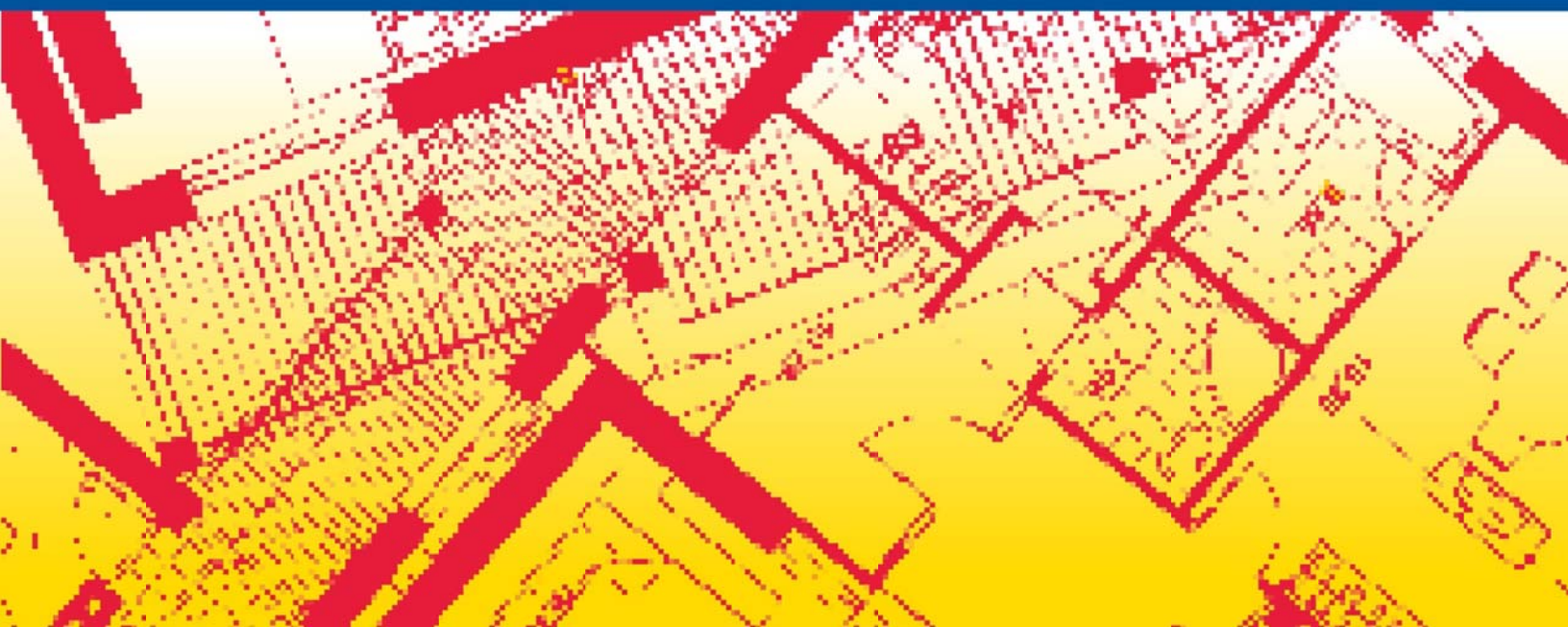
Trusted Positioning

Positioning Everywhere

TRUSTED POSITIONING NAVIGATION API

INTERNAL [TPN (F) (T), TMN(L)(A), TVN]

(VERSION M_J.M_I.P-R)



Contents

Introduction

Trusted Positioning Navigation API is a proprietary interface designed by Trusted Positioning. The document presents the function prototypes and data types available for application developers via the API. The first two sections explain the acronyms, abbreviations and definitions that are used throughout the document. Section (3) explains the naming conventions used by the API. The fourth, fifth and sixth sections present the files, function prototypes and data types that the application developers will use to embed multi-sensor based integration navigation functionality in their applications. Then, section (7) explains how to use the API by example code.

Trusted Positioning library is written in ANSI C code and does not depend on any third party libraries. The library is tested and supported on a variety of operating systems and platforms including – but not limited to – Microsoft Windows, and UNIX-Based operating systems.

1. Acronyms and Abbreviations

The list of acronyms and abbreviations used in the document is presented in Table 1.

Table 1 List of Acronyms and Abbreviations

Term	Definition
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
PVT	Position-Velocity-Time
IMU	Inertial Measurement Unit
TPP	Trusted Positioning Platform
T-PN,TPN	Trusted Portable Navigator
T-MN,TMN	Trusted Machine Navigator
T-VN,TVN	Trusted Vehicle Navigator
ZUPT	Zero Velocity Update
m	Metre
sec	Second
deg	Degree
rad	Radian
˚C	Degree Celsius
mBar	milli-Bar
mG	milli-Gauss
Hz	Hertz
mGal	milli-Gal

2. Definitions

Table 2 List of Definitions

Term	Definition
Navigation Device Frame	The navigation device is the device containing the sensor assembly. Trusted Positioning uses a device frame in which: 1. X-axis is the forward axis of the navigation device 2. Z-axis is the vertical axis of the navigation device pointing downwards 3. Y-axis is the lateral axis completing the right hand rule with the X, and Z axes
Epoch	Processing step related to the IMU data rate which is the highest available sensors' rate.
Height	Unless otherwise stated, height can be either "height above sea level" or "height above ellipsoid", but it should be consistent across all height inputs to the API functions through the same navigation session (i.e. from initialization to stop). The height from the navigation solution will follow the same definition as the input.
Heading	The heading angle is measured clockwise from the North direction.
Standard Deviation	Any standard deviation values provided to the library functions should be greater than zero.
Application Developer	The user of the library. "Application Developer" and "Developer" are used interchangeably in the document. In some instances, "User" will also be used.
Operator	The user of the application implemented by the developer.

3. Naming conventions

The naming conventions used for files, functions, and data types are defined in Table 3.

Table 3 Naming Conventions Used by Trusted Positioning API

	All files, functions and data types start with "***tpp***" which stands for **T**rusted **P**ositioning **P**latform. The first letter can be lower or upper case depending on the rules specified in this table.
Files	File names are lower-case and words are separated by underscores. **Example:** ***tpp_data_types.h***, ***tpp_api_functions.h***
Functions	Function names are lower-case and words are separated by underscores. **Example:** ***tpp_process_barometer(…)*** **Note:** The parameters are organized so that input parameters are ordered before output parameters.
Data Types	Data types start with a capital letter and a capital letter is added for each word, with no underscores between words. The type ends with the word ***Struct*** or ***Enum*** depending on the type. **Example:** ***TppImuMessageStruct***, ***TppReturnStatusEnum***
Structure Members	Structure member variables do not start with "***tpp***" because they are enclosed within the structure. Member variables are lower-case and they have a trailing underscore. **Example:** ***timetag_***
Enumerators	Enumerators are upper-case and words are separated by underscores. **Example:** ***TPP_INVALID_IMU_MESSAGE***

4. Files

There are three files that the application developer will have access to, in order to embed the functionality provided by the library in the developer's code. The files are presented in Table 4.

Table 4 Trusted Positioning Library Files

File Name	Description
tpp_data_types.h	This header file defines the initialization, input messages, output structure, and return-status data types that can be used by the developer directly in the code without any need to implement the structures from this document.
tpp_api_functions.h	This header file defines the API function pointers types that the developer can use directly when using the library.
LIBRARY- FILE	The library file that implements the functionality specified by the API. The name of the library file along with the file extension depends on the platform that the library is compiled for. The name of the library file would follow the convention below: **libtpp_<*navigator*>_<processor>_<*operating_system*>.<*extension*>** For example, the name for a TPN/Free library for Android would be: **libtpp_tpn_free_arm_android.so**

TPN_DSL

API Functions

tpp_get_api_version
tpp_create_navigation_session
tpp_initialize _dsl
tpp_advance_navigation_step
tpp_process_barometer
tpp_process_6dof_quaternions
tpp_process_imu_biases
tpp_delete_navigation_session

4.1. flag_name_

“DEVICE”
“BAROMETER”
“GNSS”
“PLATFORM_HEADING”
“POSITION”
“MODE_OF_TRANSIT”
“USE_CASE”
“GYROSCOPE_BIASES”
“ACCELEROMETER_CONFIGURATION”
“GYROSCOPE_X_CONFIGURATION”
“GYROSCOPE_Y_CONFIGURATION”
“GYROSCOPE_Z_CONFIGURATION”
“DEBUG_DATA”

4.2. Entities

Entity Id: 0x008C

5. TPN Free

API Functions

tpp_get_api_version
tpp_create_navigation_session
tpp_initialize_tpn_free
tpp_process_gnss_pvt
tpp_advance_navigation_step
tpp_process_barometer
tpp_process_magnetometer
tpp_process_speed
tpp_process_operator_2d_position
tpp_process_wireless
tpp_process_height
tpp_process_call_information
tpp_process_floor_information
tpp_process_platform_heading
tpp_process_9dof_quaternions
tpp_process_6dof_quaternions
tpp_process_imu_biases
tpp_process_misalignment
tpp_process_venue_map
tpp_set_mode_of_transit
tpp_set_zupt_mode
tpp_set_misalignment
tpp_set_magnetometer_calibration
tpp_set_use_case
tpp_set_device_heading
tpp_add_anchor_point
tpp_stop_navigation
tpp_delete_navigation_session
tpp_prerun_reset_zupt_thresholds
tpp_prerun_compute_zupt_thresholds_imu
tpp_process_gnss_observations

5.1. Flags

“DEVICE”
“START_OPTION”
“MAGNETOMETER”
“BAROMETER”
“SPEED”
“GNSS”
“WIFI”
“PLATFORM_HEADING”

“POSITION”
“MISALIGNMENT”
“MODE_OF_TRANSIT”
“USE_CASE”
“FLOOR_INFORMATION”
“GYROSCOPE_BIASES”
“ACCELEROMETER_CONFIGURATION”
GYROSCOPE_X_CONFIGURATION”
“GYROSCOPE_Y_CONFIGURATION”
“GYROSCOPE_Z_CONFIGURATION”
“MOTION_CONSTRAINTS”
“DEBUG_DATA”
“OUTPUT_POSITION_ERROR_WEIGHTING
“OUTPUT_VELOCITY_ERROR_WEIGHTING”
“OUTPUT_ATTITUDE_ERROR_WEIGHTING”
“ACCELEROMETER_BIASES”
“ZUPT_THRESHOLDS”
“DECLINATION_ANGLE”
“REPLACE_ATTITUDE_WITH_6DOF”
“PROCESSING_MODE”
“BACKWARD_SMOOTHING”
“LIBRARY_CALL_TYPE”

5.2. Structures

TppApiVersionStruct
TppGnssPvtMessageStruct
TppImuMessageStruct
TppBarometerMessageStruct
TppMagnetometerMessageStruct
TppSpeedMessageStruct
TppOperator2dPositionMessageStruct
TppWirelessMessageStruct
TppHeightMessageStruct
TppFloorInformationMessageStruct
TppCallInformationMessageStruct
TppPlatformHeadingMessageStruct
TppDeviceHeadingMessageStruct
TppQuaternionsMessageStruct
TppImuBiasesMessageStruct
TppVenueMapMessageStruct
TppProcessMisalignmentMessageStruct
TppSetMisalignmentMessageStruct
TppSetMisalignmentMessageStruct
TppSolutionStruct
TppPacketStruct
Output Array Structure

5.3. Enums

6.24. TppModeOfTransitEnum

TppZuptModeEnum
TppMagnetometerCalibrationEnum
TppUseCaseEnum
TppAnchorPointStruct
TppZuptThresholdsStruct
TppReturnStatusEnum
SUCCESS
TPP_SYSTEM_NOT_INITIALIZED

- TPP_INVALID_IMU_MESSAGE
- TPP_INVALID_GNSS_PVT_MESSAGE
- TPP_INVALID_BAROMETER_MESSAGE
- TPP_INVALID_MAGNETOMETER_MESSAGE
- TPP_INVALID_SPEED_MESSAGE
- ~~TPP_INVALID_2D_POSITION_MESSAGE~~
- TPP_INVALID_HEIGHT_MESSAGE
- TPP_INVALID_FLOOR_INFORMATION_MESSAGE
- TPP_INVALID_CALL_INFORMATION_MESSAGE
- ~~TPP_INVALID_PROCESS_2D_POSITION_CALL~~
- TPP_INVALID_SET_MODE_OF_TRANSIT_CALL
- TPP_INVALID_FLAG_MAGNETOMETER
- TPP_INVALID_FLAG_BAROMETER
- TPP_INVALID_FLAG_SPEED
- TPP_INVALID_FLAG_GNSS
- TPP_INVALID_HEADING_TO_START
- TPP_INVALID_POSITION_TO_START
- TPP_INVALID_FLAG_WIFI
- TPP_INVALID_FLAG_PLATFORM_HEADING
- TPP_INVALID_FLAG_POSITION
- TPP_INVALID_FLAG_MISALIGNMENT
- TPP_INVALID_FLAG_MODE_OF_TRANSIT
- TPP_INVALID_FLAG_MODE_OF_TRANSIT
- TPP_INVALID_FLAG_GYROSCOPE_BIASES
- TPP_INVALID_GYROSCOPE_BIASES_TO_START
- TPP_INVALID_FLAG_ACCELEROMETER_CONFIGURATION
- TPP_INVALID_FLAG_GYROSCOPE_X_CONFIGURATION
- TPP_INVALID_FLAG_GYROSCOPE_Y_CONFIGURATION
- TPP_INVALID_FLAG_GYROSCOPE_Z_CONFIGURATION
- TPP_INVALID_FLAG_DEBUG_DATA
- TPP_INVALID_FLAG_MOTION_CONSTRAINTS
- TPP_INVALID_IMU_MESSAGE_TIMETAG
- TPP_INVALID_IDENTIFIER_TO_START
- TPP_INVALID_MODE_OF_TRANSIT_TO_START
- TPP_INVALID_FLAG_BACKWARD_SMOOTHING
- TPP_BACKWARD_SMOOTHING_FILES_CREATION_FAILURE
- TPP_INVALID_RUN_BACKWARD_SMOOTHING_CALL
- TPP_RUN_BACKWARD_SMOOTHING_FAILURE
- TPP_INVALID_SYSTEM_DATE_TO_START
- TPP_INVALID_FLAG_START_OPTION
- TPP_INVALID_OPTION_SENSORS_ONLY
- TPP_INVALID_OPTION_SENSORS_RELIABLE_WIRELESS
- TPP_INVALID_OPTION_SENSORS_WIRELESS
- TPP_INVALID_FLAG_ACCELEROMETER_BIASES

- TPP_INVALID_PLATFORM_HEADING_MESSAGE
- TPP_INVALID_QUATERNIONS_MESSAGE
- TPP_INVALID_SET_ZUPT_MODE_CALL
- TPP_INVALID_SET_MISALIGNMENT_MESSAGE
- TPP_INVALID_SET_MISALIGNMENT_CALL

- TPP_INVALID_SET_MAGNETOMETER_CALIBRATION_CALL
- TPP_INVALID_SET_USE_CASE_CALL
- TPP_INVALID_FLAG_OUTPUT_POSITION_ERROR_WEIGHTING
- TPP_INVALID_FLAG_OUTPUT_VELOCITY_ERROR_WEIGHTING
- TPP_INVALID_FLAG_OUTPUT_ATTITUDE_ERROR_WEIGHTING
- TPP_INVALID_IMU_BIASES_MESSAGE
- TPP_INVALID_OPERATOR_2D_POSITION_MESSAGE
- TPP_INVALID_WIRELESS_MESSAGE
- TPP_INVALID_PROCESS_WIFI_CALL
- TPP_INVALID_PROCESS_MISALIGNMENT_MESSAGE
- TPP_INVALID_DEVICE_HEADING_MESSAGE
- TPP_INVALID_NAVIGATION_SESSION_HANDLE
- TPP_INVALID_FLAG_DECLINATION_ANGLE
- TPP_INVALID_FLAG_REPLACE_ATTITUDE_WITH_6DOF
- TPP_INVALID_ANCHOR_POINT_PACKET
- TPP_INVALID_START_ANCHOR_POINT
- TPP_INVALID_END_ANCHOR_POINT
- TPP_INVALID_FLAG_PROCESSING_MODE

5.4. Entities

Time Id: **0x00EA**

Position Id: **0x0016**

Position Standard Deviation Id: **0x00BD**

Velocity Id: **0x00A2**

Velocity Standard Deviation Id: **0x005F**

Attitude Id: **0x00E9**

Attitude Standard Deviation Id: **0x0060**

Accelerometer Bias Id: **0x006F**
Gyroscope Bias Id: **0x002E**

6. API Functions

The section presents the functions for the application developers.

6.1. tpp_get_api_version

Prototype	***void tpp_get_api_version(TppApiVersionStruct* version_struct_pointer)***
Summary	The function is used to get the version of the API
Parameters	***[OUT] version_struct_pointer***
Return Value	***[NONE]***
Notes	1. The function returns the major, minor, patch and release versions of the API 2. The major, minor, patch and release versions of the API can also be accessed through the #define statements that are provided in the file "tpp_data_types.h": ***TPP_API_VERSION_MAJOR*** ***TPP_API_VERSION_MINOR*** ***TPP_API_VERSION_PATCH*** ***TPP_API_VERSION_RELEASE_ID*** ***TPP_API_VERSION_RELEASE_NUMBER*** ***TPP_API_VERSION_SET_NUMBER***

6.2. tpp_create_navigation_session

Prototype	***TppNavigationSessionHandle tpp_create_navigation_session_handle()***
Summary	The function is used to create a navigation session handle.
Parameters	***[NONE]***
Return Value	If successful, a valid handle is returned. Otherwise, ***NULL*** is returned.
Notes	1. The function must be called once before any other functions that use a parameter of type ***TppNavigationSessionHandle*** is called. 2. If the other API functions are called with a ***NULL*** handle, ***TPP_INVALID_NAVIGATION_SESSION_HANDLE*** is returned.

6.3. tpp_initialize_tpn_free

Prototype	***TppReturnStatusEnum tpp_initialize_tpn_free(TppNavigationSessionHandle tpp_navigation_session_handle, const TppInitializationStruct* initialization_struct_pointer , TppPacketStruct* init_packet_struct_pointer)***
Summary	The function is used to initialize TPN Free.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] initialization_struct_pointer*** ***[OUT] init_packet_struct_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Section 7.28 shows all the values that can be returned by the initialization function.
Notes	1. The function must be called once after ***tpp_create_navigation_session()*** and

	before any other functions are called. 2. The ***init_packet_struct_pointer*** variable contains the populated byte array and its size that the developer can use to write such array in the beginning of any dataset file ahead of the integrated solution output packets that are generated every IMU sample. 3. The application developer is responsible for allocating the memory for the ***byte_array_pointer_*** member of the input argument ***init_packet_struct_pointer*** with the size ***TPP_INIT_BYTE_ARRAY_LENGTH*** and consequently is responsible for freeing the allocated memory space.

6.4. tpp_initialize_tpn_tethered

Prototype	***TppReturnStatusEnum tpp_initialize_tpn_tethered(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppInitializationStruct* initialization_struct_pointer ,*** ***TppPacketStruct* init_packet_struct_pointer)***
Summary	The function is used to initialize TPN Tethered.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] initialization_struct_pointer*** ***[OUT] init_packet_struct_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Section 7.28 shows all the values that can be returned by the initialization function.
Notes	1. The function must be called once before any other functions are called. 2. The ***init_packet_struct_pointer*** variable contains the populated byte array and its size that the developer can use to write such array in the beginning of any dataset file ahead of the integrated solution output packets that are generated every IMU sample. 3. The application developer is responsible for allocating the memory for the ***byte_array_pointer_*** member of the input argument ***init_packet_struct_pointer*** with the size ***TPP_INIT_BYTE_ARRAY_LENGTH*** and consequently is responsible for freeing the allocated memory space.

6.5. tpp_initialize_tmn_land

Prototype	***TppReturnStatusEnum tpp_initialize_tmn_land(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppInitializationStruct* initialization_struct_pointer ,*** ***TppPacketStruct* init_packet_struct_pointer)***
Summary	The function is used to initialize TMN Land.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] initialization_struct_pointer*** ***[OUT] init_packet_struct_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Section 7.28 shows all the values that can be returned by the initialization function.
Notes	1. The function must be called once before any other functions are called. 2. The ***init_packet_struct_pointer*** variable contains the populated byte array and its size that the developer can use to write such array in the beginning of any dataset file ahead of the integrated solution output packets that are generated every IMU sample.

	3. The application developer is responsible for allocating the memory for the ***byte_array_pointer_*** member of the input argument ***init_packet_struct_pointer*** with the size ***TPP_INIT_BYTE_ARRAY_LENGTH*** and consequently is responsible for freeing the allocated memory space.

6.6. tpp_initialize_tmn_aerial

Prototype	***TppReturnStatusEnum tpp_initialize_tmn_aerial(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppInitializationStruct* initialization_struct_pointer ,*** ***TppPacketStruct* init_packet_struct_pointer)***
Summary	The function is used to initialize TMN Aerial.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] initialization_struct_pointer*** ***[OUT] init_packet_struct_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Section 7.28 shows all the values that can be returned by the initialization function.
Notes	1. The function must be called once before any other functions are called. 2. The ***init_packet_struct_pointer*** variable contains the populated byte array and its size that the developer can use to write such array in the beginning of any dataset file ahead of the integrated solution output packets that are generated every IMU sample. 3. The application developer is responsible for allocating the memory for the ***byte_array_pointer_*** member of the input argument ***init_packet_struct_pointer*** with the size ***TPP_INIT_BYTE_ARRAY_LENGTH*** and consequently is responsible for freeing the allocated memory space.

6.7. tpp_initialize_tvn

Prototype	***TppReturnStatusEnum tpp_initialize_tvn(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppInitializationStruct* initialization_struct_pointer ,*** ***TppPacketStruct* init_packet_struct_pointer)***
Summary	The function is used to initialize TVN.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] initialization_struct_pointer*** ***[OUT] init_packet_struct_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Section 7.28 shows all the values that can be returned by the initialization function.
Notes	1. The function must be called once before any other functions are called. 2. The ***init_packet_struct_pointer*** variable contains the populated byte array and its size that the developer can use to write such array in the beginning of any dataset file ahead of the integrated solution output packets that are generated every IMU sample. 3. The application developer is responsible for allocating the memory for the ***byte_array_pointer_*** member of the input argument ***init_packet_struct_pointer*** with the size ***TPP_INIT_BYTE_ARRAY_LENGTH*** and

	consequently is responsible for freeing the allocated memory space.

6.8. tpp_process_gnss_pvt

Prototype	***TppReturnStatusEnum tpp_process_gnss_pvt(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppGnssPvtMessageStruct* gnss_pvt_message_pointer)***
Summary	This function is used when a valid data message from GNSS receiver is available.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] gnss_pvt_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise, the return value is ***TPP_INVALID_GNSS_PVT_MESSAGE*** or ***TPP_INVALID_PROCESS_GNSS_PVT_CALL***
Notes	1. The function must be used only after the system is initialized properly. 2. The function must be used only when valid GNSS data is available with a 3D fix status from the GNSS receiver.

6.9. tpp_advance_navigation_step

Prototype	***TppReturnStatusEnum tpp_advance_navigation_step(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppImuMessageStruct* imu_message_pointer,*** ***TppSolutionStruct* solution_struct_pointer,*** ***TppPacketStruct* output_packet_struct_pointer)***
Summary	This function is used when a valid IMU data message is available.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] imu_message_pointer*** ***[OUT] solution_struct_pointer*** ***[OUT] output_packet_struct_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise, the return value is ***TPP_INVALID_IMU_MESSAGE***, ***TPP_SYSTEM_NOT_INITIALIZED***, ***TPP_INVALID_IMU_MESSAGE_TIMETAG*** or ***TPP_INVALID_OUTPUT***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid IMU data is available. 3. If GNSS or any other sensor data sample is available at the same epoch at which an IMU data sample is available, ***tpp_advance_navigation_step()*** should be called last after ***tpp_process_gnss_pvt()*** and any other functions. 4. The application developer is responsible for allocating the memory for the ***byte_array_pointer_*** member of the input argument ***output_packet_struct_pointer*** with the size ***TPP_OUTPUT_BYTE_ARRAY_LENGTH*** and consequently is responsible for freeing the allocated memory space. 5. The developer can set any of the arguments ***solution_struct_pointer*** or ***output_packet_struct_pointer*** to ***NULL*** if the navigation solution is not required.

6.10. tpp_process_barometer

Prototype	***TppReturnStatusEnum tpp_process_barometer(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppBarometerMessageStruct* barometer_message_pointer)***
Summary	This function is used when a valid barometer data message is available.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] barometer_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise, the return value is ***TPP_INVALID_BAROMETER_MESSAGE*** or ***TPP_INVALID_PROCESS_BAROMETER_CALL***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid barometer data is available.

6.11. tpp_process_magnetometer

Prototype	***TppReturnStatusEnum tpp_process_magnetometer(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppMagnetometerMessageStruct* magnetometer_message_pointer)***
Summary	This function is used when a valid magnetometer data message is available.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] magnetometer_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise, the return value is ***TPP_INVALID_MAGNETOMETER_MESSAGE*** or ***TPP_INVALID_PROCESS_MAGNETOMETER_CALL***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid magnetometer data is available.

6.12. tpp_process_speed

Prototype	***TppReturnStatusEnum tpp_process_speed(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppSpeedMessageStruct* speed_message_pointer)***
Summary	This function is used when a valid speed data message is available.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] speed_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is ***TPP_INVALID_SPEED_MESSAGE*** or ***TPP_INVALID_PROCESS_SPEED_CALL***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid speed data is available.

6.13. tpp_process_multiple_antenna

Prototype	***TppReturnStatusEnum tpp_process_multiple_antenna(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppMultipleAntennaMessageStruct**** ***multiple_antenna_message_pointer)***

Summary	This function is used when a valid multiple antenna data message is available.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] multiple_antenna_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is ***TPP_INVALID_MULTIPLE_ANTENNA_MESSAGE*** or ***TPP_INVALID_PROCESS_MULTIPLE_ANTENNA_CALL***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid multiple antenna data is available.

6.14. tpp_process_operator_2d_position

Prototype	***TppReturnStatusEnum tpp_process_operator_2d_position(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppOperator2dPositionMessageStruct**** ***operator_2d_position_message_pointer)***
Summary	This function takes a valid 2D position and applies it as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] operator_2d_position_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is ***TPP_INVALID_OPERATOR_2D_POSITION_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when a valid 2D position update is available as an input from the operator. In this case, the output position will jump to the position provided to this function. Note that the GNSS position update should not be used with this function but instead it should be used with ***tpp_process_gnss_pvt()***

6.15. tpp_process_wireless

Prototype	***TppReturnStatusEnum tpp_process_wireless (*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppWirelessMessageStruct* wireless_message_pointer)***
Summary	This function takes valid wireless data and applies it as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] wireless_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is ***TPP_INVALID_WIRELESS_MESSAGE*** or ***TPP_INVALID_PROCESS_WIFI_CALL***
Notes	1. The function must be used only after the system is initialized properly.

6.16. tpp_process_height

Prototype	***TppReturnStatusEnum tpp_process_height(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,***

	const TppHeightMessageStruct* height_message_pointer)
Summary	This function takes a valid height and applies it as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] height_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator ***TPP_INVALID_HEIGHT_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when a valid height is available as an input from any external system or operator. Note that the GNSS height update and barometer height should not be used with this function but instead they should be used with their respective functions.

6.17. tpp_process_call_information

Prototype	***TppReturnStatusEnum tpp_process_call_information(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppCallInformationMessageStruct**** ***call_information_message_pointer)***
Summary	This function takes the call information (i.e. call mode, proximity detected, headset type, and speaker activity) and applies it as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] call_information_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator ***TPP_INVALID_CALL_INFORMATION_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly. 2. If the developer wants to use this function, then it must be called at the highest rate of sensor data which is the IMU data rate. It should be called before ***tpp_advance_navigation_step()***

6.18. tpp_process_floor_information

Prototype	***TppReturnStatusEnum tpp_process_floor_information(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppFloorInformationMessageStruct**** ***floor_information_message_pointer)***
Summary	This function takes floor information (i.e. floor number and height between floors) from the developer and applies it as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] floor_information_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator ***TPP_INVALID_FLOOR_INFORMATION_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid floor information is available as an input

	from the operator. 3. A zero value passed to the ***height_between_floors_*** member in the input structure will cause the navigation solution to use the recent value of the height between floors which can be one of (a) a positive value to the ***height_between_floors_*** member in the input structure in a previous call to this function (b) the height between floors specified during initialization by ***flag_value_2_*** in ***"FLOOR_INFORMATION"*** or its default value. 4. A negative value passed to the ***height_between_floors_*** member in the input structure will return ***TPP_INVALID_FLOOR_INFORMATION***

6.19. tpp_process_platform_heading

Prototype	***TppReturnStatusEnum tpp_process_platform_heading(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppPlatformHeadingMessageStruct**** ***platform_heading_message_pointer)***
Summary	This function takes a valid platform heading and applies it as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] platform_heading_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code ***TPP_INVALID_PLATFORM_HEADING_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when a valid platform heading is available as an input from any external system or operator.

6.20. tpp_process_9dof_quaternions

Prototype	***TppReturnStatusEnum tpp_process_9dof_quaternions(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppQuaternionsMessageStruct* quaternions_message_pointer)***
Summary	This function takes a valid 9dof quaternion vector to derive the pitch, roll and heading and applies the derived information as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] quaternions_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code ***TPP_INVALID_QUATERNIONS_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly.

6.21. tpp_process_6dof_quaternions

Prototype	***TppReturnStatusEnum tpp_process_6dof_quaternions(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppQuaternionsMessageStruct* quaternions_message_pointer)***
Summary	This function takes a valid 6dof quaternion vector to derive the pitch, roll and heading

	and applies the derived information as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] quaternions_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code ***TPP_INVALID_QUATERNIONS_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly.

