As discussed in the Overview section, an ACK reply should be received for every successful SET command.

When the callback function is fired, the user can read the value in register from the returned structure (stHPLS2G_Config).

- bLastCommandAck

A register value of 1 indicates that the SET command succeeded.

4.3.2. Get Configuration

In order to read the basic settings of HPLS2G device, call the following function: "hplsSdk_Get_Config".

```
void hplsSdk_Get_Config(int indx, TstCommandBuff *pstBuff);
```

The HPLS2G will reply with information about:

- Baud-Rate
- Frame-Type
- Frame-Rate
- Work-Mode
- Firmware Version

When the callback function is fired, you can read the value in register from the returned structure (stHPLS2G_Config):

- ucBaudRate - index of current baud rate
- ucFrameType - index of current selected frame type
- ucFrameRate - index of current selected frame rate
- ucWorkMode - used by the manufacturer
- usFWVer - current FW version

4.3.3. Get Firmware Version

Call function: "hplsSdk_Get_FWVer" to read the current firmware version of the connected HPLS2G device.

```
void hplsSdk_Get_FWVer(int indx, TstCommandBuff* pstBuff)
```

When the callback function is fired, you can read the value in register from the returned structure (stHPLS2G_Config):

- usFWVer - Current FW version
- usFWBuild - Current Build

4.3.4. Get Board Type

Call function: "hplsSdk_Get_BoardType" to read the current board version of the connected HPLS2G device.

```
void hplsSdk_Get_BoardType(int indx, TstCommandBuff* pstBuff)
```

When the callback function is fired, you can read the value in register from the returned structure (stHPLS2G_Config).

- ucBoardType - current Board Version/Type

4.3.5. Get MSEP

Call function: "hplsSdk_Get_Msep" to read the current settings of the GPS antenna separation from the HPLS2G device.

```
void hplsSdk_Get_Msep(int indx, TstCommandBuff *pstBuff);
```

When the callback function is fired, you can read the following value in register from the returned structure (stHPLS2G_Config):

- fMsep - current antennas separation

4.3.6. Get Heading Offset

Call function: " hplsSdk_Get_HeadingOffset" to read the current heading offset from the HPLS2G device.

```
void hplsSdk_Get_HeadingOffset(int indx, TstCommandBuff *pstBuff);
```

When the callback function is fired, you can read the following value in register from the returned structure (stHPLS2G_Config):

- fHeadingOffset - current heading offset

4.3.7. Get Alarm Thresholds

You can read the alarm thresholds (maximum and minimum) using the following functions:

```
void hplsSdk_Get_TempMaxAlaram(int indx, TstCommandBuff *pstBuff);
void hplsSdk_Get_TempMinAlaram(int indx, TstCommandBuff *pstBuff);
void hplsSdk_Get_IncMaxAlaram(int indx, TstCommandBuff *pstBuff);
void hplsSdk_Get_IncMinAlaram(int indx, TstCommandBuff *pstBuff);
```

When the callback function is fired, you can read the relative register for each command from the returned structure (stHPLS2G_Config):

- isTempMaxAlarm - current maximum temperature threshold

- isTempMinAlarm - current minimum temperature threshold
- isIncMaxAlarm - current maximum inclination threshold
- isIncMinAlarm - current minimum inclination threshold

5. DEMO APPLICATION

5.1. Overview

The demo application shows how easy it is to connect to the HPLS2G library and activate the library functions, which control settings and receive the HPLS2G messages. The Demo application uses serial port library that you can replace with your preferred serial port library in your application.

The Demo application includes the following demos:

- Test Simulation: This demo simulates data message reception without having to connect to an HPL2G device. The HPLS2G library includes special functions to build messages and send a message array of bytes through the entire data processing mechanism, which mimics actual message reception. This demo assists you in determining that the library initialization and the messages callback function work properly.
- Test Reading Configuration: This demo shows the configuration mode, used to verify ACK replays and read setting values. This demo assists you in ensuring that the library initialization and the API reply callbacks work properly.
  This demo requires a connection to a real HPLS2G device.
- Test Writing Configuration: The demo shows the configuration mode, in which you set new configuration values and verify ACK replays.
  This demo requires a connection to a real HPLS2G device.
- Test real-time: This demo shows how to connect to message callbacks and receive real-time messages.
  This demo requires a connection to a real HPLS2G device.

