1 Introduction

The purpose of this document is to provide a reference to all APIs available in ST GNSS library for all ST GNSS microcontrollers (STA2062, STA2064, STA2065, STA8088, STA8089, STA8090).

It is not recommended to use APIs that are not documented in this document.

1.1 Index

1.2 List of Tables

None.

1.3 List of Figures

None.

2 Document Management

2.1 Revision History

Rev	Date	Author	Notes
2.0	2011-04-08	A. Di Girolamo	Initial draft release. Updated to GNSS library release 7.1.1.15
2.1	2011-04-27	F. Boggia	Updated to GNSS library release 7.1.3.17
2.2	2011-07-11	F. Boggia	Updated to GNSS library release 7.1.6.27
3.0	2012-03-20	A. Di Girolamo	Added APIs for Pulse Per Second.
3.1	2012-04-03	A. Di Girolamo	Added new APIs and data structures for Pulse Per Second.
3.2	2012-04-12	A. Di Girolamo	Added API to set/get constellation rf delay. Added new fields in the pps_data_t structure.
3.2	2012-07-25	M.Renna	Added API to set/get Notch Filter functionality
3.3	2012-08-27	A. Di Girolamo	Added API to get the Master Time Base (MTB) time and the clock offset from the GNSS fix data block.
3.4	2012-09-20	A. Di Girolamo	Added API to enable/disable the PPS control logic. Added API to inject the PPS params into the PPS tracker control logic.
3.5	2012-11-09	A. Di Girolamo	Added API for setting/getting the satellite elevation mask for positioning.
3.6	2013-04-23	A. Di Girolamo	Added new APIs. Document review.
3.6.1	2013-05-23	S. Restuccia	XTAL API added
3.7	2014-10-02	A. Di Girolamo	Added missing APIs and data types General document review
4.0	2014-12-12	J. Durand	Rel8 edition
4.1	2014-04-28	J. Philippe F.Boggia	Rel8 edition. Add gnss_init_p() function. Corrected typo errors.
4.2	2015-10-27	J.Philippe	API added for RAIM/FDE exclusion

4.3	2016-03-15	J. Durand	Update of Low Power parts
	2016-05-03	A. Di Girolamo	Added not filtered PVT APIs Added WLS configuration API
4.4	2016-06-29	J. Durand	Update of Low Power parts
4.5	2016-11-07	D.Cardineau	Remove get tracker data API Remove WAAS get symbols API
4.6	2016-11-08	J. Durand	Update for Low Power and RTC calibration

2.2 Acronyms

Keyword	Definition
FDE	Fault Detection Error
GLONASS	GLObal NAvigation Satellite System (The Russian GNSS)
GNSS	Global Navigation Satellite System – It can include any combination of different satellite constellations like GPS, GLONASS, SBAS etc.
GPS	Global Positioning System (The US GNSS)
PPS	Pulse Per Second
RAIM	Receiver Autonomous Integrity Monitoring
RTC	Real Time Clock
SBAS	Satellite Based Augmentation System

2.3 Reference Documents

None

2.4 Contact info

Keyword	Definition
A. Di Girolamo	andrea.di-girolamo@st.com
F. Boggia	fulvio.boggia@st.com

3 GNSS Standard Functions

3.1 gnss_version

Returns the library version string

Synopsis:

```
#include "gnss_api.h"

const tChar *gnss_version(void);
```

Arguments:

None.

Results:

`tChar *`: Returns the pointer to GNSS library version string.

Errors:

None.

Description:

Returns the GNSS library version string.

3.2 gnss_supplier_id

Returns the name of the GNSS supplier

Synopsis:

```
#include "gnss_api.h"

const tChar *     gnss_supplier_id(void);
```

Arguments:

None.

Results:

`tChar *`: Returns the pointer to the GNSS supplier string.

Errors:

None.

Description:

Returns the name of the GNSS supplier.

3.3 gnss_get_subsystem_version

Returns the subsystem software version string

Synopsis:

```
#include "gnss_api.h"

const tChar *gnss_get_subsystem_version( void);
```

Arguments:

None.

Results:

`tChar *`: Returns the pointer to GNSS subsystem version string.

Errors:

None.

Description:

Returns the GNSS subsystem version string.

Note: This function is supported only on STA2062 and STA2064/65 platforms.

3.4 gnss_get_ekernel_version

Returns the Emerald DSP software version string

Synopsis:

```
#include "gnss_api.h"

const tChar *gnss_get_ekernel_version( void);
```

Arguments:

None.

Results:

`tChar *`: Returns the pointer to Emerald DSP software version string.

Errors:

None.

Description:

Returns the Emerald DSP software version string.

Note: This function is supported only on STA2062 platform.

3.5 gnss_get_cut_version

Returns the silicon cut version

Synopsis:

```
#include "gnss_api.h"

tracker_cut_version_t gnss_get_cut_version(void);
```

Arguments:

None.

Results:

`tracker_cut_version_t:` Returns the silicon cut version.

Errors:

None.

Description:

Returns the silicon cut version.

3.6 gnss_set_cut_version

Allow setting the silicon cut version

Synopsis:

```
#include "gnss_api.h"

void gnss_set_cut_version( tracker_cut_version_t version);
```

Arguments:

`tracker_cut_version_t`: silicon cut version

Results:

None.

Errors:

None.

Description:

Configure the GNSS library to support a specific silicon cut version.

Note: This function is supported only on STA2064/65 platforms.

3.7 gnss_init

Initializes the GNSS engine

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_init( gnss_sat_type_mask_t, gnss_sat_type_mask_t);
```

Arguments:

`gnss_sat_type_mask_t`: Constellation bit mask.

`gnss_sat_type_mask_t`: Constellation usage bit mask.

Results:

`gnss_error_t`: Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Fails if GNSS resources (including tasks and the other OS resources) initialization fails.

Description:

Initializes the GNSS engine. This does any necessary GNSS resource (tasks etc.) initialization and any global data, used by the GNSS, initialization. Once this function has been called the GNSS is ready to start searching for satellites but will not be using any processing time.

Example:

```
#include "gnss_api.h"
main()
{
   gnss_sat_type_mask_t   orbit_list_selection = 0;
   gnss_sat_type_mask_t   constellation_usage_selection = 0;

   MCR_SETBIT( orbit_list_selection, GNSS_SAT_TYPE_GPS );
   MCR_SETBIT( orbit_list_selection, GNSS_SAT_TYPE_GLONASS );

   MCR_SETBIT( constellation_usage_selection, GNSS_SAT_TYPE_GPS );
   MCR_SETBIT( constellation_usage_selection, GNSS_SAT_TYPE_GLONASS );

   /* load application into memory */
   gnss_init( orbit_list_selection, constellation_usage_selection);
   /* set the latitude to 51 deg 20.33 mins North*/
   /* set the longitude to 2 deg 30 mins West */
   /* set height to 140m above mean sea level */
   gnss_set_pos(51.33, -2.50, 140.0);
   /* set utc time to 12:35:00 on 24 Oct 1996 */
   gnss_set_time(1996, 10, 24, 12, 35, 00);
   /* allow the gnss to acquire satellites */
   gnss_start();
}
```

See also:

`gnss_init_p`, `gnss_suspend`.

3.8 gnss_init_p

Initializes the GNSS engine. This is a variant of gnss_init().

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_init_p( gnss_sat_type_mask_t, gnss_sat_type_mask_t,
boolean_t, tUInt);
```

Arguments:

gnss_sat_type_mask_t:	Constellation bit mask.
gnss_sat_type_mask_t:	Constellation usage bit mask.
Boolean_t:	indicate the Dead Reckoning presence.
tUInt:	Size of sensors sampling buffer

Results:

gnss_error_t:	Returns GNSS_NO_ERROR if the operation was successful or GNSS_ERROR on failure.

Errors:

Fails if GNSS resources (including tasks and the other OS resources) initialization fails.

Description:

Initializes the GNSS engine. This does any necessary GNSS resource (tasks etc.) initialization and any global data, used by the GNSS, initialization. Once this function has been called the GNSS is ready to start searching for satellites but will not be using any processing time.

If the Dead Reckoning is used, the sensor buffer size depends on the sampling rate and so on the sensors types but a minimum value could be 150 messages.

Example:

```
#include "gnss_api.h"
main()
{
   gnss_sat_type_mask_t   orbit_list_selection = 0;
   gnss_sat_type_mask_t   constellation_usage_selection = 0;

   MCR_SETBIT( orbit_list_selection, GNSS_SAT_TYPE_GPS );
   MCR_SETBIT( orbit_list_selection, GNSS_SAT_TYPE_GLONASS );

   MCR_SETBIT( constellation_usage_selection, GNSS_SAT_TYPE_GPS );
   MCR_SETBIT( constellation_usage_selection, GNSS_SAT_TYPE_GLONASS );

   /* Disable STAGPS */
   gnss_set_stagps_status(FALSE);
   /* load application into memory */
   gnss_init_p( orbit_list_selection, constellation_usage_selection,
FALSE, 0);
   /* set the latitude to 51 deg 20.33 mins North*/
   /* set the longitude to 2 deg 30 mins West */
   /* set height to 140m above mean sea level */
   gnss_set_pos(51.33, -2.50, 140.0);
   /* set utc time to 12:35:00 on 24 Oct 1996 */
   gnss_set_time(1996, 10, 24, 12, 35, 00);
   /* allow the gnss to acquire satellites */
   gnss_start();
}
```

See also:

`gnss_set_stagps_status`, `gnss_set_pos`, `gnss_set_time`.

3.9 gnss_start

Tells the GNSS engine to start looking for satellites

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_start( void);
```

Arguments:

None.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if it is called before the GNSS engine initialization or if it is called when the GNSS is suspended

Description:

This starts the GNSS receiver and causes the GNSS software to start looking for available satellites.

See also:

`gnss_init`, `gnss_suspend`, `gnss_restart`.

3.10 gnss_suspend

Causes the GNSS to suspend processing

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_suspend( void)
```

Arguments:

None.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if it is called when the GNSS engine is not running.

Description:

Stops the GNSS from tracking or searching for satellites and suspends all processing until a gnss_restart command is received.

See also:

`gnss_init`, `gnss_restart`.

3.11 gnss_restart

Causes the GNSS engine to start looking for satellites again.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_restart( void);
```

Arguments:

None.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if it is called when the GNSS engine is not suspended on it is not running.

Description:

This function will cause the GNSS engine to begin looking for satellites again, following a call to `gnss_suspend`.

See also:

`gnss_init`, `gnss_suspend`.

3.12 gnss_set_pos

Sets the initial position of the receiver

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_pos(
   const tDouble lat,
   const tDouble lon,
   const tDouble height
);
```

Arguments:

`tDouble lat`:	The latitude of the receiver in degrees.
`tDouble lon`:	The longitude of the receiver in degrees.
`tDouble height`:	The height of the receiver above mean sea level in meters.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` is the input position is invalid. If the operation was unsuccessful the GNSS initial position remains the same as before.

Description:

This initializes the estimate of the current position. This function is only relevant if the GNSS has not calculated its own position.

Note: It is necessary for the GNSS to have an accurate estimate of position to give a short time to first fix. If the GNSS position is not set at start-up the battery-backed position will be used.

See also:

`gnss_set_time`.

3.13 gnss_set_time

Sets the GPS time for the GNSS engine

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_time(
   const tInt year,
   const tInt month,
   const tInt day,
   const tInt hour,
   const tInt mins,
   const tInt secs,
   gnss_sat_type_t
);
```

Arguments:

`tInt year`: The year from 1996 to 2035.

`tInt month`: The month from 1 to 12.

`tInt day`: The day from 1 to 31.

`tInt hour`: The hour from 0 to 23.

`tInt mins`: The minute from 0 to 59.

`tInt secs`: The second from 0 to 59.

`gnss_sat_type_t sat_type`: GNSS constellation type.

Results:

`gnss_error_t`: Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` is the input time is not valid. If the operation was unsuccessful the GPS time remains the same as before.

Description:

This sets the constellation time. All constellation times differ from UTC Time by a fixed offset. This offset is available via the function gnss_get_utc_delta_time. This time will be used to reset

the real-time clock. It is necessary for the GNSS to have an accurate measure to time to give a short time to first fix. This time only need be accurate the nearest 15 minutes.

Note: This function is only relevant if the GNSS has not read the time from one of the satellites. Under normal circumstances the GNSS will read an accurate time from the real-time clock. It is only necessary to set the time if the real-time clock loses its battery backup power.

See also:

`gnss_set_pos`, `gnss_set_centre_freq`.

3.14 gnss_set_centre_freq

Sets the initial search frequency

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_centre_freq(const tDouble centre_freq);
```

Arguments:

tDouble:	The centre frequency in Hz for searching. This should be from -132000 to 132000.

Results:

gnss_error_t:	Returns GNSS_NO_ERROR if the operation was successful or GNSS_ERROR on failure.

Errors:

Returns GNSS_ERROR if the input frequency is not valid. If the operation was unsuccessful the centre frequency remains unchanged.

Description:

The frequency error is the error perceived on a satellite, either generated by Doppler shift on the moving satellite, or the inherent oscillator error. The GNSS uses this error to determine where to start searching for the first satellite.

The frequency need only to be set to the nearest 1KHz since the GNSS does wide searches (up to 9KHz) when looking for the first satellite. An accurate measure of the centre frequency will give a short time to first fix.

If no frequency is set the GNSS will use the previous local oscillator offset stored in battery-backed memory. This is the normal case - the centre frequency should only need to be set in exceptional circumstances e.g. if battery-backed memory is lost.

In case of successful operation, the new frequency will be stored in the battery-backed memory.

See also:

gnss_get_centre_freq.

3.15 gnss_set_freq_range

Sets the initial search range

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"
gnss_error_t gnss_set_freq_range(
   const tInt max_nco,
   const tInt min_nco
);
```

Arguments:

`tInt max_nco`:	The maximum frequency
`tInt min_nco`:	The minimum frequency.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input range is not valid. If the operation was unsuccessful the frequency range remains unchanged.

Description:

When acquiring satellites the GNSS will search over the range given. If no range is given, the GNSS searches over a default wide range from.-100 KHz to 100 KHz. In case of successful operation, the new frequency range will be stored in the battery-backed memory.

3.16 gnss_init_2_5_ppm_support

Initialize the enabling/disabling status for the 2.5ppm TCXO support

Synopsis:

```
#include "gnss_api.h"

void  gnss_init_2_5_ppm_support(const boolean_t );
```

Arguments:

`boolean_t`: ON/OFF status (TRUE=ON, FALSE=OFF)

Results:

2.5ppm TCXO support is enabled/disabled according to the input parameter

Errors:

None.

Description:

Enable/disable the 2.5ppm TCXO support inside the navigation stage.

Note: TCXO frequency accuracy and stability is important to guarantee the GNSS performances. Enabling 2.5ppm may cause degradation of GNSS performance. Make sure to follow all recommendations from ST reference design during the HW development.

3.17 gnss_set_2_5_ppm_support

Set the enabling/disabling status for the 2.5ppm TCXO support

Synopsis:

```
#include "gnss_api.h"

void  gnss_set_2_5_ppm_support(const boolean_t );
```

Arguments:

`boolean_t`: ON/OFF status (TRUE=ON, FALSE=OFF)

Results:

2.5ppm TCXO support is enabled/disabled according to the input parameter

Errors:

None.

Description:

Enable/disable the 2.5ppm TCXO support inside the navigation stage.

Note: TCXO frequency accuracy and stability is important to guarantee the GNSS performances. Enabling 2.5ppm may cause degradation of GNSS performance. Make sure to follow all recommendations from ST reference design during the HW development.

3.18 gnss_get_2_5_ppm_support

Get the status for the 2.5ppm TCXO support

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_get_2_5_ppm_support(void);
```

Arguments:

None.

Results:

`boolean_t`: ON/OFF status (TRUE=ON, FALSE=OFF)

Errors:

None.

Description:

Get the 2.5ppm TCXO support inside the navigation stage.

3.19 gnss_set_fix_rate

Sets the fix rate for the GNSS

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"
gnss_error_t gnss_set_fix_rate(const tDouble rate);
```

Arguments:

`tDouble`:	Position fixing interval for the GNSS (in seconds).

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input fix rate is not valid. If the operation was unsuccessful the fix rate remains the same as before.

Description:

Sets the rate of position fixing for the GNSS. This rate is normally set at a 1 second interval. If a fix rate other than the default is required then this must be done each time the system is reset. If the fix rate is set too high then the GNSS will fix as fast as it can, dependent on the speed of the available memory.

See also:

`gnss_get_fix_rate`

3.20 gnss_set_3D_dop_threshold

Sets the threshold for 3D position fix

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"
gnss_error_t gnss_set_3D_dop_threshold(
   const tDouble pdop,
   const tDouble hdop,
   const tDouble vdop,
   const tDouble gdop
);
```

Arguments:

`tDouble pdop`:	Maximum PDOP value for 3D fix.
`tDouble hdop`:	Maximum HDOP value for 3D fix.
`tDouble vdop`:	Maximum VDOP value for 3D fix.
`tDouble gdop`:	Maximum GDOP value for 3D fix.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if input DOPs are not valid. If the operation was unsuccessful the DOPs remain the same as before.

Description:

Sets the maximum allowable DOP (Dilution of Precision) levels for calculating a three dimensional fix. If the GNSS cannot calculate a three dimensional fix it will try to calculate a two dimensional fix.

Note: If any DOP values other than the default values are required then these must be setup after `gnss_init` is called each time the system is started.

Default values are PDOP=15, HDOP=12, VDOP=12 and GDOP=18.

See also:

`gnss_get_3D_dop_threshold`, `gnss_set_2D_dop_threshold`.

3.21 gnss_set_2D_dop_threshold

Sets the threshold for 2D position fix

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_2D_dop_threshold(
   const tDouble pdop,
   const tDouble hdop,
   const tDouble vdop,
   const tDouble gdop
);
```

Arguments:

`tDouble pdop`: Maximum PDOP value for 2D fix.

`tDouble hdop`: Maximum HDOP value for 2D fix.

`tDouble vdop`: Maximum VDOP value for 2D fix.

`tDouble gdop`: Maximum GDOP value for 2D fix.

Results:

`gnss_error_t`: Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if input DOPs are not valid. If the operation was unsuccessful the DOPs remain the same as before.

Description:

Sets the maximum allowable DOP levels for calculating a 2D fix. If the GNSS cannot calculate a 2D fix it will not calculate any fix.

Note: If any DOP values other than the default values are required then these must be setup after `gnss_init` is called each time the system is started.

Default values are PDOP=15, HDOP=12, VDOP=12 and GDOP=18.

See also:

`gnss_set_3D_dop_threshold` `gnss_get_2D_dop_threshold`.

3.22 gnss_set_elevation_mask_angle

Sets the minimum elevation for tracking a satellite

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_elevation_mask_angle(const tDouble mask_angle);
```

Arguments:

`tDouble`:	The minimum angle in degrees at which the GNSS will track a satellite. This angle should be 0 to 45 degrees.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input mask angle is not valid. If the operation was unsuccessful the mask angle remains the same as before.

Description:

This sets the minimum elevation angle at which the GNSS will track a satellite. This elevation mask angle will only come into effect once the GNSS has a valid position.

Note: If a value other than the default value is required then this must be setup after `gnss_init` is called each time the system is started. Also values greater than 15 degrees will prevent widespread use of 3D fixing.

The default value is 4.0 degrees. Usually it is set to 0.0 at every system startup (see the application `main.c` file)

See also:

`gnss_get_elevation_mask_angle`.

3.23 gnss_set_elevation_mask_angle_positioning

Sets the minimum elevation for satellite usage in the positioning stage.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_elevation_mask_angle_positioning(const tDouble
mask_angle);
```

Arguments:

`tDouble`:	The minimum angle in degrees at which the GNSS will use a satellite in the position solution. This angle should be 0 to 45 degrees.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input mask angle is not valid. If the operation was unsuccessful the mask angle remains the same as before.

Description:

This sets the minimum elevation angle at which the GNSS will use a satellite in the position solution. This elevation mask angle will only come into effect once the GNSS has a valid position.

Note: If a value other than the default value is required then this must be setup after `gnss_init` is called each time the system is started. Also values greater than 15 degrees will prevent widespread use of 3D fixing.

The default value is 1.0 degrees.

See also:

`gnss_get_elevation_mask_angle_positioning`.

3.24 gnss_set_diff_params

Sets the differential correction parameters

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_diff_params(const satid_t sat_id,
   const tInt iod,
   const tDouble de,
   const tDouble dr,
   const tDouble drr
);
```

Arguments:

`satid_t`:	The PRN number of the satellite.
`tInt`:	The issue number of the ephemeris data used to calculate the satellite position.
`tDouble de`:	The time of week of the correction in *s*.
`tDouble dr`:	The range correction in *m* for the satellite.
`tDouble drr`:	The rate of change of range correction for the satellite in ms^{-1}.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if input parameters are not valid. If the parameters are not accepted the GNSS will use previous differential corrections if they are still valid.

Description:

Sets up the differential correction parameters for the specified satellite PRN. These are added to the measured range before the satellite is used in the position solution. The GNSS will only start to use the range corrections if it has enough satellites with differential corrections available.

See also:

`gnss_set_diff_mode`.

3.25 gnss_reset_diff_params

Resets differential correction parameters for all satellites

Synopsis:

```
#include "gnss_api.h"

void gnss_reset_diff_params(void)
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Reset differential corrections parameters to default value for all satellites.

See also:

`gnss_set_diff_mode`.

3.26 gnss_diff_set_source_type

Sets the source type for differential corrections.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_diff_set_source_type(
   const gnss_diff_source_t diff_source
);
```

Arguments:

`gnss_diff_source_t`:	Source of differential corrections.

Results:

`gnss_error_t`:	Returns always `GNSS_NO_ERROR`.

Errors:

None.

Description:

Sets the current source type for differential corrections. Values allowed are: `DIFF_SOURCE_NONE`, `DIFF_SOURCE_WAAS`, `DIFF_SOURCE_RTCM` or `DIFF_SOURCE_AUTO`. It allows switching between different source types reinitializing differential corrections parameters every time the new source type if different from the current one.

See also:

`gnss_diff_get_source_type`.

3.27 gnss_diff_get_source_type

Gets current source type for differential corrections.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_diff_get_source_type(
   const gnss_diff_source_t *diff_source
);
```

Arguments:

`gnss_diff_source_t *`:	Pointer to the variable where the current source type will be stored.

Results:

`gnss_error_t`:	Returns always `GNSS_NO_ERROR`.

Errors:

None.

Description:

Allows getting current source type for differential corrections.

See also:

`gnss_diff_set_source_type`.

3.28 gnss_diff_set_RTCM_flag

Sets the RTCM status flag.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_diff_set_RTCM_flag( const boolean_t status_flag);
```

Arguments:

`boolean_t`: RTCM status flag.

Results:

`gnss_error_t`: Returns always `GNSS_NO_ERROR`.

Errors:

None.

Description:

Sets the RTCM availability flag. `TRUE` means the RTCM signal is available and `FALSE` means it is not available. This flag is used to manage the automatic switch between RTCM and SBAS (AUTO mode) when both systems are available.

See also:

`gnss_diff_get_RTCM_flag`.

3.29 gnss_diff_get_RTCM_flag

Gets the RTCM status flag.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_diff_get_RTCM_flag( boolean_t *status_flag);
```

Arguments:

`boolean_t *`: Pointer to RTCM status flag.

Results:

`gnss_error_t`: Returns always `GNSS_NO_ERROR`.

Errors:

None.

Description:

Gets the RTCM availability flag. `TRUE` means the RTCM signal is available and `FALSE` means it is not available. This flag is used to manage the automatic switch between RTCM and SBAS (AUTO mode) when both systems are available.

See also:

`gnss_diff_set_RTCM_flag`.

3.30 gnss_diff_rtcm_correction_available

Returns the RTCM correction availability status for a specific satellite.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_diff_rtcm_correction_available( const satid_t sat);
```

Arguments:

satid_t:	Satellite prn.

Results:

boolean_t:	Returns TRUE or FALSE according to the RTCM corrections availability for the input satellite.

Errors:

None.

Description:

This function allows checking if a specific satellite has been corrected with the RTCM differential correction.

3.31 gnss_diff_sbas_fast_correction_available

Returns the SBAS fast corrections availability status for a specific satellite.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_diff_sbas_fast_correction_available( const satid_t sat);
```

Arguments:

`satid_t`:	Satellite prn.

Results:

`boolean_t`:	Returns TRUE or FALSE according to the SBAS fast corrections availability for the input satellite.

Errors:

None.

Description:

This function allows checking if a specific satellite has been corrected with the SBAS fast differential correction.

3.32 gnss_diff_sbas_slow_correction_available

Returns the SBAS slow corrections availability status for a specific satellite.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_diff_sbas_slow_correction_available( const satid_t sat);
```

Arguments:

satid_t:	Satellite prn.

Results:

boolean_t:	Returns TRUE or FALSE according to the SBAS slow corrections availability for the input satellite.

Errors:

None.

Description:

This function allows checking if a specific satellite has been corrected with the SBAS slow differential correction.

3.33 gnss_diff_sbas_iono_correction_available

Returns the SBAS iono corrections availability status for a specific satellite.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_diff_sbas_iono_correction_available( const satid_t sat);
```

Arguments:

satid_t:	Satellite prn.

Results:

boolean_t:	Returns TRUE or FALSE according to the SBAS iono corrections availability for the input satellite.

Errors:

None.

Description:

This function allows checking if a specific satellite has been corrected with the SBAS iono differential correction.

3.34 gnss_set_ephemeris_params

Sets the satellite ephemeris data

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_ephemeris_params(
   const satid_t sat_id,
   const ephemeris_raw_t *ephemeris
);
```

Arguments:

`satid_t`:	The PRN number of the satellite.
`ephemeris_raw_t *`:	A pointer to a structure containing the ephemeris data for the satellite.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input parameters are not valid. If the operation is unsuccessful then ephemeris data remains unchanged.

Description:

Sets the ephemeris data for the given satellite.

3.35 gnss_ephemeris_predicted_buffer_init

Initializes the buffer for predicted ephemeris.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

void gnss_ephemeris_predicted_buffer_init(
   const ephemeris_raw_t *eph_buffer
);
```

Arguments:

`ephemeris_raw_t *`: A pointer to the predicted ephemeris buffer.

Results:

None.

Errors:

None.

Description:

Initializes the external buffer for predicted ephemeris storage. It should be used only if the STAGPS library is linked.

3.36 gnss_set_almanac_params

Sets the satellite almanac data

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_almanac_params(
   const satid_t sat_id,
   const almanac_raw_t *almanac
);
```

Arguments:

`satid_t`:	The PRN number of the satellite.
`almanac_raw_t *`:	A pointer to a structure containing the almanac parameters for the satellite.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input parameters are invalid. If the operation is unsuccessful then the almanac data remains unchanged.

Description:

Sets the almanac data for the given satellite.

3.37 gnss_set_iono_params

Sets the ionospheric data

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_iono_params( const iono_raw_t *iono, const
gnss_sat_type_t sat_type);
```

Arguments:

`iono_raw_t *`:	A pointer to the ionospheric parameters.
`sat_type`:	Satellite constellation type.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if it is called before GNSS initialization. If the operation is unsuccessful then the ionospheric data remains unchanged.

Description:

Sets the ionospheric model parameters.

3.38 gnss_set_utc_params

Sets the UTC parameters used

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_utc_params( const utc_raw_t *utc);
```

Arguments:

`utc_raw_t *`:	A pointer to the UTC time correction parameters.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if it is called before GNSS initialization or if the backup memory access fails. If the operation is unsuccessful then utc data remains unchanged.

Description:

Sets the UTC time correction parameters. The UTC correction parameters are used to correct from GPS time to UTC time.