6.22. tpp_process_imu_biases

Prototype	***TppReturnStatusEnum tpp_process_imu_biases(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppImuBiasesMessageStruct* imu_biases_message_pointer)***
Summary	This function takes the gyroscope and accelerometer biases and may apply those biases as corrections to the gyroscope and accelerometer data values input to ***tpp_advance_navigation_step()***
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] imu_biases_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code ***TPP_INVALID_IMU_BIASES***
Notes	1. The function must be used only after the system is initialized properly. 2. The function shall be called before ***tpp_advance_navigation_step()***at the same rate.

6.23. tpp_process_misalignment

Prototype	***TppReturnStatusEnum tpp_process_misalignment(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppProcessMisalignmentMessageStruct**** ***misalignment_message_pointer)***
Summary	This function takes the misalignment angle and its standard deviation and applies it as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] misalignment_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code ***TPP_INVALID_PROCESS_MISALIGNMENT_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly.

6.24. tpp_process_venue_map

Prototype	***TppReturnStatusEnum tpp_process_venue_map(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppVenueMapMessageStruct* venue_map_message_pointer)***
Summary	This function takes venue map related information (such as 2D position, height, platform heading, and the map entity) from the developer and applies it as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle***

	[IN] venue_map_message_pointer
Return Value	If successful, *TPP_SUCCESS* is returned. Otherwise the return value is the error code enumerator *TPP_INVALID_VENUE_MAP_MESSAGE*
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid map information is available.

6.25. tpp_set_mode_of_transit

Prototype	*TppReturnStatusEnum tpp_set_mode_of_transit(* *TppNavigationSessionHandle tpp_navigation_session_handle,* *TppModeOfTransitEnum mode_of_transit_enum)*
Summary	This function sets the current mode of transit of the navigator to the one specified as an input.
Parameters	*[IN] tpp_navigation_session_handle* *[IN] mode_of_transit_enum*
Return Value	If successful, *TPP_SUCCESS* is returned. Otherwise the return value is the error code enumerator *TPP_INVALID_SET_MODE_OF_TRANSIT_CALL*
Notes	1. The function must be used only after the system is initialized properly. 2. The function can be used to change the mode of transit, only if the developer has initialized mode of transit for the navigator to be automatic.

6.26. tpp_set_zupt_mode

Prototype	*TppReturnStatusEnum tpp_set_zupt_mode(* *TppNavigationSessionHandle tpp_navigation_session_handle,* *TppZuptModeEnum zupt_mode_enum)*
Summary	This function sets the current ZUPT mode of the navigator to the one specified as an input.
Parameters	*[IN] tpp_navigation_session_handle* *[IN] zupt_mode_enum*
Return Value	If successful, *TPP_SUCCESS* is returned. Otherwise the return value is the error code enumerator *TPP_INVALID_SET_ZUPT_MODE_CALL*
Notes	1. The function must be used only after the system is initialized properly.

6.27. tpp_set_misalignment

Prototype	*TppReturnStatusEnum tpp_set_misalignment(* *TppNavigationSessionHandle tpp_navigation_session_handle,* *const TppSetMisalignmentMessageStruct** *set_misalignment_message_pointer)*
Summary	This function enables or disables the automatic misalignment estimation. When the misalignment estimation is turned off, the developer can specify the misalignment angle for the navigator to use.
Parameters	*[IN] tpp_navigation_session_handle*

	[IN] set_misalignment_message_pointer
Return Value	If successful, *TPP_SUCCESS* is returned. Otherwise the return value is the error code enumerator *TPP_INVALID_SET_MISALIGNMENT_MESSAGE* or *TPP_INVALID_SET_MISALIGNMENT_CALL*
Notes	1. The function must be used only after the system is initialized properly. 2. The function can be used to turn the automatic misalignment estimation on or off, only if the developer has initialized the misalignment for the navigator to be automatic.

6.28. tpp_set_magnetometer_calibration

Prototype	***TppReturnStatusEnum tpp_set_magnetometer_calibration(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***TppMagnetometerCalibrationEnum magnetometer_calibration_enum)***
Summary	This function enables or disables the automatic magnetometer calibration.
Parameters	*[IN] tpp_navigation_session_handle* *[IN] magnetometer_calibration_enum*
Return Value	If successful, *TPP_SUCCESS* is returned. Otherwise the return value is the error code enumerator *TPP_INVALID_SET_MAGNETOMETER_CALIBRATION_CALL*
Notes	1. The function must be used only after the system is initialized properly. 2. The function can be used to enable or disable the magnetometer calibration, only if the magnetometer was initialized to be on.

6.29. tpp_set_use_case

Prototype	***TppReturnStatusEnum tpp_set_use_case(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***TppUseCaseEnum use_case_enum)***
Summary	This function sets the use case (such as pocket, hand dangling, or others).
Parameters	*[IN] tpp_navigation_session_handle* *[IN] use_case_enum*
Return Value	If successful, *TPP_SUCCESS* is returned. Otherwise the return value is the error code enumerator *TPP_INVALID_SET_USE_CASE_CALL*
Notes	1. The function must be used only after the system is initialized properly. 2. The function can be used to set the use case, only if the developer has initialized the mode of transit for the navigator to be walking or automatic.

6.30. tpp_set_orientation_based_on_pitch (INTERNAL ONLY)

Prototype	***TppReturnStatusEnum tpp_set_orientation_based_on_pitch(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***TppOrientationBasedOnPitchEnum orientation_enum)***
Summary	This function sets the orientation based on pitch angle (Vertical Up, Horizontal, or Vertical Down)
Parameters	*[IN] tpp_navigation_session_handle*

	[IN] orientation_enum
Return Value	If successful, ***TPP_SUCCESS*** is returned.
Notes	1. The function must be used only after the system is initialized properly.

6.31. tpp_set_device_heading

Prototype	***TppReturnStatusEnum tpp_set_device_heading(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppDeviceHeadingMessageStruct**** ***device_heading_message_pointer)***
Summary	This function takes a valid device angle directly, or computes the device heading from a valid platform heading and a misalignment angle, and sets it for the navigator.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] device_heading_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code ***TPP_INVALID_DEVICE_HEADING_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when a valid device heading is available, or a valid platform heading and misalignment angle are available, as an input from any external system or operator.

6.32. tpp_add_anchor_point

Prototype	TppReturnStatusEnum tpp_add_anchor_point ***(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***Const TppAnchorPointMessageStruct* anchor_point_message_pointer)***
Summary	The function adds an anchor point to the output packet.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] anchor_point_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator, ***TPP_INVALID_ANCHOR_POINT_PACKET***, ***TPP_INVALID_START_ANCHOR_POINT*** *or* ***TPP_INVALID_END_ANCHOR_POINT***
Notes	1. The function must be used only after the system is initialized properly. 2. The function must be used only when a valid anchor point is available. Those anchor points are needed if the developer is using Invensense Smoothing Library (ISL) along with the TPP library.

6.33. tpp_add_synchronization_event

Prototype	TppReturnStatusEnum tpp_add_synchronization_event ***(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***TppTypeUint64 event_number)***
Summary	The function adds a unique (i.e. within a navigation session) event number to the output packet. This event number should be greater than zero and is mainly used to give the capability to synchronize between datasets generated from multiple systems.

Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] event_number***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator ***TPP_INVALID_SYNCHRONIZATION_EVENT***
Notes	1. The function must be used only after the system is initialized properly.

6.34. tpp_stop_navigation

Prototype	***TppReturnStatusEnum tpp_stop_navigation(*** ***TppNavigationSessionHandle tpp_navigation_session_handle)***
Summary	This function is used to stop navigation
Parameters	***[IN] tpp_navigation_session_handle***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator ***TPP_STOP_NAVIGATION_FAILURE***
Notes	1. The function must be used only after the system is initialized properly. 2. The function must be called before the main application exits or before restarting navigation in the same session.

6.35. tpp_delete_navigation_session

Prototype	***TppReturnStatusEnum tpp_delete_navigation_session_handle(*** ***TppNavigationSessionHandle tpp_navigation_session_handle)***
Summary	The function is used to delete a navigation session handle.
Parameters	***[IN] tpp_navigation_session_handle***
Return Value	If successful, a TPP_SUCCESS is returned
Notes	1. The function can be called after ***tpp_stop_navigation()***to reclaim the memory allocated for a specific navigation session.

6.36. tpp_prerun_reset_zupt_thresholds

Prototype	***void tpp_prerun_reset_zupt_thresholds()***
Summary	This function is used to reset the data saved to compute the ZUPT thresholds.
Parameters	***[NONE]***
Return Value	***[NONE]***
Notes	1. The function must be called before calling the other functions that are used to compute the ZUPT thresholds.

6.37. tpp_prerun_compute_zupt_thresholds_imu

Prototype	***TppReturnStatusEnum tpp_prerun_compute_zupt_thresholds_imu(*** ***const TppImuMessageStruct* imu_message_pointer,*** ***TppZuptThresholdsStruct* zupt_threshold_struct,*** ***unsigned long* number_of_valid_samples_required)***
Summary	This function is to compute the ZUPT threshold values from IMU samples only.

Parameters	***[IN] imu_message_pointer*** ***[OUT] zupt_threshold_struct*** ***[OUT] number_of_valid_samples_required***
Return Value	1. ***TPP_ZUPT_THRESHOLDS_NOT_RESET*** is returned when ***tpp_prerun_reset_zupt_thresholds()*** is not called before calling the current function. 2. ***TPP_INVALID_IMU_MESSAGE*** is returned when the 1st parameter passed, ***imu_message_pointer***, is a null pointer. 3. ***TPP_COMPUTATION_IN_PROGRESS*** is returned if there are no errors, while the function is computing the ZUPT thresholds. The user can refer to the 3rd parameter, ***number_of_valid_samples_required***, to know the number of samples needed by the function to compute the thresholds. 4. ***TPP_SUCCESS*** is returned when the thresholds are ready. The user can refer to the 2nd parameter, ***zupt_threshold_struct***, where the threshold data is filled.
Notes	1. The function is used to compute the ZUPT threshold values from IMU samples only.

6.38. tpp_run_backward_smoothing

Prototype	***TppReturnStatusEnum tpp_run_backward_smoothing(*** ***TppNavigationSessionHandle tpp_navigation_session_handle*** ***)***
Summary	This function is used to run backward smoothing and generate a file that contains the backward smoothed navigation solution.
Parameters	***[IN] tpp_navigation_session_handle***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Section 7.28 shows all the values that can be returned by the function.
Notes	1. The function can be used only after the **BACKWARD_SMOOTHING** initialization flag is initialized successfully, with **flag_value_1_** set to "on" and **flag_value_2_** set to a folder path in which the backward smoothed navigation solution is saved. 2. The function must be used only after the system is stopped successfully. 3. A file with the name "**bs_solution.dat**" containing the backward smoothed navigation solution is generated by this function in the path specified during initialization. 4. "**bs_solution.dat**" is a *binary* file that contains the backward smoothed navigation solution. An output packet, with the structure specified in section 7.23.1, is present for every IMU sample that was input to **tpp_advance_navigation_step()**.

6.39. tpp_process_gnss_observations

Prototype	***TppReturnStatusEnum tpp_process_gnss_observations(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppGnssObservationsMessageStruct* gnss_obs_message_pointer)***
Summary	This function is used when a valid GNSS observations data message is available.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] gnss_obs_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise, the return value is the error code enumerator ***TPP_INVALID_GNSS_OBSERVATIONS_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid GNSS observations data is available.

6.40. tpp_process_glonass_ephemeris

Prototype	***TppReturnStatusEnum tpp_process_glonass_ephemeris(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppGlonassEphemerisMessageStruct**** ***glonass_eph_message_pointer)***
Summary	This function is used when a valid GLONASS Ephemeris data message is available.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] glonass_eph_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator ***TPP_INVALID_GLONASS_EPHEMERIS_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid GLONASS Ephemeris data is available.

6.41. tpp_process_gps_ephemeris

Prototype	***TppReturnStatusEnum tpp_process_gps_ephemeris(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppGpsEphemerisMessageStruct* gps_eph_message_pointer)***
Summary	This function is used when a valid GPS Ephemeris data message is available.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] gps_eph_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator ***TPP_INVALID_GPS_EPHEMERIS_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid GPS Ephemeris data is available.

6.42. tpp_multi_device_process_position_velocity

Prototype	TppReturnStatusEnum tpp_multi_device_process_position_velocity(***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***unsigned char number_of_devices,*** ***const TppMultiDeviceLatencyStruct* multi_device_latency_pointer,*** ***const TppMultiDevicePositionVelocityStruct**** ***multi_device_position_velocity_pointer,*** ***)***
Summary	The function is used to process the position and velocity information from a number of devices in a multi-device system.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] number_of_devices*** ***[IN] multi_device_latency_pointer*** ***[IN] multi_device_position_velocity_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator ***TPP_MULTI_DEVICE_INVALID_NUMBER_OF_DEVICES*** or ***TPP_MULTI_DEVICE_INVALID_LATENCY_MESSAGE*** or ***TPP_MULTI_DEVICE_INVALID_POSITION_VELOCITY_MESSAGE*** or

Notes	1. The function must be used only for a multi-device system. 2. The function must be used only after the system is initialized properly. 3. The number of devices should be at least 2 including the information generated on the device processing all the information from other devices in the multi-device system.

6.43. tpp_multi_device_process_heading

Prototype	TppReturnStatusEnum tpp_multi_device_process_heading (***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***unsigned char number_of_devices,*** ***const TppMultiDeviceLatencyStruct* multi_device_latency_pointer,*** ***const TppMultiDeviceHeadingStruct* multi_device_heading_pointer,*** ***)***
Summary	The function is used to process the heading information from a number of devices in a multi-device system.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] number_of_devices*** ***[IN] multi_device_latency_pointer*** ***[IN] multi_device_heading_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator ***TPP_MULTI_DEVICE_INVALID_NUMBER_OF_DEVICES*** or ***TPP_MULTI_DEVICE_INVALID_LATENCY_MESSAGE*** or ***TPP_MULTI_DEVICE_INVALID_HEADING_MESSAGE*** or
Notes	1. The function must be used only for a multi-device system. 2. The function must be used only after the system is initialized properly. 3. The number of devices should be at least 2 including the information generated on the device processing all the information from other devices in the multi-device system.

6.44. tpp_multi_device_process_secondary_gnss_pvt

Prototype	TppReturnStatusEnum tpp_multi_device_process_secondary_gnss_pvt (***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***signed char navigation_phase,*** ***const TppGnssPvtMessageStruct* gnss_pvt_message_pointer,*** ***)***
Summary	One of the devices in the multi-device system is assumed to share its GNSS data with the rest of the devices in the system (whether they have a GNSS receiver or not). The function is used to process the GNSS data message shared from this secondary device in the system.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] navigation_phase*** ***[IN] gnss_pvt_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise, the return value is ***TPP_INVALID_GNSS_PVT_MESSAGE*** or ***TPP_INVALID_PROCESS_GNSS_PVT_CALL***
Notes	1. The function must be used only for a multi-device system. 2. The function must be used only after the system is initialized properly.

	3. The function must be used only when valid GNSS data is available with a 3D fix status from the GNSS receiver on the secondary device.

6.45. tpp_process_street_map

Prototype	***TppReturnStatusEnum tpp_process_street_map(*** ***TppNavigationSessionHandle tpp_navigation_session_handle,*** ***const TppStreetMapMessageStruct* street_map_message_pointer)***
Summary	This function takes street map related information (such as 2D position, height, and platform heading) from the developer and applies it as an update to the navigation solution.
Parameters	***[IN] tpp_navigation_session_handle*** ***[IN] street_map_message_pointer***
Return Value	If successful, ***TPP_SUCCESS*** is returned. Otherwise the return value is the error code enumerator ***TPP_INVALID_STREET_MAP_MESSAGE***
Notes	1. The function must be used only after the system is initialized properly. 2. This function must be used only when valid street map information is available.

7. Data Types

This section defines the data types in the API.

7.1. TppNavigationSessionHandle

Defined as a type definition of pointer to void [***typedef void* TppNavigationSessionHandle***]

7.2. TppInitializationStruct

The initialization structures are used during the initialization of the navigator. The main initialization structure is "***TppInitializationStruct***" and the developer will need to correctly fill a variable of this type by specifying the number of initialization flags and an array of the type "***TppInitializationFlagStruct***". An initialization flag is a structure that is composed of six entries. The first entry is the name of the flag while the other five entries represent the values that are related to the flag's name. Section 8.1 explains how to initialize the navigator with the "***TppInitializationStruct***" structure.

	Member	Data Type	Unit	Description
1	number_of_initialization_flags_	UINT16	N/A	Number of initialization flags.
2	initialization_flag_pointer_	TppInitializationFlagStruct*	N/A	Pointer to an array of initialization flags. An entry in the array is of type ***TppInitializationFlagStruct***

7.2.1. *TppInitializationFlagStruct*

	Member	Data Type	Unit	Description
1	flag_name_	CHAR*	N/A	A pointer to characters that specifies the name of the flag.
2	flag_value_1_	CHAR*	N/A	A pointer of characters that represents the first value related to the flag name.
3	flag_value_2_	CHAR*	N/A	A pointer of characters that represents the second value related to the flag name.
4	flag_value_3_	CHAR*	N/A	A pointer of characters that represents the third value related to the flag name.
5	flag_value_4_	CHAR*	N/A	A pointer of characters that represents the fourth value related to the flag name.
6	flag_value_5_	CHAR*	N/A	A pointer of characters that represents the fifth value related to the flag name.

Table 5 shows a list of the available flag names, followed by the values that each flag can take.

Table 5 List of Flag Names

flag_name_	Description
"DEVICE"	Specifies the type of the device; whether it is a phone or a tablet or any other type.

	DEFAULT (If flag name not specified): {“DEVICE”, “phone”}
“START_OPTION”	Specifies if the navigation solution will (1) Start without wireless systems and continue to work with the sensors only (2) Start with reliable data from the wireless systems (i.e. GNSS or Wi-Fi) (3) Start immediately and accepts the wireless systems as it becomes available. **DEFAULT (If flag name not specified):** {“START_OPTION”, “sensors_reliable_wireless”}
“MAGNETOMETER”	Specifies whether the data from the magnetometer will be used in the navigation solution or not, along with other settings if the magnetometer will be used. **DEFAULT (If flag name not specified):** {“MAGNETOMETER”, “off”}
“BAROMETER”	Specifies whether the data from the barometer will be used in the navigation solution or not, along with other settings if the barometer will be used. **DEFAULT (If flag name not specified):** {“BAROMETER”, “off”}
“SPEED”	Specifies whether the data from the speed sensor will be used in the navigation solution or not, along with other settings if the speed sensor will be used. **DEFAULT (If flag name not specified):** {“SPEED”, “off”}
“GNSS”	Specifies whether the data from GNSS will be used in the navigation solution or not, along with other settings if GNSS will be used. See also “**PLATFORM_HEADING**” flag. **For TMN and TVN:** **DEFAULT (If flag name not specified):** {“GNSS”, “on”,“gnss_precision”,“1.0”,”1.0”} **For TPN:** **DEFAULT (If flag name not specified):** {“GNSS”, “on”,“gnss_high_sensitivity”,“1.0”,”1.0”} **Notes:** (1) If the developer specified the value for the “**GNSS**” flag to be “**off**”, the developer must use either the “**PLATFORM_HEADING**” flag to specify the initial heading of the platform or the “**MAGNETOMETER**” flag and set the first flag value to be “**on**”; otherwise, the initialization will not be successful (i.e. ***TPP_INVALID_HEADING_TO_START*** will be returned by ***tpp_initialize_SSS()***) (2) If the developer specified the value for the “**GNSS**” flag to be “**off**”, the developer must also use the “**POSITION**” flag to specify the initial position of the platform; otherwise, the initialization will not be successful (i.e. ***TPP_INVALID_POSITION_TO_START*** will be returned by ***tpp_initialize_SSS()***)

	For TMN and TVN: (3) If the developer specified the value of **flag_value_1_** for "**START_OPTION**" to be "**sensors_only**", then **flag_value_1_** for "**GNSS**" flag should be set to "**off**". **For TPN:** (3) If the developer specified the value for the "**GNSS**" flag to be "**off**", the "**MODE_OF_TRANSIT**" flag should be set to "**walking**"; otherwise, the initialization will not be successful (i.e. ***TPP_INVALID_MODE_OF_TRANSIT_TO_START*** will be returned by ***tpp_initialize_SSS()***). (4) If the developer specified the value of **flag_value_1_** for "**START_OPTION**" to be "**sensors_only**", then **flag_value_1_** for "**GNSS**" flag should be set to "**off**", and the "**MODE_OF_TRANSIT**" flag should be set to "**walking**" for the initialization to be successful.
"WIFI"	Specifies whether the data from Wi-Fi will be used in the navigation solution or not, along with other settings if Wi-Fi will be used. **DEFAULT (If flag name not specified):** {"WIFI", "off" }
"MULTIPLE_ANTENNA"	Specifies whether the data from the multiple antenna system will be used in the navigation solution or not, along with other settings if multiple antenna system will be used. **DEFAULT (If flag name not specified):** {"MULTIPLE_ANTENNA", "off"}
"PLATFORM_HEADING"	**For TMN and TVN:** Specifies the initial heading of the platform. If the developer wants the navigator to start computing the integrated navigation solution without waiting for good GNSS data, the developer can use this flag to specify the initial heading of the platform; this is needed if a calibrated magnetometer is unavailable or the value for the "**MAGNETOMETER**" flag is "**off**". **For TPN:** Specifies the initial heading of the platform. If the developer specified the value for the "**GNSS**" flag to be "**on**", this is an optional flag. For more information, refer to the "**GNSS**" flag description.
"POSITION"	**For TMN and TVN:** Specifies the initial position (Latitude, Longitude, and Height) of the platform. If the developer wants the navigator to start computing the integrated navigation solution without waiting for good GNSS data, the developer can use this flag to specify the initial position of the platform. **For TPN:** Specifies the initial position (Latitude, Longitude, and Height) of

	the platform. If the developer specified the value for the "**GNSS**" flag to be "**on**", this is an optional flag. For more information, refer to the "**GNSS**" flag description.
"MISALIGNMENT"	Specifies values that are related to the misalignment angle of the device with respect to the platform. **DEFAULT (If flag name not specified):** {"MISALIGNMENT", "automatic", "misalignment_normal"}
"MODE_OF_TRANSIT"	Specifies the mode of transit of the navigator. **DEFAULT (If flag name not specified):** {"MODE_OF_TRANSIT", "automatic"} For more information, refer to the "**GNSS**" flag description.
"USE_CASE"	Specifies the use case of the navigator. **DEFAULT (If flag name not specified):** {"USE_CASE", "automatic"}
"FLOOR_INFORMATION"	Specifies the initial floor information. **DEFAULT (If flag name not specified):** {"FLOOR_INFORMATION", "1", "3.0"}
"GYROSCOPE_BIASES"	Specifies the initial biases for the gyroscopes. **DEFAULT (If flag name not specified):** N/A. The user must specify the gyroscope biases during initialization. Otherwise, the initialization function will return an error.
"ACCELEROMETER_CONFIGURATION"	Specifies the configuration for the accelerometers. **For TMN and TVN:** **DEFAULT (If flag name not specified):** {"ACCELEROMETER_CONFIGURATION", "accelerometer_5"} **For TPN:** **DEFAULT (If flag name not specified):** {"ACCELEROMETER_CONFIGURATION", "accelerometer_1"}
"GYROSCOPE_X_CONFIGURATION"	Specifies the configuration for the x-axis gyroscope. **For TMN and TVN:** **DEFAULT (If flag name not specified):** {"GYROSCOPE_X_CONFIGURATION", "gyroscope_5"} **For TPN:** **DEFAULT (If flag name not specified):** {"GYROSCOPE_X_CONFIGURATION", "gyroscope_1"}
"GYROSCOPE_Y_CONFIGURATION"	Specifies the configuration for the y-axis gyroscope. **For TMN and TVN:** **DEFAULT (If flag name not specified):** {"GYROSCOPE_Y_CONFIGURATION", "gyroscope_5"} **For TPN:** **DEFAULT (If flag name not specified):** {"GYROSCOPE_Y_CONFIGURATION", "gyroscope_1"}
"GYROSCOPE_Z_CONFIGURATION"	Specifies the configuration for the z-axis gyroscope. **For TMN and TVN:** **DEFAULT (If flag name not specified):** {"GYROSCOPE_Z_CONFIGURATION", "gyroscope_5"}

	For TPN: **DEFAULT (If flag name not specified):** {“GYROSCOPE_Z_CONFIGURATION”, “gyroscope_1”}
“MOTION_CONSTRAINTS”	Specifies a configuration of two velocity constraints. **For TMN and TVN:** **DEFAULT (If flag name not specified):** {“MOTION_CONSTRAINTS”, “motion_constraints_4”} **For TPN:** **DEFAULT (If flag name not specified):** {“ MOTION_CONSTRAINTS”, “motion_constraints_1”}
“DEBUG_DATA”	Specifies whether the entities that hold the input data will be generated in the “Output Per Epoch” packet or not. Those entities are mainly used for debugging by Trusted Positioning. **DEFAULT (If flag name not specified):** {“ DEBUG_DATA”, “off”}
“OUTPUT_POSITION_ERROR_WEIGHTING”	Specifies the value of weighting for the position standard deviation computed by the navigator. **DEFAULT (If flag name not specified):** {“OUTPUT_POSITION_ERROR_WEIGHTING”, “1”, “1”, “1”}
“OUTPUT_VELOCITY_ERROR_WEIGHTING”	Specifies the value of weighting for the velocity standard deviation computed by the navigator. **DEFAULT (If flag name not specified):** {“OUTPUT_VELOCITY_ERROR_WEIGHTING”, “1”, “1”, “1”}
“OUTPUT_ATTITUDE_ERROR_WEIGHTING”	Specifies the value of weighting for the attitude standard deviation computed by the navigator. **DEFAULT (If flag name not specified):** {“OUTPUT_ATTITUDE_ERROR_WEIGHTING”, “1”, “1”, “1”}
“ACCELEROMETER_BIASES”	Specifies the initial biases for the accelerometers. **DEFAULT (If flag name not specified):** {“ACCELEROMETER_BIASES”, “0.0”, “0.0”, “0.0”}
“ZUPT_THRESHOLDS”	Specifies the initial ZUPT thresholds. **DEFAULT** 1. If flag name is not specified and **flag_value_1_** for the “**GNSS**” flag is “**on**”, the default values will {“ZUPT_THRESHOLDS”, “0.0”, “0.0”, “0.0”} 2. If flag name is not specified and **flag_value_1_** for the “GNSS” flag is “off”, there is no default value and the user must specify the ZUPT thresholds during initialization. Otherwise, the initialization function will return an error.
“DECLINATION_ANGLE”	Specifies the initial declination angle. **DEFAULT (If flag name not specified):** {“DECLINATION_ANGLE”, “0.0” }
“REPLACE_ATTITUDE_WITH_6DOF”	Specifies whether to replace the attitude computation done in the library with 6dof or not **DEFAULT (If flag name not specified):** {“REPLACE_ATTITUDE_WITH_6DOF”, “off” }

"PROCESSING_MODE"	Specifies whether the navigator is processing the data in the forward or backward mode. **DEFAULT (If flag name not specified):** {" PROCESSING_MODE", "forward" }
"BACKWARD_SMOOTHING"	Specifies whether the information required to run the backward smoothing functionality will be saved or not. If the user called **tpp_run_backward_smoothing()** with this information not being saved, the function will return an error. **DEFAULT (If flag name not specified):** {"BACKWARD_SMOOTHING", "off"}
"LIBRARY_CALL_TYPE"	Specifies whether the library will be running in the standalone mode or the ISL mode. DEFAULT (If flag name not specified): {"LIBRARY_CALL_TYPE", "standalone"}

flag_name_	Description
"CHEST_MOUNT"	"on" or "off"
"NON_HOLONOMIC"	"on" or "off"
"AUTOMATIC_ZUPT_IN_MACHINE"	"on" or "off"
"ZUPT"	"on" or "off"
"TIGHTLY_COUPLED"	"on" or "off"
~~"AIRPLANE"~~	~~"on" or "off"~~
"CHILDREN_FILTERS"	"N"
~~"GNSS_SCALE_FACTOR"~~	~~"F.F"~~
~~"MAGNETOMETER_SCALE_FACTOR"~~	~~"F.F"~~
~~"BAROMETER_SCALE_FACTOR"~~	~~"F.F"~~
"PEDESTRIAN_DEAD_RECKONING"	"F.F"
~~"WIFI"~~	~~"F.F"~~
"OLD_DATASET"	"on" or "off"
"SENSORS_RATE"	"N"
"GNSS_RATE"	"N"
~~"IMU_SYSTEM_NAME"~~	
~~"GYROSCOPE_BIASES"~~	~~"F.F", "F.F", "F.F"~~
"ACCELEROMETER_BIASES"	"F.F", "F.F", "F.F"
"GNSS_REJECTION"	"on" or "off"

7.2.1. *DEVICE flag values*

flag_name_	"DEVICE"	
flag_value_1_	"phone" or "tablet" or "head_mount" or "watch"	"phone" specifies to the navigator that the device is a phone. "tablet" specifies to the navigator that the device is a tablet. "head_mount" specifies to the navigator that the device is head-mounted such as glasses. "watch" specifies to the navigator that the device is a watch.

