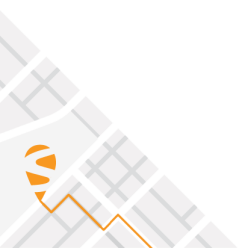




## **Post Processing SDK**

---

User Manual



Post Processing SDK

February 20, 2017

Applicable to the Post Processing SDK v4.1.8

© Copyright 2000-2017 Septentrio NV/SA. All rights reserved.

Septentrio NV  
Greenhill Campus, Interleuvenlaan 15i  
3001 Leuven, Belgium

<http://www.septentrio.com>  
[support@septentrio.com](mailto:support@septentrio.com)  
Phone: +32 16 300 800  
Fax: +32 16 221 640  
 @Septentrio

# List of Contents

<b>CONTENTS</b>	<b>5</b>
<b>1 OVERVIEW</b>	<b>8</b>
1.1 Introduction .....	8
1.2 About The Product.....	8
1.3 Requirements .....	8
1.3.1 System Requirements .....	8
1.3.2 Development Requirements .....	9
1.3.2.1 Microsoft Windows .....	9
1.4 Functional Components .....	9
1.4.1 Application and Tools .....	9
1.4.2 SDK Components .....	9
1.5 Important Definitions .....	10
1.6 Document Conventions.....	14
1.7 Contact .....	15
1.8 World Wide Web .....	15
<b>2 INSTALLATION</b>	<b>15</b>
2.1 Directories .....	16
2.2 PPSDK settings .....	16
2.2.1 Permissions File .....	16
2.2.2 NeQuick Path .....	17
2.2.3 Antenna Info File .....	17
2.2.4 Debug Log File.....	17
<b>3 LICENSE</b>	<b>17</b>
3.1 License Types.....	17
3.1.1 Single User License .....	17
3.1.2 Demo License .....	18
3.2 Permissions .....	18
<b>4 PPSDK INTERFACES</b>	<b>18</b>
4.1 Proprietary Binary Output (SBF) .....	18
4.2 SBF DiffCorrIn blocks .....	19
4.3 The SNMP' Interface.....	19
4.4 Management Information Base (MIB) .....	19
<b>5 SEPTENTRIO PPSDK POSTNAV APPLICATION</b>	<b>20</b>
5.1 PostNav Main Form .....	20
5.2 PostNav Extended Main Form .....	22
5.3 PostNav File Menu Options .....	23
5.4 PostNav Communication Menu Options .....	24
5.5 PostNav Navigation Menu Options.....	25
5.6 PostNav Tools Menu Options .....	28
5.7 PostNav Help Menu Options .....	29
<b>6 SEPTENTRIO PPSDK RINEX CONVERTER APPLICATION</b>	<b>31</b>
6.1 RINEX Converter Main Form .....	31
6.2 RINEX Converter File Menu Options .....	32
6.3 RINEX Converter Tools Menu Options .....	32
6.4 RINEX Converter Help Menu Options .....	32

<b>7</b>	<b>How To...</b>	<b>32</b>
7.1	Manage SBF Streams .....	32
7.1.1	SBF Input Requirements .....	34
7.1.2	SBF Output.....	35
7.2	Control the processing of Data .....	35
7.2.1	Using ASCII Commands .....	35
7.2.2	Using the SNMP' Interface .....	36
7.2.3	Using pvtcalc or PostNav.....	37
7.3	Check the Capabilities of your PPSDK .....	37
7.4	Handle structures returned by the PPSDK API.....	38
7.5	Return progress indication to the user .....	38
7.5.1	SSNSBFStream_pcSetCallback.....	38
7.5.2	SSNSBFStream_pcSetUserDataCallback .....	38
7.5.3	SSNSBFStream_pcSubscribe.....	38
7.5.4	SSNSBFStream_pcUnsubscribe .....	39
7.5.5	SSNSBFStream_pclIsSubscribed .....	39
7.6	Using Satellite Constellations .....	40
7.7	Merge Files .....	40
7.8	Obtain an Standalone PVT Solution .....	41
7.9	Obtain an SBAS PVT Solution .....	41
7.10	Obtain DGPS or RTK using input from a base station to increase accuracy .	42
7.10.1	Obtain an DGPS PVT Solution .....	42
7.10.2	Obtain an RTK PVT Solution .....	43
7.11	Use backwards post-processing .....	44
7.12	Use RINEX files with the PPSDK.....	44
7.13	Use a different NGS Antenna file .....	45
7.14	Check the Permission File .....	45
7.15	Use the API to Analyze SBF Files .....	46
7.16	Use with other programming languages .....	46
7.17	Create the Debug Log File whilst using the PPSDK.....	47
7.18	Interpret errors returned by the PPSDK .....	47
<b>8</b>	<b>RECEIVER (PPSDK ENGINE) OPERATION DETAILS</b>	<b>51</b>
8.1	GNSS Constellation and Signal/Satellite Usage .....	51
8.2	Generation of Measurements.....	51
8.2.1	Pilot vs. Data Component .....	52
8.3	Time Management .....	52
8.4	Computation of Position, Velocity, and Time (PVT Solution) .....	53
8.4.1	SBAS Positioning .....	54
8.4.2	DGPS Positioning (Single and Multi-Base) .....	55
8.4.3	RTK Positioning.....	55
8.4.3.1	Pseudorange versus carrier phase: ambiguity .....	55
8.4.3.2	Carrier Phase Positioning.....	56
8.4.3.3	Integer Ambiguities (RTK-fixed) .....	56
8.4.3.4	Floating Ambiguities (RTK-float) .....	56
8.4.3.5	Moving Base .....	56
8.4.3.6	Datum Transformation .....	57
8.4.3.7	Antenna Effects .....	57
8.4.3.8	Practical Considerations .....	58
8.4.4	Precise Point Positioning.....	59
8.4.4.1	PPP Seeding .....	59
8.4.4.2	PPP Datum Offset .....	59