3.39 gnss_get_utc_raw_data

Returns last UTC data package downloaded from the sky

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_get_utc_raw_data( utc_raw_t *)
```

Arguments:

utc_raw_t:	Pointer to the utc raw data structure

Results:

gnss_error_t:	Returns GNSS_NO_ERROR if the operation was successful or GNSS_ERROR on failure.

Errors:

Returns GNSS_ERROR if the API is called before the UTC module initialization. If the operation is unsuccessful then utc data remains unchanged.

Description:

Returns last UTC data downloaded from the sky. If new UTC data has been not yet downloaded from the sky since last system power up, the UTC data stored in the backup memory is returned if present otherwise an empty UTC data is returned. In such case the input structure is filled by zeroes.

3.40 gnss_clear_utc_raw_data

Clears the UTC raw data.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_get_utc_raw_data( void)
```

Arguments:

None.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the API is called before the UTC module initialization. If the operation is unsuccessful then utc data remains unchanged.

Description:

Clears the UTC raw data.

3.41 gnss_get_glonass_utc_params

Returns the UTC parameters from GLONASS constellation

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_get_glonass_utc_params( glonass_utc_raw_t * )
```

Arguments:

`glonass_utc_raw_t *`: A pointer to the glonass UTC time correction parameters.

Results:

`gnss_error_t`: Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

If the operation is unsuccessful then the area pointed to by utc remains unchanged.

Description:

The UTC correction parameters are used to correct from GLONASS time to UTC time. If no utc data is available then `utc->available` will be set to 0.

3.42 gnss_set_glonass_utc_params

Set UTC parameters for GLONASS constellation

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_set_glonass_utc_params( const glonass_utc_raw_t * )
```

Arguments:

`glonass_utc_raw_t *`:	A pointer to the glonass UTC time correction parameters.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if it is called before GNSS initialization or if the backup memory access fails. If the operation is unsuccessful then utc data remains unchanged.

Description:

Sets the UTC parameters for GLONASS constellation. The UTC correction parameters are used to correct from GLONASS time to UTC time.

3.43 gnss_glonass_get_satellite_slot

Provides the slot corresponding to a Glonass sat id.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_glonass_get_satellite_slot(
   const satid_t,
   tInt *
);
```

Arguments:

`satid_t`:	Glonass satellite id.
`tInt *`:	Slot number.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if no ephemeris or alamanc is available for the given sat id.

Description:

The slot number is an information provided in the GLONASS navigation messages. If the ephemeris or almanac is available, the slot number is extracted and provided to the caller. An error is returned if no data is available.

3.44 gnss_invalidate_glonass_utc_params

Invalidates the GLONASS UTC parameters inside the backup memory

Synopsis:

```
#include "gnss_api.h"

gnss_error_t     gnss_invalidate_glonass_utc_params( void);
```

Arguments:

None.

Results:

gnss_error_t:	Returns GNSS_NO_ERROR if the operation was successful or GNSS_ERROR on failure.

Errors:

Returns GNSS_ERROR if it is called before GNSS initialization or if the backup memory access fails. If the operation is unsuccessful then utc data remains unchanged.

Description:

Invalidate the UTC parameters for GLONASS constellation from the backup memory.

3.45 gnss_set_diff_mode

Sets the differential mode

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_diff_mode( const diff_mode_t mode);
```

Arguments:

`diff_mode_t`:	New differential mode.

Results:

`gnss_error_t`:	Returns always `GNSS_NO_ERROR`.

Errors:

Returns always `GNSS_NO_ERROR`.

Description:

Sets the differential operating mode for the GNSS engine. Available values are: `DIFF_AUTO`, `DIFF_ONLY` or `NO_DIFF`. In `DIFF_AUTO` mode the GNSS will decide when to switch in and out of differential mode. In `DIFF_ONLY` mode the GNSS will only calculate a fix when enough valid differential information is available. In `NO_DIFF` mode the GNSS will only calculate non-differential fixes.

See also:

`gnss_get_diff_mode`.

3.46 gnss_enable_sat

Enables the tracking of this satellite

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_enable_sat( const satid_t satid);
```

Arguments:

satid_t:	The PRN number of the satellite.

Results:

gnss_error_t:	Returns GNSS_NO_ERROR if the operation was successful or GNSS_ERROR on failure.

Errors:

Returns GNSS_ERROR if the input parameter is not valid. If the operation was unsuccessful the enable flag remains the same as before.

Description:

Enables a satellite that was previously disabled to be tracked again.

Note: This will not override the satellite health data transmitted from the satellite constellation. All satellites are enabled at each power up so it is necessary to disable unwanted satellites each time the system is started.

See also:

gnss_disable_sat, gnss_is_sat_enabled.

3.47 gnss_disable_sat

Disables a satellite from being tracked

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_disable_sat( const satid_t satid);
```

Arguments:

satid_t:	The PRN number of the satellite.

Results:

gnss_error_t:	Returns GNSS_NO_ERROR if the operation was successful or GNSS_ERROR on failure.

Errors:

Returns GNSS_ERROR if the input parameter is not valid. If the operation was unsuccessful the disable flag remains the same as before.

Description:

This disables a satellite from being used in the position solution. If the satellite is being tracked at the time of calling this procedure the GNSS will stop tracking the satellite immediately.

Note: This will not override the satellite health data transmitted from the satellite constellation. All satellites are enabled at each power up so it is necessary to disable unwanted satellites each time the system is started.

See also:

gnss_enable_sat, gnss_is_sat_enabled.

3.48 gnss_is_sat_enabled

Returns whether the satellite is enabled

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

boolean_t gnss_is_sat_enabled( const satid_t satid);
```

Arguments:

`satid_t`:	The PRN number of the satellite.

Results:

`boolean_t`:	Returns TRUE if the satellite is enabled, FALSE if not or if the input parameter is not valid.

Errors:

None.

Description:

Checks to see if the satellite is enabled.

Note: This will not override the satellite health data transmitted from the satellite constellation. All satellites are enabled at each power up so it is necessary to disable unwanted satellites each time the system is started.

See also:

`gnss_enable_sat`, `gnss_disable_sat`.

3.49 gnss_get_centre_freq

Returns the hardware local oscillator error

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_get_centre_freq( void);
```

Arguments:

None.

Results:

`tDouble`: the centre frequency in *Hz*.

Errors:

None.

Description:

Returns the battery-backed hardware local oscillator error, as calculated previously while performing GNSS fixes or set previously using gnss_set_centre_freq.

See also:

`gnss_set_centre_freq`.

3.50 gnss_get_fix_rate

Returns the fix rate

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_get_fix_rate( void);
```

Arguments:

None.

Results:

`tDouble`: The fix rate in *s*.

Errors:

None.

Description:

Returns the fix rate for the GNSS. This is the time between fixes in seconds.

See also:

`gnss_set_fix_rate`.

3.51 gnss_get_3D_dop_threshold

Returns the threshold for 3D positioning

Synopsis:

```
#include "gnss_api.h"

void gnss_get_3D_dop_threshold(
   tDouble *pdop,
   tDouble *hdop,
   tDouble *vdop,
   tDouble *gdop
);
```

Arguments:

`tDouble *pdop`: Pointer to maximum PDOP value for 3D fix.

`tDouble *hdop`: Pointer to maximum HDOP value for 3D fix

`tDouble *vdop`: Pointer to maximum VDOP value for 3D fix.

`tDouble *gdop`: Pointer to maximum GDOP value for 3D fix.

Results:

None.

Errors:

None.

Description:

Returns the maximum allowable DOP (Dilution of Precision) levels for calculating a three dimensional fix.

See also:

`gnss_get_2D_dop_threshold`, `gnss_set_3D_dop_threshold`.

3.52 gnss_get_2D_dop_threshold

Returns the threshold for 2D positioning

Synopsis:

```
#include "gnss_api.h"
void gnss_get_2D_dop_threshold(
   tDouble *pdop,
   tDouble *hdop,
   tDouble *vdop,
   tDouble *gdop
);
```

Arguments:

`tDouble *pdop`: Pointer to maximum PDOP value for 2D fix.

`tDouble *hdop`: Pointer to maximum HDOP value for 2D fix

`tDouble *vdop`: Pointer to maximum VDOP value for 2D fix.

`tDouble *gdop`: Pointer to maximum GDOP value for 2D fix.

Results:

None.

Errors:

None.

Description:

Returns the maximum allowable DOP (Dilution of Precision) levels for calculating a two dimensional fix.

See also:

`gnss_get_3D_dop_threshold`, `gnss_set_3D_dop_threshold`.

3.53 gnss_get_elevation_mask_angle

Returns the elevation mask angle

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_get_elevation_mask_angle( void);
```

Arguments:

None.

Results:

`tDouble`: Elevation mask angle in degrees.

Errors:

None.

Description:

Returns the elevation mask angle previously set in the GNSS. If no mask angle has been previously setup the default mask angle will be returned.

Note: This parameter will be reset each time gnss_init is called and so needs to be set up at start-up, unless the default is to be used.

See also:

`gnss_set_elevation_mask_angle`.

3.54 gnss_get_elevation_mask_angle_positioning

Returns the elevation mask angle used by positioning software.

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_get_elevation_mask_angle_positioning( void);
```

Arguments:

None.

Results:

`tDouble`: Elevation mask angle in degrees.

Errors:

None.

Description:

Returns the elevation mask angle previously set in the GNSS. If no mask angle has been previously setup the default mask angle will be returned.

Note: This parameter will be reset each time gnss_init is called and so needs to be set up at start-up, unless the default is to be used.

See also:

`gnss_set_elevation_mask_angle_positioning`.

3.55 gnss_get_diff_params

Returns the differential correction parameters

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_get_diff_params(
   const satid_t sat_id,
   tInt *iod,
   tDouble *de,
   tDouble *dr,
   tDouble *drr,
   boolean_t *available
);
```

Arguments:

`satid_t sat_id`:	The PRN number of the satellite.
`tInt *iod`:	The issue number of the ephemeris data used to calculate the satellite position.
`tDouble *de`:	The time of week of the correction in *s*.
`tDouble *dr`:	The range correction in *m* for the satellite.
`tDouble *drr`:	The rate of change of range correction for the satellite in ms^{-1}.
`boolean_t *available`:	Whether the differential data is available for this satellite.

Results:

`gnss_error_t`:	Returns always `GNSS_NO_ERROR`.

Errors:

None.

Description:

Returns the differential correction parameters for the specified satellite PRN. These are added to the measured range before the satellite is used in the position solution. The GNSS will only start to use the range corrections if it has enough satellites with differential corrections available.

See also:

`gnss_set_diff_params`.

3.56 gnss_get_sat_iode

Returns the satellite issue of data

Synopsis:

```
#include "gnss_api.h"

tInt gnss_get_sat_iode( const satid_t satid);
```

Arguments:

`satid_t`: The satellite number.

Results:

`tInt`: Returns the IODE value or -1.

Errors:

Returns -1 on error.

Description:

Returns the issue of data of the ephemeris for the specified satellite or -1 if the satellite ephemeris is not available. It returns -1 also if the input parameter is not valid or if the function is called before the GNSS initialization.

3.57 gnss_get_diff_mode

Returns the differential mode

Synopsis:

```
#include "gnss_api.h"

diff_mode_t gnss_get_diff_mode( void);
```

Arguments:

None.

Results:

`diff_mode_t`: Returns the differential mode.

Errors:

None.

Description:

Returns the differential operating mode for the GNSS engine. Expected values are: `NO_DIFF`, `DIFF_AUTO` or `DIFF_ONLY`. In `DIFF_AUTO` mode the GNSS will decide when to switch in and out of differential mode. In `DIFF_ONLY` mode the GNSS will only calculate a fix when enough valid differential information is available. In `NO_DIFF` mode the GNSS will only calculate non-differential fixes.

See also:

`gnss_get_diff_mode`.

3.58 gnss_get_sat_health

Returns the health state of the satellite

Synopsis:

```
#include "gnss_api.h"

tInt gnss_get_sat_health( const satid_t satid);
```

Arguments:

satid_t:	The PRN number of the satellite.

Results:

tInt:	Returns 0 for healthy, non-zero for unhealthy.

Errors:

None.

Description:

Returns the health of the satellite. The satellite is deemed to be unhealthy when either the almanac or the ephemeris data shows that it is unhealthy. Both must show that the satellite is healthy again before it will be used.

3.59 gnss_get_sat_pos_type

Returns the satellite position type

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

sat_pos_type_t gnss_get_sat_pos_type( const satid_t);
```

Arguments:

`satid_t`:	The PRN number of the satellite.

Results:

`sat_pos_type_t`:	Satellite position type.

Errors:

None.

Description:

Returns the current satellite position type.

3.60 gnss_get_sat_az_el

Returns the azimuth and elevation of a satellite

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_get_sat_az_el(
   const satid_t satid,
   tInt *az_p,
   tInt *el_p
);
```

Arguments:

`satid_t sat_id`:	The PRN number of the satellite.
`tInt *az_p`:	The azimuth of the satellite.
`tInt *el_p`:	The elevation of the satellite.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if provided `satid` is not valid or if position of specified satellite is not valid. If the operation is unsuccessful then the data pointed to by `az_p` and `el_p` are unspecified.

Description:

Returns the azimuth and elevation of the satellite in degrees. This is only available for satellites for which the system has either an ephemeris or almanac available.

Note: For satellites below the horizon the elevation will be returned as negative.

3.61 gnss_get_sats_visible

Returns information about the visible satellites

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

void gnss_get_sats_visible( sats_visible_list_t *sats_visible);
```

Arguments:

`sats_visible_list_t *`: A pointer to a structure containing the visible satellite information.

Results:

None.

Errors:

If the operation is unsuccessful then the area pointed to by sats_visible remains unchanged.

Description:

Returns a structure containing information about satellites visible from the present receiver position. A satellite is visible if it is above the current elevation mask.

3.62 gnss_get_sats_visible_scaled

Returns information about the visible satellites

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

void gnss_get_sats_visible_scaled( sats_visible_list_t *sats_visible);
```

Arguments:

`sats_visible_list_t *`: A pointer to a structure containing the visible satellite information.

Results:

None.

Errors:

If the operation is unsuccessful then the area pointed to by sats_visible remains unchanged.

Description:

Returns a structure containing information about satellites visible from the present receiver position. A satellite is visible if its scaled elevation is above the current elevation mask.

3.63 gnss_update_sats_visible

Trig an update of the visible sats list

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

void gnss_update_sats_visible ( void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

After the use of gnss_set_ephemeris_params (cf. §3.34) the sats visible list can be updated according to newly injected ephemeris.

3.64 gnss_get_sat_cn0

Returns satellite signal to noise ratio

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_get_sat_cn0( const satid_t sat_id);
```

Arguments:

`satid_t`: The PRN number of the satellite.

Results:

`tDouble`: Returns the signal to noise ratio for the satellite in dB.

Errors:

On an error the signal strength will be returned as 0.

Description:

Returns the signal to noise ratio for the satellite in a 1 Hz bandwidth. This number will be a number between 0 and 60 spending on the signal strength. This figure is calculated by measuring the signal plus noise in two different bandwidths and then calculating the signal to noise ratio in a 1Hz bandwidth.

3.65 gnss_sat_tracking_check

Checks if the satellite is tracked.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

boolean_t gnss_sat_tracking_check( const tInt tracker_state);
```

Arguments:

`tInt`:	Tracker state.

Results:

`boolean_t`:	Returns `TRUE` if the satellite is tracked, `FALSE` if it is not tracked.

Errors:

None.

Description:

Returns `TRUE` or `FALSE` according to the satellite tracking state.

3.66 gnss_get_sat_xyz_diff

Returns vector (x,y,z) from user position to satellite

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_get_sat_xyz_diff(
   const satid_t satid,
   tDouble *xdiff,
   tDouble *ydiff,
   tDouble *zdiff
);
```

Arguments:

`satid_t satid`:	The PRN number of the satellite
`tDouble *xdiff`:	vector x component.
`tDouble *ydiff`:	vector y component.
`tDouble *zdiff`:	vector z component.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input satellite id is not valid.

Description:

Returns components of vector (x,y,z) from user position to selected satellite.

3.67 gnss_get_ephemeris_params

Returns satellite ephemeris data

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_get_ephemeris_params(
   const satid_t satid,
   ephemeris_raw_t *ephemeris,
   boolean_t *available
);
```

Arguments:

`satid_t`:	The PRN number of the satellite.
`ephemeris_raw_t *`:	A pointer to a structure containing the ephemeris data for the satellite.
`boolean_t *`:	A pointer to a variable providing the availability of the ephemeris data for the satellite.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

If the operation is unsuccessful then the area pointed to by `ephemeris` remains unchanged.

Description:

Returns a structure containing the ephemeris parameters for a given satellite. If no ephemeris data is available for this satellite then `available` will be set to 0.

3.68 gnss_get_almanac_params

Returns satellite almanac data

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_get_almanac_params(
   const satid_t satid,
   almanac_raw_t *almanac,
   boolean_t *available

);
```

Arguments:

`satid_t`:	The PRN number of the satellite.
`almanac_raw_t *`:	A pointer to a structure containing the almanac data for the satellite.
`boolean_t *`:	A pointer to a variable providing the availability of the almanacs data for the satellite.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

If the operation is unsuccessful then the area pointed to by `almanac` remains unchanged.

Description:

Returns a structure containing almanac parameters for a given satellite. If no almanac data is available for this satellite then `available` will be set to 0.

3.69 gnss_get_ephemeris_sizeof

Returns satellite ephemeris data size.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

tInt gnss_get_ephemeris_sizeof ( const gnss_sat_type_t);
```

Arguments:

`gnss_sat_type_t`:	Satellite constellation type.

Results:

`tInt`:	Number of bytes of the ephemeris data.

Errors:

None.

Description:

Returns the number of bytes of the internal structure of ephemeris data related to the given constellation.

3.70 gnss_get_almanac_sizeof

Returns satellite almanac data size.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

tInt gnss_get_almanac_params( const gnss_sat_type_t);
```

Arguments:

`gnss_sat_type_t:`	Satellite constellation type.

Results:

`tInt:`	Number of bytes of the almanac data.

Errors:

None.

Description:

Returns the number of bytes of the internal structure of almanac data related to the given constellation.

3.71 gnss_eph_update_mask_copy_clear

Make a local copy and reset the updated ephemeris list.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

void gnss_eph_update_mask_copy_clear( tUInt *);
```

Arguments:

`tUInt *:` pointer to a mask where 1 bit is related to 1 satellite.

Results:

None.

Errors:

None.

Description:

Every new ephemeris received from a satellite set its corresponding bit to 1. This API returns a copy of the mask and reset this mask. The size of the mask varies according to the number of constellation enabled. 1 bit is dedicated to 1 satellite.

3.72 gnss_alm_update_mask_copy_clear

Make a local copy and reset the updated almanac list.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

void gnss_alm_update_mask_copy_clear( tUInt *);
```

Arguments:

`tUInt *:` pointer to a mask where 1 bit is related to 1 satellite.

Results:

None.

Errors:

None.

Description:

Every new almanac received from a satellite set its corresponding bit to 1. This API returns a copy of the mask and reset this mask. The size of the mask varies according to the number of constellation enabled. 1 bit is dedicated to 1 satellite.

3.73 gnss_get_iono_params

Returns ionospheric data

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_get_iono_params( iono_raw_t *iono, const
gnss_sat_type_t sat_type);
```

Arguments:

`iono_raw_t *`:	A pointer to the ionospheric parameters.
`gnss_sat_type_t`:	Satellite constellation type.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

If the operation is unsuccessful then the area pointed to by iono remains unchanged.

Description:

Returns a structure containing the ionospheric parameters used in the calculation of the ionospheric delay. If no ionospheric data is available then `iono->available` will be set to 0.

3.74 gnss_get_utc_params

Returns the UTC parameters used

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_get_utc_params( utc_raw_t *utc);
```

Arguments:

`utc_raw_t *`:	A pointer to the UTC time correction parameters.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

If the operation is unsuccessful then the area pointed to by utc remains unchanged.

Description:

The UTC correction parameters are used to correct from GPS time to UTC time. If no utc data is available then `utc->available` will be set to 0.

3.75 gnss_clear_all_almanacs

Clears all almanac data

Synopsis:

```
#include "gnss_api.h"

void gnss_clear_all_almanacs( void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Clears all the available almanac data from the backup memory. This ensures that the GNSS will do an autonomous start (i.e. It will have no information on satellites available).

See also:

`gnss_clear_all_ephems`.

3.76 gnss_clear_all_ephems

Clears all ephemeris data

Synopsis:

```
#include "gnss_api.h"

void gnss_clear_all_ephems( void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Clears all the available ephemeris data from backup memory. This ensures that the GNSS will do a warm start (i.e. The GNSS will have to read the ephemeris data from each satellite it tracks before a position fix will be available).

See also:

`gnss_clear_all_almanacs`.

3.77 gnss_conv_vel_to_course_speed

Converts velocity to course and speed (it is the horizontal 2D speed)

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_conv_vel_to_course_speed(
   const velocity_t *vel,
   tDouble *course,
   tDouble *speed
)
```

Arguments:

`velocity_t *vel`: The velocity to be converted.

`tDouble *course`: Returns the course in degrees.

`tDouble *speed`: Returns the speed in meters per second.

Results:

`gnss_error_t`: Returns always `GNSS_NO_ERROR`.

Errors:

None.

Description:

Converts the velocity from north and east to course and speed.

Example:

```
#include "gnss_api.h"
#include "gp_defs.h"

velocity_t *vel_p;
tDouble course;
tDouble speed;

gnss_fix_store();
gnss_fix_read_claim();
vel = gnss_fix_get_fil_vel();
gnss_conv_vel_to_course_speed(vel_p, &course, &speed);
gnss_fix_read_release();
```

See also:

`gnss_fix_get_fil_vel`.

3.78 gnss_conv_vel_to_course_speed_3D

Converts velocity to course and speed (it is the 3D speed)

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_conv_vel_to_course_speed_3D(
   const velocity_t *vel,
   tDouble *course,
   tDouble *speed
);
```

Arguments:

`velocity_t *vel`: The velocity to be converted.

`tDouble *course`: Returns the course in degrees.

`tDouble *speed`: Returns the speed in meters per second.

Results:

`gnss_error_t`: Returns always `GNSS_NO_ERROR`.

Errors:

None.

Description:

Converts the velocity from north and east and vertical to course and 3D speed.

Example:

```
#include "gnss_api.h"
#include "gp_defs.h"

velocity_t *vel_p;
tDouble course;
tDouble speed_3D;

gnss_fix_store();
gnss_fix_read_claim();
vel = gnss_fix_get_fil_vel();
gnss_conv_vel_to_course_speed_3D(vel_p, &course, &speed_3D);
gnss_fix_read_release();
```

See also:

`gnss_fix_get_fil_vel`.

3.79 gnss_cpu_clock_rate_hi

Returns the GPS time reference clock rate

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_cpu_clock_rate_hi( void);
```

Arguments:

None.

Results:

`tDouble`: clock rate in ticks per seconds.

Errors:

None.

Description:

Returns the clock rate of GPS reference time. This clock runs at approximately 1MHz.

3.80 gnss_powerdown_request

Prevents all NVM operations so that the chip may be safely powered down.

Synopsis:

```
#include "gnss_api.h"

void gnss_powerdown_request( void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Takes the NVM access semaphore, blocking all NVM operations. This is done to prevent the chip from being powered down whilst in the middle of a flash erase operation. It is available only if flash memory is used for GNSS data backup.

3.81 gnss_set_tracking_threshold

Changes the channel tracking threshold value

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_set_tracking_threshold( const tInt value);
```

Arguments:

`tInt`:	New channel tracking threshold value in dB

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

If the operation is unsuccessful the channel tracking threshold will remain unchanged.

Description:

Changes the channel tracking threshold value to the one specified by value.

See Also:

`gnss_get_tracking_threshold`.

3.82 gnss_init_tracking_threshold

Initialize the default channel tracking threshold inside the navigation stage.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_init_tracking_threshold( const tInt value);
```

Arguments:

`tInt`:	New channel tracking threshold value in dB

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

If the operation is unsuccessful the channel tracking threshold will remain unchanged.

Description:

Initialize the default channel tracking threshold value inside the navigation stage. At every restart, the navigation stage injects the default tracking threshold inside the tracking stage. When the gnss_init_tracking_threshold is called, the new tracking threshold will be operative only after the next restart of the GNSS engine. See gnss_set_tracking_threshold() for setting the tracking threshold on the fly.

See Also:

`gnss_get_tracking_threshold`.

3.83 gnss_get_tracking_threshold

Returns the channel tracking threshold and its range.

Synopsis:

```
#include "gnss_api.h"

void gnss_get_tracking_threshold(
   tInt *value,
   tInt *min,
   tInt *max
);
```

Arguments:

`tInt *value`:	Pointer to the variable where the tracking threshold will be stored.
`tInt *min`:	Pointer to the variable where the min tracking threshold will be stored.
`tInt *max`:	Pointer to the variable where the max tracking threshold will be stored.

Results:

None.

Errors:

None.

Description:

Returns the channel tracking threshold and the range limits within it could be set.

See Also:

`gnss_set_tracking_threshold`.

3.84 gnss_set_positioning_threshold

Changes the positioning threshold value

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_set_positioning_threshold ( const tInt value);
```

Arguments:

`tInt`:	New positioning threshold value in dB

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

If the operation is unsuccessful the positioning threshold will remain unchanged.

Description:

Changes the positioning threshold value to the one specified by value. Below the given value, a tracked satellite is not used to compute the position.

3.85 gnss_set_fde_status

Sets the FDE algorithm status.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

void gnss_set_fde_status( gnss_fde_status_t fde_status);
```

Arguments:

`gnss_fde_status_t`: New FDE status

Results:

None.

Errors:

None.

Description:

Turns ON/OFF the FDE algorithm according to the input parameter. Available values are: `FDE_STATUS_OFF` or `FDE_STATUS_ON`.

See Also:

`gnss_get_fde_status`.

3.86 gnss_get_fde_status

Gets the FDE algorithm status.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_fde_status_t gnss_get_fde_status( void);
```

Arguments:

None.

Results:

`gnss_fde_status_t`: Current FDE status

Errors:

None.

Description:

Returns the FDE algorithm ON/OFF status.

See Also:

`gnss_set_fde_status`.

3.87 gnss_get_noise_floor

Returns the noise floor value.

Synopsis:

```
#include "gnss_api.h"

tInt gnss_get_noise_floor( void);
```

Arguments:

None.

Results:

`tInt`: Returns the system noise floor.

Errors:

None.

Description:

Allows reading the estimated noise floor value.

3.88 gnss_turn_stop_detection_on

Allows turning ON/OFF stop detection algorithm.

Synopsis:

```
#include "gnss_api.h"

void gnss_turn_stop_detection_on( boolean_t status);
```

Arguments:

`boolean_t`:	New stop detection status

Results:

None.

Errors:

None.

Description:

Stop detection algorithm is turned OFF if the input parameter is `FALSE` or it is turned ON if the input parameter is `TRUE`. Stop detection algorithm is ON by default.

3.89 gnss_get_stop_detection_status

Get stop detection status.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_get_stop_detection_status( void);
```

Arguments:

None.

Results:

`boolean_t`: Stop detection status: ON = TRUE, OFF = FALSE

Errors:

None.

Description:

Returns stop detection status.

3.90 gnss_turn_walking_mode_on

Allows turning ON/OFF walking mode functionality.

Synopsis:

```
#include "gnss_api.h"

void gnss_turn_walking_mode_on( boolean_t status);
```

Arguments:

`boolean_t`: New turn walking mode status

Results:

None.

Errors:

None.

Description:

Walking mode is turned OFF if the input parameter is `FALSE` or it is turned ON if the input parameter is `TRUE`. Walking mode is OFF by default.

3.91 gnss_get_walking_mode_status

Get walking mode status.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_get_walking_mode_status( void);
```

Arguments:

None.

Results:

`boolean_t`: Walking mode status: ON = TRUE, OFF = FALSE

Errors:

None.

Description:

Returns walking mode status.