7.2.2. *START_OPTION flag values*

flag_name_	"START_OPTION"	
flag_value_1_	"sensors_only"	"sensors_only" specifies that only the data from the sensors will be used in

	or “sensors_reliable_wireless” or “sensors_wireless”	the navigation solution. If **flag_value_1_** for **GNSS** or **WIFI** flags is “on”, the initialization function will not be successful (i.e. ***TPP_INVALID_OPTION_SENSORS_ONLY*** will be returned by ***tpp_initialize_SSS()***) “sensors_reliable_wireless” specifies that the navigator will start with reliable wireless systems (GNSS or Wi-Fi or both) and the overall solution will be sensor/wireless integrated. The navigator will provide a reliable navigation solution only when reliable 2D position (latitude and longitude) is provided from the GNSS or the Wi-Fi systems to the navigator via ***tpp_process_gnss_pvt()*** or ***tpp_process_2d_position()***. A reliable 2D position is indicated by the standard deviation of the 2D position being less than 5 meters. If **flag_value_1_** for both **GNSS** and **WIFI** flags is “off”, the initialization function will not be successful (i.e. ***TPP_INVALID_OPTION_SENSORS_RELIABLE_WIRELESS*** will be returned by ***tpp_initialize_SSS()***) “sensors_wireless” specifies that the navigator will start with or without wireless systems (GNSS or Wi-Fi or both). The navigator will provide a reliable navigation solution without waiting for a reliable 2D position from wireless systems. When a reliable 2D position is available, the navigator will utilize it in the navigation solution. If **flag_value_1_** for both **GNSS** and **WIFI** flags is “off”, the initialization function will not be successful (i.e. ***TPP_INVALID_OPTION_SENSORS_WIRELESS*** will be returned by ***tpp_initialize_SSS()***). If “sensors_wireless” is used and the mode of transit is set to driving, the behavior of the navigator will be equivalent to “sensors_reliable_wireless”

7.2.3. *MAGNETOMETER flag values*

flag_name_	“MAGNETOMETER”	
flag_value_1_	“on” or “off”	“on” specifies that the data from the magnetometer will be used in the navigation solution. “off” specifies that the data from the magnetometer will not be used in the navigation solution.
flag_value_2_	“F.F” [1]	“F.F” specifies the value of weighting for the magnetometer standard deviation. This value must be provided if the developer specified **flag_value_1_** to be “on”. In order not to change the default behavior of the navigator, the developer can use “1.0”.

7.2.4. *BAROMETER flag values*

flag_name_	“BAROMETER”	
flag_value_1_	“on” or “off”	“on” specifies that the data from the barometer will be used in the navigation solution. “off” specifies that the data from the barometer will not be used in the navigation solution.
flag_value_2_	“F.F”	“F.F” specifies the value of weighting for the barometer standard deviation. This value must be provided if the developer specified **flag_value_1_** to be “on”. In order not to change the default behavior of the navigator, the developer can use “1.0”.

[1] “F.F” specifies that the value that should be inserted in this field is a string representing a floating number. There is no limit to the number of digits, so as an example the developer can insert “123.456”.

7.2.5. *SPEED flag values*

flag_name_	"SPEED"	
flag_value_1_	"on" or "off"	"on" specifies that the data from the speed sensor will be used in the navigation solution. "off" specifies that the data from the speed sensor will not be used in the navigation solution.
flag_value_2_	"F.F"	"F.F" specifies the value of weighting for the speed standard deviation. This value must be provided if the developer specified **flag_value_1_** to be "on". In order not to change the default behavior of the navigator, the developer can use "1.0".

7.2.6. *GNSS flag values*

flag_name_	"GNSS"	
flag_value_1_	"on" or "off"	"on" specifies that the data from the GNSS receiver will be used in the navigation solution. "off" specifies that the data from the GNSS receiver will not be used in the navigation solution.
flag_value_2_	PREDEFINED_STRING	This value specifies the receiver type. This value must be provided if the value of the first flag is "on". The values that PREDEFINED_STRING can take are defined and explained in Section 13.1.
flag_value_3_	"F.F"	"F.F" specifies the value of weighting for the GNSS position standard deviation. This value must be provided if the developer specified **flag_value_1_** to be "on". In order not to change the default behavior of the navigator, the developer can use "1.0".
flag_value_4_	"F.F"	"F.F" specifies the value of weighting for the GNSS velocity standard deviation. This value must be provided if the developer specified **flag_value_1_** to be "on". In order not to change the default behavior of the navigator, the developer can use "1.0".

7.2.7. *WIFI flag values*

flag_name_	"WIFI"	
flag_value_1_	"on" or "off"	"on" specifies that the data from Wi-Fi will be used in the navigation solution. "off" specifies that the data from Wi-Fi will not be used in the navigation solution.
flag_value_2_	"F.F"	"F.F" specifies the value of weighting for the Wi-Fi standard deviation. This value must be provided if the developer specified **flag_value_1_** to be "on". In order not to change the default behavior of the navigator, the developer can use "1.0".

7.2.8. *MULTIPLE_ANTENNA flag values*

flag_name_	"MULTIPLE_ANTENNA"	
flag_value_1_	"on" or "off"	"on" specifies that the data from the multiple antenna will be used in the navigation solution. "off" specifies that the data from the multiple antenna will not be used in the navigation solution.

7.2.9. *PLATFORM_HEADING flag values*

flag_name_	"PLATFORM_HEADING"	
flag_value_1_	"F.F"	"F.F" specifies the value of the initial platform heading in degrees that will be passed to the navigator.

7.2.10. *POSITION flag values*

flag_name_	"POSITION"	
flag_value_1_	"F.F"	"F.F" specifies the value of the initial latitude in degrees that will be passed to

		the navigator.
flag_value_2_	"F.F"	"F.F" specifies the value of the initial longitude in degrees that will be passed to the navigator.
flag_value_3_	"F.F"	"F.F" specifies the value of the initial height in metres that will be passed to the navigator.

7.2.11. *MISALIGNMENT flag values*

flag_name_	"MISALIGNMENT"	
flag_value_1_	"fixed", "automatic" or "fixed_unknown"	"fixed" specifies that the misalignment angle set in the next field will be used in the navigation solution. "automatic" specifies that the misalignment angle will be automatically calculated by the navigator. "fixed_unknown" specifies that the misalignment angle will stay fix, however it is unknown
flag_value_2_	"F.F" if **flag_value_1_** is "fixed"	**If flag_value_1_ is "fixed":** "F.F" specifies the value of the misalignment angle in degrees that will be passed to the system if the developer specified **flag_value_1_** to be "fixed".
	"misalignment_normal" or "misalignment_snap" if **flag_value_1_** is "automatic"	**If flag_value_1_ is "automatic":** "misalignment_normal" specifies that the misalignment estimation operates in normal mode without any approximations to the calculated angles. "misalignment_snap" specifies that the misalignment is snapped to specific quantized or discrete angles in handheld use cases.
	"F.F" if **flag_value_1_** is "fixed_unknown"	**If flag_value_1_ is "fixed_unknown":** "F.F" specifies the value of the misalignment angle in degrees that will be passed to the system if the developer specified **flag_value_1_** to be "fixed_unknown".
flag_value_3_	"F.F" if **flag_value_1_** is "fixed_unknown"	**If flag_value_1_ is "fixed_unknown":** "F.F" specifies the value of standard deviation of the input misalignment angle in degrees that will be passed to the system if the developer specified **flag_value_1_** to be "fixed_unknown".

7.2.12. *MODE_OF_TRANSIT flag values*

flag_name_	"MODE_OF_TRANSIT"	
flag_value_1_	"walking" or "driving" or "automatic" or "running" or "cycling"	"walking" specifies to the navigator to work in walking mode only. "driving" specifies to the navigator to work in driving mode only. "automatic" specifies to the navigator to work in automatic mode where it detects the mode of transit automatically. "running" specifies to the navigator to work in running mode only. "cycling" specifies to the navigator to work in cycling mode only.

7.2.13. *USE_CASE flag values*

flag_name_	"MODE_OF_TRANSIT"	
flag_value_1_	"automatic" or "torso_and_hand_viewing" or "pocket" or "hand_swinging" or "arm"	"automatic" Specifies to the navigator to automatically detect the use case. "torso_and_hand_viewing" specifies to the navigator that the device is "pocket" specifies to the navigator that the device is in pocket "hand_dangling" specifies to the navigator that the device is dangling in hand. "arm" specifies to the navigator that the device is on the arm

7.2.14. *FLOOR_INFORMATION flag values*

flag_name_	"FLOOR_INFORMATION"

flag_value_1_	"N"[2]	"N" represents an integer which specifies the initial floor number.
flag_value_2_	"F.F"	"F.F" specifies the height between floors in metres.

7.2.15. *GYROSCOPE_BIASES flag values*

flag_name_	"GYROSCOPE_BIASES"	
flag_value_1_	"F.F"	"F.F" specifies the bias value for the **X**-axis gyroscope in deg/sec
flag_value_2_	"F.F"	"F.F" specifies the bias value for the **Y**-axis gyroscope in deg/sec
flag_value_3_	"F.F"	"F.F" specifies the bias value for the **Z**-axis gyroscope in deg/sec

7.2.16. *ACCELEROMETER_CONFIGURATION flag values*

flag_name_	"ACCELEROMETER_CONFIGURATION"	
flag_value_1_	"accelerometer_custom" or PREDEFINED_STRING	"accelerometer_custom" specifies that a custom accelerometer configuration will be used and the following three flags must be filled accordingly. The values that PREDEFINED_STRING can take are defined and explained in Section 13.2.
flag_value_2_	"F.F"	"F.F" specifies the bias instability value for the three accelerometers in mGal
flag_value_3_	"F.F"	"F.F" specifies the velocity random walk (VRW) value for the three accelerometers in (m/sec/√hr)
flag_value_4_	"F.F"	"F.F" specifies the bias correlation time value for three accelerometers in hr

7.2.17. *GYROSCOPE_X_CONFIGURATION flag values*

flag_name_	"GYROSCOPE_X_CONFIGURATION"	
flag_value_1_	"gyroscope_custom" or PREDEFINED_STRING	"gyroscope_custom" specifies that a custom gyroscope configuration will be used and the following three flags must be filled accordingly. The values that PREDEFINED_STRING can take are defined and explained in Section 13.3
flag_value_2_	"F.F"	"F.F" specifies the bias instability value for the **X**-axis gyroscope in deg/hr
flag_value_3_	"F.F"	"F.F" specifies the angular random walk (ARW) value for the **X**-axis gyroscope in deg/√hr
flag_value_4_	"F.F"	"F.F" specifies the bias correlation time value for the **X**-axis gyroscope in hr

7.2.18. *GYROSCOPE_Y_CONFIGURATION flag values*

flag_name_	"GYROSCOPE_Y_CONFIGURATION"	
flag_value_1_	"gyroscope_custom" or PREDEFINED_STRING	"gyroscope_custom" specifies that a custom gyroscope configuration will be used and the following three flags must be filled accordingly. The values that PREDEFINED_STRING can take are defined and explained in Section 13.3
flag_value_2_	"F.F"	"F.F" specifies the bias instability value for the **Y**-axis gyroscope in deg/hr
flag_value_3_	"F.F"	"F.F" specifies the angular random walk (ARW) value for the **Y**-axis gyroscope in deg/√hr
flag_value_4_	"F.F"	"F.F" specifies the bias correlation time value for the **Y**-axis gyroscope in hr

7.2.19. *GYROSCOPE_Z_CONFIGURATION flag values*

flag_name_	"GYROSCOPE_Z_CONFIGURATION"	
flag_value_1_	"gyroscope_custom" or PREDEFINED_STRING	"gyroscope_custom" specifies that a custom gyroscope configuration will be used and the following three flags must be filled accordingly. The values that PREDEFINED_STRING can take are defined and explained in Section 13.3
flag_value_2_	"F.F"	"F.F" specifies the bias instability value for the **Z**-axis gyroscope in deg/hr
flag_value_3_	"F.F"	"F.F" specifies the angular random walk (ARW) value for the **Z**-axis gyroscope in deg/√hr
flag_value_4_	"F.F"	"F.F" specifies the bias correlation time value for the **Z**-axis gyroscope in hr

[2] "N" specifies that the value that should be inserted in this field is a string representing an integer. If a floating number is used in the corresponding field, the number will be implicitly casted to an integer.

7.2.20. *MOTION_CONSTRAINTS flag values*

flag_name_	"MOTION_CONSTRAINTS"	
flag_value_1_	"motion_constraints_custom" or PREDEFINED_STRING	"motion_constraints_custom" specifies that a custom motion constraints configuration will be used and the following two flags must be filled accordingly. The values that PREDEFINED_STRING can take are defined and explained in Section 0
flag_value_2_	"F.F"	"F.F" specifies the dynamic velocity constraint in m/sec. This value must be greater than zero.
flag_value_3_	"F.F"	"F.F" specifies the static velocity constraint in m/sec. This value must be greater than zero.

7.2.21. *DEBUG_DATA flag values*

flag_name_	"DEBUG_DATA"	
flag_value_1_	"on" or "off"	"on" specifies that the data entities used in debugging will be generated in the "Output Per Epoch" packet. "off" specifies that no debug entities will be generated in the "Output Per Epoch" packet.

7.2.22. *OUTPUT_POSITION_ERROR_WEIGHTING flag values*

flag_name_	"OUTPUT_POSITION_ERROR_WEIGHTING"	
flag_value_1_	"N"	"N" specifies the value of weighting for the latitude standard deviation. The weighting can be any integer value from 1 to 255.
flag_value_2_	"N"	"N" specifies the value of weighting for the longitude standard deviation. The weighting can be any integer value from 1 to 255.
flag_value_3_	"N"	"N" specifies the value of weighting for the height standard deviation. The weighting can be any integer value from 1 to 255.

7.2.23. *OUTPUT_VELOCITY_ ERROR_WEIGHTING flag values*

flag_name_	"OUTPUT_VELOCITY_ERROR_WEIGHTING"	
flag_value_1_	"N"	"N" specifies the value of weighting for the north velocity standard deviation. The weighting can be any integer value from 1 to 255.
flag_value_2_	"N"	"N" specifies the value of weighting for the east velocity standard deviation. The weighting can be any integer value from 1 to 255.
flag_value_3_	"N"	"N" specifies the value of weighting for the down velocity standard deviation. The weighting can be any integer value from 1 to 255.

7.2.24. *OUTPUT_ATTITUDE_ ERROR_WEIGHTING flag values*

flag_name_	"OUTPUT_ATTITUDE_ERROR_WEIGHTING"	
flag_value_1_	"N"	"N" specifies the value of weighting for the roll standard deviation. The weighting can be any integer value from 1 to 255.
flag_value_2_	"N"	"N" specifies the value of weighting for the pitch standard deviation. The weighting can be any integer value from 1 to 255.
flag_value_3_	"N"	"N" specifies the value of weighting for the heading standard deviation. The weighting can be any integer value from 1 to 255.

7.2.25. *ACCELEROMETER_BIASES flag values*

flag_name_	"ACCELEROMETER_BIASES"	
flag_value_1_	"F.F"	"F.F" specifies the bias value for the **X**-axis accelerometer in m/sec/sec
flag_value_2_	"F.F"	"F.F" specifies the bias value for the **Y**-axis accelerometer in m/sec/sec
flag_value_3_	"F.F"	"F.F" specifies the bias value for the **Z**-axis accelerometer in m/sec/sec

7.2.26. *ZUPT_THRESHOLDS flag values*

flag_name_	"ZUPT_THRESHOLDS"	
flag_value_1_	"F.F"	"F.F" specifies the ZUPT threshold for the **X**-axis

flag_value_2_	"F.F"	"F.F" specifies the ZUPT threshold for the **Y**-axis
flag_value_3_	"F.F"	"F.F" specifies the ZUPT threshold for the **Z**-axis

7.2.27. *DECLINATION_ANGLE flag values*

flag_name_	"DECLINATION_ANGLE"	
flag_value_1_	"F.F"	"F.F" specifies the declination angle in degrees within the range [-180,180]

7.2.28. *REPLACE_ATTITUDE_WITH_6DOF flag values*

flag_name_	"REPLACE_ATTITUDE_WITH_6DOF"	
flag_value_1_	"on" or "off"	"on" specifies that the 6dof will be used to compute the attitude "off" specifies that the attitude computations done in the library will be used instead of the 6dof

7.2.29. *PROCESSING_MODE flag values*

flag_name_	"PROCESSING_MODE"	
flag_value_1_	"forward" or "backward"	"forward" specifies that the navigator is processing the data in forward mode "backward" specifies that the navigator is processing the data in backward mode

7.2.30. *BACKWARD_SMOOTHING flag values*

flag_name_	"BACKWARD_SMOOTHING"	
flag_value_1_	"on" or "off"	"on" specifies to the navigator to save specific information needed to run backward smoothing. In this case, the user should set **flag_value_2_** with a folder path where a file containing the backward smoothed solution will be generated. "off" specifies to the navigator not to save any information for the backward smoothing functionality. If the user calls **tpp_run_backward_smoothing()** at the end of the navigation session, with the value of this flag being "off", an error will be returned.
flag_value_2_	FOLDER_PATH	The path of the folder where a file containing the backward smoothed solution is generated. The name of the generated file is "**bs_solution.dat**". Temporary files are also generated which contain information needed to run backward smoothing. The application using the library should have write access to the specified location on the storage device.

7.2.31. *LIBRARY_CALL_TYPE flag values*

flag_name_	"LIBRARY_CALL_TYPE"	
flag_value_1_	"standalone" or "isl"	"standalone" specifies that the TPN library will be running in the standalone mode. This is the default setting if the flag does not exist. "isl" specifies that the TPN library will be running in Invensense Smoothing Library (ISL) mode. In this case, TPN library will work together with ISL to output enhanced navigation results.

7.3. TppApiVersionStruct

	Member	Data Type	Unit	Description
1	major_	UINT8	N/A	API major version
2	minor_	UINT8	N/A	API minor version
3	patch_	UINT8	N/A	API patch version
4	release_id_	UINT8	N/A	API release id 0x00: Alpha 0x01: Beta 0x02: Release Candidate

				0x03: Ready for Production Release
5	release_number_	UINT8	N/A	API release number. The number should be greater than 0. This number is only valid when the API release is under development (i.e. Not a "Ready for Production" Release)
6	type_	UINT8	N/A	'F': Full 'M': Micro
7	set_number_	UINT16	N/A	Library set number. A unique number that identifies the library used.

The version number that is used to track the API/Library changes is as follows:

Major.Minor.Patch-ReleaseId.ReleaseNumber

Examples:

(1) 2.0.0-rc.4: 4th release candidate version of the 2.0.0 library

(2) 3.0.2: Ready for production 3.0.2 version of the library. Ready for production releases are specified by the release id being 0x03 and in that case, the release number is discarded.

7.4. TppGnssPvtMessageStruct

	Member	Data Type	Unit	Description
1	gnss_timetag_	DOUBLE64	sec	An absolute time-tag that increments according to the rate of the GNSS data provided. GPS time or UTC time can be used to fill this field, but it has to be consistent throughout the navigation session.
2	latitude_	DOUBLE64	deg	
3	longitude_	DOUBLE64	deg	
4	height_	FLOAT32	m	
5	velocity_north_	FLOAT32	m/sec	
6	velocity_east_	FLOAT32	m/sec	
7	velocity_down_	FLOAT32	m/sec	
8	position_north_standard_deviation_	FLOAT32	m	
9	position_east_standard_deviation_	FLOAT32	m	
10	height_standard_deviation_	FLOAT32	m	
11	velocity_north_standard_deviation_	FLOAT32	m/sec	
12	velocity_east_standard_deviation_	FLOAT32	m/sec	
13	velocity_down_standard_deviation_	FLOAT32	m/sec	
14	dop_data_available_	UINT8	N/A	0x00: DOP data is not available 0x01: DOP data is available If this field is set to 0x01, then the following fields (**horizontal_dop_** and **vertical_dop_**) should be filled accordingly.
15	horizontal_dop_	FLOAT32	N/A	
16	vertical_dop_	FLOAT32	N/A	

7.5. TppImuMessageStruct

	Member	Data Type	Unit	Description
1	timetag_	DOUBLE64	sec	time-tag
2	data_gyroscope_x_	DOUBLE64	deg/sec	

3	data_gyroscope_y_	DOUBLE64	deg/sec	
4	data_gyroscope_z_	DOUBLE64	deg/sec	
5	data_accelerometer_x_	DOUBLE64	m/sec/sec	
6	data_accelerometer_y_	DOUBLE64	m/sec/sec	
7	data_accelerometer_z_	DOUBLE64	m/sec/sec	

7.6. TppBarometerMessageStruct

	Member	Data Type	Unit	Description
1	height_	FLOAT32	m	
2	height_standard_deviation_	FLOAT32	m	

Notes:

(1) In case the pressure is available instead of the height, the developer can compute the height (in metres) above sea level from the pressure (in mBar) according to the below equation:

$$Height = 44330.76 * (1 - (Pressure/1013.25)^{\frac{1}{5.2558}})$$

(2) The height provided in the ***TppBarometerMessageStruct*** can be either "height above sea level" or "height above ellipsoid" through the navigation session without being consistent with the type of the height input provided to the other API functions.

(3) To get the height standard deviation, the developer can collect static data from the barometer in the normal room temperature and compute the standard deviation for the data collected which will be the value used for the member ***height_standard_deviation_***. If the height standard deviation is not available, then the field should be filled by 1.0.

7.7. TppMagnetometerMessageStruct

	Member	Data Type	Unit	Description
1	raw_data_available_	UINT8	N/A	0x00: raw data is not available 0x01: raw data is available
2	raw_data_x_	FLOAT32	mG	
3	raw_data_y_	FLOAT32	mG	
4	raw_data_z_	FLOAT32	mG	
5	raw_data_accuracy_flag_	UINT8	N/A	0x00: Flag not available 0x01: Invalid data 0x02: Valid data
6	calibrated_data_available_	UINT8	N/A	0x00: calibrated data is not available 0x01: calibrated data is available
7	calibrated_data_x_	FLOAT32	mG	
8	calibrated_data_y_	FLOAT32	mG	
9	calibrated_data_z_	FLOAT32	mG	
10	calibrated_data_accuracy_flag_	UINT8	N/A	0x00: Flag not available 0x01:Unreliable 0x02: Low Accuracy 0x03: Medium Accuracy 0x04: High Accuracy
11	calibration_status_changed_flag_	UINT8	N/A	Specifies if the status of the calibration changed when the magnetometer providing the calibrated data is reset or when the

				the calibrated signals are saturated. 0x00: Flag not available 0x01: No change in status 0x02: Calibration status changed
12	heading_available_	UINT8	N/A	0x00: heading data is not available 0x01: heading data is available
13	heading_	FLOAT32	deg	The heading is with respect to the True North (i.e. The heading provided is expected to be corrected with the magnetic declination angle) and not the Magnetic North.
14	heading_standard_deviation_	FLOAT32	deg	See notes.

Notes:

(1) If the heading standard deviation is not available, then the field should be filled by 12.0

7.8. TppSpeedMessageStruct

	Member	Data Type	Unit	Description
1	speed_	FLOAT32	m/sec	
2	reserved_1_	UINT8	N/A	
2	reverse_	UINT8	N/A	0x00: Unavailable 0x01:Forward 0x02: Reverse

7.9. TppMultipleAntennaMessageStruct

	Member	Data Type	Unit	Description
1	heading_	FLOAT32	deg	
2	heading_standard_deviation_	FLOAT32	deg	
3	roll_	FLOAT32	deg	
4	roll_standard_deviation_	FLOAT32	deg	
5	pitch_	FLOAT32	deg	
6	pitch_standard_deviation_	FLOAT32	deg	

7.10. TppOperator2dPositionMessageStruct

	Member	Data Type	Unit	Description
1	latitude_	DOUBLE64	deg	
2	longitude_	DOUBLE64	deg	

7.11. TppWirelessMessageStruct

	Member	Data Type	Unit	Description
1	source_	UINT8	N/A	0x01: Bluetooth 0x11: Wi-Fi Source (1) 0x12: Wi-Fi Source (2)
2	latency_available_	UINT8	N/A	0x00: Latency and the corresponding standard deviation are not available. 0x01: Latency and the corresponding standard deviation are available.

3	latency_	FLOAT32	sec	Delay between the time the Wi-Fi 2D position is requested and the time the position is received and sent to the navigator. If there is no delay, the user can use zero.
4	latency_standard_deviation_	FLOAT32	sec	
5	position_2d_available_	UINT8	N/A	0x00: 2d position and the corresponding standard deviation are not available 0x01: 2d position and the corresponding standard deviation are available
6	latitude_	DOUBLE64	deg	
7	longitude_	DOUBLE64	deg	
8	position_north_standard_deviation_	FLOAT32	m	
9	position_east_standard_deviation_	FLOAT32	m	
10	height_available_	UINT8	N/A	0x00: Height and the corresponding standard deviation are not available 0x01: Height and the corresponding standard deviation are available
11	height_	FLOAT32	m	
12	height_standard_deviation_	FLOAT32	m	
13	velocity_2d_available_	UINT8	N/A	0x00: 2d velocity and the corresponding standard deviation are not available 0x01: 2d velocity and the corresponding standard deviation are available
14	velocity_north_	FLOAT32	m/sec	
15	velocity_east_	FLOAT32	m/sec	
16	velocity_north_standard_deviation_	FLOAT32	m/sec	
17	velocity_east_standard_deviation_	FLOAT32	m/sec	
18	velocity_down_available_	UINT8	N/A	0x00: Down velocity and the corresponding standard deviation are not available 0x01: Down velocity and the corresponding standard deviation are available
19	velocity_down_	FLOAT32	m/sec	
20	velocity_down_standard_deviation_	FLOAT32	m/sec	
21	platform_heading_available_	UINT8	N/A	0x00: Platform heading and the corresponding standard deviation are not available 0x01: Platform heading and the corresponding standard deviation are available
22	platform_heading_	FLOAT32	deg	
23	platform_heading_standard_deviation_	FLOAT32	deg	
24	floor_information_available_	UINT8	N/A	0x00: Floor information is not available 0x01: Floor information is available
24	floor_number_	INT16	N/A	

7.12. TppHeightMessageStruct

	Member	Data Type	Unit	Description
1	height_	FLOAT32	m	
2	height_standard_deviation_	FLOAT32	m	

7.13. TppFloorInformationMessageStruct

	Member	Data Type	Unit	Description
1	floor_number_	INT16	N/A	
2	height_between_floors_	FLOAT32	m	

7.14. TppCallInformationMessageStruct

	Member	Data Type	Unit	Description
1	is_call_active_	UINT8	N/A	0x00: Call Inactive 0x01: Call Active
2	is_proximity_detected_	UINT8	N/A	0x00: No Proximity Detected 0x01: Proximity Detected
3	is_speaker_active_	UINT8	N/A	0x00: Phone speaker Inactive 0x01: Phone speaker Active
4	headset_type_	UINT8	N/A	0x00: None 0x01: Wired or wireless Earphone is used

7.15. TppPlatformHeadingMessageStruct

	Member	Data Type	Unit	Description
1	heading_	FLOAT32	deg	
2	heading_standard_deviation_	FLOAT32	deg	

7.16. TppDeviceHeadingMessageStruct

	Member	Data Type	Unit	Description
1	device_heading_available_	UINT8	N/A	**0x00:** Device heading value is not available directly and will be computed from the **platform_heading_** and **misalignment_angle_** fields. **0x01**: Device heading value is available directly from the **device_heading_** field.
2	device_heading_	FLOAT32	deg	
3	platform_heading_	FLOAT32	deg	
4	misalignment_angle_	FLOAT32	deg	

7.17. TppQuaternionsMessageStruct

	Member	Data Type	Unit	Description
1	value_0_	FLOAT32	N/A	The scalar component of the quaternion.
2	value_1_	FLOAT32	N/A	The 1st vector component of the

				quaternion.
3	value_2_	FLOAT32	N/A	The 2nd vector component of the quaternion.
4	value_3_	FLOAT32	N/A	The 3rd vector component of the quaternion.

7.18. TppImuBiasesMessageStruct

	Member	Data Type	Unit	Description
1	bias_gyroscope_source_	INT8	N/A	-1: Biases Not Available (Discard values) **Other values to be defined by ISJ and ICA**
2	bias_gyroscope_accuracy_	FLOAT32	deg/sec	Accuracy for gyroscope biases
3	bias_gyroscope_x_	FLOAT32	deg/sec	Bias for gyroscope-x
4	bias_gyroscope_y_	FLOAT32	deg/sec	Bias for gyroscope-y
5	bias_gyroscope_z_	FLOAT32	deg/sec	Bias for gyroscope-z
6	bias_accelerometer_source_	INT8	N/A	-1: Biases Not Available (Discard values) **Other values to be defined by ISJ and ICA**
7	bias_accelerometer_accuracy_	FLOAT32	m/sec/sec	Accuracy for accelerometer biases
8	bias_accelerometer_x_	FLOAT32	m/sec/sec	Bias for accelerometer-x
9	bias_accelerometer_y_	FLOAT32	m/sec/sec	Bias for accelerometer-y
10	bias_ accelerometer_z_	FLOAT32	m/sec/sec	Bias for accelerometer-z

7.19. TppVenueMapMessageStruct

	Member	Data Type	Unit	Description
1	position_2d_available_	UINT8	N/A	0x00: 2d position and the corresponding standard deviation are **not available** 0x01: 2d position and the corresponding standard deviation are **available but not reliable** 0x02: 2d position and the corresponding standard deviation are **available and reliable**
2	latitude_	DOUBLE64	deg	
3	longitude_	DOUBLE64	deg	
4	position_north_standard_deviation_	FLOAT32	m	
5	position_east_standard_deviation_	FLOAT32	m	
6	height_available_	UINT8	N/A	0x00: Height and the corresponding standard deviation are not available 0x01: Height and the corresponding standard deviation are available
7	height_	FLOAT32	m	
8	height_standard_deviation_	FLOAT32	m	
9	platform_heading_available_	UINT8	N/A	0x00: Platform heading and the corresponding standard deviation are **not available** 0x01: Platform heading and the corresponding standard deviation are **available but not reliable** 0x01: Platform heading and the

				corresponding standard deviation are **available and reliable**
10	platform_heading_	FLOAT32	deg	
11	platform_heading_standard_deviation_	FLOAT32	deg	
12	map_entity_available_	UINT8	N/A	0x00: Map entity is not available 0x01: Map entity is available
13	map_entity_	UINT16	N/A	Specifies an entity in the map. Examples for map entities are stairs, elevators, escalators, types of rooms, and others.
15	step_length_scale_available_	UINT8	N/A	Flag to notify if both fields (#16) Step Length Scale and (#17) Use case for Step length scale are available or not. 0x00: Fields are not available 0x01: Fields are available
16	step_length_scale_	FLOAT32	N/A	
17	use_case_for_step_length_scale_	UINT8	N/A	
18	mode_of_motion_for_step_length_scale_	UINT8	N/A	

7.20. TppProcessMisalignmentMessageStruct

	Member	Data Type	Unit	Description
1	angle_	FLOAT32	deg	
2	angle_standard_deviation_	FLOAT32	deg	

7.21. TppSetMisalignmentMessageStruct

	Member	Data Type	Unit	Description
1	estimation_on_	UINT8	N/A	0x00: Automatic misalignment estimation turned off 0x01: Automatic misalignment estimation turned on
2	angle_available_	UINT8	N/A	0x00: Misalignment angle is not available 0x01: Misalignment angle is available When the misalignment estimation is turned off, the user can specify a fixed misalignment angle by setting this field to '1' and setting the angle in the **angle_** field. When the misalignment estimation is turned off, and if the user specified this field to be '0', then the navigator will use the last estimated misalignment.
3	angle_	FLOAT32	deg	

7.22. TppSolutionStruct

	Member	Data Type	Unit	Description
1	navigation_phase_	INT8	N/A	-1: Pre-alignment 0: Alignment

				+1: Navigation/Available +2: Navigation/Reliable +3: Navigation/Vertical Alignment +4: Navigation/Drive to Walk
2	timetag_	DOUBLE64	sec	
3	latitude_	DOUBLE64	deg	
4	longitude_	DOUBLE64	deg	
5	height_	FLOAT32	m	
6	position_north_standard_deviation_	FLOAT32	m	
7	position_east_standard_deviation_	FLOAT32	m	
8	height_standard_deviation_	FLOAT32	m	
9	velocity_north_	FLOAT32	m/sec	
10	velocity_east_	FLOAT32	m/sec	
11	velocity_down_	FLOAT32	m/sec	
12	velocity_north_standard_deviation_	FLOAT32	m/sec	
13	velocity_east_standard_deviation_	FLOAT32	m/sec	
14	velocity_down_standard_deviation_	FLOAT32	m/sec	
15	roll_	FLOAT32	deg	
16	pitch_	FLOAT32	deg	
17	heading_	FLOAT32	deg	Device heading
18	roll_standard_deviation_	FLOAT32	deg	
19	pitch_standard_deviation_	FLOAT32	deg	
20	heading_standard_deviation_	FLOAT32	deg	Device heading standard deviation
21	platform_heading_	FLOAT32	deg	This field is equivalent to the **heading_** field

7.23. TppPacketStruct

Variable of this data type is used in the parameter list of two functions: ***tpp_initialize_SSS()*** and ***tpp_advance_navigation_step()*** to return a packed byte array and its valid size.