8.4.4.3	Tide Corrections .....	59
8.5	INS/GNSS Integration .....	60
8.5.1	Calibration .....	60
8.5.1.1	IMU Sensor Orientation .....	60
8.5.1.2	Lever Arm .....	61
8.5.2	Alignment .....	61
8.5.2.1	Static Coarse Alignment .....	62
8.5.2.2	In-Motion Alignment .....	62
8.5.3	Zero-Velocity Update (ZUPT) .....	62
8.6	Receiver Autonomous Integrity Monitoring (RAIM) .....	63
8.6.1	Integrity Algorithm .....	64
8.6.2	Internal and External Reliability Levels .....	65
<b>9</b>	<b>DISTRIBUTION</b> .....	<b>65</b>
9.1	Libraries .....	65
9.2	Environment Variables .....	66
<b>10</b>	<b>SEPTENTRIO PPSDK SAMPLE APPLICATIONS</b> .....	<b>66</b>
10.1	ComputePVT .....	66
10.2	GetMIBInfo .....	66
10.3	ListBlocks .....	67
10.4	SendCommand .....	67
10.5	ListMissingEpochs .....	67
<b>11</b>	<b>SEPTENTRIO TOOLS</b> .....	<b>68</b>
11.1	pvtcalc .....	68
11.2	rin2sbf .....	71
11.2.1	GLONASS Frequency File .....	71
11.2.2	Glonass Leap Seconds .....	72
11.3	sbf2asc .....	73
11.4	sbf2cmd .....	82
11.5	sbf2gpx .....	83
11.6	sbf2kml .....	84
11.7	sbf2rin .....	86
11.8	sbf2sbf .....	88
11.9	sbf2stf .....	91
11.10	sbfblocks .....	92
11.11	ngs2bin .....	93
<b>12</b>	<b>RTCM OVERVIEW</b> .....	<b>93</b>
<b>A</b>	<b>ATTITUDE ANGLES</b> .....	<b>94</b>
	<b>GLOSSARY</b> .....	<b>97</b>

## List of Figures

1-1	API Components .....	10
1-2	Reference Insertion Common time .....	11
1-3	Reference Insertion into an Intermediate Merged file .....	11
1-4	SBF output .....	12
4-1	PPSDK Architecture .....	18
5-1	PostNav Main Form. ....	20
5-2	PostNav SBF Analysis .....	21

5-3	PostNav Extended Main Form .....	22
5-4	PostNav Diagnostics .....	24
5-5	PostNav SBF Output .....	25
5-6	PostNav PVT Mode .....	26
5-7	PostNav Timing .....	26
5-8	PostNav Satellite Usage .....	27
5-9	PostNav Signal Usage .....	28
5-10	PostNav Base Station Parameters .....	28
5-11	PostNav Receiver ID.....	29
5-12	PostNav Receiver Permissions.....	30
5-13	PostNav Engine Interface, Permitted Capabilities .....	30
6-1	RINEX Converter .....	31
7-1	Rinex Decoder .....	45
8-1	Example of the evolution of the receiver time offset with respect to the GNSS time if $K=0.5$ .....	53
8-2	Antenna mount. ....	58
8-3	Example of values of $(\theta_x, \theta_y, \theta_z)$ for different IMU orientations on a car. X, Y and Z refer to the axes as marked on the IMU enclosure.....	61
8-4	Example of antenna/IMU relative position. In this example, $\Delta X$ and $\Delta Y$ are positive, while $\Delta Z$ is negative. ....	61
8-5	Statistical test outcomes.....	63
A-1	Vehicle reference frame. ....	94
A-2	Euler angle sequence. ....	96