3.92 gnss_set_sf_recovery_status

Allows turning ON/OFF the subframe recovery functionality.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_sf_recovery_status( boolean_t status);
```

Arguments:

`boolean_t`: New subframe recovery mode status

Results:

None.

Errors:

None.

Description:

Subframe recovery algorithm is turned OFF if the input parameter is `FALSE` or it is turned ON if the input parameter is `TRUE`.

3.93 gnss_set_constellation_mask

Allows setting the constellations to be used for the satellite tracking.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_set_constellation_mask(
   const gnss_sat_type_mask_t mask
);
```

Arguments:

`gnss_sat_type_mask_t`: Constellations bit mask.

Results:

`gnss_error_t`: Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if this function is called before the `gnss_init()` call.

Description:

According to the input bit mask the selected constellations are enabled to be used for the satellite tracking. If the corresponding bit is set the constellation is enabled; if the corresponding bit is clear the constellation is disabled. The bit position for each constellation currently supported is: `GNSS_SAT_TYPE_GPS`, `GNSS_SAT_TYPE_GLONASS`, `GNSS_SAT_TYPE_QZSS_L1_CA`, `GNSS_SAT_TYPE_COMPASS`.

Example:

```
#include "gnss_api.h"

gnss_sat_type_mask_t orbit_list_selection = 0;

// Enables all supported constellations
MCR_SETBIT( orbit_list_selection, GNSS_SAT_TYPE_GPS);
MCR_SETBIT( orbit_list_selection, GNSS_SAT_TYPE_QZSS_L1_CA);
MCR_SETBIT( orbit_list_selection, GNSS_SAT_TYPE_GLONASS);

gnss_set_constellation_mask( orbit_list_selection);
```

3.94 gnss_get_constellation_mask

Get the constellations currently used for the satellite tracking.

Synopsis:

```
#include "gnss_api.h"

gnss_sat_type_mask_t gnss_get_constellation_mask( void)
```

Arguments:

None.

Results:

`gnss_sat_type_mask_t`: Current constellations bit mask.

Errors:

None.

Description:

This function returns a bit mask reporting all constellations currently enabled for the satellite tracking. If the corresponding bit is set the constellation is enabled; if the corresponding bit is clear the constellation is disabled. The bit position for each constellation currently supported is: `GNSS_SAT_TYPE_GPS`, `GNSS_SAT_TYPE_GLONASS`, `GNSS_SAT_TYPE_QZSS_L1_CA`, `GNSS_SAT_TYPE_COMPASS`.

See also:

`gnss_set_constellation_mask`.

3.95 gnss_set_constellation_usage_mask

Allows setting the constellations to be used for the position calculation.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_set_constellation_usage_mask(
   const gnss_sat_type_mask_t mask
);
```

Arguments:

`gnss_sat_type_mask_t`: Constellations bit mask.

Results:

`gnss_error_t`: Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if this function is called before the `gnss_init()` call.

Description:

According to the input bit mask the selected constellations are enabled to be used for the position calculation. If the corresponding bit is set the constellation is enabled; if the corresponding bit is clear the constellation is disabled. The bit position for each constellation currently supported is: `GNSS_SAT_TYPE_GPS`, `GNSS_SAT_TYPE_GLONASS`, `GNSS_SAT_TYPE_QZSS_L1_CA`, `GNSS_SAT_TYPE_COMPASS`.

Example:

```
#include "gnss_api.h"

gnss_sat_type_mask_t orbit_list_selection = 0;

// Enables all supported constellations
MCR_SETBIT( orbit_list_selection, GNSS_SAT_TYPE_GPS);
MCR_SETBIT( orbit_list_selection, GNSS_SAT_TYPE_QZSS_L1_CA);
MCR_SETBIT( orbit_list_selection, GNSS_SAT_TYPE_GLONASS);

gnss_set_constellation_mask( orbit_list_selection);
```

3.96 gnss_get_constellation_usage_mask

Get the constellations currently used for the position calculation.

Synopsis:

```
#include "gnss_api.h"

gnss_sat_type_mask_t gnss_get_constellation_usage_mask( void)
```

Arguments:

None.

Results:

`gnss_sat_type_mask_t`: Current constellations bit mask.

Errors:

None.

Description:

This function returns a bit mask reporting all constellations currently enabled for the position calculation. If the corresponding bit is set the constellation is enabled; if the corresponding bit is clear the constellation is disabled. The bit position for each constellation currently supported is: `GNSS_SAT_TYPE_GPS`, `GNSS_SAT_TYPE_GLONASS`, `GNSS_SAT_TYPE_QZSS_L1_CA`, `GNSS_SAT_TYPE_COMPASS`.

See also:

`gnss_set_constellation_usage_mask`.

3.97 gnss_dynamic_set_constellation_mask

Allows setting constellation mask at run-time.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_dynamic_set_constellation_mask(const
gnss_sat_type_mask_t)
```

Arguments:

`gnss_sat_type_mask_t:` constellation mask.

Results:

`gnss_errot_t`: Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input parameter is not correct.

Description:

This function allows setting the satellite constellations to be tracked and used for positioning. It allows switching from a constellation to another at run time. This functionality can be used for low power application, turning OFF constellations when not needed for positioning.

3.98 gnss_dynamic_set_top_n_bound

Sets the max number of satellites to be tracked.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_dynamic_set_top_n_bound( const tInt)
```

Arguments:

`tInt`:	max number of satellites to be tracked.

Results:

`gnss_errot_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input parameter is not correct.

Description:

This function allows setting the max number of satellites to be tracked. If the visibility list is bigger than selected max number of satellites, only satellites with higher elevation angle will be acquired and tracked. This functionality is supported to save power consumption reducing the number of tracked satellites.

3.99 gnss_notch_filter_enable

Allows setting the notch filter functionality.

Synopsis:

```
#include "gnss_api.h"

void gnss_notch_filter_enable(
   const gnss_sat_type_t sat_type,
   tInt frequency,
   tInt mode
)
```

Arguments:

`gnss_sat_type_t sat_type:`	Notch filter path
`tInt frequency:`	Starting frequency to search
`tInt mode:`	Set the operating mode

Results:

None.

Errors:

None.

Description:

This function set the NOTCH filter operating mode in two different ways.

Notch filter on GPS path always on:

gnss_notch_filter_enable(GNSS_SAT_TYPE_GPS,starting_frequency,1);

Notch filter on GLONASS path always on:

gnss_notch_filter_enable(GNSS_SAT_TYPE_GLONASS,starting_frequency,1);

Notch filter on GPS path in auto insertion mode:

gnss_notch_filter_enable(GNSS_SAT_TYPE_GPS,starting_frequency,2);

Notch filter on GLONASS in auto insertion mode:

gnss_notch_filter_enable(GNSS_SAT_TYPE_GLONASS,starting_frequency,2);

3.100 gnss_notch_filter_disable

Allows turning OFF the notch filter functionality.

Synopsis:

```
#include "gnss_api.h"

void gnss_notch_filter_disable(const gnss_sat_type_t sat_type)
```

Arguments:

`gnss_sat_type_t sat_type`: Notch filter path

Results:

None.

Errors:

None.

Description:

This function disable the NOTCH filter functionality.

Notch filter disable on GPS path:

gnss_notch_filter_disable(GNSS_SAT_TYPE_GPS);

Notch filter disable on GLONASS path:

gnss_notch_filter_disable(GNSS_SAT_TYPE_GLONASS);

3.101 gnss_notch_filter_get_status

Get the notch filter operation mode.

Synopsis:

```
#include "gnss_api.h"
OS_error_t gnss_notch_filter_get_status(
   const gnss_sat_type_t sat_type,
   tInt* kfreq_now,
   tInt* lock_en,
   tInt* pwr,
   tInt* ovfs,
   tShort * mode)
```

Arguments:

`gnss_sat_type_t sat_type:` Notch filter path

`tInt* kfreq_now:` Notch frequency estimation actual value [Hz]

`tInt* lock_en:` Frequency lock flag

`tInt* pwr:` Band Pass Filter power estimation

`tInt* ovfs:` Notch overflows flag

`tShort * mode:` Notch mode operation

Results:

None.

Errors:

None.

Description:

This function get the NOTCH filter operation mode, RFI frequency, power estimation.

Notch mode operation:

[1 → Always ON; 2 → Auto insertion mode; 0 → disabled;]

.

3.102 gnss_get_noise_floor_raw

Get the noise floor raw estimation for a selected constellation.

Synopsis:

```
#include "gnss_api.h"

tInt gnss_get_noise_floor_raw (gnss_sat_type_t sat_type)
```

Arguments:

`gnss_sat_type_t:` satellite constellation type

Results:

`tInt:` noise floor estimation for the selected constellation

Errors:

None.

Description:

This function returns the raw noise floor estimation for the selected constellation type.

3.103 gnss_lms_get_config

Get configured parameters for the Least Mean Square (LMS) stage.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_lms_get_config (gnss_lms_config_t *lms_config_ptr)
```

Arguments:

`gnss_lms_config_t *:` pointer to the LMS config data structure.

Results:

`gnss_errot_t`: Returns always `GNSS_NO_ERROR`.

Errors:

None.

Description:

Allow getting the current configuration parameters for the LMS stage in the positioning module.

3.104 gnss_lms_set_config

Set configuration parameters for the Least Mean Square (LMS) stage.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_lms_set_config (gnss_lms_config_t *lms_config_ptr)
```

Arguments:

`gnss_lms_config_t *:` pointer to the LMS config data structure.

Results:

`gnss_errot_t`: Returns always `GNSS_NO_ERROR`.

Errors:

None.

Description:

Allow configurations for the LMS stage in the positioning module. It is suggested to use the gnss_lms_get_config() function to fill the local lms_config structure before changing and setting it again with the gnss_lms_set_config() call.

Example:

```
{
   gnss_lms_config_t local_lms_config;

   if(gnss_lms_get_config(&local_lms_config) == GNSS_NO_ERROR)
   {
      // change needed parameters here
      gnss_lms_set_config(&local_lms_config);
   }
}
```

3.105 gnss_set_wls_runtime

Allow configuration for the Weighted Least Square (WLS) algorithm

Synopsis:

```
#include "gnss_api.h"

void gnss_set_wls_runtime(const boolean_t, const tDouble, const
tDouble)
```

Arguments:

`boolean_t:`	WLS algorithm ON/OFF status (TRUE=ON, FALSE=OFF).
`tDouble:`	Coefficient for changing the satellites measurement weighting in the position filter (allowed values from 0.1 up to 10.0). 0.1 means high acceptance of satellites in the position solution; 10.0=low acceptance of satellites in the position solution.
`tDouble:`	Coefficient for changing the satellites measurement acceptance threshold in the position filter (allowed values from 1.0 up to 10.0). 1.0 means strong satellites exclusion performed by FDE; 10.0 means relaxed satellites exclusion performed by FDE.

Results:

`None.`

Errors:

None.

Description:

Allow position filter configuration to enable/disable the WLS algorithm and to set parameters to control the satellites filtering process. Using his API it is possible to change the position filter behaviour affecting the output position. Suggested input parameters are 1.0 and 2.5 as in the following example.

Example:

```
gnss_set_wls_runtime(TRUE, 1.0, 2.5)
```

3.106 gnss_get_fix_config

Return the current configuration for the position fix estimation algorithm

Synopsis:

```
#include "gnss_api.h"

void gnss_get_fix_config(gnss_fix_config_t *curr_config);
```

Arguments:

`gnss_fix_config_t *:` pointer to the position fix configuration structure

Results:

Current configuration is returned in the input structure

Errors:

`None.`

Description:

Return the current configuration for the position fix estimation algorithm

3.107 gnss_set_fix_config

Set the current configuration for the position fix estimation algorithm

Synopsis:

```
#include "gnss_api.h"

void gnss_set_fix_config(gnss_fix_config_t *new_config);
```

Arguments:

`gnss_fix_config_t *:` pointer to the position fix configuration structure

Results:

New configuration is applied according to the input data structure

Errors:

`None.`

Description:

Set a new configuration for the position fix estimation algorithm

3.108 gnss_acquisition_set_operational_mode

Enable a specific acquisition engine operative mode for a specific constellation

Synopsis:

```
#include "gnss_api.h"

void gnss_acquisition_set_operational_mode( const gnss_sat_type_t,
const tInt );
```

Arguments:

gnss_sat_type_t: Satellite constellation type

`tInt`: specific acquisition mode ON/OFF status (1 = ON, 0 = OFF)

Results:

The new acquisition engine operative mode is enabled/disabled according to the input parameter

Errors:

None.

Description:

By default the GNSS acquisition engine is configured to work for all constellations. If a constellation requires a different configuration for the acquisition engine, this API can be used.

Note: At present this API supports only a special operative mode for QZSS constellation which is designed to reduce the current peaks during the acquisition phase of QZSS satellites.

3.109 gnss_init_high_dynamics_mode

Initialize the DSP high dynamic operative mode inside the navigation stage.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_init_high_dynamics_mode(const tInt );
```

Arguments:

`tInt`:	enable/disable status (0=disabled, 1=enabled)

Results:

The high dynamic operative mode is initialized as enabled or disabled according to the input parameter.

Errors:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Description:

When high dynamics operative mode is enabled the DSP measurement rate is increased. The DSP high dynamic is required when the GNSS fix rate is configured to be higher that 5Hz. This API doesn't enable/disable the feature on the fly; it enables/disables the feature in the navigation stage which injects the status inside the DSP stage during the GNSS restart phase.

3.110 gnss_set_high_dynamics_mode

Enable/disable the DSP high dynamic operative mode.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_set_high_dynamics_mode(const tInt on_off)
```

Arguments:

`tInt`: enable/disable status (0=disabled, 1=enabled)

Results:

The high dynamic operative mode is enabled or disabled according to the input parameter.

Errors:

`gnss_error_t`: Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Description:

When high dynamics operative mode is enabled the DSP measurement rate is increased. The DSP high dynamic is required when the GNSS fix rate is configured to be higher that 5Hz. This API enable/disable the feature on the fly.

3.111 gnss_get_high_dynamics_mode

Return the ON/OFF status for the DSP high dynamic operative mode.

Synopsis:

```
#include "gnss_api.h"

tInt gnss_get_high_dynamics_mode(void)
```

Arguments:

None

Results:

tInt: enable/disable status (0=disabled, 1=enabled)

Errors:

None

Description:

Return the ON/OFF status of the DSP high dynamic operative mode.

3.112 gnss_get_user_state

Returns the user position validity

Synopsis:

```
#include "gnss_api.h"

gnss_position_validity_t  gnss_get_user_state(void);
```

Arguments:

None

Results:

Return current validity of user position

Errors:

None

Description:

Return the status of current user position.

3.113 gnss_tracker_events_on_debug_on_off

Enable/disable the event messages from tracker stage in the debug stream

Synopsis:

```
#include "gnss_api.h"

void  gnss_tracker_events_on_debug_on_off(const boolean_t on_off)
```

Arguments:

`boolean_t`: ON/OFF status (TRUE=ON, FALSE=OFF)

Results:

Event messages from tracker stage are enabled/disabled according to the input parameter.

Errors:

None.

Description:

Tracker even messages are sent by default in the debug stream. Using this API it is possible to disable the tracker event flow over the debug port. It is needed, for example, when tracker messages needs to be sent over the NMEA port.

3.114 gnss_position_get_first_fix_timestamp

Return the CPU timestamp at which the first position fix is achieved

Synopsis:

```
#include "gnss_api.h"

OS_clock_t gnss_position_get_first_fix_timestamp(void);
```

Arguments:

None.

Results:

`OS_clock_t`: CPU timestamp at which the first position fix has been achived

Errors:

None.

Description:

Return the timestamp of the first valid position fix

3.115 gnss_sat_database_lock

Lock the access to the satellite management databases.

Synopsis:

```
#include "gnss_api.h"

void gnss_sat_database_lock (void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Lock the access to the ephemeris and almanac databases.

See Also:

gnss_sat_database_unlock

3.116 gnss_sat_database_unlock

Unlock the access to the satellite management databases.

Synopsis:

```
#include "gnss_api.h"

void gnss_sat_database_unlock (void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Unlock the access to the ephemeris and almanac databases.

See Also:

gnss_sat_database_lock

3.117 gnss_set_sat_list_size

Size GNSS library databases.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_sat_list_size (tU8);
```

Arguments:

`tU8:` `Size of the GNSS databases.`

Results:

None.

Errors:

`None.`

Description:

The size of GNSS databases is set according to the number of visible satellites. It needs to be set before calling the `gnss_init` API.

See Also:

gnss_get_sat_list_size

3.118 gnss_get_sat_list_size

Get the current satellite database size.

Synopsis:

```
#include "gnss_api.h"

tU8 gnss_get_sat_list_size (void);
```

Arguments:

None.

Results:

`tU8:` Number of element in database.

Errors:

`None.`

Description:

Returns the number of element in the GNSS library databases.

See Also:

gnss_set_sat_list_size

3.119 gnss_set_tcxo_config

Select a TCXO configuration

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_set_tcxo_config(const tInt);
```

Arguments:

`tInt:`	`Configuration number.`

Results:

None.

Errors:

`gnss_error_t:`	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Description:

Select the TCXO configuration amount 10 possible settings. The default value is suitable for regular operations. If it is required, the API must be called before the `gnss_init` API.

See Also:

gnss_get_tcxo_config_selector

3.120 gnss_get_tcxo_config_selector

Returns the current TCXO configuration.

Synopsis:

```
#include "gnss_api.h"

tInt gnss_get_tcxo_config_selector(void);
```

Arguments:

None.

Results:

tInt:	Current TCXO configuration.

Errors:

None.

Description:

Returns the TCXO configuration currently in use.

See Also:

gnss_set_tcxo_config_selector

3.121 gnss_set_freq_ramp_mode

Enable/disable the Kalman Filter frequency ramp mode

Synopsis:

```
#include "gnss_api.h"

void gnss_set_freq_ramp_mode(boolean_t status)
```

Arguments:

`boolean_t status`: ON/OFF status (TRUE=ON, FALSE=OFF)

Results:

None.

Errors:

None.

Description:

Enable/disable the Kalman Filter frequency ramp mode

3.122 gnss_get_error_status

Deprecated. Always return 0.

Synopsis:

```
#include "gnss_api.h"

tUInt gnss_get_error_status( void)
```

Arguments:

None.

Results:

tUint:	Always return 0

Errors:

None.

Description:

Deprecated. Allways return 0.

3.123 gnss_set_acq_enable

Allow to enable or disable the acquisition of satellites.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_acq_enable(boolean_t acq_enable);
```

Arguments:

`boolean_t acq_enable`: ON/OFF status (TRUE=ON, FALSE=OFF)

Results:

`None.`

Errors:

`None.`

Description:

Set the tracker in a mode where it can acquire or not satellites.

3.124 gnss_set_fast_CN0_mode

Configure the fast CN0 mode to avoid GNSS fix propagation at tunnel entrance.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_fast_CN0_mode(const boolean_t on_off);
```

Arguments:

`boolean_t on_off`: ON/OFF status (TRUE=ON, FALSE=OFF)

Results:

None.

Errors:

None.

Description:

Configure the fast CN0 mode to avoid GNSS fix propagation at tunnel entrance.

3.125 gnss_get_fast_CN0_mode_status

Get the current fast CN0 mode on/off status.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_get_fast_CN0_mode_status(void);
```

Arguments:

None.

Results:

boolean_t: ON/OFF status (TRUE=ON, FALSE=OFF)

Errors:

None.

Description:

Get the current fast CN0 mode on/off status.

3.126 gnss_set_high_acc_mode

Set the kalman positioning high_acc_mode.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_set_high_acc_mode(const tInt status, const tInt th);
```

Arguments:

`tInt status`: high acc status (TRUE=ON, FALSE=OFF)

`tInt th`: velocity threshold in kmph.

Results:

`None.`

Errors:

Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Description:

Set the kalman positioning velocity filter to consider the threshold for high velocity conditions.

3.127 gnss_get_high_acc_mode

Get the kalman positioning high_acc_mode.

Synopsis:

```
#include "gnss_api.h"

tInt gnss_get_high_acc_mode(void);
```

Arguments:

None.

Results:

tInt:	ON/OFF status (TRUE=ON, FALSE=OFF)

Errors:

None.

Description:

Get the kalman positioning high_acc_mode.

3.128 gnss_flags_update_mask_setbit

Set a bit in the nav manager. Used to indicate which iono parameter has been set.

Synopsis:

```
#include "gnss_api.h"

void gnss_flags_update_mask_setbit( const tUInt bit);
```

Arguments:

`tUInt`	Number of bit to set. FLAGS_UPDATE_BIT_IONO_GPS = 0, FLAGS_UPDATE_BIT_IONO_GALILEO = 1, FLAGS_UPDATE_BIT_IONO_COMPASS = 2, FLAGS_UPDATE_BIT_GALILEO_GGTO = 3,

Results:

`None`

Errors:

`None.`

Description:

See gnss_set_iono_params.

3.129 gnss_flags_update_mask_clearbit

Clear a bit in the nav manager corresponding to the iono param.

Synopsis:

```
#include "gnss_api.h"

void gnss_flags_update_mask_clearbit( const tUInt bit);
```

Arguments:

`tUInt`	Number of bit to clear. FLAGS_UPDATE_BIT_IONO_GPS = 0, FLAGS_UPDATE_BIT_IONO_GALILEO = 1, FLAGS_UPDATE_BIT_IONO_COMPASS = 2, FLAGS_UPDATE_BIT_GALILEO_GGTO = 3,

Results:

`None`

Errors:

`None.`

Description:

Clear the bit corresponding to the iono param reset.

3.130 gnss_flags_update_mask_get_clear

Clear a bitmask in the nav manager corresponding to the iono param.

Synopsis:

```
#include "gnss_api.h"

tUInt gnss_flags_update_mask_get_clear( const tUInt flags_mask);
```

Arguments:

`tUInt`	Bitmask to clear. FLAGS_UPDATE_BIT_IONO_GPS = bit 0, FLAGS_UPDATE_BIT_IONO_GALILEO = bit 1, FLAGS_UPDATE_BIT_IONO_COMPASS = bit 2, FLAGS_UPDATE_BIT_GALILEO_GGTO = bit 3,

Results:

`tUInt`	`Value of the bits set in the flags_mask.`

Errors:

`None.`

Description:

Clear the bits corresponding to the iono param reset, and return the previous value of the bits.

3.131 gnss_flags_update_mask_reset

Reset the iono param bit mask in the nav manager.

Synopsis:

```
#include "gnss_api.h"

void gnss_flags_update_mask_reset( void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Reset the iono param bit mask. Indicates that no iono params have been set.

3.132 gnss_set_rtc_calibration_mode

Set the rtc calibration mode.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_rtc_calibration_mode ( const boolean_t);
```

Arguments:

`boolean_t`	RTC calibration is OFF – FALSE, RTC calibration is ON - TRUE.

Results:

`None.`

Errors:

`None.`

Description:

The RTC calibration consists in evaluating the 32kHz oscillator real frequency, in order to compensate the RTC clock. This improves the accuracy of the RTC.

3.133 gnss_set_ephemeris_updated_callback

Register a function to get NVM ephemeris update notification.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_ephemeris_updated_callback (const
gnss_ephemeris_updated_callback_t);
```

Arguments:

`gnss_ephemeris_updated_callback_t` Callback function.

Results:

`None.`

Errors:

`None.`

Description:

Each time the GNSS Lib update the ephemeris in NVM, the callback is called with the satellite id as parameter. This function can only be called once since there can be only one function registered. The STAGPS™ registers its own function. It must be called before gnss_start (cf. chapter 3.9) is called.

The prototype of the function

3.134 gnss_set_initvisiblesatslist_available_callback

Register a function to get a notification when the first visible list is available.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_initvisiblesatslist_available_callback (const
gnss_initvisiblesatslist_available_callback_t);
```

Arguments:

`gnss_initvisiblesatslist_available_callback_t` Callback function.

Results:

None.

Errors:

None.

Description:

At startup, some computations are required to build the visible list up. Once it is built, the callback function is called. This function can only be called once since there can be only one function registered. The STAGPS™ registers its own function. It must be called before gnss_start (cf. chapter 3.9) is called.

4 GNSS Time Management Functions

4.1 gnss_init_rtc

Initializes the real time clock.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_init_rtc(
   const rtc_switch_t rtc_switch,
   gnss_time_t *rtc_gnss_time,
   clock_t cpu_time,
   tInt rtc_accuracy
);
```

Arguments:

`rtc_switch_t`:	status of RTC
`gnss_time_t *`:	gnss time parameter for rtc
`clock_t`:	cpu time parameter for rtc
`tInt`:	rtc accuracy expressed in ms.

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input time value in not valid.

Description:

This function sets the global rtc parameters to the values supplied in the function arguments. It allows preventing the usage of the internal RTC HW by the GNSS library. This function should be used when an external RTC is used instead of the embedded one.

The accuracy parameter is used in the following way:

- `rtc_accuracy` less than 50 means accurate (it is good for fixing);
- `rtc_accuracy` bigger than or equal to 50 means rtc not accurate (it is not good enough for fixing).

4.2 gnss_rtc_get_time

Returns the time read directly on the RTC.

Synopsis:

```
#include "gnss_api.h"

void gnss_rtc_get_time(
   gnss_time_t *gnss_time,
   clock_t *cpu_time,
   rtc_status_t *rtc_status,
   time_validity_t *time_validity
);
```

Arguments:

`gnss_time_t *`:	The RTC time returned as gnss time (week and tow).
`clock_t *`:	CPU time at which the RTC has been read.
`rtc_status_t *`:	RTC status.
`time_validity_t *`:	Time validity.

Results:

None.

Errors:

None.

Description:

Allows reading the RTC time. In addition also the CPU time when the hardware has been read (it could be used to compensate delay introduced by reading procedure), the RTC status and the time validity are returned.

4.3 gnss_set_rtt_tps

Sets the RTT counter ticks per second.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_rtt_tps(tDouble tps);
```

Arguments:

`tDouble`: calibrate ticks per second

Results:

Set the accurate ticks per second for the RTT time extrapolation.

Errors:

None.

Description:

Allows setting the accurate ticks per second value to be used for the RTT time extrapolation. This function should be used to increase the RTC time accuracy if a calibrated value of the RTC/RTT ticks per second is available.

4.4 gnss_set_rtc_rtt_mode

Allow the Real Time Timer usage.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_rtc_rtt_mode(void);
```

Arguments:

None

Results:

None.

Errors:

None.

Description:

Allow the Real Time Timer usage aside the Real Time Clock Unit. This is the default mode on STA2062, STA2064 and STA2065 platforms. The API must be called before `gnss_init`_API.

4.5 gnss_set_rtc_only_mode

Disable the usage of the Real Time Timer.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_rtc_only_mode (void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Disable the usage of the Real Time Timer. Only the Real Time Clock Unit is in use. This is the default setting on STA8089, STA8089 and STA8090 platforms. The API must be called before `gnss_init` API.

4.6 gnss_rtc_write

Set the RTC or RTC/RTT timers.

Synopsis:

```
#include "gnss_api.h"

void gnss_rtc_write (
   const gnss_time_reference_t,
   const time_validity_t,
   const boolean_t,
   const tDouble
);
```

Arguments:

`gnss_time_reference_t:`	Time to be set to RTC or RTC/RTT.
`time_validity_t:`	Time validity of the constellation.
`boolean_t:`	Force the update. TRUE = Force, FALSE = Let the API decide according to time validity.
`tDouble:`	UTC delta time to first parameter.

Results:

None.

Errors:

None.

Description:

Set the RTC or RTC/RTT timers.

4.7 gnss_rtc_read

Read the RTC time.

Synopsis:

```
#include "gnss_api.h"

void gnss_rtc_read (
   gnss_time_t *,
   OS_clock_t *,
   rtc_status_t *,
   time_validity_t*,
   gnss_sat_type_t *,
   tDouble *
);
```

Arguments:

`gnss_time_t *:`	Time read from RTC.
`OS_clock_t *:`	Corresponding CPU time.
`rtc_status_t *:`	Accuracy of the returned RTC time.
`time_validity_t:`	Time validity of the constellation.
`gnss_sat_type_t *:`	Corresponding constellation.
`tDouble *:`	UTC delta time.