The structure of this byte array is dependent on which of the functions is used to populate the array.

The packed byte array which is populated when ***tpp_initialize_SSS()*** is called, is used to enable Trusted Positioning debugging. In this case, the maximum size that should be allocated for the array which the structure member ***byte_array_pointer_*** points to is **550** bytes.

The structure of the byte array which is populated when ***tpp_advance_navigation_step()*** is called is presented in Section 7.23.1. In this case, the maximum size that should be allocated for the array which the structure member ***byte_array_pointer_*** points to is **1500** bytes.

	Member	Data Type	Unit	Description
1	byte_array_pointer_	UINT8*	N/A	Pointer to the byte array.
2	returned_byte_array_length_	UINT16	N/A	This field specifies the exact used length of the byte array. This value will be used by the developer when (1) Writing the byte array directly to a dataset file which will be used for Trusted Positioning debugging. Or (2) Extracting specific entities from

				the output byte array.

7.23.1. *Output Array Structure*

The structure of the byte array that is populated by ***tpp_advance_navigation_step()*** is shown below. The entities, their identification numbers and structures are specified in Section 9.

	Field Name	Data Type	Units	Description
1	synchronization_byte_a_	UINT8	0x54	1st Synchronization Byte
2	synchronization_byte_b_	UINT8	0x50	2nd Synchronization Byte
3	synchronization_byte_c_	UINT8	0x49	3rd Synchronization Byte
4	payload_length_	UINT16	N/A	Length of the payload (i.e. packet body) attached to the packet header excluding the packet checksum.
5	packet_id_	UINT16	N/A	**0x0003**
6	stream_id_	UINT16	N/A	Identifies the stream identification number of the current epoch if more than one output stream exists
7	epoch_number_	UINT32	N/A	Number of the current epoch
8	number_of_entities_	UINT16	N/A	Number of entities (**E**) within the current epoch. The maximum number of entities is 65,535. Each entity in the packet has two fields: entity_id_ and entity_start_address_
9	entity_id_(1)	UINT16	N/A	The Identification number for the 1st entity in the payload.
10	entity_start_address_(1)	UINT16	N/A	The start address of the 1st entity
-	.....	-	-	-
9+(E-1)*2	entity_id_(E)	UINT16	N/A	The Identification number for the Eth entity in the payload.
9+(E-1)*2+1	entity_start_address_(E)	UINT16	N/A	The start address of the Eth entity
9+(E-1)*2+2	entity_(1)	-	N/A	1st entity in the packet
...	....	-	-	-
-	entity_ (E)	-	N/A	Eth entity in the packet
-	checksum	UINT8[2]	N/A	Two checksum bytes computed according to the Fletcher-8 checksum algorithm.

The Pseudo-code for computing the two checksum bytes at the end of the packet, according to the Fletcher-8 checksum algorithm, is below:

Code Sample 1 Fletcher-8 Checksum Algorithm Pseudo code

```
//N:length of the bytes on which the checksum is computed, which in this case are all the
//packet bytes excluding the first 3 header bytes and the last 2 checksum bytes
//J:The index of the first element on which the checksum is computed
//Buffer:The output packet buffer

CK[0] = 0
CK[1] = 0

for( I=J ; I<(J+N) ; I++ )
{
    CK[0] = CK[0] + Buffer[I]
    CK[1] = CK[1] + CK[0]
}
```

7.24. TppModeOfTransitEnum

	Member	Description
0	TPP_ MODE_OF_TRANSIT_DRIVING	Specifies to the navigator to work in driving mode only.
1	TPP_ MODE_OF_TRANSIT_WALKING	Specifies to the navigator to work in walking mode only.
2	TPP_MODE_OF_TRANSIT_AUTOMATIC	Specifies to the navigator to work in automatic mode where the navigator will detect automatically if the mode of transit is walking or driving.
3	TPP_MODE_OF_TRANSIT_RUNNING	Specifies to the navigator to work in running mode only.
4	TPP_MODE_OF_TRANSIT_CYCLING	Specifies to the navigator to work in cycling mode only.

7.25. TppZuptModeEnum

	Member	Description
0	TPP_ZUPT_MODE_AUTOMATIC	Specifies to the navigator to work in the normal ZUPT mode where ZUPT is detected automatically.
1	TPP_ZUPT_MODE_ON	Specifies to the navigator that the platform and device are in a static, no motion state.
2	TPP_ZUPT_MODE_OFF	Specifies to the navigator that the platform or the device is moving.

7.26. TppMagnetometerCalibrationEnum

	Member	Description
0	TPP_MAGNETOMETER_CALIBRATION_OFF	Disables the automatic magnetometer calibration in the navigator.
1	TPP_MAGNETOMETER_CALIBRATION_ON	Enables the automatic magnetometer calibration in the navigator.

7.27. TppUseCaseEnum

	Member	Description
0	TPP_USE_CASE_AUTOMATIC	Specifies to the navigator to automatically detect the use case.
1	TPP_USE_CASE_TORSO_AND_HAND_VIEWING	Example: Hand held, belt and other use cases that are not specified explicitly.
2	TPP_USE_CASE_POCKET	Specifies to the navigator that the device is in pants' pockets
3	TPP_USE_CASE_HAND_SWINGING	Specifies to the navigator that the device is swinging in hand
4	TPP_USE_CASE_ARM	Specifies to the navigator that the phone is on the arm
~~5~~	~~TPP_USE_CASE_PURSE~~	~~Specifies to the navigator that the phone is in a purse~~

7.28. TppOrientationBasedOnPitchEnum (INTERNAL only)

	Member	Description
-1	TPP_ORIENTATION_VERTICAL_DOWN	
0	TPP_ORIENTATION_HORIZONTAL	
+1	TPP_ORIENTATION_VERTICAL_UP	

7.29. TppAnchorPointStruct

	Member	Data Type	Unit	Description
1	type_	UINT8	N/A	Anchor Point Type. **0x01**: Start Anchor Point. **0x02**: End Anchor Point..
2	position_available_	UNIT8	N/A	
3	latitude_	DOUBLE64	deg	
4	longitude_	DOUBLE64	deg	
5	height_	FLOAT32	m	
6	heading_available_	UNIT8	N/A	
7	heading_	FLOAT32	deg	

7.30. TppZuptThresholdsStruct

	Member	Data Type	Unit	Description
1	value_x_	FLOAT32	deg/sec	
2	value_y_	FLOAT32	deg/sec	
3	value_z_	FLOAT32	deg/sec	

7.31. TppMultiDeviceLatencyStruct

	Member	Data Type	Unit	Description
1	latency_available_	UINT8	N/A	0x00: Latency is not available. 0x01: Latency is available
2	latency_	FLOAT32	sec	Delay between the time the information is computed on the sender device and the time the information is processed on the receiver device. If there is no delay, the user can use zero.

7.32. TppMultiDevicePositionVelocityStruct

	Member	Data Type	Unit	Description
1	device_identifier_	UINT64	N/A	A unique identifier for the device that is sending this information. When sending the device its own information, the value of the **device_identifier_** should be **zero**.
2	device_type_	UINT8	N/A	**0**: Phone **1**: Tablet **2**: Head-Mount **3**: Watch
3	navigation_phase_	INT8	N/A	Equivalent to the **navigation_phase_**

				flag in TppSolutionStruct
4	gnss_in_use_	UINT8	N/A	0x00: GNSS data was not used in computing the navigation solution within the last 2 seconds. 0x01: GNSS data was used in computing the navigation solution within the last 2 seconds.
5	magnetometer_in_use_	UINT8	N/A	0x00: Magnetometer data was not used in computing the navigation solution within the last 2 seconds. 0x01: Magnetometer data was used in computing the navigation solution within the last 2 seconds.
6	barometer_in_use_	UINT8	N/A	0x00: Barometer data was not used in computing the navigation solution within the last 2 seconds. 0x01: Barometer data was used in computing the navigation solution within the last 2 seconds.
7	speed_in_use_	UINT8	N/A	0x00: Speed data was not used in computing the navigation solution within the last 2 seconds. 0x01: Speed data was used in computing the navigation solution within the last 2 seconds.
8	use_case_	UINT8	N/A	
9	mode_of_transit_	UINT8	N/A	
10	latitude_	DOUBLE64	deg	
11	longitude_	DOUBLE64	deg	
12	height_	FLOAT32	m	
13	velocity_north_	FLOAT32	m/sec	
14	velocity_east_	FLOAT32	m/sec	
15	velocity_down_	FLOAT32	m/sec	
16	position_north_standard_deviation_	FLOAT32	m	
17	position_east_standard_deviation_	FLOAT32	m	
18	height_standard_deviation_	FLOAT32	m	
19	velocity_north_standard_deviation_	FLOAT32	m/sec	
20	velocity_east_standard_deviation_	FLOAT32	m/sec	
21	velocity_down_standard_deviation_	FLOAT32	m/sec	
22	number_of_steps_	UINT32	N/A	

7.33. TppMultiDeviceHeadingStruct

	Member	Data Type	Unit	Description
1	device_identifier_	UINT64	N/A	A unique identifier for the device that is sending this information. When sending the device its own information, the value of the **device_identifier_** should be **zero**.
2	device_type_	UINT8	N/A	**0**: Phone **1**: Tablet **2**: Head-Mount **3**: Watch
3	navigation_phase_	INT8	N/A	Equivalent to the values taken by the flag **navigation_phase_** in TppSolutionStruct

4	gnss_in_use_	UINT8	N/A	0x00: GNSS data was not used in computing the navigation solution within the last 2 seconds. 0x01: GNSS data was used in computing the navigation solution within the last 2 seconds.
5	magnetometer_in_use_	UINT8	N/A	0x00: Magnetometer data was not used in computing the navigation solution within the last 2 seconds. 0x01: Magnetometer data was used in computing the navigation solution within the last 2 seconds.
6	barometer_in_use_	UINT8	N/A	0x00: Barometer data was not used in computing the navigation solution within the last 2 seconds. 0x01: Barometer data was used in computing the navigation solution within the last 2 seconds.
7	speed_in_use_	UINT8	N/A	0x00: Speed data was not used in computing the navigation solution within the last 2 seconds. 0x01: Speed data was used in computing the navigation solution within the last 2 seconds.
8	use_case_	UINT8	N/A	
9	mode_of_transit_	UINT8	N/A	
10	device_heading_	FLOAT32	deg	
11	platform_heading_	FLOAT32	deg	
12	heading_misalignment_	FLOAT32	deg	
13	heading_standard_deviation_	FLOAT32	deg	
14	position_north_standard_deviation	FLOAT32	m	
15	position_east_standard_deviation_	FLOAT32	m	
16	number_of_steps_	UINT32	N/A	

7.34. TppReturnStatusEnum

	Member	Description
0	TPP_SUCCESS	Returned if a call to any function in the library is successful.
1	TPP_SYSTEM_NOT_INITIALIZED	Returned by ***tpp_advance_navigation_step()*** if the function is called before the system is successfully initialized.
2	TPP_INVALID_IMU_MESSAGE	Returned by ***tpp_advance_navigation_step()*** when one or more members of the input parameter are invalid.
3	TPP_INVALID_GNSS_PVT_MESSAGE	Returned by ***tpp_process_gnss_pvt()*** when one or more members of the input parameter are invalid.
4	TPP_INVALID_BAROMETER_MESSAGE	Returned by ***tpp_process_barometer()*** when one or more members of the input parameter are invalid.
5	TPP_INVALID_MAGNETOMETER_MESSAGE	Returned by ***tpp_process_magnetometer()*** when one or more members of the input parameter are invalid.
6	TPP_INVALID_SPEED_MESSAGE	Returned by ***tpp_process_speed()*** when one or more members of the input parameter are invalid.
7	TPP_INVALID_MULTIPLE_ANTENNA_MESSAGE	Returned by ***tpp_process_multiple_antenna()*** when one or more members of the input parameter are invalid.
~~8~~	~~TPP_INVALID_2D_POSITION_MESSAGE~~	~~Returned by ***tpp_process_2d_position()*** when one or more members of the input parameter are invalid.~~
9	TPP_INVALID_HEIGHT_MESSAGE	Returned by ***tpp_process_height()*** when one or more members of the input parameter are invalid.

10	TPP_INVALID_FLOOR_INFORMATION_MESSAGE	Returned by ***tpp_process_floor_information()*** when one or more members of the input parameter are invalid.
11	TPP_INVALID_CALL_INFORMATION_MESSAGE	Returned by ***tpp_process_call_information()*** when one or more members of the input parameter are invalid.
12	TPP_INVALID_OUTPUT	Returned by ***tpp_advance_navigation_step()*** when the output byte array is invalid.
13	TPP_STOP_NAVIGATION_FAILURE	Returned by ***tpp_stop_navigation()*** when the navigator failed to stop.
14	TPP_INVALID_PROCESS_GNSS_PVT_CALL	Returned by ***tpp_process_gnss_pvt()*** when the function is called after the developer specified that the data from GNSS will not be used in the navigation solution.
15	TPP_INVALID_PROCESS_BAROMETER_CALL	Returned by ***tpp_process_barometer()*** when the function is called after the developer specified that the data from the barometer will not be used in the navigation solution.
16	TPP_INVALID_PROCESS_MAGNETOMETER_CALL	Returned by ***tpp_process_magnetometer()*** when the function is called after the developer specified that the data from the magnetometer will not be used in the navigation solution.
17	TPP_INVALID_PROCESS_SPEED_CALL	Returned by ***tpp_process_speed()*** when the function is called after the developer specified that the data from the speed sensor will not be used in the navigation solution.
18	TPP_INVALID_PROCESS_MULTIPLE_ANTENNA_CALL	Returned by ***tpp_process_multiple_antenna()*** when the function is called after the developer specified that the data from the multiple antenna system will not be used in the navigation solution.
~~19~~	~~TPP_INVALID_PROCESS_2D_POSITION_CALL~~	~~Returned by ***tpp_process_2d_position()*** when the function is called after the developer specified in the initialization that Wi-Fi will not be used and then provided Wi-FI data to the function.~~
20	TPP_INVALID_SET_MODE_OF_TRANSIT_CALL	Returned by ***tpp_set_mode_of_transit()*** when the current mode of transit cannot be set to the one specified as input.
21	TPP_INVALID_FLAG_MAGNETOMETER	Returned by ***tpp_initialize_SSS()*** when the values specified for the **MAGNETOMETER** initialization flag are not valid.
22	TPP_INVALID_FLAG_BAROMETER	Returned by ***tpp_initialize_SSS()*** when the values specified for the **BAROMETER** initialization flag are not valid.
23	TPP_INVALID_FLAG_SPEED	Returned by ***tpp_initialize_SSS()*** when the values specified for the **SPEED** initialization flag are not valid.
24	TPP_INVALID_FLAG_GNSS	Returned by ***tpp_initialize_SSS()*** when the values specified for the **GNSS** initialization flag are not valid.
25	TPP_INVALID_HEADING_TO_START	Returned by ***tpp_initialize_SSS()*** when the values specified for the **GNSS**, **PLATFORM_HEADING**, and **MAGNETOMETER** flags are not valid or off.
26	TPP_INVALID_POSITION_TO_START	Returned by ***tpp_initialize_SSS()*** when the values specified for the **GNSS** and **POSITION** flags are not valid.
27	TPP_INVALID_FLAG_WIFI	Returned by ***tpp_initialize_SSS()*** when the values specified for the **WIFI** initialization flag are not valid.
28	TPP_INVALID_FLAG_MULTIPLE_ANTENNA	Returned by ***tpp_initialize_SSS()*** when the values specified for the **MULTIPLE_ANTENNA** initialization flag are not valid.
29	TPP_INVALID_FLAG_PLATFORM_HEADING	Returned by ***tpp_initialize_SSS()*** when the values specified for the **PLATFORM_HEADING** initialization flag are not valid.
30	TPP_INVALID_FLAG_POSITION	Returned by ***tpp_initialize_SSS()*** when the values specified for the **POSITION** initialization flag are not valid.
31	TPP_INVALID_FLAG_MISALIGNMENT	Returned by ***tpp_initialize_SSS()*** when the values specified for the **MISALIGNMENT** initialization flag are not valid.
32	TPP_INVALID_FLAG_MODE_OF_TRANSIT	Returned by ***tpp_initialize_SSS()*** when the values specified for the **MODE_OF_TRANSIT** initialization flag are not valid.
33	TPP_INVALID_FLAG_FLOOR_INFORMATION	Returned by ***tpp_initialize_SSS()*** when the values specified for the **FLOOR_INFORMATION** initialization flag are not valid.
34	TPP_INVALID_FLAG_GYROSCOPE_BIASES	Returned by ***tpp_initialize_SSS()*** when the values specified for

		the **GYROSCOPE_BIASES** initialization flag are not valid.
35	TPP_INVALID_GYROSCOPE_BIASES_TO_START	Returned by ***tpp_initialize_SSS()*** when the **GYROSCOPE_BIASES** flag is not used in the initialization by the user.
36	TPP_INVALID_FLAG_ACCELEROMETER_CONFIGURATION	Returned by ***tpp_initialize_SSS()*** when the values specified for the **ACCELEROMETER_CONFIGURATION** initialization flag are not valid.
37	TPP_INVALID_FLAG_GYROSCOPE_X_CONFIGURATION	Returned by ***tpp_initialize_SSS()*** when the values specified for the **GYROSCOPE_X_CONFIGURATION** initialization flag are not valid.
38	TPP_INVALID_FLAG_GYROSCOPE_Y_CONFIGURATION	Returned by ***tpp_initialize_SSS()*** when the values specified for the **GYROSCOPE_Y_CONFIGURATION** initialization flag are not valid.
39	TPP_INVALID_FLAG_GYROSCOPE_Z_CONFIGURATION	Returned by ***tpp_initialize_SSS()*** when the values specified for the **GYROSCOPE_Z_CONFIGURATION** initialization flag are not valid.
40	TPP_INVALID_FLAG_DEBUG_DATA	Returned by ***tpp_initialize_SSS()*** when the values specified for the **DEBUG_DATA** initialization flag are not valid.
41	TPP_INVALID_FLAG_MOTION_CONSTRAINTS	Returned by ***tpp_initialize_SSS()*** when the values specified for the **MOTION_CONSTRAINTS** initialization flag are not valid.
42	TPP_INVALID_IMU_MESSAGE_TIMETAG	Returned by ***tpp_advance_navigation_step()*** when the time-tag in the IMU message is incorrect with respect to the time-tag of the previous IMU message. In this case, the navigator stops and the system need to be reinitialized.
43	TPP_INVALID_IDENTIFIER_TO_START	Returned by ***tpp_initialize_SSS()***, on specific platforms, when an invalid identifier is read from the platform.
44	TPP_INVALID_MODE_OF_TRANSIT_TO_START	Returned by ***tpp_initialize_SSS()*** when the values specified for the **GNSS**, and **MODE_OF_TRANSIT** flags are not valid.
45	TPP_INVALID_FLAG_BACKWARD_SMOOTHING	Returned by ***tpp_initialize_SSS()*** when the values specified for the **BACKWARD_SMOOTHING** initialization flag are not valid.
46	TPP_BACKWARD_SMOOTHING_FILES_CREATION_FAILURE	Returned by ***tpp_initialize_SSS()*** when the library fails to create the backward smoothed solution file and other temporary files in the folder path specified by **flag_value_2_** in the **BACKWARD_SMOOTHING** flag.
47	TPP_INVALID_RUN_BACKWARD_SMOOTHING_CALL	Returned by ***tpp_run_backward_smoothing()*** when the navigator was initialized with **BACKWARD_SMOOTHING** flag being "**off**" or when the navigator is not stopped before calling this function.
48	TPP_RUN_BACKWARD_SMOOTHING_FAILURE	Returned by ***tpp_run_backward_smoothing()*** when (1) No information was saved by the navigator for backward smoothing to run successfully. (2) If the navigator is stopped before it outputs a reliable navigation solution. (3) If the library failed to save the backward smoothed navigation solution to the file "**bs_solution.dat**"
55	TPP_INVALID_SYSTEM_DATE_TO_START	Returned by ***tpp_initialize_SSS()*** when the system date is an invalid date with respect to the date of building the library.
56	TPP_INVALID_FLAG_START_OPTION	Returned by ***tpp_initialize_SSS()*** when the value specified for the **START_OPTION** initialization flag is not valid.
57	TPP_INVALID_OPTION_SENSORS_ONLY	Returned by ***tpp_initialize_SSS()*** when **flag_value_1_** for the **START_OPTION** flag is "sensors_only" and **flag_value_1_** for **GNSS** or **WIFI** flags is "on" (i.e. data from the GNSS receiver and Wi-Fi is specified to be used by the navigator)
58	TPP_INVALID_OPTION_SENSORS_RELIABLE_WIRELESS	Returned by ***tpp_initialize_SSS()*** when **flag_value_1_** for the **START_OPTION** flag is "sensors_reliable_wireless" and **flag_value_1_** for both **GNSS** and **WIFI** flags is "off" (i.e. data from the GNSS receiver and Wi-Fi is specified not to be used by the navigator)
59	TPP_INVALID_OPTION_SENSORS_WIRELESS	Returned by ***tpp_initialize_SSS()*** when **flag_value_1_** for the **START_OPTION** flag is "sensors_wireless" and **flag_value_1_** for both **GNSS** and **WIFI** flags is "off" (i.e. data from the GNSS receiver and Wi-Fi is specified not to be used by the navigator)
60	TPP_INVALID_FLAG_DEVICE	Returned by ***tpp_initialize_SSS()*** when the value specified for the **DEVICE** initialization flag is not valid.

61	TPP_INVALID_FLAG_ACCELEROMETER_BIASES	Returned by ***tpp_initialize_SSS()*** when the values specified for the **ACCELEROMETER_BIASES** initialization flag are not valid.
62	TPP_INVALID_PLATFORM_HEADING_MESSAGE	Returned by ***tpp_process_platform_heading()*** when one or more members of the input parameter are invalid.
63	TPP_INVALID_QUATERNIONS_MESSAGE	Returned by ***tpp_process_9dof_quaternions()*** or ***tpp_process_6dof_quaternions()*** when one or more members of the input parameter are invalid.
64	TPP_INVALID_SET_ZUPT_MODE_CALL	Returned by ***tpp_set_zupt_mode()*** when the ZUPT mode cannot be set to the one specified as input.
65	TPP_INVALID_SET_MISALIGNMENT_MESSAGE	Returned by ***tpp_set_misalignment()*** when one or more members of the input parameter are invalid.
66	TPP_INVALID_SET_MISALIGNMENT_CALL	Returned by ***tpp_set_misalignment()*** when the misalignment estimation cannot be set to the one specified as input.
67	TPP_INVALID_SET_MAGNETOMETER_CALIBRATION_CALL	Returned by ***tpp_set_magnetometer_calibration()*** when the magnetometer calibration setting cannot be set to the one specified as input.
68	TPP_INVALID_SET_USE_CASE_CALL	Returned by ***tpp_set_use_case()*** when the use case cannot be set to the one specified as input.
69	TPP_INVALID_FLAG_OUTPUT_POSITION_ERROR_WEIGHTING	Returned by ***tpp_initialize_SSS()*** when the values specified for the **OUTPUT_POSITION_ERROR_WEIGHTING** initialization flag are not valid.
70	TPP_INVALID_FLAG_OUTPUT_VELOCITY_ERROR_WEIGHTING	Returned by ***tpp_initialize_SSS()*** when the values specified for the **OUTPUT_VELOCITY_ERROR_WEIGHTING** initialization flag are not valid.
71	TPP_INVALID_FLAG_OUTPUT_ATTITUDE_ERROR_WEIGHTING	Returned by ***tpp_initialize_SSS()*** when the values specified for the **OUTPUT_ATTITUDE_ERROR_WEIGHTING** initialization flag are not valid.
72	TPP_INVALID_IMU_BIASES_MESSAGE	Returned by ***tpp_process_imu_biases()*** when one or more members of the input parameter are invalid.
73	TPP_INVALID_OPERATOR_2D_POSITION_MESSAGE	Returned by ***tpp_process_operator_2d_position()*** when one or more members of the input parameter are invalid.
74	TPP_INVALID_WIRELESS_MESSAGE	Returned by ***tpp_process_wireless()*** when one or more members of the input parameters are invalid.
75	TPP_INVALID_PROCESS_WIFI_CALL	Returned by ***tpp_process_wireless()*** when the function is called after the developer specified in the initialization that Wi-Fi will not be used and then provided Wi-FI data to the function.
76	TPP_INVALID_PROCESS_MISALIGNMENT_MESSAGE	Returned by ***tpp_process_misalignment()*** when one or more members of the input parameter are invalid.
77	TPP_INVALID_DEVICE_HEADING_MESSAGE	Returned by ***tpp_set_device_heading()*** when one or more members of the input parameter are invalid.
78	TPP_INVALID_FLAG_ZUPT_THRESHOLDS	Returned by ***tpp_initialize_SSS()*** when the values specified for the **ZUPT_THRESHOLDS** initialization flag are not valid.
79	TPP_INVALID_ZUPT_THRESHOLDS_TO_START	Returned by ***tpp_initialize_SSS()*** when **flag_value_1_** for the **GNSS** flag is "off" and the **ZUPT_THRESHOLDS** flag is not used in the initialization by the user.
80	TPP_INVALID_NAVIGATION_SESSION_HANDLE	Returned by any function that uses the navigation session handle when the handle passed is invalid.
81	TPP_INVALID_FLAG_DECLINATION_ANGLE	Returned by ***tpp_initialize_SSS()*** when **flag_value_1_** for the **DECLINATION_ANGLE** flag is not valid.
82	TPP_INVALID_SET_ORIENTATION_PITCH_CALL	Returned by ***tpp_set_orientation_based_on_pitch()*** when the orientation pitch exceeds its threshold.
83	TPP_INVALID_STREET_MAP_MESSAGE	Returned by ***tpp_process_street_map()*** when one or more members of the input parameter are invalid.
100	TPP_COMPUTATION_IN_PROGRESS	Returned by ***tpp_prerun_compute_zupt_thresholds_imu()*** when the computation operation is still in progress and no values are ready for the user to use.

101	TPP_ZUPT_THRESHOLDS_NOT_RESET	Returned when ***tpp_prerun_reset_zupt_thresholds()*** is not called before calling the computation functions that are used for computing the ZUPT Threshold values.
102	TPP_INVALID_VENUE_MAP_MESSAGE	Returned by ***tpp_process_venue_map()*** when one or more members of the input parameter are invalid.
103	TPP_INVALID_FLAG_REPLACE_ATTITUDE_WITH_6DOF	Returned by ***tpp_initialize_SSS()*** when the values specified for the **REPLACE_ATTITUDE_WITH_6DOF** initialization flag are not valid.
104	TPP_INVALID_ANCHOR_POINT_PACKET	Returned by ***tpp_add_anchor_point()*** when one or more members of ***anchor_point_message_pointer*** are invalid.
105	TPP_INVALID_START_ANCHOR_POINT	Returned by ***tpp_add_anchor_point()*** when one or more members of the **start** anchor point are invalid.
106	TPP_INVALID_END_ANCHOR_POINT	Returned by ***tpp_add_anchor_point()*** when one or more members of the **end** anchor point are invalid, or if the developer wanted to add an end anchor point without adding a valid start anchor point first.
107	TPP_INVALID_FLAG_PROCESSING_MODE	Returned by ***tpp_initialize_SSS()*** when the values specified for **PROCESSING_MODE** initialization flag are not valid
108	TPP_INVALID_FLAG_USE_CASE	Returned by ***tpp_initialize_SSS()*** when the values specified for **USE_CASE** initialization flag are not valid
109	TPP_INVALID_FLAG_LIBRARY_CALL_TYPE	Returned by ***tpp_initialize_SSS()*** when the values specified for **LIBRARY_CALL_TYPE** initialization flag are not valid
150	TPP_MULTI_DEVICE_INVALID_NUMBER_OF_DEVICES	Returned by the multi-device API functions when the number of devices is less than 2 or greater than 8.
151	TPP_MULTI_DEVICE_INVALID_LATENCY_MESSAGE	Returned by the multi-device API functions when the one or more fields of the ***TppMultiDeviceLatencyStruct*** argument passed to the functions is invalid.
152	TPP_MULTI_DEVICE_IDENTIFIER_ZERO_NOT_PRESENT	Returned by the multi-device API functions when the device identifier of zero is not present in the list of device identifiers in ***TppMultiDevicePositionVelocityStruct***
153	TPP_MULTI_DEVICE_INVALID_POSITION_VELOCITY_MESSAGE	Returned by ***tpp_multi_device_process_position_velocity()*** when one or more fields of ***TppMultiDevicePositionVelocityStruct*** are invalid.
154	TPP_MULTI_DEVICE_INVALID_HEADING_MESSAGE	Returned by ***tpp_multi_device_process_heading()*** when one or more fields of ***TppMultiDeviceHeadingStruct*** are invalid.
155	TPP_MULTI_DEVICE_INVALID_SECONDARY_GNSS_DEVICE_MESSAGE	Returned by ***tpp_multi_device_process_secondary_gnss_pvt()*** when one or more fields of TppGnssPvtMessageStruct are invalid.
156	TPP_MULTI_DEVICE_INVALID_PROCESS_SECONDARY_GNSS_PVT_CALL	Returned by ***tpp_multi_device_process_secondary_gnss_pvt()*** when the function is called after the developer specified that the data from GNSS will not be used in the navigation solution.
255	TPP_INVALID_SYNCHRONIZATION_EVENT	Returned by ***tpp_add_synchronization_event()*** when the value of the parameter passed is not greater than zero or when the value of the parameter passed is less than the value of the previous event number.
	TPP_INVALID_GNSS_OBSERVATIONS_MESSAGE	Returned by ***tpp_process_gnss_observations()*** when one or more members of the input parameter are invalid.
	TPP_INVALID_GLONASS_EPHEMERIS_MESSAGE	Returned by ***tpp_process_glonass_ephemeris()*** when one or more members of the input parameter are invalid.
	TPP_INVALID_GPS_EPHEMERIS_MESSAGE	Returned by ***tpp_process_gps_ephemeris()*** when one or more members of the input parameter are invalid.