## List of Tables

1-0	Document Conventions .....	14
2-0	Installation Directory.....	16
7-1	Predefined SBF Output Groups.....	33
7-2	ssn_error_t structure.....	48
7-4	Module Codes .....	48
7-5	Submodule Codes.....	48
7-6	Warning Codes.....	49
7-7	Error Codes.....	50
7-8	Error Codes.....	51
11-1	pvtcalc Arguments.....	69
11-2	pvtcalc Arguments (part 2) .....	70
11-3	rin2sbf Arguments .....	71
11-4	sb2asc Arguments .....	73
11-5	sb2asc Row Identifier .....	74
11-6	sb2asc (Short)MeasEpoch block.....	75
11-7	sb2asc PVTCartesian block .....	76
11-8	sb2asc PVTGeodetic block.....	77
11-9	sb2asc PVTcov block .....	77
11-10	sb2asc PVTDOP block .....	78
11-11	sb2asc AttitudeEuler block .....	78
11-12	sb2asc AttitudeCovEuler block.....	78
11-13	sb2asc ExtEvent block.....	79
11-14	sb2asc ReceiverStatus block .....	79
11-15	sb2asc BaseStation block.....	79
11-16	sb2asc BaseLine block .....	79

11-17	sbf2asc BaseLink block .....	80
11-18	sbf2asc GPSAlm block .....	80
11-19	sbf2asc AuxPos block .....	81
11-20	sbf2asc ExtSensorMeas block .....	81
11-21	sbf2cmd Arguments .....	82
11-22	sbf2gpx Arguments .....	83
11-24	sbf2kml Arguments .....	85
11-26	sbf2sbf Arguments .....	90
11-27	sbf2stf Arguments.....	91
11-28	sbfblocks Arguments .....	92
11-29	ngs2bin Arguments .....	93

# 1 Overview

This chapter will give a brief overview of the Post Processing Software Development Kit. It will explain the purpose of the Post Processing Software Development Kit, the system requirements and talk briefly about the files and modules that are part of the SDK.

## 1.1 Introduction

Welcome to the PPSDK, Septentrio NV/SA's Post Processing Software Development Kit for Windows. With the software and documentation included in this package, you will be able to develop applications equipped with Septentrio NV/SA's state of the art post processing technology.

## 1.2 About The Product

The PPSDK is a **kinematic and static** GNSS Post Processing Software Development Kit. The PPSDK does not only support the basic manipulation of SBF files, it also supports post processing measurement data into a full PVT solution. The PPSDK also allows the conversion from RINEX data files into SBF data files to support most commercial single and/or dual frequency receivers.

The PPSDK has the advantage of exposing an interface which is very similar to the interface exposed by the Septentrio NV/SA GNSS receivers. As an example, the PPSDK accepts SBF data as input and will also output SBF data. This is beneficial in order to keep a single tool set capable of reading the data, minimizing the effort needed by the users of the PPSDK when post processing or preparing a file. Just as when using a Septentrio NV/SA GNSS receiver you can pass commands to control the output or the Engine Operation of the PPSDK. This helps again in increasing modularity and re-usability in the software application implemented on top of the PPSDK.

## 1.3 Requirements

### 1.3.1 System Requirements

The Post Processing Software Development Kit has the following minimum requirements:

- Microsoft Windows 7 or newer
- The minimum CPU requirement is 800 MHz Pentium or above
- The minimum RAM requirement is 1 GB or more RAM is recommended

**Note 1.** : Improved CPU performance will be obtained using an up to date PC.



## 1.3.2 Development Requirements

### 1.3.2.1 Microsoft Windows

In order to build applications for Windows using the PPSDK one of the following development environments is necessary:

- Microsoft Visual Studio 2008, 2010, 2013 or 2015 (make sure you have the latest service packs).
- Express Editions of Microsoft Visual Studio 2008, 2010, 2013 or 2015 C++ or C# Editions with latest service packs.
- Msys MinGW gcc compiler version 4.9.2 (Rev2, Built by MSYS2 project)

**Important: Any PPSDK application using the ppengine (for PVT computation) requires a higher size of reserved stack in the Visual Studio Project. The supplied sample project for this application is using a value of 10000000.**



Notes: There are a few items that need to be born in mind using the Microsoft Compilers.

- Minor floating point differences may occur between the different version of the compilers. This will most likely be affected by the platform of the user, so it might be hard to get identical files.
- Make sure that the correct version of PPSDK is used with your application (either the VS2008, VS2010, VS2013 or VS2015 version). Running an application built with a Visual Studio (e.g. VS2008) different from the PPSDK library used by linking (e.g. VS2010) may result in errors.