Results:

Returns the parameters of the RTC clock.

Errors:

None.

Description:

Returns the RTC clock value, the corresponding CPU time and constellation related information.

4.8 gnss_rtc_data_invalidate

Force the RTC backup data invalidation at startup.

Synopsis:

```
#include "gnss_api.h"

void gnss_rtc_data_invalidate(void);
```

Arguments:

None

Results:

The RTC backup data (in the backup memory) is invalidated at the next GNSS restart.

Errors:

None

Description:

When the real time clock for GNSS is implemented using the RTT counter (instead of RTC counter) the backup memory is used to store time information which is needed to extrapolate the current accurate time (combining the RTT value with stored timestamps). If for some reasons the backup data is not correctly updated (it can happen when backup data is stored in RAM memory and it is saved/reloaded by the application at every system power OFF/ON) it may be required to invalidate the RTC backup data in order to avoid a wrong evaluation of the current time.

4.9 gnss_get_utc_time

Converts the UTC time (in seconds) in hours, minutes, seconds and milliseconds

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_get_utc_time(
   const tDouble utc_time,
   tInt *hours,
   tInt *mins,
   tInt *secs,
   tInt *msecs
);
```

Arguments:

`tDouble utc_time`:	UTC time in seconds to be converted
`tInt *hours`:	Pointer to hours
`tInt *mins`:	Pointer to minutes
`tInt *secs`:	Pointer to seconds
`tInt *msecs`:	Pointer to milliseconds

Results:

`gnss_error_t`:	Returns always `GNSS_NO_ERROR`.

Errors:

Returns always `GNSS_NO_ERROR`.

Description:

Converts the input UTC seconds in hours, minutes, seconds and milliseconds.

4.10 gnss_get_utc_delta_time

Returns the UTC to GPS time correction

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_get_utc_delta_time( void);
```

Arguments:

None.

Results:

`tDouble`:	time correction for converting between UTC time and GPS time.

Errors:

None.

Description:

This returns the number of seconds difference between UTC time and GPS time. UTC time can be found by subtracting this delta correction from GPS time. If UTC delta time is available in backup memory it will be returned otherwise a default value (15 seconds) is returned.

4.11 gnss_get_utc_delta_time_validity

Returns validity of the UTC delta time

Synopsis:

```
#include "gnss_api.h"

gnss_utc_delta_time_validity_t gnss_get_utc_delta_time_validity(void)
```

Arguments:

None.

Results:

Current UTC delta time validity is returned.

Errors:

None.

Description:

UTC delta time validity could be not valid until an updated value is downloaded from the sky. The UTC data is sent over the satellite navigation message every 12.5 minutes. At startup the UTC delta time is initialized with the value stored in the backup memory, if present. Using the API it is possible to get the current status for the UTC delta time validity.

4.12 gnss_utc_gps_get_tau_g

Return the UTC tau_g value

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_utc_gps_get_tau_g( const gnss_time_t *, tDouble *);
```

Arguments:

`gnss_time_t *:` pointer to current gnss time structure

`tDouble *:` pointer to tag_g parameter

Results:

Tau_g parameter is updated with current value

Errors:

Return GNSS_ERROR if the tau_g parameter is not available (e.g. it has been not yet downloaded from the sky)

Description:

Tau_g is the time difference, at nanoseconds level, between GPS time and UTC time. It is mainly used in accurate timing applications. The API allows getting the current value for tag_g parameter as soon as it has been successfully downloaded from the sky.

4.13 gnss_set_gps_utc_delta_time_default

Set the default value for the UTC delta time

Synopsis:

```
#include "gnss_api.h"

void gnss_set_gps_utc_delta_time_default(const tInt);
```

Arguments:

`tInt :` UTC delat time default value

Results:

UTC default delta time is set according to the input parameter

Errors:

`None.`

Description:

UTC delta time is the time distance (in seconds) between GPS time and UTC time. It is downloaded from the satellites navigation messages every 12.5 minutes. This parameter is also stored and kept update inside the backup memory. In case the backup memory is lost (or in case the parameter has been not yet saved) the default value is applied at startup. NOTE the UTC delta time knowledge is important to report the correct UTC time starting from GPS time (e.g. timestamp inside the NMEA messages like in GGA and RMC sentences). If the UTC delta in not available in backup memory and if the default value in not updated, the UTC time reporting will be wrong until the data is downloaded from the sky.

4.14 gnss_get_gps_utc_delta_time_default

Return the default value for the UTC delta time

Synopsis:

```
#include "gnss_api.h"

tInt gnss_get_gps_utc_delta_time_default(void);
```

Arguments:

`void`

Results:

`tInt :` current default value of the UTC delta time

Errors:

`None.`

Description:

Return the current value for the UTC default delta time.

4.15 gnss_get_time_valid

Returns the GPS time status

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_get_time_valid( void)
```

Arguments:

None

Results:

`boolean_t`: GPS time validity.

Errors:

None.

Description:

Returns TRUE if the GPS time is valid or FALSE if not.

4.16 gnss_get_time_validity

Returns the current GPS time validity

Synopsis:

```
#include "gnss_api.h"
#include "tacker_defs.h"

time_validity_t gnss_get_time_validity( void);
```

Arguments:

None.

Results:

`time_validity_t`: Returns current GPS time validity.

Errors:

None.

Description:

Returns the current GPS time validity. See section 13.17.1 for available values.

4.17 gnss_get_date

Converts GPS time format to calendar format

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_get_date(
   const tInt week,
   const tDouble tow,
   tInt * year,
   tInt * month,
   tInt * day
);
```

Arguments:

`tInt week`:	The GPS week number
`tDouble tow`:	The time of week
`tInt * year`:	The year
`tInt * month`:	The month
`tInt * day`:	The day of the month

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the date is not valid, i.e. it does not represent a real date.

Description:

Converts GPS format time to calendar format - day, month and year.

4.18 gnss_date_time_valid

Checks whether a date is valid

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_date_time_valid(
   const tInt year,
   const tInt month,
   const tInt day,
   const tInt hours,
   const tInt mins,
   const tInt secs
);
```

Arguments:

`tInt year`:	The year
`tInt month`:	The month
`tInt day`:	The day of the month
`tInt hours`:	The number of hours
`tInt mins`:	The number of minutes
`tInt secs`:	The number of seconds

Results:

`boolean_t`:	Returns TRUE if the date and time are valid, FALSE if they are not.

Errors:

None.

Description:

Used to check the validity of a date in calendar format.

4.19 gnss_date_time_to_gps_time

Coverts date and time to GPS time.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_date_time_to_gps_time(
   const tInt year,
   const tInt month,
   const tInt day,
   const tInt hours,
   const tInt mins,
   const tInt secs,
   gnss_time_t *gnss_time
);
```

Arguments:

`tInt year`:	Year
`tInt month`:	Month
`tInt day`:	Day
`tInt hours`:	Hours
`tInt mins`:	Minutes
`tInt secs`:	Seconds
`gnss_time_t *gnss_time`:	pointer to gnss time structure

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the input time value in not valid.

Description:

The date and time (supplied by the function arguments) will be converted to GPS time values and stored in the structure pointed to by `gnss_time`. The function will fail and return `GNSS_ERROR` if the values supplied in the function arguments are found to be invalid.

4.20 gnss_time_get_validity

Get GPS time validity.

Synopsis:

```
#include "gnss_api.h"

time_validity_t gnss_time_get_validity( gnss_sat_type_t sat_type);
```

Arguments:

`gnss_sat_type_t`: Constellation type (GPS, GLONASS etc.)

Results:

`time_validity_t`: Returns time validity for the given constellation type.

Errors:

None.

Description:

Allows getting the current validity of a constellation time. See section 13.17.1 for possible values.

4.21 gnss_time_now

Get current GPS time.

Synopsis:

```
#include "gnss_api.h"

gnss_time_t gnss_time_now( gnss_sat_type_t sat_type);
```

Arguments:

`gnss_sat_type_t`:	Constellation type (GPS, GLONASS etc.)

Results:

`gnss_time_t`:	Returns the constellation time.

Errors:

None.

Description:

Allows getting the current constellation time.

4.22 gnss_time_to_mtb_time

Convert constellation reference time into MTB time.

Synopsis:

```
#include "gnss_api.h"

master_timebase_t gnss_time_to_mtb_time(
   const gnss_sat_type_t sat_type,
   const gnss_time_t *ref_time
);
```

Arguments:

`gnss_sat_type_t`:	Constellation type (GPS,GLONASS etc.)
`gnss_time_t *`:	Constellation reference time.

Results:

`master_timebase_t`:	Master Time Base value corresponding to the constellation reference time.

Errors:

None.

Description:

Allows converting the constellation reference time (e.g. gps time, glonass time etc.) into the Master Time Base time.

4.23 gnss_time_to_utc_time

Convert constellation time into UTC time.

Synopsis:

```
#include "gnss_api.h"

gnss_time_t gnss_time_to_mtb_time(
   const gnss_time_t gnss_time_in,
   const gnss_sat_type_t sat_type
);
```

Arguments:

`gnss_time_t:`	Constellation time
`gnss_sat_type_t:`	Constellation type (GPS,GLONASS etc.)

Results:

`gnss_time_t:`	UTC time value corresponding to the constellation time.

Errors:

None.

Description:

Allows converting the constellation time (e.g. gps time, glonass time etc.) into UTC time.

4.24 gnss_time_then

Update the constellation reference time to the CPU timestamp.

Synopsis:

```
#include "gnss_api.h"

gnss_time_t gnss_time_then(
   const OS_clock_t cpu_timestamp,
   gnss_sat_type_t sat_type
);
```

Arguments:

`OS_clock_t`:	CPU timestamp.
`gnss_sat_type_t`:	Constellation type (GPS,GLONASS etc.)

Results:

`gnss_time_t`;	the constellation reference time updated to the CPU timestamp in input.

Errors:

None.

Description:

Allows moving ahead the constellation reference time to the CPU time stamp provided in input.

4.25 gnss_time_lminus

Subtract two TOW.

Synopsis:

```
#include "gnss_api.h"

tow_t gnss_time_lminus( const tow_t tow1, const tow_t tow2);
```

Arguments:

`tow_t tow1`: First Time Of Week

`tow_t tow2`: Second Time Of Week

Results:

`tow_t`: Returns the difference between inputs.

Errors:

None.

Description:

Allows the time difference evaluation between two Time Of Week in input. Result is provided in seconds between –half_week and +half_week.

4.26 gnss_time_lplus

Add a constant to a TOW.

Synopsis:

```
#include "gnss_api.h"

tow_t gnss_time_lplus( const tow_t tow1, const tDouble time_value);
```

Arguments:

`tow_t`:	Time Of Week
`tDouble`:	time to be added

Results:

`tow_t`:	Returns the sum between inputs.

Errors:

None.

Description:

Allows adding a constant to a Time Of Week.

4.27 gnss_time_minus

Subtract two GPS time.

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_time_minus( const gnss_time_t *t1, const gnss_time_t *t2);
```

Arguments:

`gnss_time_t *t1`: Pointer to the first GPS time

`gnss_time_t *t2`: Pointer to the second GPS time

Results:

`tDouble`: Returns the difference between two input GPS time.

Errors:

None.

Description:

Allows the time difference evaluation between two GPS time in input. Result is provided in seconds.

4.28 gnss_time_plus

Add a constant to a GPS time.

Synopsis:

```
#include "gnss_api.h"

gnss_time_t gnss_time_plus( const gnss_time_t *t1, const tDouble
delta);
```

Arguments:

`gnss_time_t *`:	Pointer to a GPS time.
`tDouble`:	time to be added

Results:

`gnss_time_t *`:	Sum of inputs.

Errors:

None.

Description:

Allows adding a constant to a GPS time.

4.29 gnss_set_min_max_week_number

Set the minimum and maximum week numbers

Synopsis:

```
#include "gnss_api.h"

void gnss_set_min_max_week_number(tInt , tInt );
```

Arguments:

`tInt:` minimum week number

`tInt:` maximum week number

Results:

Minimum and maximum week numbers are configured inside the GNSS library

Errors:

`None.`

Description:

Week number is broadcasted by GPS satellite in the navigation data stream over a 10-bit field. It means that week number wraps every 1024 weeks. The GPS library implements an algorithm to decode the correct week number along time. The algorithm works relaying on the knowledge of the minimum week number. It is able to decode the correct week number in an about 20 years time frame starting from the minimum week number (configured inside the GNSS library). After about 20 years from the minimum week number the GPS week number is no more correctly decoded. In such conditions the positioning functionality of the GPS software is not compromised but the reported date could be wrong.

Note: To guarantee that GPS software is able to decode the correct week number (and so to report the correct date info) as long as possible in the GPS product life, the minimum week number (configured in the GPS library) must be moved ahead along the years.

This API allows setting the minimum week number to allow the correct decoding of the GPS week number data. The maximum week number setting is not used by the decoding algorithm but it is used only for time validity checks.

Note: a time validity check failure may have serious impacts on the navigation and positioning performances, for this reason the max week number must be moved ahead each time the minimum week number is moved ahead.

4.30 gnss_get_min_max_week_number

Return the minimum and maximum week numbers

Synopsis:

```
#include "gnss_api.h"

void gnss_get_min_max_week_number(tInt * , tInt *);
```

Arguments:

`tInt *:` pointer to minimum week number

`tInt *:` pointer to maximum week number

Results:

Current minimum and maximum week numbers are returned through the input pointers

Errors:

`None.`

Description:

Return the current setting for minimum and maximum week numbers.

4.31 gnss_rtc_set_alarm

Deprecated. Does nothing.

Synopsis:

```
#include "gnss_api.h"

void gnss_rtc_set_alarm(tUInt delay_secs);
```

Arguments:

`tUInt *:` delay in seconds

Results:

`None.`

Errors:

`None.`

Description:

Deprecated. Does nothing.

4.32 gnss_get_tracker_time_now

Returns the current tracker time.

Synopsis:

```
#include "gnss_api.h"

OS_clock_t gnss_get_tracker_time_now( void);
```

Arguments:

None.

Results:

OS_clock_t *:	The current CPU time in microseconds.

Errors:

None.

Description:

Returns the current tracker time.

4.33 gnss_time_get_master

Returns the time of the master constellation.

Synopsis:

```
#include "gnss_api.h"

gnss_sat_type_t gnss_time_get_master(void);
```

Arguments:

None.

Results:

gnss_sat_type_t:	Satellite constellation type.

Errors:

None.

Description:

Returns the time of the master constellation.

4.34 gnss_time_get_aux

Returns the time of the auxiliary constellation.

Synopsis:

```
#include "gnss_api.h"

gnss_sat_type_t gnss_time_get_aux(void);
```

Arguments:

None.

Results:

gnss_sat_type_t:	Satellite constellation type.

Errors:

None.

Description:

Returns the time of the auxiliary constellation.

4.35 gnss_time_get_validity_raw

Returns the time validity for the requested constellation.

Synopsis:

```
#include "gnss_api.h"

time_validity_t gnss_time_get_validity_raw( const gnss_sat_type_t
sat_type);
```

Arguments:

`gnss_sat_type_t:`	Satellite constellation type.

Results:

`time_validity_t:`	Time validity for the requested constellation.

Errors:

Returns `NO_TIME` if there is no time reference available for the requested constellation .

Description:

Returns the time validity for the requested constellation.

5 GNSS Fix Related Functions

5.1 gnss_fix_store

Forces a copy to be made of the last fix data

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_store(void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

This function forces a copy to be made of all the coherent fix related information. This can be then accessed via the function calls in this section. This information will stay available until `gnss_fix_store()` is called again.

Example:

```
#include "gnss_api.h"
#include "gp_defs.h"
tInt week;
tDouble time;

gnss_fix_store(); /* Force a copy of all fix information */
gnss_fix_read_claim(); /* Claim the fix data */
if(gnss_get_fix_status() != NO_FIX)
{
   gnss_fix_get_time(&week, &time);
}
gnss_fix_read_release(); /* Release the fix data */
```

See also:

`gnss_fix_read_claim`, `gnss_fix_read_release`.

5.2 gnss_fix_read_claim

Secure access to the GNSS library for reading fix data.

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_read_claim(void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

This function allows multiple tasks to access the gnss fix information. Once `gnss_fix_read_claim()` has been called all further calls to `gnss_fix_store()` will be blocked until `gnss_fix_read_release()` is called by all tasks which have claimed the gnss fix information.

Used to protect the GNSS library from errors that may otherwise occur when multiple tasks are accessing the library. The function should be called before using any of the GNSS fix related functions, to ensure that the library data remains correct and consistent.

See also:

`gnss_fix_read_release`.

5.3 gnss_fix_read_release

Release access to the GNSS library fix data.

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_read_release(void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

The function `gnss_fix_read_release()` is used to release access to the GNSS library following operations on the fix data, allowing other tasks to store GNSS fixes and use the fix data.

See also:

`gnss_fix_read_claim`.

5.4 gnss_fix_get_time

Reads the GPS time of the stored fix

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_time(tInt *week, tDouble *time, OS_clock_t
*cpu_time);
```

Arguments:

`tInt *`:	The current week number.
`tDouble *`:	The current GPS time in seconds.
`OS_clock_t *`:	The current CPU time in microseconds at this GPS time.

Results:

None.

Errors:

None.

Description:

Returns the current GPS week and GPS time. GPS time rolls over at Saturday 11:59 pm. GPS week is related to the number of weeks since midnight January 5 1980.

5.5 gnss_fix_get_time_sat_type

Reads the type of satellite constellation at which is referred the time in fix data.

Synopsis:

```
#include "gnss_api.h"

gnss_sat_type_t gnss_fix_get_time_sat_type (void);
```

Arguments:

None.

Results:

`gnss_sat_type_t`	Satellite constellation.

Errors:

None.

Description:

Returns the type of satellite constellation at which the fix time is referred.

5.6 gnss_fix_get_time_data

Reads the GPS time, CPU timestamp and MTB timestamp of the stored fix data.

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_time_data(tInt *week, tDouble *time, OS_clock_t
*cpu_time, master_timebase_t *mtb_time);
```

Arguments:

`tInt *`:	The current week number.
`tDouble *`:	The current GPS time in seconds.
`OS_clock_t *`:	The current CPU time in microseconds at this GPS time.
`master_timebase_t *`:	The Master Time Base when satellites measurements are sampled.

Results:

None.

Errors:

None.

Description:

Returns the current GPS week, GPS tow, the CPU timestamp and the MTB timestamp at which the satellites measurements, used for the GNSS fix, have been sampled. GPS time rolls over at Saturday 11:59 pm. GPS week is related to the number of weeks since midnight January 5 1980.

Note: this is an extesion of the gnss_fix_get_time() API to provide also the MTB timestamp of the measurements.

5.7 gnss_fix_get_time_validity

Reads the GPS time validity of the stored fix

Synopsis:

```
#include "gnss_api.h"

time_validity_t gnss_fix_get_time_validity(void);
```

Arguments:

None

Results:

`time_validity_t`: time validity of fix.

Errors:

None.

Description:

Returns the time validity of current position fix.

5.8 gnss_fix_get_fil_pos

Returns the filtered receiver position

Synopsis:

```
#include "gnss_api.h"

position_t * gnss_fix_get_fil_pos( void);
```

Arguments:

None.

Results:

`position_t *`: Return the pointer to the filtered `position_t`.

Errors:

None.

Description:

Returns the filtered position data for the GNSS fix.

See also:

`gnss_fix_store`.

5.9 gnss_fix_get_fil_ecef_pos

Returns the filtered receiver position in ECEF format

Synopsis:

```
#include "gnss_api.h"

ECEF_pos_t * gnss_fix_get_fil_ecef_pos( void);
```

Arguments:

None.

Results:

`ECEF_pos_t *`: Return the pointer to the ECEF_pos_t position

Errors:

None.

Description:

Returns the filtered position data for the GNSS fix in ECEF format.

See also:

`gnss_fix_store`.

5.10 gnss_fix_get_raw_pos

Returns the not filtered (Least Mean Square - LMS) receiver position

Synopsis:

```
#include "gnss_api.h"

position_t * gnss_fix_get_raw_pos( void);
```

Arguments:

None.

Results:

`position_t *`: Return the pointer to the not filtered `position_t`.

Errors:

None.

Description:

Returns the not filtered position data for the GNSS fix.

See also:

`gnss_fix_store`.

5.11 gnss_fix_get_fil_vel

Returns the filtered receiver velocity

Synopsis:

```
#include "gnss_api.h"

velocity_t *gnss_fix_get_fil_vel(void)
```

Arguments:

None.

Results:

`velocity_t *`: pointer to a structure containing velocity information.

Errors:

None.

Description:

Returns the pointer to filtered velocity data for the GNSS fix.

See also:

`gnss_fix_store`.

5.12 gnss_fix_get_fil_ecef_vel

Returns the filtered receiver velocity in ECEF format

Synopsis:

```
#include "gnss_api.h"

ECEF_vel_t *gnss_fix_get_fil_ecef_vel(void)
```

Arguments:

None.

Results:

`ECEF_vel_t *`: pointer to a structure containing velocity in ECEF format.

Errors:

None.

Description:

Returns the pointer to filtered velocity data for the GNSS fix in ECEF format.

See also:

`gnss_fix_store`.

5.13 gnss_fix_get_raw_vel

Returns the not filtered (Least Mean Square – LMS) receiver velocity

Synopsis:

```
#include "gnss_api.h"

velocity_t *gnss_fix_get_raw_vel(void)
```

Arguments:

None.

Results:

`velocity_t *`: pointer to a structure containing velocity information.

Errors:

None.

Description:

Returns the pointer to the not filtered velocity data for the GNSS fix.

See also:

`gnss_fix_store`.

5.14 gnss_fix_get_position_covariance

Returns position covariance

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_position_covariance(
   tDouble * N_cov,
   tDouble *E_cov,
   tDouble *V_cov
);
```

Arguments:

`tDouble *N_cov`: Latitude covariance.

`tDouble *E_cov`: Longitude covariance.

`tDouble *V_cov`: Vertical covariance.

Results:

None.

Errors:

None.

Description:

Returns the Kalman Filter covariance for position.

See also:

`gnss_fix_store`.

5.15 gnss_fix_get_velocity_covariance

Returns velocity covariance

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_velocity_covariance(
   tDouble * N_cov,
   tDouble *E_cov,
   tDouble *V_cov
);
```

Arguments:

`tDouble *N_cov`: Velocity North component covariance.

`tDouble *E_cov`: Velocity East component covariance.

`tDouble *V_cov`: Velocity Vertical component covariance.

Results:

None.

Errors:

None.

Description:

Returns the Kalman Filter covariance for velocity.

See also:

`gnss_fix_store`.

5.16 gnss_fix_get_dops

Returns the DOP for the current fix

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_dops(
   tDouble *pdop,
   tDouble *hdop,
   tDouble *vdop,
   tDouble *gdop
);
```

Arguments:

`tDouble *pdop`: The Position DOP for the current fix.

`tDouble *hdop`: The Horizontal DOP for the current fix.

`tDouble *vdop`: The Vertical DOP for the current fix.

`tDouble *gdop`: The Geometric DOP for the current fix.

Results:

Returns the DOPs (Dilution of Precision) for the fix.

Errors:

None.

Description:

Returns the DOPs (Dilution of Precision) for the current fix.

See also:

`gnss_fix_store`.

5.17 gnss_fix_get_raw_pos_dops

Returns the DOP for the not filtered (LMS) position of the current fix

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_raw_pos_dops(
   tDouble *pdop,
   tDouble *hdop,
   tDouble *vdop,
   tDouble *gdop
);
```

Arguments:

`tDouble *pdop`: The not filtered Position DOP for the current fix.

`tDouble *hdop`: The not filtered Horizontal DOP for the current fix.

`tDouble *vdop`: The not filtered Vertical DOP for the current fix.

`tDouble *gdop`: The not filtered Geometric DOP for the current fix.

Results:

Returns the DOPs (Dilution of Precision) for the not filtered position.

Errors:

None.

Description:

Returns the DOPs (Dilution of Precision) for the not filtered position in the current fix.

See also:

`gnss_fix_store`.

5.18 gnss_fix_get_pos_status

Returns the status of the current GNSS fix

Synopsis:

```
#include "gnss_api.h"

fix_status_t gnss_fix_get_pos_status( void);
```

Arguments:

None.

Results:

`fix_status_t`: status of the GNSS fix.

Errors:

None.

Description:

Returns the status of the current GNSS fix. Either `NO_FIX`, `FIX_2D` or `FIX_3D`.

See also:

`gnss_fix_store`.

5.19 gnss_fix_get_raw_pos_status

Returns the status of the not filtered position of the current GNSS fix

Synopsis:

```
#include "gnss_api.h"

fix_status_t gnss_fix_get_raw_pos_status( void);
```

Arguments:

None.

Results:

`fix_status_t`: status of the not filtered position (LMS fix).

Errors:

None.

Description:

Returns the status of the current LMS fix. Either `NO_FIX`, `FIX_2D` or `FIX_3D`.

See also:

`gnss_fix_store`.

5.20 gnss_fix_get_diff_status

Returns the differential status

Synopsis:

```
#include "gnss_api.h"

diff_status_t gnss_fix_get_diff_status( void);
```

Arguments:

None.

Results:

`diff_status_t`: Returns differential fix status.

Errors:

None.

Description:

Returns whether the system is using differential data or not. Either `DIFF_ON` or `DIFF_OFF`.

See also:

`gnss_fix_store`.

5.21 gnss_fix_get_excluded_sats_doppl

A satellite may be excluded for the GNSS fix computing because of its too large velocity residual. This function returns the velocity exclusion information for the specified channel.

Synopsis:

```
#include "gnss_api.h"

Boolean_t gnss_fix_get_excluded_sats_doppl(
   const tInt chan_id,
   gnss_exclusion_type_t *exclusion_type,
);
```

Arguments:

`tInt`:	Channel ID
`gnss_exclusion_type_t *`:	Returns the type of exclusion for the satellite being tracked on that channel. The exclusion type is either GNSS_RAIM_EXCLUSION, GNSS_FDE_EXCLUSION or GNSS_NO_EXCLUSION.

Results:

`Boolean_t`:	Return the velocity exclusion indication.

Errors:

None.

Description:

The velocity exclusion indication and the exclusion type are returned for the specified channel id. If the channel id is invalid the return value is FALSE and the exclusion type is GNSS_NO_EXCLUSION.

5.22 gnss_fix_get_excluded_sats_range

A satellite may be excluded for the GNSS fix computing because of its too large range residual. This function returns the range exclusion information for the specified channel.

Synopsis:

```
#include "gnss_api.h"

Boolean_t gnss_fix_get_excluded_sats_range(
   const tInt chan_id,
   gnss_exclusion_type_t *exclusion_type,
);
```

Arguments:

`tInt`:	Channel ID
`gnss_exclusion_type_t *`:	Returns the type of exclusion for the satellite being tracked on that channel. The exclusion type is either GNSS_RAIM_EXCLUSION, GNSS_FDE_EXCLUSION or GNSS_NO_EXCLUSION.

Results:

`Boolean_t`:	Return the range exclusion indication.

Errors:

None.

Description:

The range exclusion indication and the exclusion type are returned for the specified channel id. If the channel id is invalid the return value is FALSE and the exclusion type is GNSS_NO_EXCLUSION.