7.35. TppGnssObservationsMessageStruct

	Field Name	Data Type	Unit	Description
1	gnss_week_	UINT32	N/A	Week
2	gnss_tow_	DOUBLE64	sec	Time of week
3	satellite_num_	UINT32	N/A	Number of satellites
4	gnss_prn_	UINT32	N/A	PRN
5	satellite_system_	UINT32	N/A	Satellite System: 0x00: GPS 0x01: GLONASS 0x02:COMPASS 0x03: GALILEO
6	signal_type_	UINT32	N/A	0x00: L1/CA 0x01: L2 0x02: L5
7	phase_lock_	INT32	N/A	0x00: Lose Lock 0x01: Lock
8	code_lock_	INT32	N/A	0x00: Lose Lock 0x01: Lock
9	lose_lock_indicator	UINT32	N/A	Lose Lock Indicator
10	cn0_	DOUBLE64	dB-Hz	Carrier to Noise density
11	lock_time_	DOUBLE64	sec	Time since the latest lock
12	pseudorange_	DOUBLE64	m	
13	pseudorange _std_	DOUBLE64	m	
14	doppler_	DOUBLE64	Hz	
15	doppler_std_	DOUBLE64	Hz	
16	phase_	DOUBLE64	Cycles	

7.36. TppGpsEphemerisMessageStruct

	Field Name	Data Type	Unit	Description
1	gnss_prn_	UINT32	N/A	PRN
2	gnss_week_	UINT32	N/A	Week number
3	gnss_tow_	DOUBLE64	sec	Time of week
4	code_	UINT32	N/A	Code on L2 0x00: Reserved 0x01: P code on 0x02: C/A code on
5	ura_	DOUBLE64	m	Satellite accuracy
6	health_	UINT32	N/A	Satellite health 0x00: all navigation date are ok 0x01: some navigation date are bad
7	iodc_	UINT32	N/A	
8	flag_	UINT32	N/A	Data flag on L2P
9	tgd_	DOUBLE64	sec	
10	toc_	DOUBLE64	sec	
11	af2_	DOUBLE64	sec/sec/sec	
12	af1_	DOUBLE64	sec/sec	
13	af0_	DOUBLE64	sec	
14	iode2_	UINT32	N/A	
15	crs_	DOUBLE64	m	
16	deltan_	DOUBLE64	semi-circles/sec	
17	m0_	DOUBLE64	semi-circles	
18	cuc_	DOUBLE64	rad	
19	ecc_	DOUBLE64	N/A	

20	cus_	DOUBLE64	rad	
21	sqrta_	DOUBLE64	√m	
22	toe_	DOUBLE64	sec	
23	cic_	DOUBLE64	rad	
24	omega0_	DOUBLE64	rad/sec	
25	cis	DOUBLE64	rad	
26	i0_	DOUBLE64	semi-circles	
27	crc_	DOUBLE64	m	
28	omega_	DOUBLE64	semi-circles	
29	omegadot_	DOUBLE64	semi-circles/sec	
30	iode3_	UINT32	N/A	
31	idot_	DOUBLE64	semi-circles/sec	

7.37. TppGlonassEphemerisMessageStruct

	Field Name	Data Type	Unit	Description
1	gnss_prn_	UINT32	N/A	
2	frequency_	INT32	N/A	Frequency offset
3	toe_	DOUBLE64	sec	
4	toff_	DOUBLE64	sec	
5	iode_	UINT32	N/A	
6	health_	UINT32	N/A	Satellite health 0x00: all navigation date are ok 0x01: some navigation date are bad
7	pos_x_	DOUBLE64	m	
8	pos_y_	DOUBLE64	m	
9	pos_z_	DOUBLE64	m	
10	vel_x_	DOUBLE64	m/sec	
11	vel_y_	DOUBLE64	m/sec	
12	vel_z_	DOUBLE64	m/sec	
13	acc_x_	DOUBLE64	m/sec/sec	
14	acc_y_	DOUBLE64	m/sec/sec	
15	acc_z_	DOUBLE64	m/sec/sec	
16	taun_	DOUBLE64	sec	
17	gamman_	DOUBLE64	sec/sec	
18	tof_	DOUBLE64	sec	
19	age_	INT32	N/A	

7.38. TppStreetMapMessageStruct

	Member	Data Type	Unit	Description
1	time_	DOUBLE64	s	
2	latitude_	DOUBLE64	deg	
3	longitude_	DOUBLE64	deg	
4	platform_heading_available_	UINT8	N/A	0x00: Platform heading and the corresponding standard deviation are not available 0x01: Platform heading and the corresponding standard deviation are available 0x02: Platform heading is available but not the standard deviation
5	platform_heading_	FLOAT32	deg	Measure clockwise with respect to

				the true north
6	platform_heading_standard_deviation_	FLOAT32	deg	
	postion_2d_standard_available_	UINT8		0x00: Horizontal position standard deviations are not available 0x01: Horizontal position standard deviations are available
7	position_north_standard_deviation_	FLOAT32	m	
8	position_east_standard_deviation_	FLOAT32	m	
9	height_available_	UINT8	N/A	0x00: Height and the corresponding standard deviation are not available 0x01: Height and the corresponding standard deviation are available 0x02: Height is available but the corresponding standard deviation is unavailable
10	height_	FLOAT32	m	
11	height_standard_deviation_	FLOAT32	m	
12	max_speed_available_	UINT8	N/A	0x00: Maximum speed for the segment of the road is unavailable 0x01: Maximum speed for the segment of the road is available
13	max_speed_	FLOAT32	m/s	

8. Using Trusted Positioning Library

Trusted Positioning API provides functions and data types to do the following:

1. Initialize the navigator.
2. Process the sensors' data that is pushed in by the developer in the specified format as defined herein.
3. Output a structured byte array that the developer can parse and extract the required data or save in a file.

8.1. Initialization

8.1.1. *TMN Initialization*

Following is a code sample to show how to do the initialization for TMN. First, the developer creates a navigation session handle, and then specifies an array of flags where each entry in the array specifies the flag name and the corresponding value(s).

In this sample code, the developer specifies the first flag name to be "**MAGNETOMETER**" with the first and second values of this flag being "**on**" and "**1.0**" which corresponds to the navigator utilizing the magnetometer data with original weighting as provided in the standard deviation field.

The developer then specifies the second flag name to be "**BAROMETER**" with the first and second values of this flag being "**on**" and "**1.0**" which corresponds to the navigator utilizing the barometer data with original weighting as provided in the standard deviation field.

The developer then specifies the number of initialization flags and the flags array in a variable of type ***tpp_initialization_structure*** and passes a pointer of the variable to ***tpp_initialize_tmn_SSS()***. As there are no other flags, TMN will use the default settings for those flags.

Code Sample 2 TMN Initialization (1)

```
//Creating a navigation session handle
TppNavigationSessionHandle tpp_handle = tpp_create_navigation_session_handle();
if(tpp_handle == NULL)
{
    return;
}
//Holds the number of initialization flags along with a pointer to the initialization structure
//array
TppInitializationStruct tpp_initialization_struct;
//Used as a short name instead of "tpp_initialization_struct.tpp_initialization_flag_pointer_"
TppInitializationFlagStruct* flags_struct_pointer;
//Holds the actual number of flags that the developer initialized
unsigned short flags_counter;
//Holds the number of maximum number of initialization flags the developer will allocate space
//for
const unsigned short MAXIMUM_NUMBER_OF_FLAGS = 20;

//Holds the returned array from tpp_initialize_fff()and its size to be written to a file for
//Trusted Positioning debugging
TppOutputStruct tpp_initialization_output_array_struct;
```

```
TppReturnStatusEnum tpp_return_status_enum;

tpp_initialization_output_array_struct.byte_array_pointer_ =
    (TppTypeUint8*)malloc(TPP_INIT_BYTE_ARRAY_LENGTH);

flags_counter = 0;

flags_struct_pointer =
(TppInitializationFlagStruct*) malloc(MAXIMUM_NUMBER_OF_FLAGS *
sizeof(TppInitializationFlagStruct));

memset(flags_struct_pointer, 0,MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter]. flag_name_,"BAROMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(10);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.8f" , 1.0);
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"MAGNETOMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(10);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.8f" , 1.0);
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"GYROSCOPE_BIASES");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(10);
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(10);
flags_struct_pointer[flags_counter].flag_value_3_ = (char*) malloc(10);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.5f" , 0.0);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.5f" , 0.0);
sprintf( flags_struct_pointer[flags_counter].flag_value_3_ , "%.5f" , 0.0);
flags_counter++;

tpp_initialization_structure.number_of_initialization_flags_ = flags_counter;
tpp_initialization_struct.tpp_initialization_flag_pointer_   = flags_struct_pointer;

tpp_return_status=tpp_initialize_tmn_SSS
(tpp_handle, &tpp_initialization_struct ,&tpp_initialization_output_array_struct);

if( tpp_return_status != TPP_SUCCESS )
{
    printf("[ERROR] tpp_initialize_tmn_SSS() failed\n");
}
```

After initialization, the user can free the allocated memory used in populating the flags and their values according the code sample below.

Code Sample 3 TMN Initialization (2)

```
int counter;
for(counter = 0; counter < flags_counter ; counter++){
    if(flags_struct_pointer[counter].flag_name_  != NULL){
        free(flags_struct_pointer[counter].flag_name_);
```

```
        if(flags_struct_pointer[counter].flag_value_1_ != NULL){
            free(flags_struct_pointer[counter].flag_value_1_);
        }

        if(flags_struct_pointer[counter].flag_value_2_ != NULL){
            free(flags_struct_pointer[counter].flag_value_2_);
        }

        if(flags_struct_pointer[counter].flag_value_3_ != NULL){
            free(flags_struct_pointer[counter].flag_value_3_);
        }

        if(flags_struct_pointer[counter].flag_value_4_ != NULL){
            free(flags_struct_pointer[counter].flag_value_4_);
        }

        if(flags_struct_pointer[counter].flag_value_5_ != NULL){
            free(flags_struct_pointer[counter].flag_value_5_);
        }
    }
}

free(flags_struct_pointer);
```

The following code sample shows an example where the initial position and heading is sent to TMN.

Code Sample 3 TMN Initialization (2)

```
//Holds the number of initialization flags along with a pointer to the initialization structure
array
TppInitializationStruct tpp_initialization_struct;
//Used as a short name instead of "tpp_initialization_struct.tpp_initialization_flag_pointer_"
TppInitializationFlagStruct* flags_struct_pointer;
//Holds the actual number of flags that the developer initialized
unsigned short flags_counter;
//Holds the number of maximum number of initialization flags the developer will allocate space
for
const unsigned short MAXIMUM_NUMBER_OF_FLAGS = 20;

//Holds the returned array from tpp_initialize_fff()and its size to be written to a file for
//Trusted Positioning debugging
TppOutputStruct tpp_initialization_output_array_struct;
TppReturnStatusEnum tpp_return_status_enum;

tpp_initialization_output_array_struct.byte_array_pointer_ =
    (TppTypeUint8*)malloc(TPP_INIT_BYTE_ARRAY_LENGTH);
flags_counter = 0;
flags_struct_pointer =
(TppInitializationFlagStruct*) malloc(MAXIMUM_NUMBER_OF_FLAGS *
sizeof(TppInitializationFlagStruct));
memset(flags_struct_pointer, 0,MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"BAROMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_2_,"1.0");
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"GYROSCOPE_BIASES");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc (10);
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc (10);
flags_struct_pointer[flags_counter].flag_value_3_ = (char*) malloc (10);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.5f" , 0.0);
```

```
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.5f" , 0.0);
sprintf( flags_struct_pointer[flags_counter].flag_value_3_ , "%.5f" , 0.0);
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"POSITION");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc (20);
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc (20);
flags_struct_pointer[flags_counter].flag_value_3_ = (char*) malloc (10);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.8f" , user_initial_latitude);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.8f" , user_initial_longitude);
sprintf( flags_struct_pointer[flags_counter].flag_value_3_ , "%.2f" , user_initial_height);
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"PLATFORM_HEADING");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc (20);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.8f" , user_initial_heading);
flags_counter++;

tpp_initialization_structure.number_of_initialization_flags_ = flags_counter;
tpp_initialization_struct.tpp_initialization_flag_pointer_   = flags_struct_pointer;

tpp_return_status = tpp_initialize_tmn_SSS( &tpp_initialization_struct ,
&tpp_initialization_output_array_struct);

if( tpp_return_status != TPP_SUCCESS )
{
    printf("[ERROR] tpp_initialize_tmn_SSS() failed\n");
}
```

8.1.2. *TPN Free Initialization*

Following is a code sample to show how to do the initialization for TPN free. First, the developer creates a navigation session handle, and then specifies an array of flags where each entry in the array specifies the flag name and the corresponding value(s).

In this sample code, the developer specifies the first flag name to be "**MAGNETOMETER**" with the first and second values of this flag being "**on**" and "**1.0**" which corresponds to the navigator utilizing the magnetometer data with original weighting as provided in the standard deviation field.

The developer then specifies the second flag name to be "**BAROMETER**" with the first and second values of this flag being "**on**" and "**1.0**" which corresponds to the navigator utilizing the barometer data with original weighting as provided in the standard deviation field.

The third flag specified is "**GYROSCOPE_BIASES**". The biases' values are zeros for the x-axis, y-axis and the z-axis gyroscopes.

The developer then specifies the number of initialization flags and the flags array in a variable of type ***tpp_initialization_structure*** and passes a pointer of the variable to ***tpp_initialize_tpn_free()***. As there are no other flags, TPN will use the default settings for the flags that were not initialized.

Code Sample 4 TPN Free Initialization (1)

```
//Creating a navigation session handle
TppNavigationSessionHandle tpp_handle = tpp_create_navigation_session_handle();
if(tpp_handle == NULL)
```

```
{
    return;
}
//Holds the number of initialization flags along with a pointer to the initialization structure
//array
TppInitializationStruct tpp_initialization_struct;
//Used as a short name instead of "tpp_initialization_struct.tpp_initialization_flag_pointer_"
TppInitializationFlagStruct* flags_struct_pointer;
//Holds the actual number of flags that the developer initialized
unsigned short flags_counter;
//Holds the number of maximum number of initialization flags the developer will allocate space
//for
const unsigned short MAXIMUM_NUMBER_OF_FLAGS = 20;

//Holds the returned array from tpp_initialize_fff()and its size to be written to a file for
//Trusted Positioning debugging
TppOutputStruct tpp_initialization_output_array_struct;
TppReturnStatusEnum tpp_return_status_enum;

tpp_initialization_output_array_struct.byte_array_pointer_ =
    (TppTypeUint8*)malloc(TPP_INIT_BYTE_ARRAY_LENGTH);

flags_counter = 0;

flags_struct_pointer =
(TppInitializationFlagStruct*) malloc(MAXIMUM_NUMBER_OF_FLAGS *
sizeof(TppInitializationFlagStruct));

memset(flags_struct_pointer, 0,MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"BAROMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(10);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.8f" , 1.0);
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"MAGNETOMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(10);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.8f" , 1.0);
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"GYROSCOPE_BIASES");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(10);
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(10);
flags_struct_pointer[flags_counter].flag_value_3_ = (char*) malloc(10);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.5f" , 0.0);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.5f" , 0.0);
sprintf( flags_struct_pointer[flags_counter].flag_value_3_ , "%.5f" , 0.0);
flags_counter++;

tpp_initialization_structure.number_of_initialization_flags_ = flags_counter;
tpp_initialization_struct.tpp_initialization_flag_pointer_   = flags_struct_pointer;

tpp_return_status=tpp_initialize_tpn_free
(tpp_handle, &tpp_initialization_struct ,&tpp_initialization_output_array_struct);
```

```
if( tpp_return_status != TPP_SUCCESS )
{
    printf("[ERROR] tpp_initialize_tpn_free() failed\n");
}
```

After initialization, the user can free the allocated memory used in populating the flags and their values according the code sample below.

Code Sample 5 TPN Free Initialization (2)

```
int counter;
for(counter = 0; counter < flags_counter ; counter++){
    if(flags_struct_pointer[counter].flag_name_  != NULL){
        free(flags_struct_pointer[counter].flag_name_);

        if(flags_struct_pointer[counter].flag_value_1_ != NULL){
            free(flags_struct_pointer[counter].flag_value_1_);
        }

        if(flags_struct_pointer[counter].flag_value_2_ != NULL){
            free(flags_struct_pointer[counter].flag_value_2_);
        }

        if(flags_struct_pointer[counter].flag_value_3_ != NULL){
            free(flags_struct_pointer[counter].flag_value_3_);
        }

        if(flags_struct_pointer[counter].flag_value_4_ != NULL){
            free(flags_struct_pointer[counter].flag_value_4_);
        }

        if(flags_struct_pointer[counter].flag_value_5_ != NULL){
            free(flags_struct_pointer[counter].flag_value_5_);
        }
    }
}

free(flags_struct_pointer);
```

The following code sample shows an example where TPN free is initialized to work without GNSS and without magnetometer, and the developer provided the initial position and heading of the platform.

Code Sample 6 TPN Free Initialization (3)

```
//Holds the number of initialization flags along with a pointer to the initialization structure
array
TppInitializationStruct tpp_initialization_struct;
//Used as a short name instead of "tpp_initialization_struct.tpp_initialization_flag_pointer_"
TppInitializationFlagStruct* flags_struct_pointer;
//Holds the actual number of flags that the developer initialized
unsigned short flags_counter;
//Holds the number of maximum number of initialization flags the developer will allocate space
for
const unsigned short MAXIMUM_NUMBER_OF_FLAGS = 20;

//Holds the returned array from tpp_initialize_fff()and its size to be written to a file for
//Trusted Positioning debugging
TppOutputStruct tpp_initialization_output_array_struct;
TppReturnStatusEnum tpp_return_status_enum;

tpp_initialization_output_array_struct.byte_array_pointer_ =
    (TppTypeUint8*)malloc(TPP_INIT_BYTE_ARRAY_LENGTH);
flags_counter = 0;
flags_struct_pointer =
```

```
(TppInitializationFlagStruct*) malloc(MAXIMUM_NUMBER_OF_FLAGS *
sizeof(TppInitializationFlagStruct));
memset(flags_struct_pointer, 0,MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"BAROMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_2_,"1.0");
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"GYROSCOPE_BIASES");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc (10);
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc (10);
flags_struct_pointer[flags_counter].flag_value_3_ = (char*) malloc (10);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.5f" , 0.0);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.5f" , 0.0);
sprintf( flags_struct_pointer[flags_counter].flag_value_3_ , "%.5f" , 0.0);
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"GNSS");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"off");
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"POSITION");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc (20);
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc (20);
flags_struct_pointer[flags_counter].flag_value_3_ = (char*) malloc (10);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.8f" , user_initial_latitude);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.8f" , user_initial_longitude);
sprintf( flags_struct_pointer[flags_counter].flag_value_3_ , "%.2f" , user_initial_height);
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_name_,"PLATFORM_HEADING");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc (20);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.8f" , user_initial_heading);
flags_counter++;

tpp_initialization_structure.number_of_initialization_flags_ = flags_counter;
tpp_initialization_struct.tpp_initialization_flag_pointer_   = flags_struct_pointer;

tpp_return_status = tpp_initialize_tpn_free( &tpp_initialization_struct ,
&tpp_initialization_output_array_struct);

if( tpp_return_status != TPP_SUCCESS )
{
    printf("[ERROR] tpp_initialize_tpn_free() failed\n");
}
```

8.1.3. *TPN Tethered Initialization*

Following is an example of code to show how to do the initialization for TPN tethered. First, the developer creates a navigation session handle, and then specifies an array of flags where each entry in the array specifies the flag name and the corresponding value(s).

In this sample code, the developer specifies the first flag name to be "**MAGNETOMETER**" with the first and second values of this flag being "**on**" and "**1.0**" which corresponds to the navigator utilizing the magnetometer data with original weighting as provided in the standard deviation field.

The developer then specifies the second flag name to be "**BAROMETER**" with the first and second values of this flag being "**on**" and "**1.0**" which corresponds to the navigator utilizing the barometer data with original weighting as provided in the standard deviation field.

The developer then specifies the number of initialization flags and the flags array in a variable of type ***tpp_initialization_structure*** and passes a pointer of the variable to ***tpp_initialize_tpn_tethered()***. As there are no other flags, TPN will use the default settings for those flags.

Code Sample 7 TPN Tethered Initialization (1)

```
//Creating a navigation session handle
TppNavigationSessionHandle tpp_handle = tpp_create_navigation_session_handle();
if(tpp_handle == NULL)
{
    return;
}
```

```
//Holds the number of initialization flags along with a pointer to the initialization structure array
TppInitializationStruct tpp_initialization_struct;
//Used as a short name instead of "tpp_initialization_struct.tpp_initialization_flag_pointer_"
TppInitializationFlagStruct* flags_struct_pointer;
//Holds the actual number of flags that the developer initialized
unsigned short flags_counter;
//Holds the number of maximum number of initialization flags the developer will allocate space for
const unsigned short MAXIMUM_NUMBER_OF_FLAGS = 20;

//Holds the returned array from tpp_initialize_fff()and its size to be written to a file for
//Trusted Positioning debugging
TppOutputStruct tpp_initialization_output_array_struct;
TppReturnStatusEnum tpp_return_status_enum;

tpp_initialization_output_array_struct.byte_array_pointer_ =
        (TppTypeUint8*)malloc(TPP_INIT_BYTE_ARRAY_LENGTH);

flags_counter = 0;

flags_struct_pointer =
(TppInitializationFlagStruct*) malloc(MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

memset(flags_struct_pointer, 0,MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"BAROMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_2_,"1.0");
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"MAGNETOMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_2_,"1.0");
flags_counter++;

tpp_initialization_structure.number_of_initialization_flags_ = flags_counter;
tpp_initialization_struct.tpp_initialization_flag_pointer_   = flags_struct_pointer;

tpp_return_status = tpp_initialize_tpn_tethered( &tpp_initialization_struct ,
&tpp_initialization_output_array_struct);

if( tpp_return_status != TPP_SUCCESS )
{
    printf("[ERROR] tpp_initialize_tpn_tethered() failed\n");
}
```

The following code sample shows an example where TPN Tethered is initialized to work without GNSS and without magnetometer, and the developer provided the initial position and heading.

Code Sample 8 TPN Tethered Initialization (2)

```
//Holds the number of initialization flags along with a pointer to the initialization structure array
TppInitializationStruct tpp_initialization_struct;
//Used as a short name instead of "tpp_initialization_struct.tpp_initialization_flag_pointer_"
TppInitializationFlagStruct* flags_struct_pointer;
//Holds the actual number of flags that the developer initialized
unsigned short flags_counter;
//Holds the number of maximum number of initialization flags the developer will allocate space for
const unsigned short MAXIMUM_NUMBER_OF_FLAGS = 20;

//Holds the returned array from tpp_initialize_fff()and its size to be written to a file for
//Trusted Positioning debugging
TppOutputStruct tpp_initialization_output_array_struct;
TppReturnStatusEnum tpp_return_status_enum;

tpp_initialization_output_array_struct.byte_array_pointer_ =
        (TppTypeUint8*)malloc(TPP_INIT_BYTE_ARRAY_LENGTH);
flags_counter = 0;
flags_struct_pointer =
(TppInitializationFlagStruct*) malloc(MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));
memset(flags_struct_pointer, 0,MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"BAROMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_2_,"1.0");
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"GNSS");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"off");
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"POSITION");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc (20);
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc (20);
flags_struct_pointer[flags_counter].flag_value_3_ = (char*) malloc (10);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.8f" , user_initial_latitude);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.8f" , user_initial_longitude);
sprintf( flags_struct_pointer[flags_counter].flag_value_3_ , "%.2f" , user_initial_height);
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"HEADING");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc (20);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.8f" , user_initial_heading);
flags_counter++;

tpp_initialization_structure.number_of_initialization_flags_ = flags_counter;
tpp_initialization_struct.tpp_initialization_flag_pointer_   = flags_struct_pointer;

tpp_return_status = tpp_initialize_tpn_tethered( &tpp_initialization_struct ,
&tpp_initialization_output_array_struct);

if( tpp_return_status != TPP_SUCCESS )
{
    printf("[ERROR] tpp_initialize_tpn_tethered() failed\n");
}
```

8.1.4. *TVN Initialization*

Following is a code sample to show how to do the initialization for TVN. First, the developer creates a navigation session handle, and then specifies an array of flags where each entry in the array specifies the flag name and the corresponding value(s).

In this sample code, the developer specifies the first flag name to be "**MAGNETOMETER**" with the first and second values of this flag being "**on**" and "**1.0**" which corresponds to the navigator utilizing the magnetometer data with original weighting as provided in the standard deviation field.

The developer then specifies the second flag name to be "**BAROMETER**" with the first and second values of this flag being "**on**" and "**1.0**" which corresponds to the navigator utilizing the barometer data with original weighting as provided in the standard deviation field.

The developer then specifies the number of initialization flags and the flags array in a variable of type ***tpp_initialization_structure*** and passes a pointer of the variable to ***tpp_initialize_tvn()***. As there are no other flags, TVN will use the default settings for those flags.

Code Sample 9 TVN Initialization (1)

```
//Creating a navigation session handle
TppNavigationSessionHandle tpp_handle = tpp_create_navigation_session_handle();
if(tpp_handle == NULL)
{
    return;
}
```

```
//Holds the number of initialization flags along with a pointer to the initialization structure array
TppInitializationStruct tpp_initialization_struct;
//Used as a short name instead of "tpp_initialization_struct.tpp_initialization_flag_pointer_"
TppInitializationFlagStruct* flags_struct_pointer;
//Holds the actual number of flags that the developer initialized
unsigned short flags_counter;
//Holds the number of maximum number of initialization flags the developer will allocate space for
const unsigned short MAXIMUM_NUMBER_OF_FLAGS = 20;

//Holds the returned array from tpp_initialize_fff()and its size to be written to a file for
//Trusted Positioning debugging
TppOutputStruct tpp_initialization_output_array_struct;
TppReturnStatusEnum tpp_return_status_enum;

tpp_initialization_output_array_struct.byte_array_pointer_ =
        (TppTypeUint8*)malloc(TPP_INIT_BYTE_ARRAY_LENGTH);

flags_counter = 0;

flags_struct_pointer =
(TppInitializationFlagStruct*) malloc(MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

memset(flags_struct_pointer, 0,MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"BAROMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_2_,"1.0");
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"MAGNETOMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_2_,"1.0");
flags_counter++;

tpp_initialization_structure.number_of_initialization_flags_ = flags_counter;
tpp_initialization_struct.tpp_initialization_flag_pointer_   = flags_struct_pointer;

tpp_return_status = tpp_initialize_tvn( &tpp_initialization_struct ,
&tpp_initialization_output_array_struct);

if( tpp_return_status != TPP_SUCCESS )
{
    printf("[ERROR] tpp_initialize_tvn() failed\n");
}
```

The following code sample shows an example where the initial position and heading is sent to TVN.

Code Sample 10 TVN Initialization (2)

```
//Holds the number of initialization flags along with a pointer to the initialization structure array
TppInitializationStruct tpp_initialization_struct;
//Used as a short name instead of "tpp_initialization_struct.tpp_initialization_flag_pointer_"
TppInitializationFlagStruct* flags_struct_pointer;
//Holds the actual number of flags that the developer initialized
unsigned short flags_counter;
//Holds the number of maximum number of initialization flags the developer will allocate space for
const unsigned short MAXIMUM_NUMBER_OF_FLAGS = 20;

//Holds the returned array from tpp_initialize_fff()and its size to be written to a file for
//Trusted Positioning debugging
TppOutputStruct tpp_initialization_output_array_struct;
TppReturnStatusEnum tpp_return_status_enum;

tpp_initialization_output_array_struct.byte_array_pointer_ =
        (TppTypeUint8*)malloc(TPP_INIT_BYTE_ARRAY_LENGTH);

flags_counter = 0;

flags_struct_pointer =
(TppInitializationFlagStruct*) malloc(MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

memset(flags_struct_pointer, 0,MAXIMUM_NUMBER_OF_FLAGS * sizeof(TppInitializationFlagStruct));

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"BAROMETER");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"on");
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_2_,"1.0");
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"POSITION");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc (20);
flags_struct_pointer[flags_counter].flag_value_2_ = (char*) malloc (20);
flags_struct_pointer[flags_counter].flag_value_3_ = (char*) malloc (10);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.8f" , user_initial_latitude);
sprintf( flags_struct_pointer[flags_counter].flag_value_2_ , "%.8f" , user_initial_longitude);
sprintf( flags_struct_pointer[flags_counter].flag_value_3_ , "%.2f" , user_initial_height);
flags_counter++;

flags_struct_pointer[flags_counter].flag_name_ = (char*) malloc(40);
strcpy(flags_struct_pointer[flags_counter].flag_value_1_,"HEADING");
flags_struct_pointer[flags_counter].flag_value_1_ = (char*) malloc (20);
sprintf( flags_struct_pointer[flags_counter].flag_value_1_ , "%.8f" , user_initial_heading);
flags_counter++;


tpp_initialization_structure.number_of_initialization_flags_ = flags_counter;
tpp_initialization_struct.tpp_initialization_flag_pointer_   = flags_struct_pointer;

tpp_return_status = tpp_initialize_tvn( &tpp_initialization_struct ,
&tpp_initialization_output_array_struct);

if( tpp_return_status != TPP_SUCCESS )
{
    printf("[ERROR] tpp_initialize_tvn() failed\n");
}
```

8.2. Data Processing

For each sensor, there is a specific function that the developer can use to process the data from the corresponding sensor. Code Sample 11 shows how to set the members of a structure of type ***TppBarometerMessageStruct*** before passing a pointer of this structure to the function ***tpp_process_barometer()***. After calling the function ***tpp_process_barometer()***, the developer checks that the function returns ***TPP_SUCCESS*** which indicates that the call was successful. It is recommended to check on the return value when calling any of the API functions to ensure that the call is successful before proceeding.