## 1.4 Functional Components

### 1.4.1 Application and Tools

The main functional components of the PPSDK which can be used after installation without programming are the following:

- **PostNav**: a GUI application to post process SBF files (See section 5).
- **RINEX Converter**: a GUI application to convert RINEX files into SBF.
- **pvtcalc**: a CMD line application to post process SBF files.
- **rin2sbf**: a CMD line application to convert RINEX files into SBF.
- A set of executable (Windows Console) applications to process, analyze and convert SBF files.

### 1.4.2 SDK Components

The main components of the PPSDK which can be used for development of a custom application are the following:

- An **Application Program Interface (API)** enabling the creation of applications to preprocess SBF input files, decode RINEX, control the PVT calculation and analyze or edit the resulting output SBF file
- The core engine that performs the actual computation of a PVT solution
- Dynamic and static libraries for linking to the final application

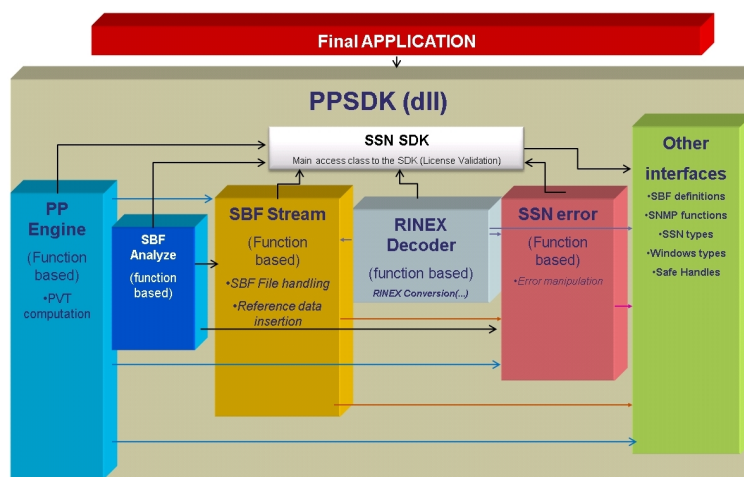
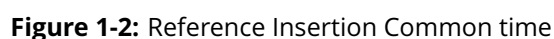


Figure 1-1: API Components

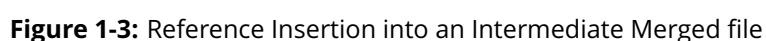
## 1.5 Important Definitions

When working with the PPSDK it is important to consider the following definitions which will allow you to understand the main functionality of the product:

- **SBF Stream:** It refers to an SBF file which is loaded in memory. Note that the file can be either an Input or a Reference file.
- **Input:** An Input file normally refers to an SBF file (or stream) which can be used for post-processing. The Input file may be a Rover Kinematic or Static file. While it is recommended to work with SBF files from Septentrio receivers, the PPSDK allows you to use SBF files which have been obtained from a RINEX to SBF conversion using the appropriate tools delivered with the PPSDK.
- **Reference:** It refers to the SBF file which will be used as a Reference file. As such the file is normally a Static file (sometimes referred as a Base file or Reference Network file) but may also be a Kinematic file in case the Moving Base feature is supported in your version of the PPSDK. While it is also recommended to use SBF files from Septentrio receivers as Reference files, you can also use RINEX files from other sources by using the appropriate RINEX to SBF conversions tools included with the PPSDK (e.g. using RINEX files from downloadable reference networks and converting them to SBF so that they can be used in the PPSDK post-processing).
- **DiffCorr:** DiffCorr messages are normally applicable for real time transmissions of RTCM or CMR streams. However in the PPSDK DiffCorr data is generated when inserting a Reference Station data file (SBF) into an Input file (Rover or Static). The process of insertion (converting SBF measurements into DiffCorr RTCM) allows the PPSDK to be able to post-process files where the final solution is improved thanks to the use of differential GNSS techniques (e.g. DGPS or RTK). While you can still use SBF files with CMR on the post-processing engine (using a log file from a Septentrio receiver), the PPSDK can only generate RTCM messages for insertion or intermediate files.