5.23 gnss_fix_get_geoid_msl

Returns the mean sea level offset

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_fix_get_status( void);
```

Arguments:

None.

Results:

`tDouble`: mean sea level offset from the WGS84 geoid.

Errors:

None.

Description:

Returns the mean sea level offset from the WGS84 geoid in meters.

See also:

`gnss_fix_store`.

5.24 gnss_fix_get_num_sats_excluded

Some satellites may be excluded because of their too large range or velocity residuals. This function returns the number of sats excluded for the current GNSS fix.

Synopsis:

```
#include "gnss_api.h"

tInt gnss_fix_get_num_sats_excluded( void);
```

Arguments:

None.

Results:

`tInt`: number of excluded satellites for the current GNSS fix.

Errors:

None.

Description:

Returns the number of satellites excluded for the current GNSS fix.

See also:

`gnss_fix_store`.

5.25 gnss_fix_get_num_sats_used

Returns the number of sats used in the current GNSS fix

Synopsis:

```
#include "gnss_api.h"

tInt gnss_fix_get_num_sats_used( void);
```

Arguments:

None.

Results:

`tInt`: number of satellites used in the current GNSS fix.

Errors:

None.

Description:

Returns the number of satellites used in the current GNSS fix.

See also:

`gnss_fix_store`.

5.26 gnss_fix_get_raw_pos_sats

Returns the number of sats used in the current not filtered position

Synopsis:

```
#include "gnss_api.h"

tInt gnss_fix_get_raw_pos_sats( void);
```

Arguments:

None.

Results:

`tInt`: number of satellites used in the current LMS fix.

Errors:

None.

Description:

Returns the number of satellites used in the current LMS fix.

See also:

`gnss_fix_store`.

5.27 gnss_fix_get_raw_measurements

Returns the raw fix measurements

Synopsis:

```
#include "gnss_api.h"

raw_measurement_list_t *gnss_fix_get_raw_measurements(void);
```

Arguments:

None.

Results:

`raw_measurement_list_t *`: list of raw measurements used in the fix.

Errors:

None.

Description:

Returns the measurement data from the satellites used in the current fix.

See also:

`gnss_fix_store`.

5.28 gnss_fix_get_position_residual

Gets the position residual value for the specified channel.

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_position_residual(
   const tInt chan_id,
   tDouble *residual,
   boolean_t *valid
);
```

Arguments:

`tInt`:	Channel ID
`tDouble *`:	Returns the position residual value in meters for the satellite being tracked on that channel.
`boolean_t *`:	Returns the satellite's position residual validity flag

Results:

None.

Errors:

None.

Description:

The position residual value for the specified channel and its validity are returned. If the channel id is invalid or not kept for the GNSS fix computing then a residual of zero will be returned with a FALSE validity flag.

5.29 gnss_fix_get_position_residual_used

Gets the position residual value for the specified channel. Even if the satellite is discarded for the GNSS fix computing because of a too large residual, the position residual is available.

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_position_residual_used(
   const tInt chan_id,
   tDouble *residual,
   boolean_t *valid
);
```

Arguments:

`tInt`:	Channel ID
`tDouble *`:	Returns the position residual value in meters for the satellite being tracked on that channel.
`boolean_t *`:	Returns the satellite's position residual validity flag

Results:

None.

Errors:

None.

Description:

The position residual value for the specified channel and its validity is returned. If the channel id is invalid then a residual of zero will be returned with a FALSE validity flag.

5.30 gnss_fix_get_position_rms_residual

Gets the position rms residual value.

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_fix_get_position_rms_residual( void);
```

Arguments:

None.

Results:

`tDouble`: Returns the position RMS residual value in meters.

Errors:

None.

Description:

This function returns the position RMS residual value.

5.31 gnss_fix_get_velocity_residual

Gets the velocity residual value for the specified channel.

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_velocity_residual(
   const tInt chan_id,
   tDouble *residual,
   boolean_t *valid
);
```

Arguments:

`tInt`:	Channel ID
`tDouble *`:	Returns the velocity residual value in meters per second for the satellite being tracked on that channel
`boolean_t *`:	Returns the satellite's velocity residual validity flag

Results:

None.

Errors:

None.

Description:

The velocity residual value for the specified channel and its validity is returned. If the channel id is invalid or not kept for the GNSS fix computing then a residual of zero will be returned with a FALSE validity flag.

5.32 gnss_fix_get_velocity_rms_residual

Gets the velocity rms residual value.

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_fix_get_velovity_rms_residual( void);
```

Arguments:

None.

Results:

`tDouble`:	Returns the velocity RMS residual value in meters per second.

Errors:

None.

Description:

This function returns the velocity RMS residual value.

5.33 gnss_fix_get_stopped_duration

Returns the length of time the fix has remained stationary.

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_stopped_duration( tInt *stopped_duration);
```

Arguments:

`tInt *`: pointer to stopped duration value to be returned (secs)

Results:

None.

Errors:

None.

Description:

This function returns the length of time that the fix has remained static. This value can then be used for example, to determine if the receiver has stopped moving and is stationary.

5.34 gnss_fix_get_pos_algo

Gets the current positioning algorithm.

Synopsis:

```
#include "gnss_api.h"

pos_algo_t gnss_fix_get_pos_algo( void);
```

Arguments:

None.

Results:

pos_algo_t:	current positioning algorithm.

Errors:

None.

Description:

This function returns the current positioning algorithm. This could be either Least Mean Squares (LMS) algorithm or a Kalman Filter Algorithm (KALMAN).

5.35 gnss_fix_get_kf_config_status

Return the positioning algorithms status

Synopsis:

```
#include "gnss_api.h"

tUShort  gnss_fix_get_kf_config_status();
```

Arguments:

None

Results:

Return the position algorithms status from last fix (bit field mask)

bit0=walking mode (0=off 1=on)

bit1 = stop detection (0=off 1=on)

bit2=frequency ramp (0=not detected 1=detected)

bit3=velocity estimator (0=single model 1=multiple model)

bit4=velocity filter (0=fast 1=slow)

bit5=fde status (0=off 1=on)

Errors:

None

Description:

Report the status of dynamic algorithms in the positioning stage.

5.36 gnss_fix_is_chan_used

Gets the channel usage condition

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_fix_is_chan_used( const tInt chan);
```

Arguments:

tInt:	tracker channel ID

Results:

boolean_t:	Returns TRUE is the specific channel is used for the position fix; FALSE if not used.

Errors:

None.

Description:

This function returns if a specific channel is used in the current position fix.

5.37 gnss_fix_get_clock_drift

Gets the clock drift

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_fix_get_clock_drift(void)
```

Arguments:

None

Results:

`tDouble`: clock drift estimation.

Errors:

None.

Description:

This function returns the clock drift estimation computed by kalman filter.

5.38 gnss_fix_get_clock_offset

Gets the clock offset

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_fix_get_clock_offest(void)
```

Arguments:

None

Results:

`tDouble`: clock offset estimation [m].

Errors:

None.

Description:

This function returns the clock offset estimation computed by kalman filter. It represents the error in meters on the satellite range due to the clock frequency drift.

5.39 gnss_fix_get_ehpe

Gets the Estimated Horizontal Position Error (EHPE)

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_fix_get_ehpe(void)
```

Arguments:

None

Results:

`tDouble`: estimated horizontal position error [m].

Errors:

None.

Description:

This function returns the horizontal position error estimation of the current gnss position.

5.40 gnss_fix_get_error_ellipse

Gets the horizontal position error ellipse parameters

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_error_ellipse (math_ellipse_t *)
```

Arguments:

None

Results:

`math_ellipse_t *`: pointer to the horizontal position error ellipse parameters.

Errors:

None.

Description:

This function returns the parameters (East semiaxis, North semiaxis and inclination angle) of the horizontal position error ellipse.

5.41 gnss_fix_get_position_all_covariance

Return position covariance

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_position_all_covariance(tDouble *, tDouble *,tDouble
*, tDouble *,tDouble *,tDouble *);
```

Arguments:

`tDouble *:`	pointer to position North covariance
`tDouble *:`	pointer to position East covariance
`tDouble *:`	pointer to position Vertical covariance
`tDouble *:`	pointer to position North-East covariance
`tDouble *:`	pointer to position East-Vertical covariance
`tDouble *:`	pointer to position North-Vertical covariance

Results:

Return position covariance from last fix

Errors:

None

Description:

Return the elements of the position covariance matrix.

5.42 gnss_fix_get_velocity_all_covariance

Return velocity covariance

Synopsis:

```
#include "gnss_api.h"

void gnss_fix_get_velocity_all_covariance(tDouble *, tDouble *,tDouble
*, tDouble *,tDouble *,tDouble *);
```

Arguments:

`tDouble *:`	pointer to velocity North covariance
`tDouble *:`	pointer to velocity East covariance
`tDouble *:`	pointer to velocity Vertical covariance
`tDouble *:`	pointer to velocity North-East covariance
`tDouble *:`	pointer to velocity East-Vertical covariance
`tDouble *:`	pointer to velocity North-Vertical covariance

Results:

Return velocity covariance from last fix

Errors:

None

Description:

Return the elements of the velocity covariance matrix.

5.43 gnss_fix_get_position_q_matrix_diag

Return position Q matrix diagonal

Synopsis:

```
#include "gnss_api.h"

tDouble* gnss_fix_get_position_q_matrix_diag(void *);
```

Arguments:

`tDouble *:` pointer to the Q matrix diagonal array (5 elements array)

Results:

Return the position Q matrix diagonal from last fix

Errors:

None

Description:

Return the elements of the position Q matrix.

5.44 gnss_fix_get_velocity_q_matrix_diag

Return velocity Q matrix diagonal

Synopsis:

```
#include "gnss_api.h"

tDouble* gnss_fix_get_velocity_q_matrix_diag(void *);
```

Arguments:

`tDouble *:` pointer to the Q matrix diagonal array (5 elements array)

Results:

Return the velocity Q matrix diagonal from last fix

Errors:

None

Description:

Return the elements of the velocity Q matrix.

5.45 gnss_fix_get_glonass_path_delay

Gets the extra path introduced by RF on GLONASS signal.

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_fix_get_glonass_path_delay (void)
```

Arguments:

None

Results:

`tDouble:` GLONASS path delay estimation [m].

Errors:

None.

Description:

The RF chain adds on GLONASS signal an extra delay compared to the GPS signal; it is estimated and compensated by the GNSS positioning stage. This function allows getting the estimation of GLONASS path delay.

5.46 gnss_fix_get_measure_requested_time

Gets the CPU Time of the measurement.

Synopsis:

```
#include "gnss_api.h"

OS_clock_t gnss_fix_get_measure_requested_time(void);
```

Arguments:

None

Results:

`OS_clock_t:` CPU Time of the last measurement used for the fix.

Errors:

None.

Description:

Return the measurement CPU Time used for the fix computing.

5.47 gnss_fix_get_measure_get_data_time

Gets the CPU Time of the current fix calculation.

Synopsis:

```
#include "gnss_api.h"

OS_clock_t gnss_fix_get_measure_get_data_time(void);
```

Arguments:

None

Results:

`OS_clock_t:` CPU Time of the current fix computing.

Errors:

None.

Description:

Return the CPU Time of the current fix calculation.

5.48 gnss_fix_get_fix_available_time

Gets the CPU Time of the last fix.

Synopsis:

```
#include "gnss_api.h"

OS_clock_t gnss_fix_get_fix_available_time(void);
```

Arguments:

None

Results:

`OS_clock_t:` CPU Time of the last fix.

Errors:

None.

Description:

Return the CPU Time of the last fix.

6 GNSS Fix Related Functions With Local Buffering

GNSS fix data is shared among all processes that need to get updated positioning info. The data access is protected by a semaphore based mechanism to guarantee the data consistency at any time during read/write operations. The APIs involved in the data update and protection process are gnss_fix_store(), gnss_fix_read_claim() and gnss_fix_read_release() as described in the previous chapter. Such GNSS Fix data access method can cause task priority inversion issues when more than one OS task, at different priority, need to access to the fix data at the same time. To avoid this issue (e.g. when there are high priority tasks that require a fast access to the GNSS fix data), the local buffering version of GNSS fix related APIs can be used. Local buffering means that GNSS fix data is stored into a private buffer that can be accessed only by the process that is using the buffer pointer. Having a private buffer, the semaphore protection is no more needed and so gnss_fix_read_claim() and gnss_fix_read_release() APIs must not be used when local buffering method is selected.

Note: local buffering method is memory consuming due to the duplication of GNSS fix data, it should be used only is strictly necessary in the application.

GNSS fix related APIs with local buffering capability have similar name of standard fix related functions (described in previous chapter), they have the "_local" addendum in the function name and an additional input parameters (at the end of the standard parameter list) which is the pointer to the local buffer. If the pointer to the local buffer is NULL, the "_local()" API behaves in the same way of the standard function accessing to the common fix data buffer (in such case the access protection must to be used and so gnss_fix_read_claim() and gnss_fix_read_release() must be used before and after the data access).

For APIs documentation refers to the corresponding standard API described in previous chapter.

Only a new API has been added for the fix data pointer initialization and it is described here after.

6.1 gnss_fix_create_local_copy

Initialize the pointer to the GNSS fix data local buffer copy.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_fix_create_local_copy( OS_partition_t *partition,
void **local_data_p)
```

Arguments:

`OS_partition_t *`	Memory partition pointer for OS resources allocation. If NULL, the data is allocated in the standard OS partition.
`void **`	pointer to the local buffer pointer which is used as input parameter for all other "_local" API calls.

Results:

The local buffer is allocated and the input pointer is initialized.

Errors:

Returns `GNSS_ERROR` in case there is not enough memory space to allocate the GNSS fix data buffer copy. In case of error the pointer to the local buffer is left not initialized.

Description:

Allocate memory for the GNSS fix data buffer copy and initialize the pointer to the buffer. This API must be called only one time at beginning before the usage of any "_local" API. As soon as the buffer pointer is initialized, it can be used as input parameter for the all "_local" calls.

6.2 gnss_fix_get_time_best_local

Reads the GNSS time of the stored fix

Synopsis:

```
#include "gnss_api.h"

gnss_sat_type_t gnss_fix_get_time_best_local(
   gnss_time_t *time,
   OS_clock_t *cpu_time,
   time_validity_t *validity,
   void* local_data_p
);
```

Arguments:

`gnss_time_t *:`	Pointer to the GNSS time of the fix (time of week and week number).
`OS_clock_t *:`	Pointer to the CPU time stamp of the fix.
`time_validity_t *:`	Pointer to the Time Validity status.
`void *:`	Pointer to the fix data structure stored

Results:

`gnss_sat_type_t:`	Returns the master sat type used for the fix

Errors:

None.

Description:

Returns the master type sat used to compute the last stored fix.

Returns the current GNSS week and GNSS week number of the last stored fix. GNSS time rolls over at Saturday 11:59 pm. GNSS week is related to the number of weeks since midnight January 5 1980.

Returns the CPU time stamp of the stored fix, the Time Validity for this fix, and all the fix data elements.

6.3 gnss_fix_get_time_master_local

Reads the GNSS time of the master constellation used for the stored fix. Similar to gnss_fix_get_time_best_local

Synopsis:

```
#include "gnss_api.h"

gnss_sat_type_t gnss_fix_get_time_master_local(
   gnss_time_t *time,
   time_validity_t *validity,
   void* local_data_p);
```

Arguments:

`gnss_time_t *:`	Pointer to the GNSS time of the master constellation used for the stored fix
`time_validity_t *:`	Pointer to the master Time Validity status.
`void *:`	Pointer to the fix data structure stored

Results:

`gnss_sat_type_t:`	returns the master sat type used for the fix

Errors:

None.

Description:

Returns the master type sat used to compute the last stored fix.

Returns the current GNSS week and GNSS week number of the master constellation used for last stored fix, the master Time Validity for this fix, and all the fix data elements.

6.4 gnss_fix_get_time_aux_local

Reads the GNSS time of the auxiliary constellation.

Synopsis:

```
#include "gnss_api.h"

gnss_sat_type_t gnss_fix_get_time_aux_local(
   gnss_time_t *time,
   time_validity_t *validity,
   void* local_data_p
);
```

Arguments:

`gnss_time_t *:`	Pointer to the auxiliary GNSS time. This is the GNSS time computed with the auxiliary constellation used.
`time_validity_t *:`	Pointer to the Time Validity of the auxiliary constellation.
`void *:`	Pointer to the fix data structure stored

Results:

`gnss_sat_type_t:`	returns the auxiliary sat type used for the fix

Errors:

None.

Description:

Returns the auxiliary type sat used to compute the last stored fix.

Returns the auxiliary GNSS week and GNSS week number of the last stored fix, the auxiliary Time Validity for this fix, and all the fix data elements.

The auxiliary constellation are chosen in the following order if requested in the software configuration:

GNSS_SAT_TYPE_GPS

GNSS_SAT_TYPE_COMPASS

GNSS_SAT_TYPE_GLONASS

GNSS_SAT_TYPE_GALILEO

7 GNSS PPS Functions

7.1 gnss_pps_init

Initialize the OS resources for the Pulse Per Second (PPS) management.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_pps_init( OS_partition_t *part, const tDouble
pulse_delay, const tDouble pulse_duration, const boolean_t
inverted_polarity);
```

Arguments:

`OS_partition_t *`	Memory partition pointer for OS resources allocation.
`const tDouble`	Pulse delay [s]. To compensate the time offset introduced by the antenna cable and RF chain.
`const tDouble`	Pulse duration [s].
`const boolean_t`	Pulse polarity inversion.

Results:

gnss_error_t	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Return `GNSS_ERROR` on failure.

Description:

Initialize the GNSS internal Pulse Per Second control. All required operating system resources are allocated and initialized.

NOTE1: this function should not be called if the PPS control by the GNSS library is not required.

NOTE2: this function must be called before the usage of any gnss_pps related API. If an gnss_pps API is called before the gnss_pps_init() call, it may return not valid data. It is recommended to avoid using the gnss_pps_ API when the gnss_pps_init() is not called if not explicitly indicated in the API description.

7.2 gnss_pps_set_signal_on_off_status

Enable or disable the PPS signal generation.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_signal_on_off_status( const boolean_t on_off);
```

Arguments:

const boolean_t on_off	PPS signal status (TRUE = enabled; FALSE = disabled)

Results:

None

Errors:

None

Description:

According to the input parameter, the Pulse Per Second signal will be enable or disabled. When disabled the PPS signal is kept low.

7.3 gnss_pps_get_signal_on_off_status

Allow getting the PPS signal generation ON/OFF status.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_pps_get_signal_on_off_status( void);
```

Arguments:

None.

Results:

boolean_t	PPS signal status (TRUE = enabled; FALSE = disabled)

Errors:

None

Description:

According to the input parameter, the Pulse Per Second signal will be enable or disabled. When disabled the PPS signal is kept low.

7.4 gnss_pps_enable_control

Allow enabling/disabling the PPS registers control by tracker process.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_enable_control( boolean_t);
```

Arguments:

boolean_t	PPS control status (TRUE = enabled; FALSE = disabled)

Results:

None

Errors:

None

Description:

According to the input parameter, the PPS registers update control logic in the tracker process is enabled or disabled.

The PPS control logic should be disabled if an external management of the PPS registers is implemented.

The control logic is disabled by default, it must be enabled before starting using it.

Note: this function can be used even is the gnss_pps_init() is not used.

7.5 gnss_pps_set_params

Inject the PPS params into the PPS control logic of the tracker process.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_params( const gnss_pps_params_t *);
```

Arguments:

const gnss_pps_params_t * Pointer to the PPS params data structure.

Results:

None.

Errors:

None.

Description:

Allow feeding the PPS parameters into the tracker process PPS control logic. This function should be called once a second to updates the PPS values which are written on the PPS registers by the tracker process. It must be called in advance to the PPS rising edge event.

NOTE1: if the function is not called in time or for other reasons the call is lost, the PPS registers are updated by the tracker process propagating the previous values. The propagation is done adding the CPL_STEP and the CPH_STEP values from the last available PPS params block.

NOTE2: to use the tracker PPS control logic it must be enabled. See gnss_pps_enable_control() API.

NOTE3: this function can be used even is the gnss_pps_init() is not used.

7.6 gnss_pps_set_pulse_duration

Sets the PPS pulse duration.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_pulse_duration( const tDouble pulse_duration);
```

Arguments:

const tDouble pulse_duration Pulse duration [s].

Results:

None.

Errors:

None.

Description:

Allow setting the pulse duration of the PPS signal. The pulse duration is the time interval in which the PPS signal is high (if inverted polarity is disabled).

7.7 gnss_pps_get_pulse_duration

Allow getting the PPS pulse duration.

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_pps_get_pulse_duration( void);
```

Arguments:

None.

Results:

tDouble	Pulse duration [s].

Errors:

None.

Description:

Return the current setting for the PPS signal pulse duration.

7.8 gnss_pps_set_time_delay

Set a time delay for the PPS signal generation.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_time_delay( const tDouble pulse_delay);
```

Arguments:

const tDouble pulse_delay	Pulse delay [s]

Results:

None.

Errors:

None.

Description:

Allow setting a time delay for the PPS signal generation. The pulse delay setting is required to compensate any time offset introduced by the antenna cable and by the RF chain.

7.9 gnss_pps_get_time_delay

Allow getting the PPS time delay setting.

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_pps_get_time_delay( void);
```

Arguments:

None.

Results:

tDouble	Pulse delay [s].

Errors:

None.

Description:

Returns the current setting for the PPS pulse delay.

7.10 gnss_pps_set_rf_compensation

Set a time delay for the RF path compensation. It is applied to the specific constellation type.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_rf_compensation(gnss_sat_type_t sat_type, tDouble
rf_correction)
```

Arguments:

gnss_sat_type_t sat_type	Constellation type: GNSS_SAT_TYPE_GPS or GNSS_SAT_TYPE_GLONASS
tDouble rf_correction	Time delay [s]

Results:

None.

Errors:

None.

Description:

Allow setting a time delay for a specific constellation in order to compensate the effects of different RF paths.

7.11 gnss_pps_get_rf_compensation

Allow getting the configured time delay from a specific constellation.

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_pps_get_rf_compensation(gnss_sat_type_t sat_type)
```

Arguments:

gnss_sat_type_t sat_type	Constellation type: GNSS_SAT_TYPE_GPS or GNSS_SAT_TYPE_GLONASS

Results:

tDouble rf_correction	Time delay [s].

Errors:

None.

Description:

Allow getting the time delay currently used to compensate the RF path delay for the selected constellation.

7.12 gnss_pps_set_polarity

Sets the PPS signal polarity.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_polarity( const boolean_t inverted_polarity);
```

Arguments:

const boolean_t inverted_polarity PPS polarity inversion status (FALSE = not inverted; TRUE = inverted).

Results:

None.

Errors:

None.

Description:

Enable or disable the PPS signal polarity inversion.

7.13 gnss_pps_get_polarity

Allow getting the PPS polarity inversion setting.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_pps_get_polarity( void);
```

Arguments:

None.

Results:

`boolean_t`	Polarity inversion status (FALSE = not inverted; TRUE = inverted).

Errors:

None.

Description:

Return the current setting for the PPS polarity inversion flag.

7.14 gnss_pps_get_used_sats

Allow getting the number of satellites used by the timing filter.

Synopsis:

```
#include "gnss_api.h"

tUChar gnss_pps_get_used_sats( void);
```

Arguments:

None.

Results:

tUChar	number of satellites used in last time correction.

Errors:

None

Description:

Return the number of satellites used by the timing filter to correct the accurate system time.

7.15 gnss_pps_get_fix_status

Allow getting the status of position fix during the timing correction.

Synopsis:

```
#include "gnss_api.h"

fix_status_t gnss_pps_get_fix_status( void);
```

Arguments:

None.

Results:

fix_status_t	GNSS position fix status: 1 = NO FIX, 2 = 2D FIX, 3 = 3D FIX.

Errors:

None.

Description:

Return the status of the GNSS position fix when last time correction has been applied.

7.16 gnss_pps_set_sat_threshold

Set the minimum number of satellites to control the PPS signal enabling/disabling.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_sat_threshold( const tUChar sat_th);
```

Arguments:

const tUChar sat_th	minimum number of satellites.

Results:

None.

Errors:

None.

Description:

Set a satellite threshold to control the PPS signal generation: PPS signal is enabled if the number of satellites, used by the time correction filter, is bigger or equal then the satellite threshold. When the number of satellites is less then threshold the PPS signal generation is disabled. Setting "0" as satellite threshold, means don't use the satellite threshold and so the PPS signal will be generated independently by the number of satellites.

7.17 gnss_pps_get_sat_threshold

Allow getting the current satellite threshold used by the timing filter.

Synopsis:

```
#include "gnss_api.h"

tUChar gnss_pps_get_sat_threshold( void);
```

Arguments:

None.

Results:

tUChar	Satellite threshold

Errors:

None.

Description:

Return the current setting of the satellite threshold for timing usage.

7.18 gnss_pps_set_fix_condition

Set the position fix status for PPS signal control.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_fix_condition( const fix_status_t fix_status);
```

Arguments:

const fix_status_t fix_status	position fix status. 1=NO Fix, 2=2D Fix, 3 = 3D Fix

Results:

None.

Errors:

None.

Description:

Set the position fix condition to control the PPS signal generation. The PPS signal is enabled if the GNSS position fix status is better or equal then fix condition setting.

7.19 gnss_pps_get_fix_condition

Allow getting the current fix condition setting.

Synopsis:

```
#include "gnss_api.h"

fix_status_t gnss_pps_get_fix_condition( void);
```

Arguments:

None.

Results:

fix_status_t	The GNSS position fix status during last time correction. 1 = NO FIX, 2 = 2D FIX, 3 = 3D FIX.

Errors:

None.

Description:

Returns the GNSS fix position status when the time correction has been applied last time.

7.20 gnss_pps_set_elevation_mask

Set the satellite elevation mask for the timing filter.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_elevation_mask( const tInt elevation_mask);
```

Arguments:

elevation_mask	Minimum satellite elevation for usage in the timing filtering.

Results:

None.

Errors:

None.

Description:

Allow setting the minimum satellite elevation to be used by the timing filter. Satellites below the configured elevation mask will be not used to correct the accurate time reference.

Note: this elevation mask parameter is only for timing usage purpose. It is independent by the .mask angle parameter used to select satellites for the visible sat list,

7.21 gnss_pps_get_elevation_mask

Allow getting the current satellite elevation mask setting.

Synopsis:

```
#include "gnss_api.h"

tInt gnss_pps_get_elevation_mask( void);
```

Arguments:

None.

Results:

Int	Minimum satellite elevation for usage in the timing filtering

Errors:

None.

Description:

Returns the current setting for the satellite elevation mask.

7.22 gnss_pps_set_constellation_mask

Set the constellation mask for the timing filter.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_constellation_mask( const gnss_sat_type_mask_t
constellation_mask);
```

Arguments:

const gnss_sat_type_mask_t constellation_mask Bit mask for constellations enabling/disabling. bit0 enable/disable the GPS constellation, bit1 enable/disable the GLONASS constellation and bit3 enable/disable the QZSS constellation.

Results:

None.

Errors:

None.

Description:

Allow selecting the satellite constellations to be used to correct the current reference time.

7.23 gnss_pps_get_constellation_mask

Allow getting the constellation mask setting.

Synopsis:

```
#include "gnss_api.h"

gnss_sat_type_mask_t gnss_pps_get_constellation_mask( void);
```

Arguments:

None.

Results:

gnss_sat_type_mask_t	Constellation bitmask. bit0 enable/disable the GPS constellation, bit1 enable/disable the GLONASS constellation and bit3 enable/disable the QZSS constellation

Errors:

None.

Description:

Returns the mask of satellite constellations enabled for timing correction.

7.24 gnss_pps_set_reference_constellation

Set the reference constellation used for PPS Signal generation.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_reference_constellation(const gnss_sat_type_t
reference_constellation);
```

Arguments:

`gnss_sat_type_t`: Constellation type (GPS,GLONASS etc.)

Results:

None.

Errors:

None.

Description:

Deprecated.

The parameter is now updated according to the master constellation type used.

7.25 gnss_pps_get_reference_constellation

Get the constellation type used for PPS Signal generation.

Synopsis:

```
#include "gnss_api.h"

gnss_sat_type_t gnss_pps_get_reference_constellation( void);
```

Arguments:

None

Results:

`gnss_sat_type_t`	Satellite constellation.

Errors:

None.

Description:

Returns the type of satellite constellation associated with the time reference type used for the PPS Signal.

If PPS process is not started the return value is GNSS_SAT_TYPE_GPS.

7.26 gnss_pps_set_clock_speed

Set the PPS Clock in the GNSS manager

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_clock_speed(tU8 clk);
```

Arguments:

`tU8:` Clock : 16, 32, 64 (Unit = MHz)

Results:

None

Errors:

None.