Code Sample 11 Processing the Barometer Data

```
TppReturnStatusEnum tpp_return_status;

tpp_barometer_message.height                = user_barometer_height;
tpp_barometer_message.height_standard_deviation_     = sqrt(user_barometer_height_variance);


tpp_return_status = tpp_process_barometer(tpp_handle,&tpp_barometer_message );
if( tpp_return_status != TPP_SUCCESS )
{
    printf("[ERROR] tpp_process_barometer() failed\n");
}
```

The following code samples show how to use the library depending on the method by which the developer acquires the GNSS and sensors' data.

In event-based applications, the developer registers specific events and the corresponding event handlers to get the GNSS and sensors' data. When the data from any of the sensors is available, the developer will fill a variable of the corresponding sensor message and will pass it to the corresponding ***tpp_process_SSS()***function. Code Sample 12 shows an example of using the library in an event-based application.

Code Sample 12 Data processing using event handlers

```
user_imu_data_event_handler(event e)
{
    ...
    tpp_return_status =
        tpp_advance_navigation_step(tpp_handle, &tpp_imu_message , &solution_structure,
        &packet_structure);
    ...
}

user_gnss_data_event_handler(event e)
{
    ...
     tpp_return_status = tpp_process_gnss_pvt(tpp_handle, &tpp_gnss_pvt_message );
    ...
}

user_barometer_data_event_handler(event e)
{
    ...
     tpp_return_status = tpp_process_barometer(tpp_handle, &tpp_barometer_message );
    ...
}
```

The same applies for multi-threaded applications that do not use events to acquire the sensors data which is the case when multiple threads exist where data is acquired from one sensor in each thread.

If one of the threads acquires multiple sensors data at the same epoch, then the order of calling the processing functions matter where the ***tpp_advance_navigation_step()*** should be called last. This case is presented in Code Sample 13.

Code Sample 13 Data processing when multiple sensors' data is available at the same time

```
user_imu_barometer_thread(...)
{
    ...
    tpp_return_status = tpp_process_barometer(tpp_handle, &tpp_barometer_message );
    ...
    tpp_return_status =
        tpp_advance_navigation_step(tpp_handle, &tpp_imu_message , &solution_structure,
        &packet_structure);
    ...
}

user_gnss_data_thread(...)
{
    ...
    tpp_return_status = tpp_process_gnss_pvt(tpp_handle, &tpp_gnss_pvt_message );
    ...
}

user_magnetometer_data_thread(...)
{
    ...
    tpp_return_status = tpp_process_magnetometer(tpp_handle, &tpp_magnetometer_message );
    ...
}
```

8.3. Output Array Parsing

When the developer calls ***tpp_advance_navigation_step()*** and if ***TPP_SUCCESS*** is returned. The developer can expect a valid output byte array which holds the navigation solution according to the structure specified in Section 7.23.1 that will be used to parse the output byte array. First, the developer can check that the three header bytes and the checksum are valid. The output packet is structured in this way, so that the developer can use this structure directly to write it to a file or send it over a communication channel.

Once the developer validated the byte array, the developer can start with parsing the packet according to the packet structure in Section 7.23.1.

Instead of parsing the output byte array, the developer can use the output of type ***TppSolutionStruct*** which provides direct access to the nine states of the integrated navigation solution along with the platform heading.

9. Output Array Entities[3]

The Ids for the different entities that can be specified with the 'Output per Epoch' packet along with the entities' structures are defined below.

9.1. Time

Entity Id: **0x00EA**

	Field Name	Data Type	Unit	Description
1	timetag_	DOUBLE64	sec	

9.2. Position

Entity Id: **0x0016**

	Field Name	Data Type	Unit	Description
1	latitude_	DOUBLE64	deg	
2	longitude_	DOUBLE64	deg	
3	height_	FLOAT32	m	

9.3. Position Standard Deviation

Entity Id: **0x00BD**

	Field Name	Data Type	Unit	Description
1	position_north_standard_deviation_	FLOAT32	m	
2	position_east_standard_deviation_	FLOAT32	m	
3	height_standard_deviation_	FLOAT32	m	

9.4. Velocity

Entity Id: **0x00A2**

	Field Name	Data Type	Unit	Description
1	velocity_north_	FLOAT32	m/sec	
2	velocity_east_	FLOAT32	m/sec	
3	velocity_down_	FLOAT32	m/sec	

9.5. Velocity Standard Deviation

Entity Id: **0x005F**

	Field Name	Data Type	Unit	Description
1	velocity_north_standard_deviation_	FLOAT32	m/sec	
2	velocity_east_standard_deviation_	FLOAT32	m/sec	

[3] Entities with a star (*) next to their names correspond to entities that hold the input data and are mainly used for Trusted Positioning Debugging. Those entities can be generated by initializing the "DEBUG_DATA" flag accordingly.

3	velocity_down_standard_deviation_	FLOAT32	m/sec	

9.6. Attitude

Entity Id: **0x00E9**

	Field Name	Data Type	Unit	Description
1	roll_	FLOAT32	deg	
2	pitch_	FLOAT32	deg	
3	heading_	FLOAT32	deg	Device heading

9.7. Attitude Standard Deviation

Entity Id: **0x0060**

	Field Name	Data Type	Unit	Description
1	roll_standard_deviation_	FLOAT32	deg	
2	pitch_standard_deviation_	FLOAT32	deg	
3	heading_standard_deviation_	FLOAT32	deg	Device heading standard deviation

9.8. Accelerometer Bias

Entity Id: **0x006F**

	Field Name	Data Type	Unit	Description
1	bias_accelerometer_x_	FLOAT32	m/sec/sec	
2	bias_accelerometer_y_	FLOAT32	m/sec/sec	
3	bias_accelerometer_z_	FLOAT32	m/sec/sec	

9.9. Gyroscope Bias

Entity Id: **0x002E**

	Field Name	Data Type	Unit	Description
1	bias_gyroscope_x_	FLOAT32	deg/sec	
2	bias_gyroscope_y_	FLOAT32	deg/sec	
3	bias_gyroscope_z_	FLOAT32	deg/sec	

9.10. Heading Misalignment

Entity Id: **0x004D**

	Field Name	Data Type	Unit	Description
1	heading_misalignment_	FLOAT32	deg	

9.11. Platform Heading

Entity Id: **0x00B3**

	Field Name	Data Type	Unit	Description
1	platform_heading_	FLOAT32	deg	

9.12. Stride Information

Entity Id: **0x005E**

	Field Name	Data Type	Unit	Description
1	stride_distance_	FLOAT32	m	
2	stride_velocity_	FLOAT32	m/sec	

9.13. Number of Steps

Entity Id: **0x0099**

	Field Name	Data Type	Unit	Description
1	number_of_steps_	UINT32	N/A	Number of steps for the whole navigation session.

9.14. Floor Number

Entity Id: **0x00DD**

	Field Name	Data Type	Unit	Description
1	floor_number_	Unit16	N/A	

9.15. Converged Gyroscope Bias

Entity Id: **0x002F**

	Field Name	Data Type	Unit	Description
1	is_valid_	UINT8	N/A	0x00: Invalid bias values. The values should not be used. 0x01: Valid bias values. The values can be saved to be used later on in the initialization of the gyroscope bias values.
2	bias_gyroscope_x_	FLOAT32	deg/sec	
3	bias_gyroscope_y_	FLOAT32	deg/sec	
4	bias_gyroscope_z_	FLOAT32	deg/sec	

9.16. Flags - Magnetometer and Barometer

Entity Id: **0x0091**

	Field Name	Data Type	Unit	Description

1	flags_	UINT8	N/A	**Bit[0]**:Barometer Data Available Flag 0: Barometer data is not available from user 1: Barometer data is available from user **Bit[1]**:Magnetometer Data Available Flag 0: Magnetometer data is not available from user 1: Magnetometer data is available from user **Bit[2]**:Barometer Data In Use Flag 0: Barometer data is not used in navigator 1: Barometer data is used in navigator **Bit[3]**:Magnetometer In Use Flag 0: Magnetometer data is not used in navigator 1: Magnetometer data is used in navigator **Bit[7:4]**: Reserved

9.17. Mode of Transit

Entity Id: **0x00C8**

	Field Name	Data Type	Unit	Description
1	mode_	UINT8	N/A	Current mode of transit 0x00: Driving 0x01: Walking 0x02: Elevator 0x03: Stairs 0x04: Escalator walking 0x05: Escalator standing 0x06: Fidgeting 0x07: Conveyer walking 0x08: Conveyer standing 0x09: Running 0x0A: Cycling

9.18. Use Case

Entity Id: **0x00EE**

	Field Name	Data Type	Unit	Description
1	use_case_	UINT8	N/A	0x00: Automatic 0x01: Others 0x02: Pocket 0x03: Hand dangling

				0x04: Phone on ear ~~0x05: Purse~~

9.19. Flags – GNSS, Speed and Static Status

Entity Id: **0x002B**

	Field Name	Data Type	Unit	Description
1	flags_	UINT8	N/A	**Bit[0]**:Static Status Flag 0: In motion 1: Static **Bit[1]**:Multiple Antenna Available Flag 0:Multiple antenna data is not available from user 1: Multiple Antenna Data is available from user **Bit[2]**:GNSS/PVT Available Flag 0: GNSS/PVT data is not available from user 1: GNSS/PVT data is available from user **Bit[3]**:Multiple Antenna Data In Use Flag 0:Multiple antenna data is not used in navigator 1: Multiple antenna data is used in navigator **Bit[4]**:GNSS/PVT Data In Use Flag 0: GNSS/PVT data is not used in navigator 1: GNSS/PVT Data is used in navigator **Bit[5]**:GNSS Available for One Second Flag 0: GNSS is not available for the current second 1: GNSS is available for the current second **Bit[6]**:Speed Data Available Flag 0: Speed data is not available from user 1: Speed data is available from user **Bit[7]**:Speed Data In Use Flag 0: Speed data is not used in navigator 1: Speed data is used in navigator

9.20. Flags – Navigation Phase

Entity Id: **0x00E7**

	Field Name	Data Type	Unit	Description

1	navigation_phase_	INT8	N/A	-1: Pre-alignment 0: Alignment +1: Navigation/Available +2: Navigation/Reliable +3: Navigation/Vertical Alignment +4: Navigation/Drive to Walk

9.21. Orientation Based On Pitch

Entity Id: **0x00DF**

	Field Name	Data Type	Unit	Description
1	orientation_	INT8	N/A	-1: Vertical Down 0: Horizontal +1: Vertical Up

9.22. Distance Travelled

Entity Id: **0x0097**

	Field Name	Data Type	Unit	Description
1	distance_	FLOAT32	m	

9.23. DSL Height (For TPP/DSL only)

Entity Id: **0x00BC**

	Field Name	Data Type	Unit	Description
1	height_	FLOAT32	m	
2	height_standard_deviation_	FLOAT32	m	

9.24. ZUPT Thresholds

Entity Id: **0x006A**

	Field Name	Data Type	Unit	Description
1	zupt_threshold_x_	FLOAT32	deg/sec	Gyro x standard deviation during zupt
2	zupt_threshold_y_	FLOAT32	deg/sec	Gyro y standard deviation during zupt
3	zupt_threshold_z_	FLOAT32	deg/sec	Gyro z standard deviation during zupt

9.25. Heading Misalignment STD Output:

Entity Id: **0x004F**

	Field Name	Data Type	Unit	Description
1	misalignment_standard_deviation_	FLOAT32	deg	

9.26. Raw IMU Data*

This entity is present in the packet when the user passes data to ***tpp_advance_navigation_step()***

Entity Id: **0x00A1**

	Field Name	Data Type	Unit	Description
1	timetag_	DOUBLE64	sec	time-tag
2	data_gyroscope_x_	DOUBLE64	deg/sec	
3	data_gyroscope_y_	DOUBLE64	deg/sec	
4	data_gyroscope_z_	DOUBLE64	deg/sec	
5	data_accelerometer_x_	DOUBLE64	m/sec/sec	
6	data_accelerometer_y_	DOUBLE64	m/sec/sec	
7	data_accelerometer_z_	DOUBLE64	m/sec/sec	

9.27. GNSS PVT Data*

This entity is present in the packet when the user passes data to ***tpp_process_gnss_pvt()***

Entity Id: **0x001C**

	Field Name	Data Type	Unit	Description
1	gnss_timetag_	DOUBLE64	sec	An absolute time-tag that increments according to the rate of the GNSS data provided.
2	latitude_	DOUBLE64	deg	
3	longitude_	DOUBLE64	deg	
4	height_	FLOAT32	m	
5	velocity_north_	FLOAT32	m/sec	
6	velocity_east_	FLOAT32	m/sec	
7	velocity_down_	FLOAT32	m/sec	
8	position_north_standard_deviation_	FLOAT32	m	
9	position_east_standard_deviation_	FLOAT32	m	
10	height_standard_deviation_	FLOAT32	m	
11	velocity_north_standard_deviation_	FLOAT32	m/sec	
12	velocity_east_standard_deviation_	FLOAT32	m/sec	
13	velocity_down_standard_deviation_	FLOAT32	m/sec	
14	dop_data_available_	UINT8	N/A	0x00: DOP data is not available 0x01: DOP data is available
15	horizontal_dop_	FLOAT32	N/A	
16	vertical_dop_	FLOAT32	N/A	

9.28. Barometer Data*

This entity is present in the packet when the user passes data to ***tpp_process_barometer()***

Entity Id: **0x0083**

	Field Name	Data Type	Unit	Description
1	height_	FLOAT32	m	
2	height_standard_deviation_	FLOAT32	m	

9.29. Magnetometer Data*

This entity is present in the packet when the user passes data to ***tpp_process_magnetometer()***

Entity Id: **0x0062**

	Field Name	Data Type	Unit	Description
1	raw_data_available_	UINT8	N/A	0x00: raw data is not available 0x01: raw data is available
2	raw_data_x_	FLOAT32	mG	
3	raw_data_y_	FLOAT32	mG	
4	raw_data_z_	FLOAT32	mG	
5	raw_data_accuracy_flag_	UINT8	N/A	0x00: Flag not available 0x01: Invalid data 0x02: Valid data
6	calibrated_data_available_	UINT8	N/A	0x00: calibrated data is not available 0x01: calibrated data is available
7	calibrated_data_x_	FLOAT32	mG	
8	calibrated_data_y_	FLOAT32	mG	
9	calibrated_data_z_	FLOAT32	mG	
10	calibrated_data_accuracy_flag_	UINT8	N/A	0x00: Flag not available 0x01:Unreliable 0x02: Low Accuracy 0x03: Medium Accuracy 0x04: High Accuracy
11	calibration_status_changed_flag_	UINT8	N/A	Specifies if the status of the calibration changed when the magnetometer providing the calibrated data is reset or when the the calibrated signals are saturated. 0x00: Flag not available 0x01: No change in status 0x02: Calibration status changed
12	heading_available_	UINT8	N/A	0x00: heading data is not available 0x01: heading data is available
13	heading_	FLOAT32	deg	The heading is with respect to the True North (i.e. The heading is corrected with the magnetic declination angle) and not the Magnetic North.
14	heading_standard_deviation_	FLOAT32	deg	

9.30. Speed Data*

This entity is present in the packet when the user passes data to ***tpp_process_speed()***.

Entity Id: **0x00A7**

	Field Name	Data Type	Unit	Description
1	speed_	FLOAT32	m/sec	
2	reserved_1_	UINT8	N/A	
2	reverse_	UINT8	N/A	0x00: Unavailable 0x01: Forward 0x02: Reverse

9.31. Operator 2D Position Data*

This entity is present in the packet when the user passes data to ***tpp_process_2d_position()*** with the source being operator.

Entity Id: **0x0043**

	Field Name	Data Type	Unit	Description
1	latitude_	DOUBLE64	deg	
2	longitude_	DOUBLE64	deg	

9.32. Wireless Data*

This entity is present in the packet when the user passes data to ***tpp_process_wireless()*** with the source being Wi-Fi.

Entity Id: **0x0056**

	Member	Data Type	Unit	Description
1	source_	UINT8	N/A	0x01: Bluetooth 0x11: Wi-Fi Source (1) 0x12: Wi-Fi Source (2) 0x13: Wi-Fi Source (3)
2	latency_available_	UINT8	N/A	0x00: Latency and the corresponding standard deviation are not available. 0x01: Latency and the corresponding standard deviation are available.
3	latency_	FLOAT32	sec	Delay between the time the Wi-Fi 2D position is requested and the time the position is received and sent to the navigator. If there is no delay, the user can use zero.
4	latency_standard_deviation_	FLOAT32	sec	
5	position_2d_available_	UINT8	N/A	0x00: 2d position and the corresponding standard deviation are not available 0x01: 2d position and the corresponding standard deviation are available
6	latitude_	DOUBLE64	deg	
7	longitude_	DOUBLE64	deg	
8	position_north_standard_deviation_	FLOAT32	m	
9	position_east_standard_deviation_	FLOAT32	m	
10	height_available_	UINT8	N/A	0x00: Height and the corresponding standard deviation are not available 0x01: Height and the corresponding standard deviation are available
11	height_	FLOAT32	m	
12	height_standard_deviation_	FLOAT32	m	
13	velocity_2d_available_	UINT8	N/A	0x00: 2d velocity and the corresponding standard deviation are not available

				0x01: 2d velocity and the corresponding standard deviation are available
14	velocity_north_	FLOAT32	m/sec	
15	velocity_east_	FLOAT32	m/sec	
16	velocity_north_standard_deviation_	FLOAT32	m/sec	
17	velocity_east_standard_deviation_	FLOAT32	m/sec	
18	velocity_down_available_	UINT8	N/A	0x00: Down velocity and the corresponding standard deviation are not available 0x01: Down velocity and the corresponding standard deviation are available
19	velocity_down_	FLOAT32	m/sec	
20	velocity_down_standard_deviation_	FLOAT32	m/sec	
21	platform_heading_available_	UINT8	N/A	0x00: Platform heading and the corresponding standard deviation are not available 0x01: Platform heading and the corresponding standard deviation are available
22	platform_heading_	FLOAT32	deg	
23	platform_heading_standard_deviation_	FLOAT32	deg	
24	floor_information_available_	UINT8	N/A	0x00: Floor information is not available 0x01: Floor information is available
24	floor_number_	INT16	N/A	

9.33. Floor Information Data*

This entity is present in the packet when ***tpp_process_floor_information()*** is called.

Entity Id: **0x004A**

	Field Name	Data Type	Unit	Description
1	floor_number_	INT16	N/A	
2	height_between_floors_	FLOAT32	m	

9.34. Call Information Data*

This entity is present in the packet when ***tpp_process_call_information()*** is called.

Entity Id: **0x00AE**

	Field Name	Data Type	Unit	Description
1	is_call_active_	UINT8	N/A	0x00: Call Inactive 0x01: Call Active
2	is_proximity_detected_	UINT8	N/A	0x00: No Proximity Detected 0x01: Proximity Detected
3	is_speaker_active_	UINT8	N/A	0x00: Phone speaker Inactive 0x01: Phone speaker Active
4	headset_type_	UINT8	N/A	0x00: None 0x01: Wired or wireless earphone

9.35. Set Mode of Transit Event Data*

This entity is present in the packet when ***tpp_set_mode_of_transit()*** is called.

Entity Id: **0x001D**

	Field Name	Data Type	Unit	Description
1	mode_	UINT8	N/A	In-run change of user mode 0x00: Driving 0x01: Walking 0x02: Automatic 0x03: Running 0x04: Cycling

9.36. Platform Heading Data*

This entity is present in the packet when ***tpp_process_platform_heading()*** is called.

Entity Id: **0x00BA**

	Member	Data Type	Unit	Description
1	heading_	FLOAT32	deg	
2	heading_standard_deviation_	FLOAT32	deg	

9.37. Device Heading Data*

This entity is present in the packet when ***tpp_set_device_heading()*** is called.

Entity Id: **0x00BB**

	Member	Data Type	Unit	Description
1	device_heading_available_	UINT8	N/A	**0x00:** Device heading value is not available directly and will be computed from the **platform_heading_** and **misalignment_angle_** fields. **0x01**: Device heading value is available directly from the **device_heading_** field.
2	device_heading_	FLOAT32	deg	
3	platform_heading_	FLOAT32	deg	
4	misalignment_angle_	FLOAT32	deg	

9.38. 9-DOF Quaternions Data*

This entity is present in the packet when ***tpp_process_9dof_quaternions()*** is called.

Entity Id: **0x007A**

	Member	Data Type	Unit	Description
1	value_0_	FLOAT32	N/A	The scalar component of the quaternion.

2	value_1_	FLOAT32	N/A	The 1st vector component of the quaternion.
3	value_2_	FLOAT32	N/A	The 2nd vector component of the quaternion.
4	value_3_	FLOAT32	N/A	The 3rd vector component of the quaternion.

9.39. IMU Biases Data*

This entity is present in the packet when ***tpp_process_imu_biases()*** is called.

Entity Id: **0x0008**

	Member	Data Type	Unit	Description
1	bias_gyroscope_source_	UINT8	N/A	0: Biases Not Available (Discard values) **Other values to be defined by ISJ and ICA**
2	bias_gyroscope_accuracy_	FLOAT32	deg/sec	Accuracy for gyroscope biases
3	bias_gyroscope_x_	FLOAT32	deg/sec	Bias for gyroscope-x
4	bias_gyroscope_y_	FLOAT32	deg/sec	Bias for gyroscope-y
5	bias_gyroscope_z_	FLOAT32	deg/sec	Bias for gyroscope-z
6	bias_accelerometer_source_	UINT8	N/A	0: Biases Not Available (Discard values) **Other values to be defined by ISJ and ICA**
7	bias_accelerometer_accuracy_	FLOAT32	m/sec/sec	Accuracy for accelerometer biases
8	bias_accelerometer_x_	FLOAT32	m/sec/sec	Bias for accelerometer-x
9	bias_accelerometer_y_	FLOAT32	m/sec/sec	Bias for accelerometer-y
10	bias_ accelerometer_z_	FLOAT32	m/sec/sec	Bias for accelerometer-z

9.40. Process Misalignment Data*

This entity is present in the packet when ***tpp_process_misalignment()*** is called.

Entity Id: **0x00A6**

	Member	Data Type	Unit	Description
1	angle_	FLOAT32	deg	
2	angle_standard_deviation_	FLOAT32	deg	

9.41. Set ZUPT Mode Event Data*

This entity is present in the packet when ***tpp_set_zupt_mode()*** is called.

Entity Id: **0x009F**

	Member	Data Type	Unit	Description
1	zupt_mode_	UINT8	N/A	0x00: Automatic 0x01: On 0x02: Off

9.42. Set Misalignment Event Data*

This entity is present in the packet when ***tpp_set_misalignment()*** is called.

Entity Id: **0x00A4**

	Member	Data Type	Unit	Description
1	estimation_on_	UINT8	N/A	0x00: Automatic misalignment estimation turned off 0x01: Automatic misalignment estimation turned on
2	angle_available_	UINT8	N/A	0x00: Misalignment angle is not available 0x01: Misalignment angle is available
3	angle_	FLOAT32	deg	

9.43. Set Magnetometer Calibration Status Event Data*

This entity is present in the packet when ***tpp_set_magnetometer_calibration()*** is called.

Entity Id: **0x00E4**

	Member	Data Type	Unit	Description
1	magnetometer_calibration_	UINT8	N/A	0x00: calibration disabled 0x01: calibration enabled

9.44. Set Use Case Event Data*

This entity is present in the packet when ***tpp_set_use_case()*** is called.

Entity Id: **0x0098**

	Member	Data Type	Unit	Description
1	use_case_	UINT8	N/A	0x00: Automatic 0x01: Torso and hand viewing 0x02: Pocket 0x03: Hand swinging 0x04: Arm ~~0x05: Purse~~

9.45. 6-DOF Quaternions Data*

This entity is present in the packet when ***tpp_process_6dof_quaternions()*** is called.

Entity Id: **0x007B**

	Member	Data Type	Unit	Description
1	value_0_	FLOAT32	N/A	The scalar component of the quaternion.
2	value_1_	FLOAT32	N/A	The 1st vector component of the quaternion.
3	value_2_	FLOAT32	N/A	The 2nd vector component of the

				quaternion.
4	value_3_	FLOAT32	N/A	The 3rd vector component of the quaternion.

9.46. Venue Map Data*

This entity is present in the packet when ***tpp_process_map_information()*** is called.

Entity Id: **0x004B**

	Member	Data Type	Unit	Description
1	position_2d_available_	UINT8	N/A	0x00: 2d position and the corresponding standard deviation are **not available** 0x01: 2d position and the corresponding standard deviation are **available but not reliable** 0x02: 2d position and the corresponding standard deviation are **available and reliable**
2	latitude_	DOUBLE64	deg	
3	longitude_	DOUBLE64	deg	
4	position_north_standard_deviation_	FLOAT32	m	
5	position_east_standard_deviation_	FLOAT32	m	
6	height_available_	UINT8	N/A	0x00: Height and the corresponding standard deviation are not available 0x01: Height and the corresponding standard deviation are available
7	height_	FLOAT32	m	
8	height_standard_deviation_	FLOAT32	m	
9	platform_heading_available_	UINT8	N/A	0x00: Platform heading and the corresponding standard deviation are **not available** 0x01: Platform heading and the corresponding standard deviation are **available but not reliable** 0x01: Platform heading and the corresponding standard deviation are **available and reliable**
10	platform_heading_	FLOAT32	deg	
11	platform_heading_standard_deviation_	FLOAT32	deg	
12	map_entity_available_	UINT8	N/A	0x00: Map entity is not available 0x01: Map entity is available
13	map_entity_	UINT16	N/A	Specifies an entity in the map. Examples for map entities are stairs, elevators, escalators, types of rooms, and others.
15	step_length_scale_available_	UINT8	N/A	Flag to notify if both fields (#16) Step Length Scale and (#17) Use case for Step length scale are available or not. 0x00: Fields are not available 0x01: Fields are available
16	step_length_scale_	FLOAT32	N/A	
17	use_case_for_step_length_scale_	UINT8	N/A	
18	mode_of_motion_for_step_length_scale_	UINT8	N/A	

9.47. Anchor Point Data*

This entity is present in the packet when ***tpp_add_anchor_point()*** is called.

Entity Id: **0x00FE**

	Member	Data Type	Unit	Description
1	type_	UINT8	N/A	Anchor Point Type. **0x01**: Start Anchor Point. **0x02**: End Anchor Point.
2	position_available_	UNIT8	N/A	
3	latitude_	DOUBLE64	deg	
4	longitude_	DOUBLE64	deg	
5	height_	FLOAT32	m	
6	heading_available_	UNIT8	N/A	
7	heading_	FLOAT32	deg	

9.48. Synchronization Event Data*

This entity is present in the packet when ***tpp_add_synchronization_event()*** is called.

Entity Id: **0x00FF**

	Member	Data Type	Unit	Description
1	event_number_	UINT64	N/A	

9.49. Orientation Based On Pitch Data*

This entity is present in the packet when ***tpp_set_orientation_based_on_pitch()*** is called.

Entity Id: **0x00FD**

	Member	Data Type	Unit	Description
1	orientation_	INT8	N/A	

9.50. Street Map Message*

This entity is present in the packet when the user passes data to ***tpp_process_street_map()***.

Entity Id: **0x0026**

	Member	Data Type	Unit	Description
1	time_	DOUBLE64	s	
2	latitude_	DOUBLE64	deg	
3	longitude_	DOUBLE64	deg	
4	platform_heading_available_	UINT8	N/A	0x00: Platform heading and the corresponding standard deviation are not available 0x01: Platform heading and the corresponding standard deviation are available 0x02: Platform heading is available

				but not the standard deviation
5	platform_heading_	FLOAT32	deg	Measure clockwise with respect to the true north
6	platform_heading_standard_deviation_	FLOAT32	deg	
7	postion_2d_standard_available_	UINT8		0x00: Horizontal position standard deviations are not available 0x01: Horizontal position standard deviations are available
8	position_north_standard_deviation_	FLOAT32	m	
9	position_east_standard_deviation_	FLOAT32	m	
10	height_available_	UINT8	N/A	0x00: Height and the corresponding standard deviation are not available 0x01: Height and the corresponding standard deviation are available 0x02: Height is available but the corresponding standard deviation is unavailable
11	height_	FLOAT32	m	
12	height_standard_deviation_	FLOAT32	m	
13	max_speed_available_	UINT8	N/A	0x00: Maximum speed for the segment of the road is unavailable 0x01: Maximum speed for the segment of the road is available
14	max_speed_	FLOAT32	m/s	

9.51. Vehicle Frame Message*

This entity provides acceleration and rotation rates in vehicle frame.