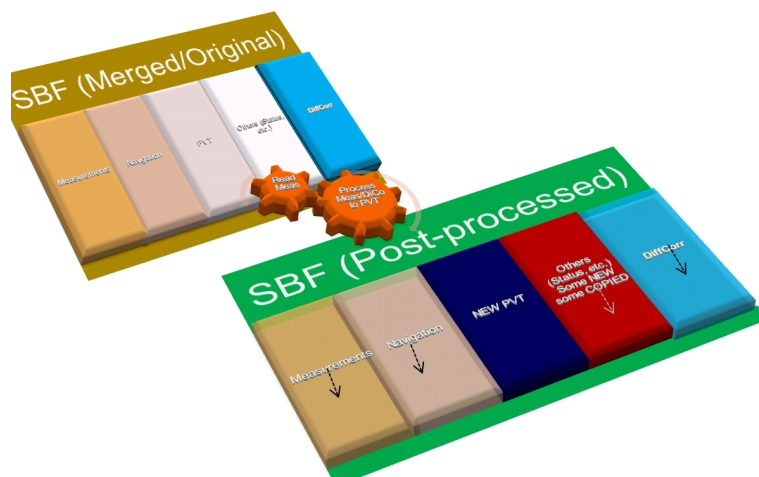


- **Intermediate or Merged file:** It is the SBF file after insertion of DiffCorr data which may be created with the PPSDK and which contains the original Input file (rover) merged with the Reference data transformed into DiffCorr messages, this intermediate file is then used for post-processing with Differential Corrections. With the PPSDK you can create intermediate files containing either RTCM2 or RTCM3 messages (option of the PPSDK). You must make sure that the Reference file you are inserting corresponds to the same time interval of the Input file. While an intermediate file normally contains DiffCorr data, you may also create intermediate files which only have new inserted commands.



- **PPSDK API parameters:** Via the PPSDK API you may pass parameters which allow you to control multiple functionality of the PPSDK. However you need to write an application on top of the PPSDK libraries to be able to access this. Some of these parameters are exposed on the command line or in the gui applications delivered with the PPSDK. **PPSDK Setting Commands:** Just like in a Septentrio receiver you may control specific settings of the engine by using set or get commands. These settings allow you to control things such as Elevation Mask, PVT Mode, etc.

- **SBF Output:** Just like in a Septentrio receiver the PPSDK allows you to output SBF files from the post-processed file. This has enormous advantages since you will be able to use RxTools or any other SBF capable tool-set for analyzing your results.



**Figure 1-4: SBF output**

- **PPSDK PVT modes:** When using the PPSDK users must bear in mind that the processing modes may not entirely reflect the names used in other post-processing SW applications; the reason being that the PPSDK tries to match the PVT modes used in the Septentrio receivers (which also eases compatibility among other Septentrio tools). The following list gives a mapping of how the PVT Modes are matched with other common names in post-processing:
  - autonomous or standalone (StandAlone in the PPSDK)
  - code - differential (DGPS in the PPSDK)
  - phase - differential with fixed ambiguities (RTK-fixed in the PPSDK)
  - phase - differential with float ambiguities (RTK-float in the PPSDK)
  - SBAS is not typical for other post-processing applications, however the PPSDK is capable of this post-processing in the same way as the Septentrio receivers.

Within the PostNav application you will see the possibility to post-process files in Static Positioning Mode (normally used for Reference files), this feature is however not needed for normal Post-processing and is mainly only used when the automatic insertion of DiffCorr data is done (internally). If you would like to use the PPSDK for post-processing a static solution it is recommended to use a Differential mode such as RTK which should give the appropriate accurate solution (you can then choose the point with the highest accuracy by taking a look at the covariances and the PVT Mode obtained). Basically all the settings in the top menu's of PostNav reflect the settings to be used for the processing of the already merged file, while the some of the settings in the main screen are used for the generation of the Merged data file.

- **PPSDK Reference Position:** When using the PPSDK with Differential positioning modes (e.g. DGPS, RTK) users must bear in mind that the Position of the Reference Station is important for having an accurate post-processed solution. When generating a Merged data file the PPSDK by default tries to retrieve this position from the SBF Reference file by looking for any Position blocks on the SBF file (Geodetic or Cartesian). The PPSDK however also provides a way to force the Position which should be embedded in the DiffCorr merged data file. When converting RINEX files to SBF files the RINEX converter will generate a single Position block which will be then used for the DiffCorr insertion.
- **PPSDK Antenna Type:** When using the PPSDK with Differential positioning modes (e.g. DGPS, RTK) users must bear in mind that the Antenna types of both the Rover and the Reference receiver are important for having an accurate post-processed solution. When generating a Merged data file the PPSDK by default tries to retrieve the antenna type from the SBF Reference file by looking for any `ReceiverSetup` blocks

on the SBF file (Geodetic or Cartesian). The PPSDK however also provides a way to force the Antenna type which should be embedded in the `DiffCorrIn` merged data file. When converting RINEX files to SBF files the RINEX converter will generate a single `ReceiverSetup` block which will be then used for the `DiffCorrIn` insertion. The Antenna type for the Rover can be changed by using the PPSDK settings command `setAntennaOffset`. Both antennas need to be known by the PPSDK (see 7.13). More information about the antenna effects can also be found at 8.4.3.7.