Description:

Set the PPS Clock in the GNSS manager.

By default the Software config value is used (id PPS_CLOCK_SETTING_ID)

This function shall be called before gnss_start() function.

The G3 component always use the 64MHz value.

7.27 gnss_pps_get_clock_speed

Get the GNSS manager PPS Clock value

Synopsis:

```
#include "gnss_api.h"

tU8 gnss_pps_get_clock_speed(void);
```

Arguments:

None

Results:

`tU8:` PPS Clock : 16, 32, 64 (Unit = MHz)

Errors:

None.

Description:

Return the GNSS manager PPS Clock value.

By default the Software config value is returned (id PPS_CLOCK_SETTING_ID)

This function shall be called after gnss_init () function.

7.28 gnss_pps_started

Get the PPS process status

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_pps_started(void);
```

Arguments:

None

Results:

`Boolean_t`	PPS Status: TRUE means that PPS process is started, FALSE means PPS process is not started

Errors:

None.

Description:

Return the PPS process status.

The PPS process is not started if not requested in the Software configuration (id 200 Mask PPS_ON_OFF_SWITCH)

7.29 gnss_pps_set_reference_time

Allow setting the reference time for PPS signal synchronization.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_reference_time(const pps_reference_time_t ref_time);
```

Arguments:

pps_reference_time_t ref_time	Reference time for PPS synchronization: PPS_TIME_REFERENCE_UTC PPS_TIME_REFERENCE_GPS_UTC PPS_TIME_REFERENCE_GLONASS_UTC

Results:

None.

Errors:

None.

Description:

Set the reference time on which the PPS signal is synchronized. Three different reference times are available: PPS_TIME_REFERENCE_UTC, PPS_TIME_REFERENCE_GPS_UTC and PPS_TIME_REFERENCE_GLONASS_UTC.

7.30 gnss_pps_get_reference_time

Allow getting the selected reference time for PPS signal synchronization.

Synopsis:

```
#include "gnss_api.h"

pps_reference_time_t gnss_pps_get_reference_time(void);
```

Arguments:

None.

Results:

pps_reference_time_t	Reference time for PPS synchronization: PPS_TIME_REFERENCE_UTC PPS_TIME_REFERENCE_GPS_UTC PPS_TIME_REFERENCE_GLONASS_UTC

Errors:

None.

Description:

Return the reference time currently selected for PPS signal synchronization. Three different reference times are available: PPS_TIME_REFERENCE_UTC, PPS_TIME_REFERENCE_GPS_UTC and PPS_TIME_REFERENCE_GLONASS_UTC.

7.31 gnss_pps_get_timing_data

Allow getting the current timing data block.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_get_timing_data(timing_data_t *timing_data);
```

Arguments:

timing_data_t * timing_data Pointer to the timing data structure.

Results:

None.

Errors:

None.

Description:

Returns the current time data block which includes all the timing information used for the PPS signal generation on the reported second.

7.32 gnss_pps_get_data

Allow getting the current PPS data block.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_get_data(pps_data_t *pps_data);
```

Arguments:

pps_data_t *pps_data	Pointer to the PPS data structure.

Results:

None.

Errors:

None.

Description:

Returns the current time data block which includes all the information used for the PPS signal generation on the reported second.

Note: The PPS data structure includes the timing data block.

7.33 gnss_pps_set_output_mode

Set the PPS output mode

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_output_mode( const pps_output_mode_t out_mode);
```

Arguments:

`const pps_output_mode_t`	PPS signal output mode. PPS_OUT_MODE_ALWAYS, PPS_OUT_MODE_ON_EVEN_SECONDS, PPS_OUT_MODE_ON_ODD_SECONDS

Results:

None.

Errors:

None.

Description:

Allow setting different ways for the PPS signal generation.

7.34 gnss_pps_get_output_mode

Allow getting the current output mode for the PPS signal.

Synopsis:

```
#include "gnss_api.h"

pps_output_mode_t gnss_pps_get_output_mode( void);
```

Arguments:

None

Results:

`pps_output_mode_t`	PPS signal output mode.
	PPS_OUT_MODE_ALWAYS,
	PPS_OUT_MODE_ON_EVEN_SECONDS,
	PPS_OUT_MODE_ON_ODD_SECONDSPPS Errors:

None.

Description:

Returns the selected PPS signal output mode.

7.35 gnss_pps_set_position_hold_status

Enable/Disable the Position Hold algorithm.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_position_hold_status( const boolean_t status);
```

Arguments:

const boolean_t status	Position hold enabling/disabling status. TRUE = enabled, FALSE = disabled.

Results:

None.

Errors:

None.

Description:

Allow setting the ON/OFF status for the Position Hold algorithm.

7.36 gnss_pps_get_position_hold_status

Allow getting the Position Hold enabling/disabling status.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_pps_get_position_hold_status( void);
```

Arguments:

None.

Results:

boolean_t	Position hold enabling/disabling status. TRUE = enabled, FALSE = disabled.

Errors:

None.

Description:

Returns the ON/OFF status of the Position Hold algorithm.

7.37 gnss_pps_set_position_hold_ECEF_pos

Set the ECEF position for the Position Hold algorithm.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_position_hold_ECEF_pos( const tDouble x, const
tDouble y, const tDouble z);
```

Arguments:

tDouble *x	pointer to ECEF x coordinate [m].
tDouble *y	pointer to ECEF y coordinate [m].
tDouble *z	pointer to ECEF z coordinate [m].

Results:

None.

Errors:

None.

Description:

Set the ECEF position for the Position Hold algorithm.

Note: two methods are available to set the Position Hold position; both do the same thing getting in input data in different position reference format (ECEF or LLH)

7.38 gnss_pps_get_position_hold_ECEF_pos

Allow getting the ECEF position used by the Position Hold algorithm.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_get_position_hold_ECEF_pos( tDouble *x, tDouble *y,
tDouble *z);
```

Arguments:

tDouble *x	pointer to ECEF x coordinate [m].
tDouble *y	pointer to ECEF y coordinate [m].
tDouble *z	pointer to ECEF z coordinate [m].

Results:

None.

Errors:

None.

Description:

Returns the ECEF position used by the Position Hold algorithm.

7.39 gnss_pps_set_position_hold_llh_pos

Set the LLH position for the Position Hold algorithm.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_position_hold_llh_pos( const tDouble lat,const
tDouble lon,const tDouble h);
```

Arguments:

tDouble *lat	pointer to LLH latitude coordinate [deg].
tDouble *lon	pointer to LLH longitude coordinate [deg].
tDouble *h	pointer to LLH altitude coordinate [m]. It is intended as the mean see level altitude.

Results:

None.

Errors:

None.

Description:

Set the LLH position used by the Position Hold algorithm.

Note: two methods are available to set the Position Hold position; both do the same thing getting in input data in different position reference format (ECEF or LLH)

7.40 gnss_pps_get_position_hold_llh_pos

Allow getting the LLH position used by the Position Hold algorithm.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_get_position_hold_llh_pos( tDouble *lat,tDouble
*lon,tDouble *h);
```

Arguments:

tDouble *lat	pointer to LLH latitude coordinate [deg].
tDouble *lon	pointer to LLH longitude coordinate [deg].
tDouble *h	pointer to LLH altitude coordinate [m]. It is intended as the mean see level altitude.

Results:

None.

Errors:

None.

Description:

Returns the LLH position used by the Position Hold algorithm.

7.41 gnss_pps_set_auto_hold_samples

Set the number of position samples for the Auto Position Hold operating mode.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_auto_hold_samples( const tUInt samples);
```

Arguments:

const tUInt samples	Number of position samples on which the average position is estimated for the Auto Position Hold operating mode.

Results:

None.

Errors:

None.

Description:

Allow setting the number of position samples to be captured for the average position evaluation. At the end of position samples collecting the Position Hold algorithm is automatically enabled using the average position.

In the position samples is set to “0” the Auto Position Hold operating mode is disabled.

7.42 gnss_pps_enable_traim

Enable the TRAIM algorithm.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_enable_traim ( const tDouble alarm);
```

Arguments:

const tDouble alarm	Time error [s] threshold for satellite removal from the timing filter.

Results:

None.

Errors:

None.

Description:

Enable the TRAIM algorithm and set the time error threshold for the bad satellite removal from the timing filetring.

7.43 gnss_pps_disable_traim

Disable the TRAIM algorithm.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_disable_traim ( void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Disable the TRAIM algorithm.

7.44 gnss_pps_get_traim_data

Allow getting the TRAIM algorithm data

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_get_traim_data( traim_data_t *traim_data);
```

Arguments:

traim_data_t *traim_data pointer to the TRAIM data block.

Results:

None.

Errors:

None.

Description:

Returns a copy of the TRAIM data.

7.45 gnss_pps_set_time_filter_feedback

Allow setting the timing filter feedback parameter

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_time_filter_feedback ( const tInt)
```

Arguments:

tInt	timing filter feedback value. High values give a slow response to the filter; low values (min value is 1) give a fast response. Default value is 25.

Results:

None.

Errors:

None.

Description:

Allow setting the timing filter response in the time error correction. A fast filter (low feedback value) could be noisy while a slow filter (high feedback values) could have a slow convergence in presence of a frequency jump. The right balance should be selected according to the application field.

7.46 gnss_pps_get_time_filter_feedback

Allow getting the timing filter feedback parameter

Synopsis:

```
#include "gnss_api.h"

tInt gnss_pps_get_time_filter_feedback ( void)
```

Arguments:

None.

Results:

tInt	timing filter feedback value. High values give a slow response to the filter; low values (min value is 1) give a fast response. Default value is 25.

Errors:

None.

Description:

Allow getting the timing filter response in the time error correction.

7.47 gnss_pps_set_cps_and_cpr_regs

Allow setting the PPS rising edge and falling edge registers.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_cps_and_cpr_regs(tUShort cpsh, tUShort cpsl, tUShort
cprh, tUShort cprl);
```

Arguments:

tUShort cpsh	high half part of the rising edge register.
tUShort cpsl	low half part of the rising edge register.
tUShort cprh	high half part of the falling edge register.
tUShort cprl	low half part of the falling edge register.

Results:

None.

Errors:

None.

Description:

Allow setting the PPS registers with the rising edge and falling edge timestamps. This API is supported to allow GNSS external control of the PPS signal.

Note: if the external control of the PPS signal is required the internal one should be disabled (see gnss_pps_init() API).

7.48 gnss_pps_set_cps_regs

Allow setting the PPS rising edge registers.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_cps_regs(tUShort cpsh, tUShort cpsl);
```

Arguments:

tUShort cpsh	high half part of the rising edge register.
tUShort cpsl	low half part of the rising edge register.

Results:

None.

Errors:

None.

Description:

Allow setting the PPS registers with the rising edge timestamp. This API is supported to allow GNSS external control of the PPS signal.

Note: if the external control of the PPS signal is required the internal one should be disabled (see gnss_pps_init() API).

7.49 gnss_pps_set_cpr_regs

Allow setting the PPS falling edge registers.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_set_cpr_regs(tUShort cprh, tUShort cprl);
```

Arguments:

tUShort cprh	high half part of the falling edge register.
tUShort cprl	low half part of the falling edge register.

Results:

None.

Errors:

None.

Description:

Allow setting the PPS registers with the falling edge timestamp. This API is supported to allow GNSS external control of the PPS signal.

Note: if the external control of the PPS signal is required the internal one should be disabled (see gnss_pps_init() API).

7.50 gnss_pps_get_mtb_timer

Return the current value of the Master Time Base counter

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_get_mtb_timer(tUInt *, tUInt *);
```

Arguments:

tUInt *: pointer to the TBH 16-bit counter (ms)

tUInt *: pointer to the TBL 14-bit counter (fraction of ms)

Results:

Current value of time base counter registers are returned.

Errors:

None.

Description:

Return current reading of the master time base counter. The MTB value is compared by HW with the PPS registers to set the rising and falling edges of the PPS signal.

7.51 gnss_pps_signal_disable

Disable the PPS signal generation.

Synopsis:

```
#include "gnss_api.h"

void gnss_pps_signal_disable(void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Allow disabling the PPS signal generation. This API is supported to allow GNSS external control of the PPS signal.

Note: if the external control of the PPS signal is required the internal one should be disabled (see gnss_pps_init() API). If the GNSS internal control is used for the PPS signal, the signal generation should be enabled/disabled using the gnss_pps_set_signal_on_off_status() API.

8 GNSS WAAS Functions

8.1 gnss_waas_set_status

Sets the WAAS satellite and the ON/OFF status.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_waas_set_status(
   const satid_t sat_id,
   const gnss_waas_status_t status
);
```

Arguments:

`satid_t`: WAAS satellite PRN.

`gnss_waas_status_t`: WAAS status (ON/OFF)

Results:

`gnss_error_t`: Returns always `GNSS_NO_ERROR`.

Errors:

Returns always `GNSS_NO_ERROR`.

Description:

Sets the WAAS satellite and starts or stops the WAAS engine according to the input parameter. If WAAS was OFF and the status parameter is `WAAS_STATUS_ON`, the selected WAAS satellite will be put in the tracker and WAAS activity starts. If WAAS was running and the input parameter is `WAAS_STATUS_OFF`, the WAAS satellite will be removed from tracker and WAAS activity stops.

Note: When this function is used in presence of a waas multichannel capability, only one channel (channel 0) is set, the other one is forced to `WAAS_STATUS_OFF` status.

See Also:

`gnss_waas_get_status`.

8.2 gnss_waas_get_status

Gets current WAAS satellite and status.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_waas_get_status(
   satid_t *sat_id,
   gnss_waas_status_t *status
);
```

Arguments:

`satid_t *`: Pointer to WAAS satellite PRN.

`gnss_waas_status_t *`: Pointer to ON/OFF WAAS status

Results:

`gnss_error_t`: Returns always `GNSS_NO_ERROR`.

Errors:

Returns always `GNSS_NO_ERROR`.

Description:

Allows getting the current WAAS satellite and the WAAS engine ON/OFF status.

See Also:

`gnss_waas_set_status`.

8.3 gnss_waas_get_multi_ch_prn_and_status

Gets current WAAS satellite and status on one channel.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_waas_get_multi_ch_prn_and_status(
   const chanid_t chan_id,
   satid_t *sat_id,
   gnss_waas_status_t *status
);
```

Arguments:

`chanid_t`:	WAAS channel of multichannel engine (0 or 1).
`satid_t *`:	Pointer to WAAS satellite PRN.
`gnss_waas_status_t *`:	Pointer to ON/OFF WAAS status

Results:

`gnss_error_t`:	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

Returns `GNSS_ERROR` if the channel ID in input is wrong.

Description:

Allows getting the current WAAS satellite and the status of a specific channel.

See Also:

`gnss_waas_set_multi_ch_prn_and_status`.

8.4 gnss_waas_set_ multi_ch_prn_and_status

Sets WAAS satellite and status on one channel.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_waas_set_multi_ch_prn_and_status(
   const chanid_t chan_id,
   satid_t sat_id,
   gnss_waas_status_t status
);
```

Arguments:

`chanid_t`:	WAAS channel of multichannel engine (0 or 1).
`satid_t`:	WAAS satellite PRN.
`gnss_waas_status_t`:	ON/OFF WAAS status

Results:

`gnss_error_t`:	Returns always `GNSS_NO_ERROR`.

Errors:

Returns always `GNSS_NO_ERROR`.

Description:

Allows setting the WAAS satellite and the ON/OFF status on a specific channel.

See Also:

`gnss_waas_get_multi_ch_prn_and_status`.

8.5 gnss_waas_set_prn_to_decode

Sets WAAS satellite to be used for differential corrections.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_waas_set_prn_to_decode( const satid_t sat_id);
```

Arguments:

`satid_t`: WAAS satellite PRN.

Results:

`gnss_error_t`: Returns always `GNSS_NO_ERROR`.

Errors:

Returns always `GNSS_NO_ERROR`.

Description:

Allows setting the WAAS satellite to be used for differential corrections.

See Also:

`gnss_waas_get_prn_to_decode`.

8.6 gnss_waas_get_prn_to_decode

Gets WAAS satellite which is used for differential corrections.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_waas_get_prn_to_decode( satid_t *sat_id);
```

Arguments:

`satid_t *`: Pointer to WAAS satellite PRN.

Results:

`gnss_error_t`: Returns always `GNSS_NO_ERROR`.

Errors:

Returns always `GNSS_NO_ERROR`.

Description:

Allows getting the WAAS satellite currently used for differential corrections.

See Also:

`gnss_waas_set_prn_to_decode`.

8.7 gnss_waas_is_tracking

Checks if the WAAS satellite is tracked.

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

boolean_t gnss_waas_is_tracking( const tInt tracker_state);
```

Arguments:

`tInt`:	Tracker state

Results:

`boolean_t`:	Returns `TRUE` if the WAAS satellite is tracked, `FALSE` if it is not tracked.

Errors:

None.

Description:

Returns `TRUE` or `FALSE` according to the WAAS satellite tracking state.

8.8 gnss_set_sbas_diff_params

Sets a specific set of differential correction parameters

Synopsis:

```
#include "gnss_api.h"
#include "gp_defs.h"

gnss_error_t gnss_set_sbas_diff_params (
   const satid_t sat_id,
   const tInt iod,
   const tDouble de,
   const tDouble dr,
   const tDouble drr,
   const diff_correction_t diff_correction_type
);
```

Arguments:

satid_t:	The PRN number of the satellite.
tInt:	The issue number of the ephemeris data used to calculate the satellite position.
tDouble de:	The time of week of the correction in *s*.
tDouble dr:	The range correction in *m* for the satellite.
tDouble drr:	The rate of change of range correction for the satellite in ms^{-1}.
diff_correction_t:	The diff correction type.

Results:

gnss_error_t:	Returns GNSS_NO_ERROR if the operation was successful or GNSS_ERROR on failure.

Errors:

Returns GNSS_ERROR if input parameters are not valid. If the parameters are not accepted the GNSS will use previous differential corrections if they are still valid.

Description:

Sets up the differential correction parameters for the specified satellite PRN. These are added to the measured range before the satellite is used in the position solution. The GNSS will only start to use the range corrections if it has enough satellites with differential corrections available.

9 GNSS STAGPS Functions

9.1 gnss_ephemeris_broadcast_enable_sat

Enable usage of broadcasted ephemeris for a specific satellite.

Synopsis:

```
#include "gnss_api.h"
void gnss_ephemeris_broadcast_enable_sat( const satid_t satid)
```

Arguments:

`satid_t`: Satellite PRN.

Results:

None.

Errors:

None.

Description:

Allows enabling the usage of broadcasted ephemeris for the satellites in input. If the input satellite is "0" all satellites are enabled to use the broadcasted ephemeris.

9.2 gnss_ephemeris_broadcast_disable_sat

Enable usage of broadcasted ephemeris for a specific satellite.

Synopsis:

```
#include "gnss_api.h"

void gnss_ephemeris_broadcast_disable_sat( const satid_t satid);
```

Arguments:

`satid_t`: Satellite PRN.

Results:

None.

Errors:

None.

Description:

Allows disabling the usage of broadcasted ephemeris for the satellites in input. If the input satellite is "0" all satellites are enabled to use the broadcasted ephemeris. This function is useful for STAGPS testing because disabling broadcasted ephemeris usage the GNSS library will use only predicted ephemeris if available.

9.3 gnss_get_ephemeris_broadcast_onoff_status

Gets the status of current broadcasted ephemeris usage enabling mask.

Synopsis:

```
#include "gnss_api.h"

tUInt gnss_get_ephemeris_broadcast_onoff_status( void)
```

Arguments:

None.

Results:

`unsigned`:	bit mask reporting the status of broadcasted ephemeris usage.

Errors:

None.

Description:

Allows getting the status of broadcasted ephemeris usage for all satellite.

9.4 gnss_get_sat_ephemeris_broadcast_onoff_status

Gets the status of broadcasted ephemeris usage for a specific satellite.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_get_sat_ephemeris_broadcast_onoff_status(
   const satid_t satid
);
```

Arguments:

satid_t:	Satellite PRN.

Results:

boolean_t:	TRUE if broadcasted ephemeris usage is enabled for the input satellite; returns FALSE if disabled.

Errors:

None.

Description:

Allows getting the broadcasted ephemeris usage status for a specific satellite.

9.5 gnss_get_multiconst_ephemeris_broadcast_onoff_status

Return the broadcast ephemeris usage ON/OFF status mask for the specified constellation

Synopsis:

```
#include "gnss_api.h"

tUInt gnss_get_multiconst_ephemeris_broadcast_onoff_status(
gnss_sat_type_t );
```

Arguments:

`gnss_sat_type_t`: Satellite constellation type

Results:

`tUInt`:	broadcast ephemeris usage ON/OFF status mask. It is a bitfield mask that represents the ON/OFF status for each satellite of the constellation. The bit position is referred to the satellite ID (from sat1=bit0 up to sat32=bit31)

Errors:

None.

Description:

Return the ON/OFF status concerning the usage of broadcasted ephemeris for a specific constellation.

9.6 gnss_ephemeris_predicted

Checks if satellites is using predicted ephemeris.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_ephemeris_predicted( const satid_t satid)
```

Arguments:

satid_t:	Satellite PRN.

Results:

boolean_t:	TRUE if satellite is using predicted ephemeris; returns FALSE if satellite is not using predicted ephemeris.

Errors:

None.

Description:

Allows checking if the satellite in input is using predicted ephemeris.

9.7 gnss_get_ephemeris_predict_params

Gets predicted parameters for a specific satellite.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_get_ephemeris_predict_params(
   const satid_t satid,
   pred_data_t *params
);
```

Arguments:

`satid_t`:	Satellite PRN.
`pred_data_t *`:	Pointer to the predicted ephemeris parameters block.

Results:

`gnss_error_t`:	Returns `GNSS_ERROR` is predicted params are not available; `GNSS_NO_ERROR` is returned when predicted params are available for the input satellite.

Errors:

Returns `GNSS_ERROR` is predicted params are not available.

Description:

Allows getting predicted ephemeris info (availability, age, number of ephemeris used for prediction, time distance between ephemeris) for a specific satellite.

9.8 gnss_clear_sat_ephemeris

Clear ephemeris for a specific satellite.

Synopsis:

```
#include "gnss_api.h"

void gnss_clear_sat_ephemeris( const satid_t satid);
```

Arguments:

`satid_t`: Satellite PRN.

Results:

Clear ephemeris for input satellite.

Errors:

None.

Description:

Allows clearing ephemeris for a single satellite.

9.9 gnss_tracker_events_on_debug_on_off

Enable/disable the tracker event messages over the debug channel.

Synopsis:

```
#include "gnss_api.h"

void gnss_tracker_events_on_debug_on_off ( const boolean_t on_ff);
```

Arguments:

`boolean_t`: enabling/disabling flag. ON=TRUE, OFF=FALSE.

Results:

According to the input parameter the tracker event messages over the debug channel are enabled or disabled.

Errors:

None.

Description:

Allows enabling/disabling tracker event messages over the debug channel.

9.10 gnss_set_stagps_status

Enable/Disable the STAGPS.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_stagps_status ( boolean_t);
```

Arguments:

`boolean_t`: enabling/disabling flag. ON=TRUE, OFF=FALSE.

Results:

`None`.

Errors:

None.

Description:

Enable/Disable the STAGPS.

9.11 gnss_get_stagps_status

Returns the STAGPS status.

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_get_stagps_status ( void);
```

Arguments:

None.

Results:

boolean_t: STAGPS status. ON=TRUE, OFF=FALSE.

Errors:

None.

Description:

Returns the STAGPS status.

10 GNSS Test Functions

10.1 gnss_test_set_user_pos

Changes the user position stored in the backup memory

Synopsis:

```
#include "gnss_api.h"

void gnss_test_set_user_pos(const tDouble lat,
const tDouble lon,
const tDouble height);
```

Arguments:

`tDouble lat`:	New latitude value
`tDouble lon`:	New longitude value
`tDouble height`:	New height value

Results:

None.

Errors:

None.

Description:

Changes the user position in backup memory to that supplied by the arguments. This is usually done for initiating time to first fix test values.

10.2 gnss_test_invalidate_user_pos

Invalidates the user position stored in the backup memory.

Synopsis:

```
#include "gnss_api.h"

void gnss_test_invalidate_user_pos( void);
```

Arguments:

None

Results:

None

Errors:

None

Description:

Invalidates the user position stored in the backup memory. Again this is usually used to initialize a time to first fix test scenario.

10.3 gnss_test_set_nco_value

Changes the nco value stored in the backup memory

Synopsis:

```
#include "gnss_api.h"

void gnss_test_set_nco_value(
   boolean_t availability,
   const nco_t nco_value
);
```

Arguments:

`boolean_t`: nco availability flag.

`nco_t`: nco_value

Results:

None.

Errors:

None.

Description:

Allow changing the value of the nco stored in the backup memory without checking the validity of the new value. This should only be used when initializing a time to first fix test scenario.

10.4 gnss_test_set_nco_range

Changes the nco range value stored in the backup memory

Synopsis:

```
#include "gnss_api.h"

void gnss_test_set_nco_range(const nco_t max, const nco_t min);
```

Arguments:

`nco_t`: max nco value.

`nco_t`: min nco value

Results:

None.

Errors:

None.

Description:

Allow changing the value of the nco range stored in the backup memory.

10.5 gnss_test_invalidate_nco_value

Invalidates the nco value stored in the backup memory.

Synopsis:

```
#include "gnss_api.h"

void gnss_test_invalidate_nco_value (void);
```

Arguments:

None

Results:

None

Errors:

None

Description:

Invalidates the nco value stored in the GNSS backup memory.

10.6 gnss_test_invalidate_nco_range

Invalidates the nco range parameters stored in the backup memory.

Synopsis:

```
#include "gnss_api.h"

void gnss_test_invalidate_nco_range (void);
```

Arguments:

None

Results:

None

Errors:

None

Description:

Invalidates the nco range parameters stored in the GNSS backup memory.

10.7 gnss_set_rf_test_mode_on_sat_n

Enables the RF test mode on a specific satellite.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_rf_test_mode_on_sat_n( const satid_t);
```

Arguments:

`satid_t`: satellite PRN used in RF test mode

Results:

The RF test mode is enabled on the input satellite.

Errors:

None.

Description:

Enables the RF test mode on the input satellites. This test mode configures the GNSS engine to acquire and track only the satellite specified in the input parameter. To start the RF test mode the gnss_suspend/gnss_restart sequence should be executed.

10.8 gnss_reset_rf_test_mode

Resets the RF test mode enabling all satellites to be acquired and tracked.

Synopsis:

```
#include "gnss_api.h"

void gnss_reset_rf_test_mode(void);
```

Arguments:

None.

Results:

None.

Errors:

None.

Description:

Disables the RF test mode. To start again all satellites acquiring and tracking the gnss_suspend/gnss_restart sequence should be executed.

10.9 gnss_rf_test_mode_del_sat_n

Remove a satellite from the RF test mode list

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_rf_test_mode_del_sat_n (const satid_t test_sat_id);
```

Arguments:

satid_t:	Satellite PRN to be removed.

Results:

boolean_t:	Returns TRUE if the input satellite is present in the RF test mode list. Returns FALSE if selected satellite is not present in the list.

Errors:

None.

Description:

The RF test mode has the capability to configure a list of satellites to be acquired and tracked in test mode. This function allows removing satellites from the RF test mode satellite list.