5.2. Include headers

The demo application includes two header files (see Code 14 and Code 15), required for the main application that uses the HPLS2G library:

Code 14. HPLS2G – Header Files

```
#include "Hpls_SDK_DataTypes.h"
#include "hpls_sdk_dll_based.h"
```

A different header is used for the serial port library (which the user can replace), not described in this document:

Code 15. Serial Port Header File

```
#include "SerialPortDLL_V2.h"
```

5.3. Static Library Loading

This demo uses a simple static library to load both the HPLS2G library and the serial port library. You can find the "*.dll" files and the "*.lib" files in the Debug and the Release Project directories.

Code 16. Static Library Loading

```
//load static Lib (dll) - can also be done through the project properties or using a dynamic load
#pragma comment(lib, "hpls_sdk_dll_based.lib")    //load hpls2g library
#pragma comment(lib, "SerialPortDLL_V2.lib")      //load serial port library - to be used for example
```

5.4. Declared Functions

The demo application declares functions that manage event functions and test functions.

5.4.1. HPLS2G Callback Functions

The following functions were declared to connect with the HPLS2G driver callback functions:

Code 17. HPLS2G – Callback Functions

```
void OnStatusRcvd(void* pObj, TeStatusCommand status, unsigned char value);
void OnMessageRcvd(void* pObj, stHpls2g_Header *MsgBase, void* pMsg);
void OnApiReplyRcvd(void* pObj, TeAPI_CmndsIDs apiCmndId, stHPLS2G_Config *ApiReply);
```

Use these functions to collect HPLS2G data information, status information and command replays.

Connect these functions with the HPLS2G object in order to call them (Section 2.2.3).

5.4.2. Serial Port Callback Function

The serial port is an object that wraps the PC RS232 port. The Serial Port object receives data on the physical RS232 port and delivers the incoming bytes to the user application. When those bytes reach the user application, the OnSerialPortDataReady callback function is called to collect the bytes, as well as the total number of bytes collected.

Then use function hplsSdk_WriteData (Section 2.2.4) to send the data to the HPLS2G main object for processing.

Code 18. SerialPort – Callback Function

```
void OnSerialPortDataReady(void* pObj, const unsigned char *data, const unsigned short &len);
```

Code 19. OnSerialPortDataReady Implementation

```
void OnSerialPortDataReady(void* pObj, const unsigned char *data, const unsigned short
&len)
{
        hplsSdk_WriteData(g_iHplsObj , (uint8_t*)data, len);
}
```

5.5. Main function

The main function initializes a handler for:

- the RS232 communication port (which may be a virtual communication port) and
- the HPLS2G device.

When the HPLS2G handler is initialized successfully (calling: hplsSdk_CreateCHplsSdkObject() function), it returns a value >= 0 to variable **g_iHplsObj**. This variable should be passed as the first argument to any function of the HPLS2G handler.

Following successful initialization, set the callback functions to handle incoming status messages and API ACKs or API reply messages.

Then call a function "hpls_run_user_interface()" to organize a list of options to be used in the demo.

```
INFORM: Comm Port Settings passed OK!
INFORM: Comport picked and configured: COM-858993460
+----------------------------------------------------+
List of commands:
~~~~~~~~~~~~~~~~~~~~~~~~~~~
1: Open COM port
2: Close COM port
3: Simulate Messages
4: Read Configuration
5: Write Configuration
6: Receive Messages
0: Exit
Command Chosen :
```

Figure 2. List of Demo Options

5.5.1. Simulate Messages

Option #3 (*Simulate Messages*) connects the "OnMessageRcvd" callback to the HPLS2G handler and then calls all simulation functions, one-by-one.

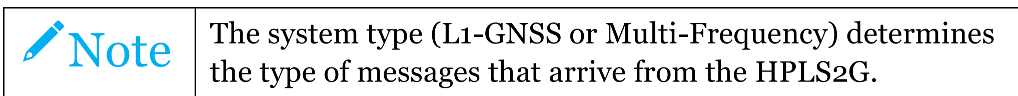
```
else if (cUserInput == '3') {

    // Hook to the messages received and start showing them.
    hplsSdk_SetMsgRcvd_Callback(g_iHplsObj, OnMessageRcvd);

    SimulateMsgM1(g_iHplsObj);
    SimulateMsgM2(g_iHplsObj);
    SimulateMsgM3(g_iHplsObj);
    SimulateMsgGM_M1(g_iHplsObj);
    SimulateMsgGM_M2(g_iHplsObj);
    SimulateMsgGM_M3(g_iHplsObj);
    SimulateMsgGM_M4(g_iHplsObj);

}
```

Figure 3. Simulate Messages

Figure 4 is a simulation of successful message execution.