- **PPSDK Reference ID:** Just like the Reference Position, the Reference ID is an option of the PPSDK which allows you to force the Reference ID which will be put in the `DiffCorr` merged data. This feature should not be confused with the option in the Navigation menu's of the PostNav application (or the `setDiffCorrUsage` command) where the user can select which Reference station to use for the already merged `DiffCorr` messages (e.g. when having an SBF file with multiple station data).

## 1.6 Document Conventions

Throughout the manual several formatting styles are used. Please see Table 1-0 for a detailed list of formatting styles.

Convention	Type Of Information
<b>Bold</b> type	Used to refer to titles of chapters or sections.
<i>Italic</i> type	Used to refer to titles of manuals and to emphasize certain words, such as new terms.
Monospaced type	Sets of code examples and shows syntax spacing.
KEYCAPS	Indicates a key on your keyboard.

**Table 1-0:** Document Conventions

## 1.7 Contact

Septentrio NV/SA wants you to get the most from its software. Feel free to contact us with your suggestions or questions at one of the following addresses:

### **Septentrio Headquarters**

Greenhill Campus  
Interleuvenlaan 15i  
3001 Leuven  
Belgium  
Tel: +32 (0) 16 300 800  
Fax: +32 (0) 16 221 640  
<http://www.septentrio.com>  
[support@septentrio.com](mailto:support@septentrio.com)

### **USA Office**

20725 Western Avenue Suite #144  
Torrance, CA 90501  
USA  
Phone: +1 (888) 655-9998  
Fax: +1 (323) 297-4648  
<http://www.septentrio.com>  
[support@septentrio.com](mailto:support@septentrio.com)

When contacting Septentrio NV/SA with a question about the PPSDK, make sure to supply the following information:

- The version number of the PPSDK as mentioned on the title page of the documentation set.
- The application environment: architecture of the application.
- The operation system and version numbers.
- For a PPSDK problem, the error code returned from the API call and its corresponding error string, all parameters used to call the API.
- The latest debug log file if available. See sections 2.2.4 and 7.17 for details as to how to create a log file.

## 1.8 World Wide Web

You can visit Septentrio NV/SA on the World Wide Web to find out about new products, software releases, available patches and information about various other fields of interest.

URL: <http://www.septentrio.com>

## 2 Installation

The following chapter explains the installation process of the PPSDK and gives an overview of all the files included in the PPSDK installation package.

Installation is accomplished by executing the following file:

Septentrio-Post\_Processing\_SDK-4.1.8-Setup.exe is the main installation routine. Run the file and follow the instructions display on the screen. The only choice to be made is one of installation folder if the default folder offered is not wanted.

Uninstallation is done by running the uninstall.exe file found in the main installation directory. This will remove the PPSDK. The installation folder will also be removed provided that no files have added after installation. This is to preserve any user files.

## 2.1 Directories

After the PPSDK has been installed a number of folders are created in the installation directory.

Directory	Description
applications	Contains PostNav and the RINEX Converter, Windows applications to post process SBF files and an application to convert RINEX files into SBF using the PPSDK API
doc	Contains the documentation of the PPSDK such as the manuals, API reference and release notes
includes	Contains the C header files of the PPSDK API
libdata	Contains any data files that may be required by the PPSDK in a computation
library	Contains the DLL and LIB files
license	Contains the permissions file (after the license installation is executed)
prerequisites	Contains prerequisites needed by the PPSDK should you not have Visual Studio or the Microsoft run-time libraries installed on your system
samples	Contains sample applications and source code to demonstrate the usage of the PPSDK API
tools	Contains pvtcalc, rin2sbf and some other basic tools to perform some of the most common basic tasks available through the PPSDK API

**Table 2-0:** Installation Directory

The PPSDK license installer also registers the environment variables described in the next section.

## 2.2 PPSDK settings

The PPSDK depends on some settings to access a debug log file, the permissions file or the PPSDK library required data. These variables are installed automatically by the PPSDK Settings Manager during installation. The are stored in the <home\_dir>\.septentrio\receiver.conf file. After installation the settings can be changed by running the PPSDK Settings Manager. Alternatively those settings can also be set via environment variables. Note that when a setting is present in both the settings file and the environment variables the setting from the settings file is used. More information about the permissions files of the PPSDK can be found in chapter 3.

If one of the settings is absent it is likely not all functionality available through the PPSDK is allowed and a corresponding error code will be returned by the PPSDK API functions. Please check the presence of the settings variables in case of unexpected behavior or error codes regarding the license with the PPSDK Settings Manager.

### 2.2.1 Permissions File

The most important setting is the one pointing to the location of the permissions file and is called SSN\_PPSDK\_INFOPATH\_PERMSFILE.