10.10 gnss_set_rf_test_mode_on_sat_n_cn0_trk_thr

Enables the RF test mode on a specific satellite setting a CN0 threshold.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_rf_test_mode_on_sat_n_cn0_trk_thr(
   satid_t satid,
   tUChar threshold
);
```

Arguments:

`satid_t`: satellite PRN used in RF test mode

`tUChar`: satellite acquisition CN0 threshold

Results:

None.

Errors:

None.

Description:

Enables the RF test mode on the input satellites. This test mode configures the GNSS engine to acquire and track only the satellite specified in the input parameter. It sets also a CN0 threshold for satellite acquisition. The selected satellite will be acquired only if its CN0 is bigger that threshold. To start the RF test mode the gnss_suspend/gnss_restart sequence should be executed.

10.11 gnss_set_rf_test_mode_on_sat_n_cn0_trk_xtal_thr

Enables the RF test mode on a specific satellite setting the CN0 threshold and the minimum CN0 for the Xtal calibration.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_rf_test_mode_on_sat_n_cn0_trk_xtal_thr ( satid_t
test,tUChar, tUChar );
```

Arguments:

satid_t:	satellite PRN used in RF test mode
tUChar:	satellite acquisition CN0 threshold
tUChar:	minimum CN0 value for the Xtal calibration

Results:

None.

Errors:

None.

Description:

Enable the RF test mode on the input satellite. This test mode configures the GNSS engine to acquire and track only the satellite specified in the input parameter. It sets also a CN0 threshold for satellite acquisition. The selected satellite will be acquired only if its CN0 is bigger that threshold. This API is designed to support Xtal calibration during the execution of the RF test. For this reason the minimum CN0 value for Xtal calibration is required as input parameter.

To start the RF test mode the gnss_suspend/gnss_restart sequence should be executed.

10.12 gnss_set_tracker_jammer

Enables or disables the jamming tracker mode.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_tracker_jammer(tInt on_off);
```

Arguments:

`tInt`: ON/OFF status

Results:

None.

Errors:

None.

Description:

Enables or disables the jamming tracking mode according to the input parameter. (1 = enable and 0 = disable).

Jamming mode allows tracking any interferer signals; in this operating mode the tracker is forced to work without any code phase check on the tracked signal. It is useful to measure the signal level of an external interfere or to check if the system generates a self-jamming in the GNSS frequency band.

10.13 gnss_test_invalidate_rtc

Invalidates the realtime clock value

Synopsis:

```
#include "gnss_api.h"

void gnss_test_invalidate_rtc(void);
```

Arguments:

None

Results:

None

Errors:

None

Description:

Invalidates the realtime clock by setting it to a dummy time. This is usually used when initializing time to first fix test scenarios.

10.14 gnss_test_invalidate_iono_params

Invalidates the iono parameters stored in the backup memory.

Synopsis:

```
#include "gnss_api.h"

void gnss_test_invalidate_iono_params (void);
```

Arguments:

None

Results:

None

Errors:

None

Description:

Invalidates ionospheric parameters stored in the GNSS backup memory.

10.15 gnss_test_invalidate_utc_params

Invalidates the UTC parameters stored in the backup memory.

Synopsis:

```
#include "gnss_api.h"

void gnss_test_invalidate_utc_params(void);
```

Arguments:

None

Results:

None

Errors:

None

Description:

Invalidates UTC parameters stored in the GNSS backup memory.

10.16 gnss_test_force_satellite_healthy_status

Force healthy status on a specific satellite

Synopsis:

```
#include "gnss_api.h"

void gnss_test_force_satellite_healthy_status(
   const satid_t satid,
   const tInt on_off_status
);
```

Arguments:

`satid_t`:	Satellite PRN
`tInt`:	ON/OFF status (1 = ON, 0 = OFF)

Results:

None.

Errors:

None.

Description:

This function allows forcing the healthy status on a single satellite. It is useful for testing in presence of unhealthy satellites which should be used in the positioning solution.

10.17 gnss_test_get_forced_satellites_healthy_status

Reports the list of forced satellites on debug channel

Synopsis:

```
#include "gnss_api.h"

void gnss_test_get_forced_satellites_healthy_status(void)
```

Arguments:

None

Results:

None

Errors:

None

Description:

Print all forced satellites on debug channel. If there is no satellite forced to be healthy, no message is sent on debug.

10.18 gnss_set_area_selection_limits

Provides Japanese area limits.

Synopsis:

```
#include "gnss_api.h"

void gnss_set_area_selection_limits (tDouble, tDouble);
```

Arguments:

`tDouble`: Minimum longitude.

`tDouble`: Maximum longitude.

Results:

`None`.

Errors:

None.

Description:

Set the Japanese area limits to consider QZSS satellites or not. Default values are -2.61 and +1.05 radians.

10.19 gnss_test_dsp_verification

Perform specific DSP verification tests.

Synopsis:

```
#include "gnss_api.h"

void gnss_test_dsp_verification(
   tInt type,
   tInt satid,
   tInt par1,
   tInt par2,
   tInt par3,
   tInt par4,
   tInt par5);
```

Arguments:

`tInt type:`	Command Id
`tInt satid:`	Satellite Id
`tInt par1 -> tInt par5:`	Test parameters

Results:

None

Errors:

None

Description:

Execute the specified DSP verification test.

This procedure is called by the NMEA command $PSTMTEST,2000,xxx. Please refer to the DSP Verification Suite document for more details.

10.20 gnss_test_set_operation

Set positioning filtering in some given operation modes.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_test_dsp_verification(
   const operation_property_t operation_property,
   const tInt value,
   const tInt ext_argc,
   const tDouble *ext_argv
);
```

Arguments:

`operation_property_t:`	Required operation.
`tInt:`	Enable or Diasble the operation mode
`tInt:`	Number of argument in the array.
`tDouble *:`	Array of arguments. Used to provide user position when in GNSS_OPERATION_POSITION_LOCK mode, or the delay when in GNSS_OPERATION_GLONASS_DELAY_LOCK mode.

Results:

`gnss_error_t:`	Returns `GNSS_NO_ERROR` if the operation was successful or `GNSS_ERROR` on failure.

Errors:

`None`

Description:

Set positioning filtering in some given operation modes.

11 GNSS Low power management functions

11.1 gnss_low_power_init_prv_config_params

Configure the low power algorithm implemented in the GNSS library. This function is used before the function gnss_start() and gnss_low_power_init_config_params() has been called.

It is advised to call `gnssapp_low_power_setup` instead of this function.

Synopsis:

```
#include "gnss_api.h"

void gnss_low_power_set_config_params(gnss_low_power_private_config_t
*);
```

Arguments:

`gnss_low_power_private_config_t *` pointer to private configuration data structure

Results:

New low power private configuration is applied.

Errors:

None

Description:

Configure the low power management supported by the GNSS library. Does not start the low power management until gnss_low_power_set_config_params and gnss_low_power_set_status are called.

11.2 gnss_low_power_set_config_params

Configure the low power algorithm implemented in the GNSS library. This function is used after the function gnss_start() has been called. See chapter 11.3.

It is advised to call `gnssapp_low_power_setup` instead of this function.

Synopsis:

```
#include "gnss_api.h"

void gnss_low_power_set_config_params(gnss_low_power_cyclic_mode_t *,
gnss_low_power_periodic_mode_t *);
```

Arguments:

`gnss_low_power_cyclic_mode_t *`	pointer to cyclic configuration data structure
`gnss_low_power_periodic_mode_t *`	pointer to periodic configuration data structure

Results:

New low power management configuration is applied. NULL pointer can be given in the API, this deactivates the related feature. Both cyclic and periodic cannot be activated at the same time.

Errors:

None

Description:

Allow the configuration for the low power management supported by the GNSS library.

11.3 gnss_low_power_init_config_params

Configure the low power algorithm implemented in the GNSS library for the specific case of the initialisation phase. It must be used only before the gnss_start function is called.

It is advised to call `gnssapp_low_power_setup` instead of this function.

Synopsis:

```
#include "gnss_api.h"

void gnss_low_power_init_config_params(gnss_low_power_cyclic_mode_t *,
gnss_low_power_periodic_mode_t *);
```

Arguments:

`gnss_low_power_cyclic_mode_t *`	pointer to cyclic configuration data structure
`gnss_low_power_periodic_mode_t *`	pointer to periodic configuration data structure

Results:

New low power management configuration is applied. NULL pointer can be given in the API, this deactivates the related feature. Both cyclic and periodic cannot be activated at the same time.

Errors:

None

Description:

Allow the configuration for the low power management supported by the GNSS library.

11.4 gnss_low_power_get_config_params

Return current configuration of low power algorithm implemented in the GNSS library

Synopsis:

```
#include "gnss_api.h"

void gnss_low_power_get_config_params(gnss_low_power_cyclic_mode_t *,
gnss_low_power_periodic_mode_t *);
```

Arguments:

`gnss_low_power_cyclic_mode_t *`	pointer to cyclic configuration data structure
`gnss_low_power_periodic_mode_t *`	pointer to periodic configuration data structure

Results:

Current configuration is returned in the input parameter

Errors:

None

Description:

Return the current configuration of the low power management.

11.5 gnss_low_power_set_status

Turn ON/OFF the low power management in the GNSS library

It is advised to call `gnssapp_low_power_setup` instead of this function.

Synopsis:

```
#include "gnss_api.h"

void gnss_low_power_set_status(boolean_t);
```

Arguments:

`boolean_t:` ON/OFF status (0=OFF 1=ON)

Results:

Low power management is enabled or disabled according to the input parameter.

Errors:

None

Description:

Enable/Disable the low power management algorithm.

11.6 gnss_low_power_get_status

Return the ON/OFF status of the low power management in the GNSS library

Synopsis:

```
#include "gnss_api.h"

boolean_t gnss_low_power_get_status(void);
```

Arguments:

None.

Results:

`boolean_t:` ON/OFF status (0=OFF 1=ON)

Errors:

None

Description:

Return the Enable/Disable status of the low power management algorithm.

11.7 gnss_low_power_get_data

Return data from low power management algorithm

Synopsis:

```
#include "gnss_api.h"

void  gnss_low_power_get_data(gnss_low_power_data_t *);
```

Arguments:

`gnss_low_power_data_t *:` pointer to the low power algorithm data structure

Results:

Current data from low power algorithm is returned in the input structure

Errors:

None

Description:

Return current status data from the low power algorithm.

11.8 gnss_low_power_get_nvm_sat_valid

Return the number of valid satellites stored by low power in NVM

Synopsis:

```
#include "gnss_api.h"

void  gnss_low_power_get_nvm_sat_valid (tUInt *);
```

Arguments:

`tUInt *:` pointer to a variable

Results:

Number of valid satellites stored in NVM.

Errors:

None

Description:

The low power algorithm compute the number of valid satellites in selected constellations (cf. gnss_set_constellation_mask chapter §3.93). It is used to know how much downloaded data (ephemeris and almanac) are expected from the constellation. This number is stored in NVM, and can is returned in the pointer on the function call.

11.9 gnss_low_power_set_long_eph_refresh_timer

Set the low power ephemeris refresh period interval

Synopsis:

```
#include "gnss_api.h"

void  gnss_low_power_set_long_eph_refresh_timer (boolean_t);
```

Arguments:

`boolean_t`: Short ephemeris refresh interval FALSE, Long ephemeris refresh interval TRUE

Results:

None

Errors:

None

Description:

Set the interval between ephemeris refresh intervals. Short is approximately 30 minutes, while long is about 10 hours. It is used by the STAGPS™ feature as soon as at least one ephemeris is available for each visible satellite of the constellations. With one ephemeris on a GPS satellite, STAPGS™ is able to make ephemeris predictions.

11.10 gnss_duty_cycle_set_params

Set the time period for the duty-cycle OFF state.

Synopsis:

```
#include "gnss_api.h"

gnss_error_t gnss_duty_cycle_set_params( const tInt);
```

Arguments:

`tInt:` Duty cycle OFF time period (ms)

Results:

Dutycycle ON-OFF time periods are configured.

Errors:

Returns GNSS_ERROR in case the input time period is out of [min,max] limits [100ms,750ms]

Description:

DutyCycle is supported inside the 1 second time frame. The API sets the time period for the OFF state; the ON state is evaluated as complement of 1 second time frame.

11.11 gnss_duty_cycle_get_state

Return the status of the duty-cycle process

Synopsis:

```
#include "gnss_api.h"

boolean_t            gnss_duty_cycle_get_state( void);
```

Arguments:

None.

Results:

`boolean_t:` status of duty-cycle process (FALSE=idle TRUE=running)

Errors:

None.

Description:

Return the current status of the duty-cycle process.

11.12 gnss_duty_cycle_start

Start the duty-cycle process

Synopsis:

```
#include "gnss_api.h"

void  gnss_duty_cycle_start( void);
```

Arguments:

None.

Results:

Start the duty-cycle process

Errors:

None.

Description:

Start the duty-cycle process.

11.13 gnss_duty_cycle_stop

Stop the duty-cycle process

Synopsis:

```
#include "gnss_api.h"

void  gnss_duty_cycle_stop( void);
```

Arguments:

None.

Results:

Stop the duty-cycle process

Errors:

None.

Description:

Stop the duty-cycle process.

12 GNSS XTAL functions

12.1 gnss_init_xtal_support

Initialize the enabling/disabling status for the SmartXtal support inside the navigation stage

Synopsis:

```
#include "gnss_api.h"

void  gnss_init_xtal_support(const boolean_t );
```

Arguments:

`boolean_t`: ON/OFF status (TRUE=xtal suppurt enabled, FALSE=xtal support disabled)

Results:

SmartXtal feature is enabled/disabled according to the input parameter.

Errors:

None.

Description:

Initialize the ON/OFF status for the SmartXtal feature inside the navigation stage.

12.2 gnss_xtal_mgr_support_set

Sets (or disables) XTAL support

Synopsis:

```
#include "gnss_api.h"

void gnss_xtal_mgr_support_set(tInt support_value);
```

Arguments:

`Int:` support_value (1: XTAL support on/ 0:Xtal support off)

Results:

Xtal support is ebabled/disable according to the input parameter

Errors:

None.

Description:

This function sets (or disables) XTAL functionalities according to the support_value input parameter.

See also:

12.3 gnss_xtal_set_Beta_and_res_params

Sets Beta, Rto and Rs values used to compute, in XTAL SW, the temperature from the ADC read.

Synopsis:

```
#include "gnss_api.h"
void gnss_xtal_set_Beta_and_res_params(gnss_xtal_mgr_res_net_data_t
res_network_param)
```

Arguments:

gnss_xtal_mgr_res_net_data_t: res_network_param(data structure containing Beta, Rto and Rs)

Results:

Xtal parameters are configured according to the input values

Errors:

None.

Description:

This function sets the Beta, Rto and Rs values that will be used to convert the ADC reading into a temperature value to be used by XTAL SW.

See also:

12.4 gnss_xtal_set_input_data_callback

Used to set the function to read ADC (or temp) according to the selector

Synopsis:

```
#include "gnss_api.h"
void gnss_xtal_set_input_data_callback(tInt (* p)(void),tInt sel)
```

Arguments:

`tInt (* p)(void)` : function pointer used to indicate the function that will be used to read the ADC (or Temp)

`tInt:` sel (selector:1 to indicate an ADC value will be the input, selector: 2 to indicate that a temperature will be the input)

Results:

The callback function is configured according to the input pointer

Errors:

None.

Description:

Callback function to set the function used to read ADC (or temp) according to the selector parameter.

See also:

12.5 gnss_xtal_mgr_copy_state_and_cov_params

Used to copy state and covariance values

Synopsis:

```
#include "gnss_api.h"
gnss_error_t
gnss_xtal_mgr_copy_state_and_cov_params(gnss_xtal_mgr_nvm_data_t
*xtal_data_copy)
```

Arguments:

gnss_xtal_mgr_nvm_data_t: *xtal_data_copy Address to the structure where state and covariance values will be copied

Results:

Gnss_error_t: If semaphore (xtal_params_update_sem) has been initialized the function returns no error; otherwise a GNSS_ERROR will be returned

Errors:

Description:

It copies (into the address xtal_data_copy) the state and the covariance computed by Xtal SW, if xtal_params_update_sem semaphore has already been initialized.

See also:

12.6 gnss_xtal_mgr_set_state_and_cov_from_ext

Used to set state and covariance from extern

Synopsis:

```
#include "gnss_api.h"
 void gnss_xtal_mgr_set_state_and_cov_from_ext(gnss_xtal_mgr_nvm_data_t
xtal_values )
```

Arguments:

`gnss_xtal_mgr_nvm_data_t`: data structure containing the state, covariance values and extern configurator indicator ext_set (to be set to 1 if external settings will be used /0 otherwise)

Results:

Errors:

None.

Description:

This function sets the initial state and covariance form extern, if the ext_set is set to 1.

See also:

12.7 gnss_xtal_mgr_set_compression_factor

Allow setting a compression factor to adapt the ADC voltage range to the thermistor voltage range

Synopsis:

```
#include "gnss_api.h"

void gnss_xtal_mgr_set_compression_factor(const tInt , const tInt );
```

Arguments:

`tInt`: ADC voltage range [mV] (it is referred to ground)

`tInt`: thermistor voltage range [mV] (it is referred to ground)

Results:

A compression factor is configured according to the input parameters

Errors:

`None.`

Description:

In case the ADC input voltage range doesn't match the xtal thermistor voltage range, the API can be used to configure the correct scaling factor to align the ADC range with the thermistor range. The correct scaling factor is needed to have an accurate estimation of the xtal temperature.

12.8 gnss_xtal_mgr_get_compression_factor

Return the current value of the compression factor

Synopsis:

```
#include "gnss_api.h"

tDouble gnss_xtal_mgr_get_compression_factor(void );
```

Arguments:

`void`

Results:

`tDouble:` compression factor

Errors:

`None.`

Description:

Allow getting the configured compression factor to adapt the ADC voltage range to the thermistor voltage range

12.9 gnss_xtal_mgr_set_Q_values

Set Q matrix initial values.

Synopsis:

```
#include "gnss_api.h"

void gnss_xtal_mgr_get_compression_factor(tDouble *);
```

Arguments:

`tDouble *:` Q element value.

Results:

`None.`

Errors:

`None.`

Description:

Set Q matrix initial values, used in XTAL compensation modelling.

12.10 gnss_xtal_get_data

Return the current status of the xtal management data

Synopsis:

```
#include "gnss_api.h"

void gnss_xtal_get_data(gnss_xtal_monitor_t *);
```

Arguments:

`gnss_xtal_monitor_t *`	pointer to xtal data structure

Results:

Xtal data structure is updated with current values

Errors:

`None.`

Description:

Allow getting last updated data from the Xtal management algorithm.

13 Type Definitions

Print all forced satellites on debug channel. If there is no satellite forced to be healthy, no message is sent on debug.

13.1 Satellite Data

13.1.1 gnss_sat_type_t

Type	Values		Description
enum	-1	GNSS_SAT_TYPE_NOT_VALID	Not valid satellite type
	0	GNSS_SAT_TYPE_GPS	GPS constellation satellite
	1	GNSS_SAT_TYPE_GLONASS	GLONASS constellation satellite
	2	GNSS_SAT_TYPE_QZSS_L1_CA	QZSS constellation satellite
	3	GNSS_SAT_TYPE_GALILEO	GALILEO constellation satellite
	4	GNSS_SAT_TYPE_SBAS	SBAS constellation satellite
	5	GNSS_SAT_TYPE_QZSS_L1_SAIF	QZSS constellation satellite
	6	GNSS_SAT_TYPE_QZSS_L1C	QZSS constellation satellite
	7	GNSS_SAT_TYPE_COMPASS	COMPASS constellation satellite
	8	GNSS_SAT_TYPE_PSEUDOLITE	PSEUDOLITE constellation satellite
	9	GNSS_SAT_TYPE_L2C	L2C constellation satellite

13.1.2 satid_t

Type	Description
tUShort	The satellite ID as reported below: GPS: from 1 to 32 GLONASS: from 65 to 92 SBAS: from 120 to 138 QZSS: from 193 to 197? GALILEO: from 301 to 325 COMPASS: from 141 to 170

13.1.3 gnss_sat_type_mask_t

Synonym of tUInt. The enum gnss_sat_type_t gives the bit numbering for each constellation.

13.2 Visible Satellite Data

13.2.1 visible_t

Type	Structure Member	Description
satid_t	satid	The PRN number of the satellite.
tInt	azimuth	The azimuth of the satellite in degrees.
tInt	elevation	The elevation of the satellite in degrees.

13.2.2 visible_sats_data_t

Type	Structure Member	Description
Int	list_size	The number of satellites visible.
visible_t	list[32]	A list of satellite visibility data.

13.2.3 gnss_initvisiblesatslist_available_callback_t

```
typedef tVoid (*gnss_initvisiblesatslist_available_callback_t)(void)
```

13.3 Satellite Tracking Data

13.3.1 available_locked_t

Bits	Structure Member	Description
1	available	The availability of satellite tracking data, TRUE or FALSE.
1	preamble_locked	

13.3.2 dsp_data_t

Type	Structure Member	Description
tDouble	carrier_phase	Carrier phase.
fp_U32_t	pseudorange	The measured pseudo range data in m.
fp_U32_t	frequency	The measured frequency
tInt	tracked_time	Number of seconds the satellite is in tracking state
tUShort	signal_strength	The satellite signal strength
available_locked_t	available_and_preamble_locked	The tracking status.

13.3.3 sat_data_t

Type	Structure Member	Description
ECEF_pos_fp_t	sat_pos	The satellite position vector.
ECEF_vel_fp_t	sat_vel	The satellite velocity vector.
fp_s32_t	range_correction	The range correction in m.
fp_s16_t	atmospheric_correction	The atmospheric correction in m.
fp_s16_t	range_rate_correction	The range rate correction in s.
boolean_t	available	The availability of satellite data, TRUE or FALSE.

13.3.4 pred_data_t

Bits	Structure Member	Description
8	age_h	
2	ephems_n	
6	time_distance_h	
1	available	

13.3.5 raw_t

Type	Structure Member	Description
dsp_data_t	dsp	Structure containing the dsp measurement data.
sat_data_t	sat	Structure containing the satellite data.
diff_data_t	diff	Structure containing the differential data.
pred_data_t	pred	Structure containing the prediction data (if STAGPS is available)

13.3.6 raw_measurement_list_t

Type	Structure Member	Description
tInt	list_size	The number of satellite measurements available.
boolean_t	chans_used[24]	Whether the channel data was used in the fix.
raw_t	list[24]	A list of measurement data.

13.4 NCO Data

13.4.1 nco_t

Type	Description
`tInt`	The NCO frequency value [Hz].

13.5 Ephemeris Data

13.5.1 gps_ephemeris_raw_t

Bits	Structure Member	Description
16	week	Week number of the Issue of Data
16	toe	Time of week for ephemeris epoch
16	toc	Time of week for clock epoch
8	iode1	Issue of data 1
8	iode2	Issue of data 2
10	iodc	Issue of data clock
14	i_dot	Rate of inclination angle.
8	age_h	Age of predicted ephemeris (hours)
24	omega_dot	Rate of right ascension.
2	ephems_n	Number of ephemeris (1 or 2) used for prediction.
6	time_distance_h	Time interval (hours) between two ephemeris used for prediction.
16	crs	Amplitude of the sine harmonic correction to the orbit radius.
16	crc	Amplitude of the cosine harmonic correction to the orbit radius.
16	cus	Amplitude of the sine harmonic correction to the argument of latitude.
16	cuc	Amplitude of the cosine harmonic correction to the argument of latitude.
16	cis	Amplitude of the sine harmonic correction to the angle of inclination.
16	cic	Amplitude of the cosine harmonic correction to the angle of inclination.
16	motion_difference	Mean motion difference from computed value
32	inclination	Inclination angle at reference time
32	eccentricity	Eccentricity.

32	root_a	Square root of major axis.
32	mean_anomaly	Mean anomaly at reference time.
32	omega_zero	Longitude of ascending node of orbit plane at weekly epoch.
32	perigee	Argument of perigee.
8	time_group_delay	Estimated group delay differential.
8	af2	Second order clock correction.
16	af1	First order clock correction.
22	af0	Constant clock correction.
1	subframe1_available	Reserved for use by GNSS library
1	subframe2_available	Reserved for use by GNSS library
1	subframe3_available	Reserved for use by GNSS library
1	available	Contains 1 if ephemeris is available, 0 if not
1	health	Contains 1 if the satellite is unhealthy, 0 if healthy
1	predicted	Contains 1 if the ephemeris is predicted, 0 if not
4	accuracy	Accuracy

13.5.2 glonass_ephemeris_raw_t

Bits	Structure Member	Description
16	week	Week number of the Issue of Data.
16	toe	Time of week for ephemeris epoch.
4	toe_lsb	Time of week for ephemeris epoch (LBS).
11	NA	Calendar day number within the four-year period since the beginning of last leap year (almanac).
7	tb	Time of ephemeris index.
2	M	Type of satellite 00=GLONASS 01=GLONASS-M .
2	P1	Time interval between two adjacent tb parameters.
1	P3	Number of satellites for which almanac is transmitted within this frame 0=4 1=5.
5	Not Used	

27	xn	Satellite PZ-90 x coordinate at epoch tb.
5	xn_dot_dot	Satellite PZ-90 x velocity at epoch tb.
24	xn_dot	Satellite PZ-90 x acceleration component at epoch tb.
5	n	Slot number (1…24).
3	Bn	Healthy flags.
27	yn	Satellite PZ-90 y coordinate at epoch tb.
5	yn_dot_dot	Satellite PZ-90 y acceleration component at epoch tb.
24	yn_dot	Satellite PZ-90 y velocity at epoch tb.
8	age_h	Age of predicted ephemeris (hours)
27	zn	Satellite PZ-90 z coordinate at epoch tb.
5	zn_dot_dot	Satellite PZ-90 z acceleration component at epoch tb.
24	zn_dot	Satellite PZ-90 z velocity at epoch tb.
2	ephems_n	Number of ephemeris (1 or 2) used for prediction.
6	time_distance_h	Time interval (hours) between two ephemeris used for prediction.
11	gamma_n	Satellite clock frequency drift at epoch tb.
5	E_n	Age of the ephemeris information.
16	Reserved	
22	tau_n	Satellite clock correction at epoch tb.
10	Not Used	
32	tau_c	GLONASS to UTC(SU) time correction.
22	tau_GPS	GLONASS to GPS system time correction.
10	Not Used	
11	NT	Calendar day number of ephemeris within the four-year period since the beginning of last leap year.
5	N4	Four-year interval number starting from 1996.
12	tk	Satellite time referenced to the beginning of the frame.
4	Not Used	
32	Reserved	

32	Not Used	
25	Not Used	
1	available	Contains 1 if ephemeris is available, 0 if not.
1	health	Contains 1 if the satellite is unhealthy, 0 if healthy.
1	predicted	Contains 1 if the ephemeris is predicted, 0 if not.
4	Not Used	

13.5.3 galileo_ephemeris_raw_t

Bits	Structure Member	Description
16	week	Week number of the Issue of Data
14	toe	Time of week for ephemeris epoch
2	ephems_n	Number of ephemeris (1 or 2) used for prediction.
14	toc	Time of week for clock epoch
10	iod_nav	Issue of data
8	SISA	Signal in space accuracy
10	Not Used	
10	BGD_E1_E5a	Broadcast Group Delay on E5a
10	BGD_E1_E5b	Broadcast Group Delay on E5b
2	E1BHS	Health status on E1B
32	inclination	Inclination angle at reference time
32	eccentricity	Eccentricity.
32	root_a	Square root of major axis.
32	mean_anomaly	Mean anomaly at reference time.
32	omega_zero	Longitude of ascending node of orbit plane at weekly epoch.
32	perigee	Argument of perigee.
14	i_dot	Rate of inclination angle.
1	available	Contains 1 if ephemeris is available, 0 if not
1	health	Contains 1 if the satellite is unhealthy, 0 if healthy
16	motion_difference	Mean motion difference from computed value
16	crs	Amplitude of the sine harmonic correction to the orbit radius.
16	crc	Amplitude of the cosine harmonic correction to the orbit radius.
16	cus	Amplitude of the sine harmonic correction to the argument of latitude.