```
INFORM: Comm Port Settings passed OK!
INFORM: Comport picked and configured: COM-858993460
+-------------------------------------------------+
List of commands:
~~~~~~~~~~~~~~~~~~~
1: Open COM port
2: Close COM port
3: Simulate Messages
4: Read Configuration
5: Write Configuration
6: Receive Messages
0: Exit
Command Chosen : 3

Message M1
Data Info: 35.500000, 2.050000, 1.280000, 123.349998
Flags: 0, 0, 0
ExtFlags: 0, 0

Message M2
Data Info: 35.500000, 2.050000, 1.280000, 123.349998, 1.000000
Flags: 0, 0, 0
ExtFlags: 0, 0

Message M3
Data Info: 35.500000, 2.050000, 1.280000, 123.349998, 3-10-2019
Flags: 0, 0, 0
ExtFlags: 0, 0

Message GM-M1
Data Info: 35.500000, 2.050000, 1.280000, 123.349998
Flags: 0, 0, 0
ExtFlags: 0, 0

Message GM-M2
Data Info: 35.500000, 2.050000, 1.280000, 123.349998, 123.349998
Flags: 0, 0, 0
ExtFlags: 0, 0

Message GM-M3
Data Info: 35.500000, 2.050000, 1.280000, 123.349998, 3-10-2019
Flags: 0, 0, 0
ExtFlags: 0, 0

Message GM-M4
Data Info: 35.500000, 2.050000, 1.280000, 123.349998
IMU Info: 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000
Flags: 0, 0, 0
ExtFlags: 0, 0
```

Figure 4. Simulation Demo Execution



Note	The system type (L1-GNSS or Multi-Frequency) determines the type of messages that arrive from the HPLS2G.

5.5.2. Read Configuration

Option #4 (*Read Configuration*) connects the " OnApiReplyRcvd" callback to the HPLS2G handler.

Before activating this demo, select and open a communication port attached to the HPLS2G device and ensure that the HPLS2G is powered up accordingly.

```
else if (cUserInput == '4') {

    // Check if the comport is opened first.
    if(g_bComportOpen) {

        // Reading all the configuration values.
        hpls_read_all_config_values(g_iHplsObj);

        // Inform user we're done reading configuration.
        printf("INFORM: Done reading HPLS configuration.\n\n");
    }
    else{

        // Inform user we're done reading configuration.
        printf("ERROR: Comport must be opened first.\n\n");
    }
}
```

Figure 5. Read Configuration

In order to read or write configuration settings values, the system must be in Configuration mode. First, send the "hplsSdk_EnterConfig(iObjIndx, &g_stCmndBuff);" command to the HPLS2G device before reading the configuration settings values.

If an ACK is received, you can read the settings values, including:

- Read Firmware version
- Read current MSEP value
- Read current Heading Offset
- Read Board Type
- Read Temperature Max & Min Alarm Threshold
- Read Inclinometer Max & Min Alarm Threshold

After reading the values, exit the configuration mode and enter the real-time mode using this command:
"hplsSdk_ExitConfig(iObjIndx, &g_stCmndBuff);"

5.5.3. Write Configuration

Option #5 (*Write Configuration*) connects the " OnApiReplyRcvd" callback to the HPLS2G handler.

Before activating this demo, select and open a communication port, attached to the HPLS2G device, and ensure that the HPLS2G is powered up accordingly.

```
else if (cUserInput == '5') {

    // Check if the comport is opened first.
    if(g_bComportOpen){

        // Writing all the configuration values.
        hpls_write_all_config_values(g_iHplsObj);

        // Inform user we're done reading configuration.
        printf("INFORM: Done writing HPLS configuration.\n\n");
    }
    else{

        // Inform user we're done reading configuration.
        printf("ERROR: Comport must be opened first.\n\n");
    }
}
```

Figure 6. Write Configuration

Note	To read or write settings, the system must be in Configuration mode.

The first command sent to the HPLS2G device before writing configuration values is: "hplsSdk_EnterConfig(iObjIndx, &g_stCmndBuff);".