## 2.2.2 NeQuick Path

The `SSN_NEQUICK_PATH` defines the location of NeQuick data to be used by the PPSDK Engine when using the Galileo Constellation (note that although not used for processing GPS or GLONASS, the location of these files must be specified). This data provides a measure of the electron density in the Ionosphere and the Total Electron Content between the receiver and any satellite.

## 2.2.3 Antenna Info File

The `SSN_ANTINFO_FILE` defines the location of Antenna Info file (NGS) to be used by the PPSDK Engine when RTK computation is done. The file allows to compensate for the phase center variations and compute the ARP position depending on the antenna used (See section 11.11 for understanding how to make a new NGS antenna file. Section 8.4.3.7 also talks about the effects that an antenna type has on the final computed solution).

## 2.2.4 Debug Log File

The `SSN_LOGFILE` is optional and allows the basic debug output generated by the PPSDK to be redirected to an ASCII text file. The setting is the full path to and the name of the log file. This may be helpful in debugging a user application and is necessary when requesting Septentrio NV/SA support. See section 7.17 in order to create a logfile.

# 3 License

The following chapter explains the license mechanism of the PPSDK.

All license types require the presence of certain environment variables set during the installation process. See chapter 2.2 for more information.

## 3.1 License Types

The PPSDK has two main types of licensing schemes. Each scheme has its own custom features and possibilities but all of them use a permissions file to handle the PVT computation options. See chapter 3.2 for more information about the permissions file.

The following chapters give a brief overview of the different license schemes and describe the possibilities, limitations and requirements.

A USB dongle is required to use the PVT related functionality of the PPSDK and is required at all time during the PVT computation. The license scheme does not prevent the PPSDK from being installed on multiple PCs. In order to perform a PVT calculation the dongle must be attached to a USB port on the PC. However, processing and manipulation of SBF files such as conversion from RINEX to SBF does not require the presence of the dongle. Please contact Septentrio sales team to obtain the more suitable license for your needs.

### 3.1.1 Single User License

The *Single User License* type is designed for customers who require only single instances of the PPSDK. The Single User License uses a USB dongle to verify the permissions file. Each USB dongle belongs to one single permissions file.

### 3.1.2 Demo License

The *Demo License* type is the second license type and is designed to give a brief overview of the PPSDK functionality and possibilities. The Demo License exposes all functionality in the permission file but limits the number of PVT calculations that can be performed. The limit is contained in a dongle delivered with the demo version.

## 3.2 Permissions

Just like the SSN receivers the PPSDK is protected via a permissions file used to enable or disable settings used by the PVT computation functionality. The permissions file defines which satellites, signals and commands can be used by the PPSDK. The file can only be generated by Septentrio NV/SA and is protected by electronic signature.

To install a permission file you need to run the PPSDK License installer (Septentrio-Post Processing SDK License-4.1.8-Setup.exe) delivered by Septentrio NV/SA.

## 4 PPSDK Interfaces

This chapter gives an overview of the interfaces to the PPSDK Engine, providing the user with the facilities to control the parameters of a computation.

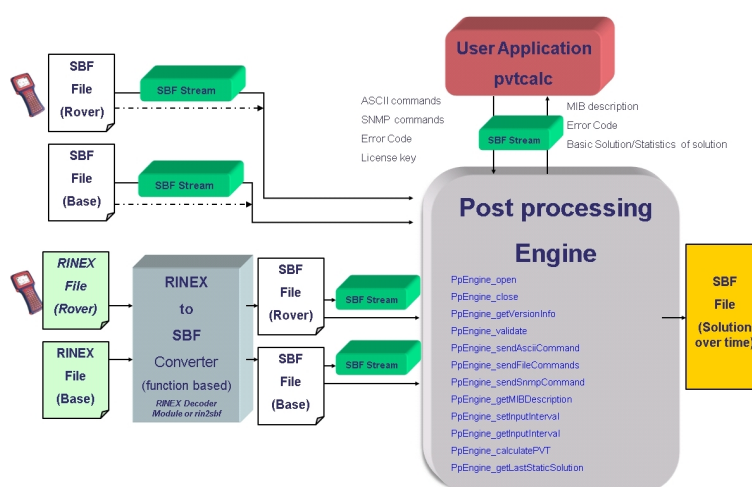


Figure 4-1: PPSDK Architecture

## 4.1 Proprietary Binary Output (SBF)

The PPSDK operates on and produces output conforming to the Septentrio Binary Format (SBF) definition. In SBF files data are arranged in SBF blocks identified by block IDs. All the blocks begin with the SBF identifier \$@. Please refer to the SBF Reference Guide for a complete definition of SBF. Please note that the Reference Guide was written from the point of view of the receiver. The PPSDK cannot produce all SBF blocks in the Guide. It is however capable of passing any block in the input SBF File through to the output file. SBF blocks may have different versions having different block IDs. For example there are version 1 and 2 PVTGeodetic blocks. The version 1 blocks are produced by earlier PolaRx receivers. The version 2 blocks being produced by the AsteRx receivers. The PPSDK makes no distinction and will process all such blocks.