Entity Id: **0x0017**

	Member	Data Type	Unit	Description
1	leveled_gyro_x_	FLOAT32	deg/sec	
2	leveled_gyro_y_	FLOAT32	deg/sec	
3	leveled_gyro_z_	FLOAT32	deg/sec	
4	leveled_gyro_status_	UINT8	N/A	0x00: leveled gyroscope measurement not present 0x01: leveled gyroscope measurement present but not used 0x02: leveled gyroscope measurement present and used
5	leveled_accel_x_	FLOAT32	m/sec^2	
6	leveled_accel_y_	FLOAT32	m/sec^2	
7	leveled_accel_z_	FLOAT32	m/sec^2	
8	leveled_accel_status_	UINT8	N/A	0x00: leveled accelerometer measurement not present 0x01: leveled accelerometer measurement present but not used 0x02: leveled accelerometer measurement present and used

9.52. Multi-Device Position Velocity Data*

When ***tpp_multi_device_process_position_velocity()*** is called, the data passed to this function is constructed in the form of one common entity and 2 or more data entities. When parsing the

packet the user can check the availability of the position and velocity common entity first. If it exists, the user can parse the number of devices. The number of position and velocity data entities that will exist in the packet is equal to the number of devices.

9.52.1. *Common Entity*

Entity Id: **0x0101**

	Member	Data Type	Unit	Description
1	number_of_devices_	UINT8	N/A	
2	latency_available_	UINT8	N/A	0x00: Latency is not available. 0x01: Latency is available
3	latency_	FLOAT32	sec	Delay between the time the information is computed on the sender device and the time the information is processed on the receiver device. If there is no delay, the user can use zero.

9.52.2. *Data per Device Entity*

Entity Id: **0x0102**

	Member	Data Type	Unit	Description
1	device_identifier_	UINT64	N/A	A unique identifier for the device that is sending this information. When sending the device its own information, the value of the **device_identifier_** should be **zero**.
2	device_type_	UINT8	N/A	**0**: Phone **1**: Tablet **2**: Head-Mount **3**: Watch
3	navigation_phase_	INT8	N/A	Equivalent to the values taken by the flag **navigation_phase_** in TppSolutionStruct
4	gnss_in_use_	UINT8	N/A	0x00: GNSS data was not used in computing the navigation solution within the last 2 seconds. 0x01: GNSS data was used in computing the navigation solution within the last 2 seconds.
5	magnetometer_in_use_	UINT8	N/A	0x00: Magnetometer data was not used in computing the navigation solution within the last 2 seconds. 0x01: Magnetometer data was used in computing the navigation solution within the last 2 seconds.
6	barometer_in_use_	UINT8	N/A	0x00: Barometer data was not used in computing the navigation solution within the last 2 seconds.

				0x01: Barometer data was used in computing the navigation solution within the last 2 seconds.
7	speed_in_use_	UINT8	N/A	0x00: Speed data was not used in computing the navigation solution within the last 2 seconds. 0x01: Speed data was used in computing the navigation solution within the last 2 seconds.
8	use_case_	UINT8	N/A	
9	mode_of_transit_	UINT8	N/A	
10	latitude_	DOUBLE64	deg	
11	longitude_	DOUBLE64	deg	
12	height_	FLOAT32	m	
13	velocity_north_	FLOAT32	m/sec	
14	velocity_east_	FLOAT32	m/sec	
15	velocity_down_	FLOAT32	m/sec	
16	position_north_standard_deviation_	FLOAT32	m	
17	position_east_standard_deviation_	FLOAT32	m	
18	height_standard_deviation_	FLOAT32	m	
19	velocity_north_standard_deviation_	FLOAT32	m/sec	
20	velocity_east_standard_deviation_	FLOAT32	m/sec	
21	velocity_down_standard_deviation_	FLOAT32	m/sec	
22	number_of_steps_	UINT32	N/A	

9.53. Multi-Device Heading Data*

When ***tpp_multi_device_process_heading()*** is called, the data passed to this function is constructed in the form of one common entity and 2 or more data entities. When parsing the packet the user can check the availability of the heading common entity first. If it exists, the user can parse the number of devices. The number of heading data entities that will exist in the packet is equal to the number of devices.

9.53.1. *Common Entity*

Entity Id: **0x0103**

	Member	Data Type	Unit	Description
1	number_of_devices_	UINT8	N/A	
2	latency_available_	UINT8	N/A	0x00: Latency is not available. 0x01: Latency is available
3	latency_	FLOAT32	sec	Delay between the time the information is computed on the sender device and the time the information is processed on the receiver device. If there is no delay, the user can use zero.

9.53.2. *Data per Device Entity*

Entity Id: **0x0104**

	Member	Data Type	Unit	Description
1	device_identifier_	UINT64	N/A	A unique identifier for the device that is sending this information. When sending the device its own information, the value of the **device_identifier_** should be **zero**.
2	device_type_	UINT8	N/A	**0**: Phone **1**: Tablet **2**: Head-Mount **3**: Watch
3	navigation_phase_	INT8	N/A	Equivalent to the values taken by the flag **navigation_phase_** in TppSolutionStruct
4	gnss_in_use_	UINT8	N/A	0x00: GNSS data was not used in computing the navigation solution within the last 2 seconds. 0x01: GNSS data was used in computing the navigation solution within the last 2 seconds.
5	magnetometer_in_use_	UINT8	N/A	0x00: Magnetometer data was not used in computing the navigation solution within the last 2 seconds. 0x01: Magnetometer data was used in computing the navigation solution within the last 2 seconds.
6	barometer_in_use_	UINT8	N/A	0x00: Barometer data was not used in computing the navigation solution within the last 2 seconds. 0x01: Barometer data was used in computing the navigation solution within the last 2 seconds.
7	speed_in_use_	UINT8	N/A	0x00: Speed data was not used in computing the navigation solution within the last 2 seconds. 0x01: Speed data was used in computing the navigation solution within the last 2 seconds.
8	use_case_	UINT8	N/A	
9	mode_of_transit_	UINT8	N/A	
10	device_heading_	FLOAT32	deg	
11	platform_heading_	FLOAT32	deg	
12	heading_misalignment_	FLOAT32	deg	
13	heading_standard_deviation_	FLOAT32	deg	
14	position_north_standard_deviation	FLOAT32	m	
15	position_east_standard_deviation_	FLOAT32	m	
16	number_of_steps_	UINT32	N/A	

9.54. Multi-Device Secondary GNSS PVT Data*

This entity is present in the packet when the user passes data to
tpp_multi_device_process_secondary_gnss_pvt()

Entity Id: **0x0105**

	Field Name	Data Type	Unit	Description
1	gnss_timetag_	DOUBLE64	sec	An absolute time-tag that increments according to the rate of the GNSS data provided.
2	latitude_	DOUBLE64	deg	
3	longitude_	DOUBLE64	deg	
4	height_	FLOAT32	m	
5	velocity_north_	FLOAT32	m/sec	
6	velocity_east_	FLOAT32	m/sec	
7	velocity_down_	FLOAT32	m/sec	
8	position_north_standard_deviation_	FLOAT32	m	
9	position_east_standard_deviation_	FLOAT32	m	
10	height_standard_deviation_	FLOAT32	m	
11	velocity_north_standard_deviation_	FLOAT32	m/sec	
12	velocity_east_standard_deviation_	FLOAT32	m/sec	
13	velocity_down_standard_deviation_	FLOAT32	m/sec	
14	dop_data_available_	UINT8	N/A	0x00: DOP data is not available 0x01: DOP data is available
15	horizontal_dop_	FLOAT32	N/A	
16	vertical_dop_	FLOAT32	N/A	
17	secondary_device_navigation_phase_	INT8	N/A	Equivalent to the values taken by the flag **navigation_phase_** in TppSolutionStruct

9.55. Multi-Device GNSS PVT Data Source*

This entity is present in the packet to indicate the source of the GNSS PVT data when more than one source is available. The source can be the device's own data (i.e. main device data which can be referenced from the entity in Section 9.27) or other device's data (i.e. secondary device data, such as the master device in a multi-device system, which can be referenced from the entity in Section 9.54)

Entity Id: **0x0106**

	Field Name	Data Type	Unit	Description
1	data_source_	UINT8	N/A	0x00: Secondary Device 0x01: Main Device

9.56. In-run Magnetometer Calibration Information^ [Internal Debugging Only]

Entity Id: **0x000F**

	Field Name	Data Type	Unit	Description
1	heading_	FLOAT32	deg	The heading derived from the automatic magnetometer calibration algorithms.
2	used_status_	UINT8	N/A	
3	calibration_type_	UINT8	N/A	Flag to indicate whether 2D or 3D calibration is used. 0x00 : No Calibration 0x01 : 2D

				0x02 : 3D 0x03 : 3D gravity
4	used_signals_type_	UINT8	N/A	Flag to indicate whether the raw or calibrated magnetometer signals are used in the calibration. 0x00: raw data 0x01: calibrated data
5	collection_completed_2d_	UINT8	N/A	Flag to indicate if 2D collection is completed. 0x00: Not Completed 0x01: Completed
6	collection_completed_3d_	UINT8	N/A	Flag to indicate if 3D collection is completed. 0x00: Not Completed 0x01: Completed
7	collection_completed_gravity_3d_	UINT8	N/A	Flag to indicate if 3D gravity collection is completed. 0x00: Not Completed 0x01: Completed
8	post_calibration_checks_failure_2d_	UINT8	N/A	Flag to indicate that the 2D post-calibration checks failed. 0x00: No Failure 0x01:Scale Factor check failed 0x02:Heading check failed 0x03:Signal check failed
9	post_calibration_checks_failure_3d_	UINT8	N/A	Flag to indicate that the 3D post-calibration check failed. 0x00: No Failure 0x01:Scale Factor Check Failed 0x02:Heading Check Failed 0x03:Signal Check Failed
10	post_calibration_checks_failure_gravity_3d_	UINT8	N/A	Flag to indicate that the 3D gravity post-calibration check failed. 0x00: No Failure 0x01:Scale Factor Check Failed 0x02:Heading Check Failed 0x03:Signal Check Failed
11	periodic_checks_failure_2d_	UINT8	N/A	Flag to indicate that the 2D periodic check failed. 0x00: No Failure 0x01: Signal Check Failed
12	periodic_checks_failure_3d_	UINT8	N/A	Flag to indicate that the 3D periodic check failed. 0x00: No Failure 0x01: Signal Check Failed
13	periodic_checks_failure_gravity_3d_	UINT8	N/A	Flag to indicate that the 3D gravity periodic check failed. 0x00: No Failure 0x01: Signal Check Failed
14	mag_bias_x_	FLOAT32	mG	
15	mag_bias_y_	FLOAT32	mG	

16	mag_bias_z_	FLOAT32	mG	
17	mag_scale_factor_x_	FLOAT32	N/A	
18	mag_scale_factor_y_	FLOAT32	N/A	
19	mag_scale_factor _z_	FLOAT32	N/A	
20	pitch_roll_sector_id_	INT16	N/A	Pitch-roll sector ID
21	pitch_defining_angle_for_sector_	FLOAT32	deg	Pitch defining angle for pitch-roll sector
22	roll_defining_angle_for_sector_	FLOAT32	deg	Roll defining angle for pitch-roll sector

9.57. Internal Debugging Information (1)^ [Internal Debugging Only]

Entity Id: **0x008C**

	Field Name	Data Type	Unit	Description
1	imu_sample_processing_time_	FLOAT32	µSec	The time taken, in micro-seconds, by the core main function in the navigation algorithm to compute the integrated navigation solution.
2	radius_of_rotation_	DOUBLE64	m	
3	radius_of_rotation_x_	DOUBLE64	m	
4	radius_of_rotation_y_	DOUBLE64	m	
5	radius_of_rotation_z_	DOUBLE64	m	
6	orientation_based_on_pitch_	INT8	N/A	
7	automatic_ear_flag_	UINT8	N/A	
8	sensors_ear_flag_	UINT8	N/A	
9	walking_fidgeting_flag_	UINT8	N/A	
10	height_changing_modes_	INT16	N/A	
11	children_info_	UINT8	N/A	
12	misalignement_driving_gps_	FLOAT32	deg	
13	misalignement_driving_radius_	FLOAT32	deg	
14	misalignement_driving_kalman_filter _	FLOAT32	deg	
15	misalignment_driving_sensors_	FLOAT32	deg	
16	misalignment_driving_sensors_comb ined_	FLOAT32	deg	
17	misalignment_driving_children_	FLOAT32	deg	
18	misalignment _driving_sensors_align_average_	FLOAT32	deg	
19	misalignment_driving_sensors_ratio_	FLOAT32	N/A	
20	misalignment_driving_sensors_count _	INT16	N/A	
21	misalignment_walking_pca_p1_	FLOAT32	deg	
22	misalignment_walking_pca_p2_	FLOAT32	deg	
23	misalginment_walking_outdoors_	FLOAT32	deg	
24	misalginment_walking_averaged_	FLOAT32	deg	
25	misalginment_walking_snap_	FLOAT32	deg	
26	primary_device_heading_	FLOAT32	deg	
27	roll_misalignment_	FLOAT32	deg	
28	pitch_misalignment_	FLOAT32	deg	
29	internal_roll_	DOUBLE64	deg	
30	internal_pitch_	DOUBLE64	deg	
31	accelerometer_roll_	DOUBLE64	deg	
32	accelerometer_pitch_	DOUBLE64	deg	
33	leveled_data_gyroscope_x_	DOUBLE64	deg/sec	
34	leveled_data_gyroscope_y_	DOUBLE64	deg/sec	

35	leveled_data_gyroscope_z_	DOUBLE64	deg/sec	
36	leveled_data_accelerometer_x_	FLOAT32	m/sec/sec	
37	leveled_data_accelerometer_y_	FLOAT32	m/sec/sec	
38	leveled_data_accelerometer_z_	FLOAT32	m/sec/sec	
39	magnitude_compensated_gyroscope_	DOUBLE64	deg/sec	
40	vertical_speed_from_height_	DOUBLE64	m/sec	
41	step_time_	DOUBLE64	sec	
42	step_frequency_	DOUBLE64	1/sec	
43	acceleration_variance_per_step_	DOUBLE64	m*m/sec/sec/sec/sec	
44	step_leveled_accelerometer_value_	DOUBLE64	m/sec/sec	
45	leveled_smoothed_vertical_accelerometer_data_	FLOAT32	m/sec/sec	
46	leveled_smoothed_vertical_acceleration_	FLOAT32	m/sec/sec	
47	pdr_latitude_	DOUBLE64	deg	
48	pdr_longitude_	DOUBLE64	deg	
49	attitude_filter_roll_	FLOAT32	deg	
50	attitude_filter_ptich_	FLOAT32	deg	
51	attitude_filter_heading_	FLOAT32	deg	
52	attitude_filter_roll_standard_deviation_	FLOAT32	deg	
53	attitude_filter_ptich_ standard_deviation_	FLOAT32	deg	
54	attitude_filter_heading_ standard_deviation_	FLOAT32	deg	
55	attitude_filter_bias_gyroscope_x_	FLOAT32	deg/sec	
56	attitude_filter_bias_gyroscope_y_	FLOAT32	deg/sec	
57	attitude_filter_bias_gyroscope_z_	FLOAT32	deg/sec	
58	attitude_filter_bias_standard_deviation_gyroscope_x_	FLOAT32	deg/sec	
59	attitude_filter_bias_standard_deviation_gyroscope_y_	FLOAT32	deg/sec	
60	attitude_filter_bias_standard_deviation_gyroscope_z_	FLOAT32	deg/sec	
61	declination_angle_	FLOAT32	deg	
62	magnetometer_derived_heading_from_calibrated_	FLOAT32	deg	
63	magnetometer_heading_used_	FLOAT32	deg	
64	magnetometer_heading_source_	UINT8	N/A	0x00: Not Available 0x01: External heading available in the magnetometer input data. 0x02:Heading derived from quaternions 0x03: Heading derived from calibrated data. 0x04: Heading derived from in-run magnetometer calibration.
65	barometer_filter_height_	FLOAT32	m	
66	barometer_filter_height_standard_deviation_	FLOAT32	m	
67	quaternions_derived_roll_	FLOAT32	deg	
68	quaternions_derived_pitch_	FLOAT32	deg	
69	quaternions_derived_heading_	FLOAT32	deg	
70	secondary_device_roll_	FLOAT32	deg	
71	secondary_device_pitch_	FLOAT32	deg	

72	Secondary_ bias_gyroscope_x_	FLOAT32	deg/sec	
73	Secondary_ bias_gyroscope_y_	FLOAT32	deg/sec	
74	Secondary_ bias_gyroscope_z_	FLOAT32	deg/sec	
75	Secondary_ bias_accelerometer_x_	FLOAT32	m/sec/sec	
76	Secondary_ bias_accelerometer _y_	FLOAT32	m/sec/sec	
77	Secondary_ bias_accelerometer _z_	FLOAT32	m/sec/sec	
78	step_distance_sensors_	FLOAT32	m	
79	step_speed_sensors_	FLOAT32	m/sec	
80	step_distance_gnss_	FLOAT32	m	
81	step_speed_gnss_	FLOAT32	m/sec	

10. DEPRECATED ENTITIES

10.1. Magnetometer Data* [Deprecated – As of version 3.0.0]

This entity is present in the packet when the user passes data to ***tpp_process_magnetometer()***

Entity Id: **0x00B7**

	Field Name	Data Type	Unit	Description
1	raw_data_available_	UINT8	N/A	0x00: raw data is not available 0x01: raw data is available
2	raw_data_x_	FLOAT32	mG	
3	raw_data_y_	FLOAT32	mG	
4	raw_data_z_	FLOAT32	mG	
5	raw_data_accuracy_flag_	INT8	N/A	0x00: Flag not available 0x01: Invalid 0x02: Valid
6	calibrated_data_available_	UINT8	N/A	0x00: calibrated data is not available 0x01: calibrated data is available
7	calibrated_data_x_	FLOAT32	mG	
8	calibrated_data_y_	FLOAT32	mG	
9	calibrated_data_z_	FLOAT32	mG	
10	calibrated_data_accuracy_flag_	UINT8	N/A	0x00: Flag not available 0x01:Unreliable 0x02: Low Accuracy 0x03: Medium Accuracy 0x04: High Accuracy
11	heading_available_	UINT8	N/A	0x00: heading data is not available 0x01: heading data is available
12	heading_	FLOAT32	deg	
13	heading_standard_deviation_	FLOAT32	deg	

10.2. Wi-Fi 2D Position Data* [Deprecated – As of version 4.1.0]

This entity is present in the packet when the user passes data to ***tpp_process_2d_position()*** with the source being Wi-Fi.

Entity Id: **0x0051**

	Field Name	Data Type	Unit	Description
1	latitude_	DOUBLE64	deg	
2	longitude_	DOUBLE64	deg	
3	position_north_standard_deviation_	FLOAT32	m	
4	position_east_standard_deviation_	FLOAT32	m	

10.3. Operator 2D Position Data* [Deprecated as of version 4.1.0]

This entity is present in the packet when the user passes data to ***tpp_process_2d_position()*** with the source being operator.

Entity Id: **0x0042**

	Field Name	Data Type	Unit	Description
1	latitude_	DOUBLE64	deg	
2	longitude_	DOUBLE64	deg	
3	position_north_standard_deviation_	FLOAT32	m	
4	position_east_standard_deviation_	FLOAT32	m	

10.4. Wi-Fi Data* [Deprecated]

This entity is present in the packet when the user passes data to ***tpp_process_wifi()*** with the source being Wi-Fi.

Entity Id: **0x0055**

	Member	Data Type	Unit	Description
1	latency_available_	UINT8	N/A	0x00: Latency and the corresponding standard deviation are not available. 0x01: Latency and the corresponding standard deviation are available.
2	latency_	FLOAT32	sec	Delay between the time the Wi-Fi 2D position is requested and the time the position is received and sent to the navigator. If there is no delay, the user can use zero.
3	latency_standard_deviation_	FLOAT32	sec	
4	position_2d_available_	UINT8	N/A	0x00: 2d position and the corresponding standard deviation are not available 0x01: 2d position and the corresponding standard deviation are available
5	latitude_	DOUBLE64	deg	
6	longitude_	DOUBLE64	deg	
7	position_north_standard_deviation_	FLOAT32	m	
8	position_east_standard_deviation_	FLOAT32	m	
9	height_available_	UINT8	N/A	0x00: Height and the corresponding standard deviation are not available 0x01: Height and the corresponding standard deviation are available
10	height_	FLOAT32	m	
11	height_standard_deviation_	FLOAT32	m	
12	velocity_2d_available_	UINT8	N/A	0x00: 2d velocity and the corresponding standard deviation are not available 0x01: 2d velocity and the corresponding standard deviation are available
13	velocity_north_	FLOAT32	m/sec	
14	velocity_east_	FLOAT32	m/sec	
15	velocity_north_standard_deviation_	FLOAT32	m/sec	
16	velocity_east_standard_deviation_	FLOAT32	m/sec	
17	velocity_down_available_	UINT8	N/A	0x00: Down velocity and the corresponding standard deviation are not available 0x01: Down velocity and the

				corresponding standard deviation are available
18	velocity_down_	FLOAT32	m/sec	
19	velocity_down_standard_deviation_	FLOAT32	m/sec	
20	platform_heading_available_	UINT8	N/A	0x00: Platform heading and the corresponding standard deviation are not available 0x01: Platform heading and the corresponding standard deviation are available
21	platform_heading_	FLOAT32	deg	
22	platform_heading_standard_deviation_	FLOAT32	deg	
23	floor_information_available_	UINT8	N/A	0x00: Floor information is not available 0x01: Floor information is available
24	floor_number_	INT16	N/A	

11. SUGGESTED ENTITIES

11.1. Accelerometer Bias Standard Deviation

Entity Id: **0x0054**

	Field Name	Data Type	Unit	Description
1	bias_accelerometer_standard_deviation_x_	FLOAT32	m/sec/sec	
2	bias_accelerometer_standard_deviation_y_	FLOAT32	m/sec/sec	
3	bias_accelerometer_standard_deviation_z_	FLOAT32	m/sec/sec	

11.2. Gyroscope Bias STD Outputs

Entity Id: **0x007E**

	Field Name	Data Type	Unit	Description
1	BiasGyroXStdv	FLOAT32	deg/sec	
2	BiasGyroYStdv	FLOAT32	deg/sec	
3	BiasGyroZStdv	FLOAT32	deg/sec	

11.3. Corrected Accel Outputs:

Entity Id: **0x00E3**

	Field Name	Data Type	Unit	Description
1	CorrectedAccX	DOUBLE64	m/sec/sec	
2	CorrectedAccY	DOUBLE64	m/sec/sec	
3	CorrectedAccZ	DOUBLE64	m/sec/sec	

11.4. Corrected Gyro Outputs:

Entity Id: **0x00BE**

	Field Name	Data Type	Unit	Description
1	CorrectedGyroX	DOUBLE64	deg/sec	
2	CorrectedGyroY	DOUBLE64	deg/sec	
3	CorrectedGyroZ	DOUBLE64	deg/sec	

11.5. Declination Angle

Entity Id: **0x0050**

	Field Name	Data Type	Unit	Description

1	declination_angle_	FLOAT32	deg	

11.6. Mag Pre-Calibration Outputs:

Entity Id: **0x00DE**

	Field Name	DataType	Unit	Description
1	CalibrationType	UINT8	N/A	
2	MagBiasX	DOUBLE64	mG	
3	MagBiasY	DOUBLE64	mG	
4	MagBiasZ	DOUBLE64	mG	
5	MagSfX	DOUBLE64	N/A	
6	MagSfY	DOUBLE64	N/A	
7	MagSfZ	DOUBLE64	N/A	

11.7. Mag Online Calibration Outputs:

Entity Id: **0x00B1**

	Field Name	Data Type	Unit	Description
1	CalibrationType	UINT8	N/A	
2	MagBiasX	DOUBLE64	mG	
3	MagBiasY	DOUBLE64	mG	
4	MagBiasZ	DOUBLE64	mG	
5	MagSfX	DOUBLE64	N/A	
6	MagSfY	DOUBLE64	N/A	
7	MagSfZ	DOUBLE64	N/A	

11.8. Magnetometer Derived Heading

Entity Id: **0x0077**

	Field Name	Data Type	Unit	Description
1	magnetometer_heading_	FLOAT32	deg	

11.9. Corrected Mag Outputs:

Entity Id: **0x0025**

	Field Name	Data Type	Unit	Description
1	CorrectedMagX	DOUBLE64	mG	
2	CorrectedMagY	DOUBLE64	mG	
3	CorrectedMagZ	DOUBLE64	mG	

11.10. Barometer Height

Entity Id: **0x00E0**

	Field Name	Data Type	Unit	Description
1	barometer_filter_height_	FLOAT32	m	

2	barometer_filter_height_standard_deviation_	FLOAT32	m	

11.11. Odometer Scale Factor:

Entity Id: **0x0030**

	Field Name	Data Type	Unit	Description
1	OdomF	DOUBLE64	N/A	
2	OdomFStdv	DOUBLE64	N/A	

11.12. Corrected Odometer Speed:

Entity Id: **0x00C9**

	Field Name	Data Type	Unit	Description
1	CorrectedOdSpeed	DOUBLE64	m/sec	

11.13. Corrected odometer Speed STD:

Entity Id: **0x008A**

	Field Name	Data Type	Unit	Description
1	CorrectedOdSpeedStdv	DOUBLE64	m/sec	

11.14. Roll Misalignment Outputs:

Entity Id: **0x003D**

	Field Name	Data Type	Unit	Description
1	RollMisalignment	DOUBLE64	deg	

11.15. Roll Misalignment STD Output:

Entity Id: **0x009A**

	Field Name	Data Type	Unit	Description
1	RollMisalignmentStdv	DOUBLE64	deg	

11.16. Pitch Misalignment Output:

Entity Id: **0x004E**

	Field Name	Data Type	Unit	Description
1	PitchMisalignment	DOUBLE64	deg	

11.17. Pitch Misalignment STD Output:

Entity Id: **0x000A**

	Field Name	Data Type	Unit	Description
1	PitchMisalignmentStdv	DOUBLE64	deg	

11.18. Resource Consumption

Entity Id: **0x00D5**

	Field Name	Data Type	Unit	Description
1	prediction_time_	DOUBLE64	milli-sec	
2	update_time_	DOUBLE64	milli-sec	
3	average_epoch_time_	DOUBLE64	milli-sec	
4	cpu_percentage_	UINT8	N/A	
5	battery_level_	UINT8	N/A	
6	**RESERVED**	DOUBLE64	N/A	

11.19. Flags - Events

Entity Id: **0x0004**

	Field Name	Data Type	Unit	Description
1	flags_	UINT8	N/A	**Bit[2:0]**: Alignment Info (3 bits) **Bit[3]**: User Event Flag (1 bit) **Bit[4]**: ~~Call On Flag (1 bit)~~ **Bit[5]**: ~~Proximity Sensor Reading (1 bit)~~ **Bit[7:6]**:~~Headset Reading (2 bits)~~

*"Alignment info" indicates the Child Filters

11.20. Flags – External Position

Entity Id: **0x00C0**

	Field Name	Data Type	Unit	Description
1	flags_	UINT8	N/A	**Bit[0]**: WiFi Valid Flag (1 bit) **Bit[1]**: WiFi In Use Flag (1 bit) **Bit[2]**: User Position Valid Flag (1 bit) **Bit[3]**: User Position In Use Flag (1 bit) **Bit[4]**: External Wireless Valid Flag (1 bit) **Bit[5]**: External Wireless In Use Flag (1 bit) **Bit[6]**: Map Position Valid Flag (1 bit) **Bit[7]**: Map Position In Use Flag (1 bit)

11.21. Flags – Vertical and Secondary Filter Info

Entity Id: **0x0040**

	Field Name	Data Type	Unit	Description
1	flags_	UINT8	N/A	**Bit[1:0]**: Vertical Orientation (2 bits) **Bit[5:2]**: Secondary Info (4 bits) **Bit[7:6]**: Reserved (2 bits)

11.22. GNSS Raw Data*

Entity Id: **0x0063**

	Field Name	Data Type	Unit	Description
1	gnss_week_	UINT32	N/A	Week
2	gnss_tow_	DOUBLE64	sec	Time of week
3	satellite_num_	UINT32	N/A	Number of satellites
4	gnss_prn_	UINT32	N/A	PRN
5	satellite_system_	UINT32	N/A	Satellite System: 0x00: GPS 0x01: GLONASS 0x02:COMPASS 0x03: GALILEO
6	signal_type_	UINT32	N/A	0x00: L1/CA 0x01: L2 0x02: L5
7	phase_lock_	INT32	N/A	0x00: Lose Lock 0x01: Lock
8	code_lock_	INT32	N/A	0x00: Lose Lock 0x01: Lock
9	lose_lock_indicator	UINT32	N/A	Lose Lock Indicator
10	cn0_	DOUBLE64	dB-Hz	Carrier to Noise density
11	lock_time_	DOUBLE64	sec	Time since the latest lock
12	pseudorange_	DOUBLE64	m	
13	pseudorange _std_	DOUBLE64	m	
14	doppler_	DOUBLE64	Hz	
15	doppler_std_	DOUBLE64	Hz	
16	phase_	DOUBLE64	Cycles	

* Packets are repeated <Num_of_Sat> times

11.23. GPS Ephemeris Data*

Entity Id: **0x0059**

	Field Name	Data Type	Unit	Description
1	gnss_prn_	UINT32	N/A	PRN
2	gnss_week_	UINT32	N/A	Week number
3	gnss_tow_	DOUBLE64	sec	Time of week
4	code_	UINT32	N/A	Code on L2 0x00: Reserved 0x01: P code on 0x02: C/A code on

5	ura_	DOUBLE64	m	Satellite accuracy
6	health_	UINT32	N/A	Satellite health 0x00: all navigation date are ok 0x01: some navigation date are bad
7	iodc_	UINT32	N/A	
8	flag_	UINT32	N/A	Data flag on L2P
9	tgd_	DOUBLE64	sec	
10	toc_	DOUBLE64	sec	
11	af2_	DOUBLE64	sec/sec/sec	
12	af1_	DOUBLE64	sec/sec	
13	af0_	DOUBLE64	sec	
14	iode2_	UINT32	N/A	
15	crs_	DOUBLE64	m	
16	deltan_	DOUBLE64	semi-circles/sec	
17	m0_	DOUBLE64	semi-circles	
18	cuc_	DOUBLE64	rad	
19	ecc_	DOUBLE64	N/A	
20	cus_	DOUBLE64	rad	
21	sqrta_	DOUBLE64	√ m	
22	toe_	DOUBLE64	sec	
23	cic_	DOUBLE64	rad	
24	i0_	DOUBLE64	semi-circles	
25	crc_	DOUBLE64	m	
26	omega_	DOUBLE64	semi-circles	
27	omegadot_	DOUBLE64	semi-circles/sec	
28	iode3_	UINT32	N/A	
29	idot_	DOUBLE64	semi-circles/sec	