16	cuc	Amplitude of the cosine harmonic correction to the argument of latitude.
16	cis	Amplitude of the sine harmonic correction to the angle of inclination.
16	cic	Amplitude of the cosine harmonic correction to the angle of inclination.
24	omega_dot	Rate of right ascension.
6	SVID	Satellite vehicle ID
1	E1BDVS	Data validity status on E1B.
1	predicted	Contains 1 if the ephemeris is predicted, 0 if not
6	af2	Second order clock correction.
21	af1	First order clock correction.
5	word_available	Contains 1 if ephemeris is available, 0 if not.
31	af0	Constant clock correction.
1	Not Used	

13.5.4 compass_ephemeris_raw_t

Bits	Structure Member	Description
32	inclination	Inclination angle at reference time
32	eccentricity	Eccentricity.
32	root_a	Square root of major axis.
32	mean_anomaly	Mean anomaly at reference time.
32	omega_zero	Longitude of ascending node of orbit plane at weekly epoch.
32	perigee	Argument of perigee.
17	toe	Time of week for ephemeris epoch
10	time_group_delay	Estimated group delay differential.
5	aode	Age of data ephemeris
24	omega_dot	Rate of right ascension.
8	A0	Beidou iono modem parameter.

24	af0	Constant clock correction.
8	A1	Beidou iono modem parameter.
20	sow	Number of seconds within a week since Saturday midnight
11	af2	Second order clock correction.
1	is_geo	Contains 1 if sat is GEO. 0 if not.
22	af1	First order clock correction.
10	subframe_avail	Bitmask of available subframes.
16	motion_difference	Mean motion difference from computed value
8	A2	Beidou iono modem parameter.
8	A3	Beidou iono modem parameter.
18	crs	Amplitude of the sine harmonic correction to the orbit radius.
8	B2	Beidou iono modem parameter.
4	urai	User range accuracy index.
2	Not Used	
18	crc	Amplitude of the cosine harmonic correction to the orbit radius.
8	B3	Beidou iono modem parameter.
5	aodc	Age of data clock.
1	Not Used	
18	cus	Amplitude of the sine harmonic correction to the argument of latitude.
14	i_dot	Rate of inclination angle.
18	cuc	Amplitude of the cosine harmonic correction to the argument of latitude.
8	B0	Beidou iono modem parameter.
6	Not Used	
18	cis	Amplitude of the sine harmonic correction to the angle of inclination.
8	B1	Beidou iono modem parameter.

6	Not Used	
18	cic	Amplitude of the cosine harmonic correction to the angle of inclination.
1	Reserved	
13	Not Used	
17	toc	Time of week for clock epoch
13	week	Week number of the Issue of Data
1	available	Contains 1 if ephemeris is available, 0 if not
1	health	Contains 1 if the satellite is unhealthy, 0 if healthy

13.5.5 ephemeris_raw_t

Union	Structure Member	Description
gps_ephemeris_raw_t	gps	GPS ephemeris
glonass_ephemeris_raw_t	glonass	GLONASS ephemeris
galileo_ephemeris_raw_t	galileo	GALILEO ephemeris
compass_ephemeris_raw_t	compass	COMPASS ephemeris

13.5.6 gnss_ephemeris_updated_callback_t

```
typedef tVoid (*gnss_ephemeris_updated_callback_t)(satid_t sat_id)
```

13.6 Almanac Data

13.6.1 gps_almanac_raw_t

Bits	Structure Member	Description
8	satid	The satellite number
16	week	The week number for the epoch
8	toa	Reference time almanac.
16	e	Eccentricity.
16	delta_i	Rate of inclination angle.
16	omega_dot	Rate of right ascension.
24	root_A	Square root of semi-major axis.
24	omega_zero	Longitude of ascending node of orbit plane at weekly epoch.
24	perigee	Argument of perigee.
24	mean_anomaly	Mean anomaly at reference time.
11	af0	Constant clock correction.
11	af1	First order clock correction.
1	health	Contains 1 if the satellite is unhealthy 0 if healthy.
1	available	Contains 1 if almanac is available 0 if not.

13.6.2 glonass_almanac_raw_t

Bits	Structure Member	Description
8	satid	The satellite number.
16	week	The week number for the epoch.
8	toa	Reference time almanac.
5	n_A	Slot number (1…24).
5	H_n_A	Carrier frequency channel number.
2	M_n_A	Type of satellite 00=GLONASS 01=GLONASS-M.
10	tau_n_A	Satellite clock correction.

15	epsilon_n_A	Eccentricity.
21	t_lambda_n_A	Time of the first ascending node passage.
21	lambda_n_A	Longitude of ascending node of orbit plane at almanac epoch.
18	delta_i_n_A	Inclination angle correction to nominal value.
7	delta_T_n_dot_A	Draconian period rate of change.
22	delta_T_n_A	Draconian period correction.
16	omega_n_A	Argument of perigee.
1	health	Contains 1 if the satellite is unhealthy 0 if healthy.
1	available	Contains 1 if almanac is available 0 if not.
32	Tau_c	
11	NA	
5	N4	
16	Spare	

13.6.3 galileo_almanac_raw_t

Bits	Structure Member	Description
16	satid	The satellite number
6	svid	Satellite vehicle ID
16	week	The week number for the epoch
20	toa	Reference time almanac.
11	eccentricity	Eccentricity.
16	perigee	Argument of perigee.
11	delta_i	Rate of inclination angle.
16	omega_zero	Longitude of ascending node of orbit plane at weekly epoch.
11	omega_dot	Rate of right ascension.
16	mean_anomaly	Mean anomaly at reference time.
16	af0	Constant clock correction.

Bits	Structure Member	Description
13	af1	First order clock correction.
2	E5bHS	Health status on E5b
2	E1BHS	Health status on E1B
4	ioda_1	Issue of data almanac.
4	ioda_2	Issue of data almanac.
2	word_available	Contains 1 if ephemeris is available, 0 if not.
1	health	Contains 1 if the satellite is unhealthy 0 if healthy.
1	available	Contains 1 if almanac is available 0 if not.

13.6.4 Compass_almanac_raw_t

Bits	Structure Member	Description
8	prn	The satellite number
16	week	Week number of the Issue of Data
8	toa	Reference time almanac.
17	eccentricity	Eccentricity.
24	af0	Constant clock correction.
1	is_geo	Contains 1 if sat is GEO. 0 if not.
3	Not used	
17	omega_dot	Rate of right ascension.
11	af1	First order clock correction.
4	Not used	
24	root_a	Square root of major axis.
24	omega_zero	Longitude of ascending node of orbit plane at weekly epoch.
24	perigee	Argument of perigee.
24	mean_anomaly	Mean anomaly at reference time.
16	delta_i	Rate of inclination angle.
1	health	Contains 1 if the satellite is unhealthy 0 if healthy.

Bits	Structure Member	Description
8	prn	The satellite number
16	week	Week number of the Issue of Data
1	available	Contains 1 if almanac is available 0 if not.

13.6.5 almanac_raw_t

Union	Structure Member	Description
gps_almanac_raw_t	gps	GPS almanac
glonass_almanac_raw_t	glonass	GLONASS almanac
galileo_almanac_raw_t	galileo	GALILEO almanac
compass_almanac_raw_t	compass	COMPASS almanac

13.7 Ionospheric Data

13.7.1 iono_raw_t

Bits	Structure Member	Description
8	A0	Ionospheric cubic coefficient alpha0.
8	A1	Ionospheric cubic coefficient alpha1.
8	A2	Ionospheric cubic coefficient alpha2.
8	A3	Ionospheric cubic coefficient alpha3.
8	B0	Ionospheric cubic coefficient beta0.
8	B1	Ionospheric cubic coefficient beta1.
8	B2	Ionospheric cubic coefficient beta2.
8	B3	Ionospheric cubic coefficient beta3.
8	available	Contains 1 if ionospheric data is available 0 if not

13.8 UTC Data

13.8.1 utc_raw_t

Bits	Structure Member	Description
32	A0	Constant terms of polynomial.
24	A1	First order term of polynomial.
8	delta_tls	Delta time due to leap seconds.
8	delta_tlsf	Leap second value at change over.
8	DN	Day number of UTC change over.
8	tot	Reference time for UTC data.
8	WNt	UTC reference week number.
8	WNlsf	Current week number derived from subframe 1.
8	available	If not zero the utc data is available
16	Page18_offset	TOW of first page 18 occurrence I the week

13.8.2 gnss_utc_delta_time_validity_t

Type	Values		Description
enum	0	UTC_GPS_OFFEST_NOT_VALID	The UTC delta time is not available in backup memory and it has been not yet downloaded from the sky
	1	UTC_GPS_OFFEST_STORED	The UTC delta time has been initialized from the backup memory
	2	UTC_GPS_OFFEST_VALID	The UTC delta time has been updated with recent data downloaded from the sky

13.8.3 glonass_utc_raw_t

Bits	Structure Member	Description
16	week_lsf	
16	spare0	

32	tow_lsf	
11	B1	
10	B2	
2	KP	
7	Spare1	
1	expired	
1	available	

13.9 Position Data

13.9.1 position_t

Type	Structure Member	Description
tDouble	latitude	The latitude of position in degrees and decimal degrees.
tDouble	longitude	The longitude of position in degrees and decimal degrees.
tDouble	height	The height of position in m.

13.9.2 gnss_position_validity_t

Type	Values		Description
enum	0	GNSS_POSITION_UNKNOWN	User position not valid
	1	GNSS_POSITION_VALID	User position valid

13.9.3 operation_property_t

Type	Values		Description
enum	0	GNSS_OPERATION_ORBIT_LIST_POPULATE	Not used
	1	GNSS_OPERATION_LMS_SET_LS_ONLY	LMS only operation
	2	GNSS_OPERATION_POSITION_LOCK	Position locked operation
	10	GNSS_OPERATION_GLONASS_USE_MEASURES	Usage of GLONASS for LMS and Kalman
	11	GNSS_OPERATION_GLONASS_DELAY_LOCK	GLONASS Delay lock operation

13.10 Velocity Data

13.10.1 velocity_t

Type	Structure Member	Description
tDouble	vel_north	The velocity along the north axis in ms-1.
tDouble	vel_east	The velocity along the east axis in ms-1.
tDouble	vel_vert	The vertical velocity in ms-1

13.11 Sat Position Data

13.11.1 ECEF_pos_t

Type	Structure Member	Description
tDouble	x	The satellite position in m in the x axis.
tDouble	y	The satellite position in m in the y axis.
tDouble	z	The satellite position in m in the z axis.

13.11.2 ECEF_pos_fp_t

Type	Structure Member	Description
fp_s32_t	x	The satellite position in m in the x axis.
fp_s32_t	y	The satellite position in m in the y axis.
fp_s32_t	z	The satellite position in m in the z axis.

13.12 Sat Velocity Data

13.12.1 ECEF_vel_t

Type	Structure Member	Description
tDouble	x	The satellite velocity in ms-1 the x axis.
tDouble	y	The satellite velocity in ms-1 the y axis.
tDouble	z	The satellite velocity in ms-1 the z axis.

13.13 Tracker Data

13.13.1 tracker_data_t

Type	Structure Member	Description
satid_t	sat_id	The satellite id number
tInt	timer	The length of time the satellite has been tracking
tInt	state	The satellite state
tInt	frequency	The satellite frequency
tInt	ave_phase_noise	The average tracking phase noise in 1/1000th of a degree
tInt	CN0	The signal strength.
tInt	sync_lost	Whether the tracker has lost sync

13.13.2 tracker_cut_version_t

Type	Values	
enum	0	TRK_CUT_VERSION_1_0
	1	TRK_CUT_VERSION_2_0
	2	TRK_CUT_T2_CUT_AA
	3	TRK_CUT_T2_CUT_BB
	4	TRK_CUT_T2_CUT_BC
	5	TRK_CUT_T3_AA
	6	TRK_CUT_UNKNOWN

13.14 Positioning Data

13.14.1 gnss_exclusion_type_t

Type	Values		Description
Enum	0	GNSS_NO_EXCLUSION	No satellite exclusion
	1	GNSS_RAIM_EXCLUSION	RAIM exclusion
	2	GNSS_FDE_EXCLUSION	FDE exclusion

13.14.2 gnss_fde_status_t

Type	Values		Description
Enum	0	FDE_STATUS_OFF	FDE algorithm is OFF
	1	FDE_STATUS_ON	FDE algorithm is ON

13.14.3 pos_algo_t

Type	Values		Description
enum	0	NONE	Positioning algorithm is not running
	1	LMS_ALGO	LMS algorithm is running
	2	KF_ALGO	KALMAN algorithm is running

13.14.4 fix_status_t

Type	Values		Description
enum	1	NO_FIX	Position fix is not available
	2	FIX_2D	2D fix available
	3	FIX_3D	3D fix available

13.14.5 math_ellipse_t

Type	Structure Member	Description
tDouble	semiaxis_e	The East semiaxis
tDouble	semiaxis_n	The North semiaxis
tDouble	theta	The inclination angle

13.14.6 gnss_lms_config_t

Type	Structure Member	Description
boolean_t	enable_2D_fixes	Enable/disable the 2DFix positioning.
tDouble	position_residual_thr	Position residual threshold [m]
tDouble	position_residual_thr_after_raim	Position residual threshold after RAIM [m]
tInt	min_sat_gnss_fix	Minimum number of satellites to have the position fix in GNSS mode
tInt	min_sat_single_const_fix	Minimum number of satellites to have the position fix in single constellation mode.
boolean_t	check_residual_hdop_product	Enable/disable the check on residual by hdop product.
tDouble	raim_min_angle_separation	Minimum angle separation for RAIM algorithm
tDouble	raim_thr_coeff_1	Threshold for RAIM algorithm
tDouble	min_tracked_time	Minimum satellite tracking time
tDouble	glonass_path_delay_init_value	Initialization value for GLONASS path delay
boolean_t	lock_glonass_path_delay	Enable/disable the lock of GLONASS path delay.

13.14.7 gnss_fix_config_t

Type	Structure Member	Description
boolean_t	few_sats_pos_estimation_enable	Enabling/Disabling of position fix estimation when the number of available satellites is below the configured threshold
tChar	pos_estimation_sat_th	Satellites threshold for the position fix estimation algorithm

13.15 Differential Data

13.15.1 gnss_diff_source_t

Type	Values		Description
enum	0	DIFF_SOURCE_NONE	Differential corrections source is not available.
	1	DIFF_SOURCE_WAAS	Differential corrections source is WAAS
	2	DIFF_SOURCE_RTCM	Differential corrections source is RTCM
	3	DIFF_SOURCE_AUTO	Differential corrections source is AUTO (WAAS or RTCM according to their availability)

13.15.2 diff_correction_t

Type	Values		Description
enum	0	DIFF_CORRECTION_NONE	
	1	DIFF_CORRECTION_RTCM	
	2	DIFF_CORRECTION_IONO	
	4	DIFF_CORRECTION_SLOW	
	8	DIFF_CORRECTION_FAST	

13.15.3 diff_status_t

Type	Values		Description
enum	0	DIFF_OFF	Differential corrections OFF
	1	DIFF_ON	Differential corrections ON

13.15.4 diff_mode_t

Type	Values		Description
enum	0	DIFF_ONLY	GNSS will calculates the fix only if enough differential corrections are available.
	1	AUTO_DIFF	GNSS will decides when use or not use differential corrections.
	2	NO_DIFF	Differential corrections are never used in the fix.

13.15.5 diff_data_t

Type	Structure Member	Description
`fp_s16_t`	`range_correction`	The range correction in *m*.
`fp_s16_t`	`range_rate_correction`	The range rate correction in ms^{-1}.
`diff_correction_t`	`available`	The availability of differential data.

13.16 WAAS Data

13.16.1 gnss_waas_status_t

Type	Values		Description
enum	0	WAAS_STATUS_OFF	WAAS engine OFF
	1	WAAS_STATUS _ON	WAAS engine ON

13.17 Time Data

13.17.1 time_validity_t

Type	Values		Description
enum	0	NO_TIME	Time is not available
	1	FLASH_TIME	Time has been read from backup memory
	2	USER_TIME	Time has been initialized by user
	3	USER_RTC_TIME	RTC has been initialized by user (no more used)
	4	RTC_TIME	The RTC time is available with accuracy bigger than 50ms
	5	RTC_TIME_ACCURATE	The RTC time is available with accuracy less than 50ms
	6	APPROX_TIME	If more than 300 seconds are elapsed from last ephemeris time update.
	8	ACCURATE_TIME	Either POSITION_TIME or EPHEMERIS time
	9	POSITION_TIME	Time has been corrected using position
	10	EPHEMERIS_TIME	Time has been downloaded from the sky

13.17.2 gnss_time_t

Type	Structure Member	Description
tInt	week_n	Week number
tow_t	tow	Number of second in the current week

13.17.3 master_timebase_t

Type	Structure Member	Description
tInt	ms	MTB milliseconds
tInt	timestamp	1/16368 ms

13.17.4 RTC Data

13.17.5 rtc_status_t

Type	Values		Description
enum	0	RTC_STATUS_INVALID	RTC is not available
	1	RTC_STATUS_STORED	RTC time has been read from backup memory
	2	RTC_STATUS_APPROXIMATE	RTC time has been read from RTC hardware and it is valid

13.17.6 rtc_switch_t

Type	Values		Description
enum	0	RTC_SWITCH_OFF	RTC is OFF. The gnss library cannot acces to RTC hardware
	1	RTC_SWITCH_ON	RTC is ON. The gnss library can access to RTC hardware

13.17.7 gnss_time_reference_t

Type	Structure Member	Description
gnss_time_t	gps_time	Constellation time
OS_clock_t	cpu_time	System cpu time
tDouble	ticks_per_sec	Number of tick per ms of a 1.023MHz clock
gnss_time_t	last_ephemeris_time	Time of the last transition to Ephemeris time validity
gnss_time_t	mtb_gps_time	MTB time converted to GPS time
master_timebase_t	mtb_time	Master Timebase time
tDouble	mtb_rate	MTB rate
gnss_sat_type_t	sat_type	Constellation
time_validity_t	time_validity	Time validity

13.18 PPS Data

13.18.1 pps_output_mode_t

Type	Values		Description
enum	0	PPS_OUT_MODE_ALWAYS	PPS signal always enabled.
	1	PPS_OUT_MODE_ON_EVEN_SECONDS	PPS signal enabled on even seconds.
	2	PPS_OUT_MODE_ON_ODD_SECONDS	PPS signal enabled on odd seconds.

13.18.1 pps_reference_time_t

Type	Values		Description
enum	0	PPS_TIME_REFERENCE_UTC	PPS signal synchronous with UTC time.
	1	PPS_TIME_REFERENCE_GPS_UTC	PPS signal synchronous with GPS UTC time.
	2	PPS_TIME_REFERENCE_GLONASS_UTC	PPS signal synchronous with GLONASS UTC time.

13.18.1 timing_data_t

Type	Structure Member	Description
tUChar	used_sats	Number of satellites which have contributed to the time correction.
tUChar	fix_status	GNSS position fix status when the time has been corrected.
tUChar	constellation_mask	Constellations which are enabled for timing correction.
tUChar	elevation_mask	Minimum satellite elevation angle for timing correction.
boolean_t	traim_enabled	TRAIM algorithm ON/OFF status.
tShort	traim_alarm	TRAIM max time error threshold.
traim_data_t	traim_data	TRAIM data.

13.18.2 traim_mode_t

Type	Values		Description
enum	0	TRAIM_UNDER_ALARM	All satellites time errors are below the TRAIM alarm.
	1	TRAIM_OVER_ALARM	At least one satellite time error is above the TRAIM alarm.
	2	TRAIM_UNKNOWN	TRAIM algorithm is not operating.

13.18.1 traim_data_t

Type	Structure Member	Description
boolean_t	traim_valid	TRAIM validity FALSE = not valid TRUE = valid
traim_mode_t	traim_solution	TRAIM status
tShort	ave_error	Average time error [ns]
satid_t	used_sat_id_table[]	Table of satellites used for time correction. Table size = 24
satid_t	removed_sat_id_table[]	Table of satellites removed by time correction. Table size = 24
satid_t	used_sat_id	Number of used satellites.
satid_t	removed_sat_id	Number of removed satellites.
tUChar	ref_second	Second at which the timing data will be applied on the PPS signal. According to the selected reference time the reference second could be a UTC or GPS or GLONASS time second.
tShort	residual[]	Table of time error residuals [ns]
tDouble	mod_range[]	
tDouble	mod_time_data_received	

13.18.2 pps_data_t

Type	Structure Member	Description
tDouble	rf_correction	RF pulse delay compensation.

Type	Structure Member	Description
tDouble	gps_rf_correction	RF delay compensation for the GPS path.
tDouble	pulse_duration	Pulse duration.
boolean_t	inverted_polarity	PPS signal polarity: FALSE = not inverted. TRUE = inverted.
boolean_t	enabled	PPS signal enable/disable status: FALSE = disabled TRUE = enabled
boolean_t	pps_valid	
boolean_t	pps_synch_valid	Indicates if there are all conditions to have a valid PPS synchronization (e.g. utc delta time data valid, good reference time validity, etc.)
tUChar	sat_threshold	Minimum satellites for PPS signal generation.
fix_status_t	fix_condition	GNSS position fix condition for PPS signal generation.
pps_output_mode_t	output_mode	PPS output mode.
pps_reference_time_t	reference_time	Reference time for PPS synchronization.
tShort	gps_utc_delta_time_s	UTC time leap seconds.
tShort	gps_utc_delta_time_ns	UTC - GPS time difference (ns)
tDouble	quantization_error	Quantization error [s] on PPS rising edge setting.
tDouble	pps_clk_freq_Hz	Internal PPS counters clock [Hz]
tDouble	tcxo_clk_freq_Hz	
tChar	pps_clk_setting	
timing_data_t	applied_timing_data	Timing data which is applied on the reference second.
tDouble	glonass_rf_correction	RF delay compensation for the GLONASS path.

Type	Structure Member	Description
tShort	glonass_utc_delta_time_ns	UTC - GLONASS time difference (ns)
tDouble	compass_rf_correction	RF delay compensation for the COMPASS path.
tShort	compass_utc_delta_time_ns	UTC - COMPASS time difference (ns)

13.18.3 gnss_pps_params_t

Type	Structure Member	Description
boolean_t	pps_enabled	PPS signal enable/disable status. 0: PPS signal disabled. 1: PPS signal enabled.
tUShort	pps_cpsh	Value to be written on CPSH register value [ms]
tUShort	pps_cprh	Value to be written on CPRH register value [ms]
tUInt	pps_cpsl	High resolution value to be written on CPSL register value. It is a 24-bit value. Only the upper 16-bits are written on the CPSL register. The 8 lower bits are needed to avoid accuracy degradation during propagation.
tUInt	pps_cprl	High resolution value to be written on CPRL register value. It is a 24-bit value. Only the upper 16-bits are written on the CPRL register. The 8 lower bits are needed to avoid accuracy degradation during propagation.
tUInt	pps_cpl_step	High resolution value representing the ticks step to be added to the cpl registers to move ahead by one second.
tUShort	pps_cph_step	value representing the ms step to be added to the cph registers to move ahead by one second.
OS_clock_t	rising_edge_cpu_time	CPU time stanp of the next PPS rising edge event.

`OS_clock_t`	`Pulse_duration_ticks`	CPU time ticks representing the PPS pulse duration.

13.19 XTAL data

13.19.1 gnss_xtal_mgr_res_net_data_t

Type	Structure Member	Description
tInt	B	Beta(3380 or 4250) value used in the logarithmic function relating temperature and ADC read
tInt	Rto	Rto value(10k or 100K) used in the logarithmic function relating temperature and ADC read
tInt	Rs	Rs (serial resistor value in the xtal polarization network typ.10K) used in the logarithmic function relating temperature and ADC read

13.19.2 gnss_xtal_mgr_nvm_data_t

Type	Structure Member	Description
tDouble	Coeff_val[4]	Table containing the states[A3 A2 A1 A0]
tDouble	variance_va[16]	Covariance values table
tInt	Ext_sel	Selector that must be set to 1 to indicate that state coefficients and covariance will be set from extern. It must be set to zero to utilize NVM pre-stored values (or standard values)
gnss_time_t	time_info	

13.19.3 gnss_xtal_monitor_t

Type	Structure Member	Description
tDouble	T	Temperature [Celsius deg]
tDouble	filtered_ramp_rate	Xtal filtered ramp rate [Hz/s]
tDouble	raw_ramp_rate	Xtal raw ramp rate [Hz/s]
tDouble	nav_xtal_ADC_rate	Thermistor voltage rate [ADC steps/s]

tDouble	nav_xtal_adc_stored_value	Reserved
boolean_t	kf_ramp_rate_status	Reserved
boolean_t	acq_reactivation_status	Reserved

13.20 Low Power data

13.20.1 gnss_low_power_cyclic_mode_t

Type	Structure Member	Description
tU8	ehpe_threshold;	Set the EHPE threshold for automatic activation of adaptive and cyclic mode
tU8	N_sats_reduced;	Number of maximum satellites of the adaptive algorithm
gnss_sat_type_mask_t	const_mask_init;	Set the constellation mask restored when EHPE is below the threshold
boolean_t	reduced_type;	Enable/Disable adaptive algorithm
boolean_t	duty_cycle_on_off;	Enable/Disable cyclic mode
tShort	ms_off;	Duration of OFF period of the cyclic mode

13.20.2 gnss_low_power_periodic_mode_t

Type	Structure Member	Description
tU32	fix_period;	Distance between fix activities
tU8	fix_on_time;	Number of fix reported for each fix activities
tU8	EPH_refresh;	Enable/Disable ephemeris refresh
tU8	RTC_refresh;	Enable/Disable RTC calibration
tU8	NoFixTimeout;	Time to declare fix loss
tU16	NoFixOffTime	OFF duration time after No Fix Timeout occured
boolean_t	periodic_mode	Enable/Disable periodic mode

13.20.3

13.20.4 gnss_low_power_data_t

Type	Structure Member	Description
boolean_t	steady_state;	If TRUE, the GNSS engine has completed the satellite acquisition phase
boolean_t	no_fix;	Reserved
boolean_t	glonass_tow_referesh;	Reserved
tChar	counter_glonass_eph_ON;	Reserved
gnss_sat_type_mask_t	eph_const_mask;	Reserved
tUInt	eph_interval;	Reserved
boolean_t	eph_long_refresh_timer	Reserved
gnss_low_power_periodic_data_t	cyclic	Adaptive and Cyclic data
gnss_low_power_periodic_data_t	periodic	Periodic data

13.20.5 gnss_low_power_cyclic_data_t

Type	Structure Member	Description
gnss_low_power_state_t	state;	Low Power algorithm status
tDouble	ehpe;	Current Estimation of the Horizontal Position Error
tDouble	average_ehpe;	EHPE Average
tDouble	average_ehpe_nmea;	Reserved
tUChar	counter_ehpe_IN;	Reserved
tUChar	counter_ehpe_OUT;	Reserved
tUChar	counter_NO_FIX_IN;	Reserved
tUChar	counter_NO_FIX_OUT;	Reserved
tShort	average_NO_FIX;	Reserved
tShort	sats_used;	Reserved
boolean_t	reduced_type;	Reserved
boolean_t	duty_cycle_on_off;	Reserved
tInt	ms_off;	Reserved
boolean_t	duty_cycle_state;	Reserved

13.20.6 gnss_low_power_periodic_data_t

Type	Structure Member	Description
gpOS_clock_t	SavedTimerTicks;	Reserved
tU16	NoFixCnt;	Reserved
tU8	FixOnCnt;	Reserved
boolean_t	NoFixCntSelector;	Reserved
boolean_t	RTCTrimRequired;	Reserved

14 Disclaimer

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.