If an ACK is received, you can begin writing the following settings values:

- Write Baud-Rate
- Write Frame-Type
- Write Frame-Rate
- Write Temperature Max & Min Alarm Threshold
- Write Inclinometer Max & Min Alarm Threshold
- Write MSEP value
- Write Heading Offset

When you have finished reading the values, exit the configuration mode and enter the real-time mode using this command: "hplsSdk_ExitConfig(iObjIndx, &g_stCmndBuff);"

If the HPLS2G successfully received a command, it will reply with an API ACK.

5.5.4. Receive Messages (real-time)

Option #6 (*Receive Messages*) connects the " OnMessageRcvd " callback to the HPLS2G handler.

Before activating this demo, select and open a communication port attached to the HPLS2G device and ensure that the HPLS2G is powered up accordingly.

In this demo, the application receives and parses a message from the HPLS2G according to the last configuration settings.
Unlike the simulation demo, only one type of message is displayed.

When calling the data message callback function, use the stHpls2_Header argument type to identify the message structure. The code snippet below shows how to:

- identify the message type received,
- type-cast the message structure, and
- send the message data fields to the screen.

```
void OnMessageRcvd(void* pObj, stHpls2g_Header *MsgBase, void* pMsg){

    TCHpls2g_handler* hpls2g = (TCHpls2g_handler*)pObj;

    //hpls-2g
    stHPLS2G_M1_MSG* msgM1;
    stHPLS2G_M2_MSG* msgM2;
    stHPLS2G_M3_MSG* msgM3;
    stHPLS2G_FLAGS      stFlags = { 0 };
    stHPLS2G_EXT_FLAGS  stExtFlags = { 0 };
    //hpls-2g-gm
    stHPLS2G_GM_M1_MSG* msgGM_M1;
    stHPLS2G_GM_M2_MSG* msgGM_M2;
    stHPLS2G_GM_M3_MSG* msgGM_M3;
    stHPLS2G_GM_M4_MSG* msgGM_M4;
    stHPLS2G_GM_FLAGS       stGmFlags = { 0 };
    stHPLS2G_GM_EXT_FLAGS   stGmExtFlags = { 0 };

    if (MsgBase->ucPacketType == 0x11)
    {
        msgM1 = (stHPLS2G_M1_MSG*)pMsg;

        //ProcessMsgM1(msgM1);

        printf("\nMessage M1\n");
        printf("Data Info: %f, %f, %f, %f \n", msgM1->fTemp, msgM1->fRoll, msgM1->fPitch, msgM1->fYaw);

        hpls2g->FlagsToStruct(msgM1->usFlags, stFlags);
        hpls2g->ExtFlagsToStruct(msgM1->usReserved, stExtFlags);

        printf("Flags: %d, %d, %d\n", stFlags.GpsPwrOn, stFlags.IncPwrOn, stFlags.DgpsLock);
        printf("ExtFlags: %d, %d\n", stExtFlags.TempAboveMaxTH, stExtFlags.TempBelowMinTH);

    }else
    if (MsgBase->ucPacketType == 0x2a)
    {
        msgM2 = (stHPLS2G_M2_MSG*)pMsg;

        printf("\nMessage M2\n");
        printf("Data Info: %f, %f, %f, %f, %f \n", msgM2->fTemp, msgM2->fRoll, msgM2->fPitch, msgM2->fYaw, msgM2->fMsep);

        hpls2g->FlagsToStruct(msgM2->usFlags, stFlags);
        hpls2g->ExtFlagsToStruct(msgM2->ExtFlags, stExtFlags);

        printf("Flags: %d, %d, %d\n", stFlags.GpsPwrOn, stFlags.IncPwrOn, stFlags.DgpsLock);
        printf("ExtFlags: %d, %d\n", stExtFlags.TempAboveMaxTH, stExtFlags.TempBelowMinTH);

    }else
```

Figure 7. Example of OnMessageRcvd

AUTHORIZATION TABLE

<table>
<tr><td colspan="5">Document Information</td></tr>
<tr><td>Document:</td><td colspan="2"><Document Name></td><td>Site:</td><td><SITE NAME></td></tr>
<tr><td>Purpose:</td><td colspan="4"><Operational Document/Acceptance Document/Technical Reference Document></td></tr>
<tr><td colspan="5">Authorization</td></tr>
<tr><td rowspan="7">Approved by:</td><td>Role</td><td>Name</td><td>Approval Date</td><td>Signature</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>

TABLE OF REVISIONS

Ver. #	Description	Author/s	Date
<0.0>			<DAY> <MONTH> <YEAR>