11.24. GLONASS Ephemeris Data*

Entity Id: **0x00E8**

	Field Name	**Data Type**	**Unit**	**Description**
1	gnss_prn_	UINT32	N/A	
2	frequency_	INT32	N/A	Frequency offset
3	toe_	DOUBLE64	sec	
4	toff_	DOUBLE64	sec	
5	iode_	UINT32	N/A	
6	health_	UINT32	N/A	Satellite health 0x00: all navigation date are ok 0x01: some navigation date are bad
7	pos_x_	DOUBLE64	m	
8	pos_y_	DOUBLE64	m	
9	pos_z_	DOUBLE64	m	
10	vel_x_	DOUBLE64	m/sec	
11	vel_y_	DOUBLE64	m/sec	
12	vel_z_	DOUBLE64	m/sec	
13	acc_x_	DOUBLE64	m/sec/sec	
14	acc_y_	DOUBLE64	m/sec/sec	

15	acc_z_	DOUBLE64	m/sec/sec	
16	taun_	DOUBLE64	sec	
17	gamman_	DOUBLE64	sec/sec	
18	tof_	DOUBLE64	sec	
19	age_	INT32	N/A	

11.25. Ionospheric Parameters*

Entity Id: **0x00BF**

	Field Name	Data Type	Unit	Description
1	GPSTimeTag	DOUBLE64	sec	
2	A1	DOUBLE64		
3	A2	DOUBLE64		
4	A3	DOUBLE64		
5	A4	DOUBLE64		
6	B1	DOUBLE64		
7	B2	DOUBLE64		
8	B3	DOUBLE64		
9	B4	DOUBLE64		

* Packets are repeated <Num_of_Sat> times

11.26. Satellite Parameters*

Entity Id: **0x0052**

	Field Name	Data Type	Unit	Description
1	Num_of_Sat	DOUBLE64	N/A	
2	GPSweek*	UINT16	N/A	
3	GPSTimeTag*	DOUBLE64	sec	
4	Num_of_Sat*	UINT16	N/A	
5	SatelliteID*	UINT8	N/A	
6	Satellite_X *	DOUBLE64	m	
7	Satellite_Y*	DOUBLE64	m	
8	Satellite_Z*	DOUBLE64	m	
9	Satellite_Vx*	DOUBLE64	m/sec	
10	Satellite_Vy*	DOUBLE64	m/sec	
11	Satellite_Vz*	DOUBLE64	m/sec	
12	Satellite_Elev*	DOUBLE64	deg	
13	Satellite_Azi*	DOUBLE64	Deg	
14	Sat_Ion_Corr*	DOUBLE64	Sec	

* Packets are repeated <Num_of_Sat> times

11.27. Multiple Antenna Data*

Entity Id: **0x0075**

	Field Name	Data Type	Unit	Description
1	heading_	FLOAT32	deg	
2	heading_standard_deviation_	FLOAT32	deg	
3	roll_	FLOAT32	deg	
4	roll_standard_deviation_	FLOAT32	deg	

5	pitch_	FLOAT32	deg	
6	pitch_standard_deviation_	FLOAT32	deg	

11.28. Temperature Sensor Data*

Entity Id: **0x0071**

	Field Name	Data Type	Unit	Description
1	GPSTimeTag	DOUBLE64	Seconds	
2	Temperature	FLOAT32	K	

11.29. Heading Data*

Entity Id: **0x009B**

	Field Name	Data Type	Unit	Description
1	GPSTimeTag		Seconds	
2	Heading	DOUBLE64	Degrees	
3	HeadingErr_Std	FLOAT32	Degrees	

11.30. Map Matching Data*

Entity Id: **0x00C7**

	Field Name	Data Type	Unit	Description
1	GPSTimeTag	DOUBLE64	Seconds	
2	Map_ Derived_Lat	DOUBLE64	Degrees	
3	Map_ Derived_Long	DOUBLE64	Degrees	
4	Map_ Derived_Head	DOUBLE64	Degrees	

11.31. Raw Map database Option1*

Entity Id: **0x00C2**

	Field Name	Data Type	Unit	Description
1	num_of_map_db_entries			
2	Seg_Start_Lat*	DOUBLE64	Degrees	
3	Seg_Start_Long*	DOUBLE64	Degrees	
4	Seg_End_Lat*	DOUBLE64	Degrees	
5	Seg_End_Long*	DOUBLE64	Degrees	

11.32. Raw Map database Option2*

Entity Id: **0x00A0**

	Field Name	Data Type	Unit	Description
1	num_of_map_db_entries	DOUBLE64	Degrees	
2	Init_Lat	DOUBLE64	Degrees	

3	Init_Long	DOUBLE64	Metres	
4	Init_Alt	FLOAT32	m	
5	Seg_Start_X*	FLOAT32	m	
6	Seg_Start_Y*	FLOAT32	m	
7	Seg_End_X*	FLOAT32	m	
8	Seg_End_Y*	FLOAT32	m	
9	Seg_Slope*	DOUBLE64	N/A	

* Packets are repeated < num_of_map_db_entries > times

11.33. WiFi Beacon Message*

Entity Id: **0x0018**

	Field Name	Data Type	Unit	Description
1	GPSTimeTag	FLOAT32	Sec	
2	MAC	UINT8[6]	N/A	
3	RSSI	FLOAT32	dBm	
4	Timestamp	ULONG	Ms	
5	HostTimestamp	ULONG	100ns	
6	NetworkType	UNIT8	N/A	
7	SSID	BYTE[80]	N/A	
8	Beacon Period	DOUBLE64	Ms	
9	Channel Frequency	ULONG	KHZ	
10	Channel	INT32	N/A	
11	Lat	DOUBLE64	Degrees	
12	Long	DOUBLE64	Degrees	
13	Alt	FLOAT32	Degrees	

11.34. Height Data*

Entity Id: **0x0081**

	Field Name	Data Type	Unit	Description
1	height_	FLOAT32	m	
2	height_standard_deviation_	FLOAT32	m	

11.35. Heading Data*

Entity Id: **0x0074**

	Field Name	Data Type	Unit	Description
1	Heading	DOUBLE64	Degrees	
2	HeadingErr_Std	FLOAT32	Degrees	

11.36. Other Wireless Position/Heading Data*

Entity Id: **0x00CB**

	Field Name	Data Type	Unit	Description
1	GPSTimeTag	DOUBLE64	sec	

2	Latitude	DOUBLE64	deg	
3	Longitude	DOUBLE64	deg	
4	Heading	DOUBLE64	deg	
5	Position std	FLOAT32	m	
6	Heading std	FLOAT32	deg	

12. Output Entities For Each Version

12.1. 2.0.0-b

	Entity	Entity Id	Entity Type
1	Time	0x00ea	Standard
2	Position	0x0016	Standard
3	Position Standard Deviation	0x00bd	Standard
4	Velocity	0x00a2	Standard
5	Velocity Standard Deviation	0x005f	Standard
6	Attitude	0x00e9	Standard
7	Attitude Standard Deviation	0x0060	Standard
8	Accelerometer Bias	0x006f	Standard
9	Gyroscope Bias	0x002e	Standard
10	Heading Misalignment	0x004d	TPN-Free
11	Platform Heading	0x00b3	TPN-Free
12	Number of Steps	0x0099	TPN-Free
13	Floor Number	0x00dd	TPN-Free
14	Flags – Magnetometer and Barometer	0x0091	Standard
15	Mode of Transit	0x00c8	TPN-Free
16	Flags – GNSS, Speed and Static Status	0x002b	Standard
17	Flags – Navigation Phase	0x00e7	Standard
18	Raw IMU Data*	0x00a1	Standard / Debug
19	GNSS PVT Data*	0x001c	Standard / Debug
20	Barometer Data*	0x0083	Standard / Debug
21	Magnetometer Data*	0x00b7	Standard / Debug
22	Multiple Antenna Data*	0x0075	Standard / Debug
23	Speed Data*	0x00a7	Standard / Debug
24	Wi-Fi 2D Position Data*	0x0051	TPN-Free / Debug
25	Operator 2D Position Data*	0x0042	TPN-Free / Debug
26	Floor Information Data*	0x004a	TPN-Free / Debug
27	Call Information Data*	0x00ae	TPN-Free / Debug
28	Changed User Mode Event Data*	0x001d	TPN-Free / Debug

12.2. 2.1.0-rc/3.0.0

	Entity	Entity Id	Entity Type
1	Time	0x00ea	Standard
2	Position	0x0016	Standard
3	Position Standard Deviation	0x00bd	Standard
4	Velocity	0x00a2	Standard
5	Velocity Standard Deviation	0x005f	Standard
6	Attitude	0x00e9	Standard
7	Attitude Standard Deviation	0x0060	Standard
8	Accelerometer Bias	0x006f	Standard
9	Gyroscope Bias	0x002e	Standard
10	Heading Misalignment	0x004d	TPN-Free
11	Platform Heading	0x00b3	TPN-Free
12	Number of Steps	0x0099	TPN-Free
13	Floor Number	0x00dd	TPN-Free
14	Flags – Magnetometer and Barometer	0x0091	Standard
15	Mode of Transit	0x00c8	TPN-Free
16	Flags – GNSS, Speed and Static Status	0x002b	Standard
17	Flags – Navigation Phase	0x00e7	Standard

18	Raw IMU Data*	0x00a1	Standard / Debug
19	GNSS PVT Data*	0x001c	Standard / Debug
20	Barometer Data*	0x0083	Standard / Debug
	~~Magnetometer Data*~~	~~0x00b7~~	~~Standard / Debug~~ [Deprecated]
21	Magnetometer Data*	0x0062	Standard / Debug
	~~Multiple Antenna Data*~~	~~0x0075~~	~~Standard / Debug~~ [Removed in this release]
22	Speed Data*	0x00a7	Standard / Debug
23	Wi-Fi 2D Position Data*	0x0051	TPN-Free / Debug
24	Operator 2D Position Data*	0x0042	TPN-Free / Debug
25	Floor Information Data*	0x004a	TPN-Free / Debug
26	Call Information Data*	0x00ae	TPN-Free / Debug
27	Changed User Mode Event Data*	0x001d	TPN-Free / Debug

12.3. 4.0.0-x

	Entity	Entity Id	Entity Type
1	Time	0x00ea	Standard
2	Position	0x0016	Standard
3	Position Standard Deviation	0x00bd	Standard
4	Velocity	0x00a2	Standard
5	Velocity Standard Deviation	0x005f	Standard
6	Attitude	0x00e9	Standard
7	Attitude Standard Deviation	0x0060	Standard
8	Accelerometer Bias	0x006f	Standard
9	Gyroscope Bias	0x002e	Standard
10	Heading Misalignment	0x004d	TPN-Free
11	Platform Heading	0x00b3	TPN-Free
12	Number of Steps	0x0099	TPN-Free
13	Floor Number	0x00dd	TPN-Free
14	Flags – Magnetometer and Barometer	0x0091	Standard
15	Mode of Transit	0x00c8	TPN-Free
16	Flags – GNSS, Speed and Static Status	0x002b	Standard
17	Flags – Navigation Phase	0x00e7	Standard
	~~Use Case~~	~~0x00ee~~	~~TPN-Free~~ (Not output in this version)
	~~Stride Information~~	~~0x005e~~	~~TPN-Free~~ (Not output in this version)
18	Raw IMU Data*	0x00a1	Standard / Debug
19	GNSS PVT Data*	0x001c	Standard / Debug
20	Barometer Data*	0x0083	Standard / Debug
21	Magnetometer Data*	0x0062	Standard / Debug
22	Speed Data*	0x00a7	Standard / Debug
23	Wi-Fi 2D Position Data*	0x0051	TPN-Free / Debug
24	Operator 2D Position Data*	0x0042	TPN-Free / Debug
25	Floor Information Data*	0x004a	TPN-Free / Debug
26	Call Information Data*	0x00ae	TPN-Free / Debug
27	Set Mode of Transit Event Data*	0x001d	TPN-Free / Debug [Name Changed from "Changed User Mode Event Data"]
28	Platform Heading Data*	0x00ba	TPN-Free / Debug
29	Quaternions Data*	0x007a	TPN-Free / Debug
30	Gyroscope Delta Biases Data*	0x0008	TPN-Free / Debug
31	Set ZUPT Mode Event Data*	0x009f	TPN-Free / Debug
32	Set Misalignment Event Data*	0x00a4	TPN-Free / Debug
33	Set Magnetometer Calibration Status Event Data*	0x00e4	TPN-Free / Debug
34	Set Use Case Event Data*	0x0098	TPN-Free / Debug
35	6 DOF Quaternions	0x007B	TPN-Free/Debug

36	In-run Magnetometer Calibration Information ^	0x000f	TPN-Free / Internal [Engine Mode = 0]
37	Internal Debugging Information(1) ^	0x008c	TPN-Free / Internal [Engine Mode = 0]

12.4. 4.1.0-x

	Entity	Entity Id	Entity Type
1	Time	0x00ea	Standard
2	Position	0x0016	Standard
3	Position Standard Deviation	0x00bd	Standard
4	Velocity	0x00a2	Standard
5	Velocity Standard Deviation	0x005f	Standard
6	Attitude	0x00e9	Standard
7	Attitude Standard Deviation	0x0060	Standard
8	Accelerometer Bias	0x006f	Standard
9	Gyroscope Bias	0x002e	Standard
10	Heading Misalignment	0x004d	TPN-Free
11	Platform Heading	0x00b3	TPN-Free
12	Number of Steps	0x0099	TPN-Free
13	Floor Number	0x00dd	TPN-Free
14	Converged Gyroscope Bias	0x002f	TPN-Free [Added in 4.1.0-a.5]
15	Flags – Magnetometer and Barometer	0x0091	Standard
16	Mode of Transit	0x00c8	TPN-Free
17	Flags – GNSS, Speed and Static Status	0x002b	Standard
18	Flags – Navigation Phase	0x00e7	Standard
19	Use Case	0x00ee	TPN-Free
20	Stride Information	0x005e	TPN-Free
21	Raw IMU Data*	0x00a1	Standard / Debug
22	GNSS PVT Data*	0x001c	Standard / Debug
23	Barometer Data*	0x0083	Standard / Debug
24	Magnetometer Data*	0x0062	Standard / Debug
25	Speed Data*	0x00a7	Standard / Debug
	~~Wi-Fi 2D Position Data*~~	~~0x0051~~	~~TPN-Free / Debug~~
	~~Operator 2D Position Data*~~	~~0x0042~~	~~TPN-Free / Debug~~
26	Operator 2D Position Data*	0x0043	TPN-Free / Debug
27	Wi-Fi Data*	0x0055	TPN-Free / Debug
28	Floor Information Data*	0x004a	TPN-Free / Debug
29	Call Information Data*	0x00ae	TPN-Free / Debug
30	Set Mode of Transit Event Data*	0x001d	TPN-Free / Debug [Name Changed from “Changed User Mode Event Data”]
31	Platform Heading Data*	0x00ba	TPN-Free / Debug
32	Quaternions Data*	0x007a	TPN-Free / Debug
33	Gyroscope Delta Biases Data*	0x0008	TPN-Free / Debug
34	Process Misalignment Data*	0x00a6	TPN-Free/Debug [Added in 4.1.0-a.5]
35	Set ZUPT Mode Event Data*	0x009f	TPN-Free / Debug
36	Set Misalignment Event Data*	0x00a4	TPN-Free / Debug
37	Set Magnetometer Calibration Status Event Data*	0x00e4	TPN-Free / Debug
38	Set Use Case Event Data*	0x0098	TPN-Free / Debug
39	6 DOF Quaternions	0x007B	TPN-Free/Debug
40	Synchronization Event Data*	0x00ff	TPN-Free/Debug [Added in 4.1.0-a.5]
41	In-run Magnetometer Calibration Information ^	0x000f	TPN-Free / Internal [Engine Mode = 0]
42	Internal Debugging Information(1) ^	0x008c	TPN-Free / Internal [Engine Mode = 0]

13. Appendix A: Predefined Configurations

13.1. Predefined GNSS Configurations

PREDEFINED_STRING	Examples of GNSS Receivers
"gnss_high_sensitivity"	Qualcomm-GNSS(GPS+GLONASS) (SIII), Default High Sensitivity
"gnss_high_sensitivity_1"	SiRFstar (Nexus)
"gnss_high_sensitivity_2"	Broadcom-GNSS(GPS+GLONASS) (Note1)
"gnss_high_sensitivity_3"	u-blox (when used with low grade gyroscopes with bias instability ≥ 100 deg/hr)
"gnss_high_sensitivity_4"	Low cost GNSS receivers that provide unusable GPS velocities
"gnss_high_sensitivity_5"	u-blox (when used with high grade gyroscopes with bias instability < 100 deg/hr)
"gnss_high_sensitivity_6"	BlackBerry Playbook GNSS Receiver
"gnss_high_sensitivity_7"	Not Specified
"gnss_high_sensitivity_8"	Samsung Note 3
"gnss_precision"	NovAtel, Hemisphere, Default Precision Receivers
"gnss_precision_1"	Trimble

13.2. Predefined Accelerometer Configurations

PREDEFINED_STRING	Bias Instability (mGal)	VRW (m/sec/√hr)	Correlation time (hr)	Examples of Triad accelerometers
"accelerometer_1"	1700	2.4	2.0	MPU3050(Nexus), MPU6050(SIII), MPU9150, Note(STMICRO)
"accelerometer_2"	100 Turning: 5000	0.09	4.0 Turning: 0.5	ADIS16485,ADIS16385
"accelerometer_3"	3700	3.4	2.0	BlackBerry Playbook
"accelerometer_4"	1000	0.5	1.0	Murata SCC1300-D04
"accelerometer_5"	500	0.1	1.0	**TMN/Airplane**[4] Murata SCC1300-D04

13.3. Predefined Gyroscope Configurations

PREDEFINED_STRING	Bias Instability (deg/hr)	ARW (deg/√hr)	Correlation time (hr)	Examples of triad gyroscopes
"gyroscope_1"	100	1.8	1.0	MPU6050(SIII), MPU9150
"gyroscope_2"	150	1.8	1.0	MPU3050(Nexus), STMICRO (Note)
"gyroscope_3"	4	0.15	4.0	ADIS16485
"gyroscope_4"	10	1.8	1.0	Embedded library
"gyroscope_5"	20	1.0	1.0	Murata SCR1100-D04

[4] Not applicable for the TPN/Free navigator.

	Turning: 35			
“gyroscope_6”	1	0.15	4.0	NG Litef uFORS-6U
“gyroscope_7”	21	1.9	1.0	ADIS16385(X,Y GYROS)
“gyroscope_8”	6	0.75	1.0	ADIS16385(Z GYRO)
“gyroscope_9”	15	2.0	1.0	**TMN/Airplane** Murata SCR1100-D04
“gyroscope_10”	350	5	1.0	Lumia820 Gyros for walk

13.4. Predefined Motion Constraints Configurations

PREDEFINED_STRING	Dynamic Velocity Constraint (m/sec)	Static Velocity Constraint (m/sec)	Examples of Triad Accelerometers
“motion_constraints_1”	0.1	0.1	MPU6050(SIII),MPU9150
“motion_constraints_2”	0.3	0.1	Embedded library
“motion_constraints_3”	0.4	0.5	Murata SCC1300-D04
“motion_constraints_4”	0.15	0.5	ADIS16485, ADIS16385
“motion_constraints_5”	50.0	0.1	**TMN/Airplane** MPU3050(Nexus), STMicro (Note),MPU6050(SIII), MPU9150
“motion_constraints_6”	35.0	0.5	**TMN/Airplane** Murata SCC1300-D04, ADIS16385, ADIS16485

Change Log

Table 6 Change Log

Version	Changes
May 02-2016	1. Added Street Map API 2. Change section for misalignment to add “fixed-unknown” option with standard deviation to accommodate Sierra’s work
v4.5.0-alpha	1. Adding functions to create and delete a navigation session handle. This navigation session handle needs to be passed to all the functions. The use of the navigation session handle would allow the developer to use multiple navigation sessions in the same process that loads the TPP library. 2. Adding the multi-device API functions, structures, return values. v4.5.0-alpha.11 1. Adding the initialization flag DECLINATION_ANGLE 2. Adding the function **tpp_multi_device_process_secondary_gnss_pvt()** 4. Adding temporary use-cases for testing 5. Adding the “running” mode of transit v4.5.0-alpha.16 1. Adding the function **tpp_process_venue_map(),** the function return types, the corresponding structure **TppVenueMapMessageStructure**, and the corresponding entity. v4.5.0-alpha.20 1. Updating the type of **solution_reliable_** field in TppSolutionStruct to INT8 instead of UINT8. The reason is to use the value (-1) to indicate a new phase which is pre-alignment. In this phase, the solution shall be discarded entirely in plotting or as a reference. 2. In the entity ‘**Flags – Navigation Phase**’, the reserved 4 bits are removed, and the whole byte is allocated for the navigation phase. The type of the byte also changed from INT8 to UINT8 to reflect the change related to the point above, 3. Updating the name of the flag **solution_reliable_** in TppSolutionStruct to **navigation_phase_** to be consistent in what both values represent. v4.5.0-alpha.21 1. Adding values 0x07 and 0x08 for ‘Conveyer Walking’ and ‘Conveyer Standing’ v4.5.0-alpha.30 1. Updating the structure definition for **TppApiVersionStruct**. 2. Renaming the name of the function **tpp_process_gyroscope_delta_biases()** to **tpp_process_imu_biases()**. The corresponding structures and enumerations’ names are changed as well. 3. Updating the structure definition for **TppGyroscopeDeltaBiasesMessageStructure** accordingly. v4.5.0-alpha. 1. Adding the function **tpp_add_anchor_point (),** the function return types, the corresponding structure **TppAnchorPoint**, and the corresponding entity.

	2. Adding the three fields related to the "step length scale" to **TppVenueMapMessageStruct** and the corresponding entity. 3. Adding the function **tpp_set_orientation_based_on_pitch()**, the corresponding enumeration **TppOrientationBasedOnPitch**, return types and entity "**Orientation Based On Pitch Data**". 4. Adding the entity **Orientation Based On Pitch** 5. Adding the initialization flag **PROCESSING_MODE** v4.5.0-alpha.60 1. Replacing tpp_process_wifi() with tpp_process_wireless() and updating the structures accordingly. 2. Adding the "cycling" mode of transit 3. Adding the entities "Distance Travelled" and "DSL Height"
v4.2.0-beta	1. Beta release of v4.2.0 of the library 2. Updating the **solution_reliable_** flag definition in **TppSolutionStruct** 3. Adding the function **tpp_set_device_heading()**
v4.1.0-alpha	1. Alpha release of v4.1.0 of the library 2. For "**MISALIGNMENT**" initialization flag**, flag_value_2_** now takes a value when **flag_value_1_** is "**automatic**". The value for **flag_value_2_** can be "**misalignment_normal**" or "**misalignment_snap**" 3. **tpp_process_2d_position()** is no more available. To process operator 2d position updates, the user can use **tpp_process_operator_2d_position().** 4. To process Wi-Fi data, **tpp_process_wifi()** is added. 5. **tpp_process_quaternions()** is renamed to **tpp_process_9dof_quaternions()** 6. The name of the structure **TppMisalignmentStruct** is changed to **TppSetMisalignmentStruct** 7. **tpp_process_misalignment()**, and its corresponding structure, entity, and return value is added.
v4.0.0-alpha	1. Alpha release of v4.0.0 of the library 2. Added Initialization Flags: a. "**OUTPUT_POSITION_ERROR_WEIGHTING**" b. "**OUTPUT_VELOCITY_ERROR_WEIGHTING**" c. "**OUTPUT_ATTITUDE_ERROR_WEIGHTING**" d. "**ACCELEROMETER_BIASES**" 3. Added Functions. The corresponding structures/enumerators and data/debug entities are added as well: a. **tpp_process_platform_heading** b. **tpp_process_quaternions** c. **tpp_process_6dof_quaternions** d. **tpp_process_gyroscope_delta_biases** e. **tpp_set_zupt_mode** f. **tpp_set_misalignment** g. **tpp_set_magnetometer_calibration** h. **tpp_set_use_case** 4. Added output entities: a. **Stride Information** – entity Id: **0x005E** b. **Use Case** – entity Id: **0x00EE**

	5. Additional values are used for "**Mode of Transit**" entity – entity Id: **0x00C8**
v3.0.0	1. Official release of the ready for production version of the library 2. "**START_OPTION**" initialization flag is added to the list of initialization flags 3. The name of the function **tpp_change_mode_of_transit**() changed to **tpp_set_mode_of_transit()** 4. The raw magnetometer entity structure is revised and the entity ID changed. 5. The behavior of **tpp_process_2d_position()** is changed. If the **source_** field is equal to 0x00 (i.e. Operator position update), the standard deviation will not take effect.
v2.0.0-beta	1. Official release of the beta version of the library

Revision History

Table 7 Document Revision History

Date	Revision	Changes
2012-November-10	1	Initial release.
2012-December-06	2 (1.5.1)	1. Adding a couple of functions to process (1) the 2D position (2) height from a source other than GPS or barometer (3) multiple antenna data (4) floor information 2. Correcting the measurement units 3. Adding a table for the list of definitions 4. Adding pseudo-code for checksum computation
2013-February-28	3 (1.5.2)	1. Adding an API function to get the major, minor and patch versions for the API. 2. Removing the default values that specific options will take if the application developer used a flag name without a flag value. 3. Revising the code samples provided. 4. Changing the order of the first three sections. 5. Changing the patch number due to fixing bugs, and adding a new function. 6. Removing the offset column in the output array structure table. 7. Updating the footer of the document. 8. Adding a specific title on the cover page to specify whether the API is for TPN/Free, TPN/Tethered, TMN or TVN. 9. TMN initialization in 7.1. is revised. 10. Tightly coupled entities added by Tao.
2013-April-23 2013-May-13 2013-May-28	4 (2.0.0 Beta)	1. Adding a function to initialize TMN/Aerial. 2. Changing the name of the function used to initialize TMN system **tpp_initialize_tmn()** to **tpp_intialize_tmn_land()** 3. Adding **tpp_process_call_information()** function and the corresponding input structure 4. Removing the **timetag_** member/field from four input structures ▪ Tpp2dPositionMessageStruct ▪ TppHeightMessageStruct ▪ TppFloorInformationMessageStruct ▪ TppCallInformationMessageStruct 5. Error codes enumeration values are updated to match more logical and consistent values. 6. Some entities are revised such as ▪ Wi-Fi Data ▪ 2D Position Data ▪ Height Data ▪ Floor Information Data 6. Tightly coupled functions **tpp_process_gnss_observations(), tpp_process_gps_ephemeris()**, **tpp_process_glonass_ephemeris()** and the corresponding input structures along with the error codes are defined. 7. **TppPacketStruct** replaces **TppOutputStruct** (just a name

		replacement) 8. **TppSolutionStruct** is added for customers 9. **sec** is used instead of **s** in some areas. 10. **Height** definition is refined. **Developer** and **operator** definitions are added. 11. **tpp_advance_navigation_step**() is restructured to accommodate for another parameter of type **TppSolutionStruct** 12. **tpp_change_mode_of_transit**() is added along with corresponding enum and error code to return. 13. For every initialization flag, we have now up to 9 flag values that can be set instead of 4. 14. **GNSS** initialization flag: adding **scalefactor**, and receiver type 15. **WIFI** initialization flag added: adding **scalefactor** 16. **IMU_CONFIGURATION**, **ACCELEROMETER_CONFIGURATION**, **GYROSCOPE_CONFIGURATION** and **DEBUG_DATA** flags are added 17. **MAGNETOMETER**,**BAROMETER**, and **SPEED** flags: adding **scalefactor** 18. **HEADING**: on/off flag removed 19. **message_id_**, and **message_sub_id_** members are removed from the input structures. 20. GNSS/PVT input structure: members **dop_data_available_** added & **pdop_,gdop_,gnss_velocity_timetag** are removed 21. Restructuring the barometer and magnetometer structures 22. Barometer, Magnetometer, Raw IMU, GNSS entities are restructured 23. Stars (*) are added to identify data/debug entities 24. Names of some entities were refined 25. Call Information Data entity is added 26. Error codes are added to make it easier for the developer to identify the problems 27. The word 'developer' is used instead of 'user' 28. **sensor_status_** member is removed from RAW IMU structure 29. **tpp_change_mode_of_transit**() is added 30. Adding "**MOTION_CONSTRAINTS**" flag 31. No default values for "**GYROSCOPE_BIASES**". Has to be set by the user, or an error will be returned. Adding corresponding error code. 32. Wired and wireless earphones events are combined in one flag instead of two. 33. Changes to the types of some variables/members in the structures and the entities. 34. Heading is used instead of azimuth in all locations 35. All flag values use non-capital letters 36. **platform_heading_** is added to **TppSolutionStructure** 37. **HEADING** flag is now **PLATFORM_HEADING** 38. Changing the name of the initialize TPN/Free function to **tpp_initialize_tpn_free()** instead of

		tpp_initialize_tpn_free_motion() 39. The configurations table is refined 40. "**GNSS**" flag values are refined 41. Correcting the default GNSS flag values for TPN 42. "Examples of GNSS Receivers" is used instead of "Included Receivers"
2013-June-24	5 (Customized Library)	1. Customized API functions are added
2013-July-19	6	1. Adding an API function, initialization flag, and return enumerations to run the backward smoothing functionality.
2013-August-27	7	1. Changing the word 'scaling' to 'weighting'. Other wording is changed accordingly.
2013-September-30	8	Refer to the change log for v4.0.0-alpha
2013-November-14	9	Refer to the change log for v4.1.0-alpha
2014-January-22	10	Changes related to the TMN and TVN Libraries 1. **START_OPTION** initialization flag is now supported for TMN 2. **flag_value_1_** for **GNSS** can be set to "on" or "off" where previously it was reserved
2014-July-25	11	Adding the initialization flag "**DECLINATION_ANGLE**"