The benefit of SBF is its compactness. This format should be your first choice if you wish to process the detailed information from the receiver.

The list of supported SBF messages on your particular PPSDK version can be found in the Command and Log Reference Card. An intuitive GUI, 'SBF Converter', is part of the RxTools package, allowing SBF conversion into RINEX, KML, GPX, ASCII, and other possible formats. 'SBF Analyzer' is another application which will allow you to analyze SBF files in an easy and intuitive way. To obtain either 'SBF Converter' or 'SBF Analyzer' please refer to the Septentrio NV/SA Website.

## 4.2 SBF DiffCorrIn blocks

In order to compute a DGPS, RTKFloat or an RTKFixed solution the PPSDK Engine requires the differential corrections contained in `DiffCorrIn` SBF blocks. The PPSDK can utilize the `DiffCorrIn` blocks contained in the original input stream however it also contains the means to regenerate `DiffCorrIn` blocks as follows.

If the original input SBF stream contains `DiffCorrIn` blocks in RTCM 2.x, RTCM 3.x or CMR 2.0 format and the stream is not merged with a reference stream the `DiffCorrIn` blocks will be used.

If the original input SBF stream is merged with a reference stream any `DiffCorrIn` blocks contained in the original stream may be optionally be discarded. The merged stream will contain `DiffCorrIn` blocks generated from the data in the reference SBF stream again in RTCM 2.x or 3.x format as specified by parameters to the `SSNSBFStream_insertReferenceStream` function. Any `DiffCorrIn` blocks contained in the input streams may be optionally removed or retained. The user has control over the RTCM format and generated RTCM messages as follows:

Using either PostNav or pvtcalc three RTCM options (RTCM Version) are available.

- RTCM version 2 All RTCM 2.x messages are generated when data in the reference stream permits;
- RTCM version 2 DGPS + RTK RTCM 2.x messages 18, 19 and 23 and 24 are generated when data in the reference stream permits;
- RTCM version 3 All RTCM 3.x messages are generated when data in the reference stream permits;
- RTCM version 3 DGPS + RTK RTCM 2.x messages 1004, 1006, 1007 and 1012 are generated when data in the reference stream permits;

Using the API Function `SSNSBFStream_insertReferenceStream` directly the exact RTCM message type may be specified. Refer to section 12 where all RTCM messages are listed with the required input SBF blocks to ensure their creation.

## 4.3 The SNMP' Interface

The Septentrio NV/SA's receivers can be controlled via a binary interface similar to the well known Simple Network Management Protocol (SNMP) interface. Similarly the PPSDK will interpret and process SNMP' commands using the `SSNPPEngine_sendSnmpCommand` API function. Within one SNMP' message any number (from 1 till 255) of variable bindings in any order can be used as long as the total length of the SNMP' message does not exceed the maximum size of 2kByte.

For a completer description of the SNMP' interface refer to the SNMP' Technical Note contained in the manual folder contained in the 'doc' directory.

## 4.4 Management Information Base (MIB)

The MIB is a hierarchical, tree structured database and comprises a set of objects used to manage the Septentrio NV/SA's receivers and PPSDK. It defines the commands that the PPSDK will process such as get and set, as well as holding their parameters. It allows an SNMP' command message to be interpreted and executed provided that the message has a corresponding Object ID in the MIB. The PPSDK contains an associated MIB enabling SNMP' commands embedded in SBF files, or received via the API to be executed.

An extract from a MIB is illustrated in section 7.2.2.

## 5 Septentrio PPSDK PostNav Application

The PPSDK installation contains the PostNav application, to be found in the 'applications\postnav' subdirectory of the main installation folder. It is recommended to make a shortcut to the executable file and place it in a convenient folder or on the desktop if PostNav is to be used frequently. PostNav is a Graphical User Interface (GUI) Application that uses the PPSDK dll in order to post process SBF files using many of the facilities provided by the PPSDK. The main form contains the main options for post processing with fine tuning being provided by the menu options. You will notice that the interface is very similar to the RxControl interface when connected to a Septentrio receiver; as such, you will be able to configure the PPSDK in a very similar way as you would do it on a real time Septentrio receiver.

The following sections give an overview of the processing options provided by PostNav and relate these to the main functions in the PPSDK API. The User Interface is intuitive and PostNav is best learnt by using it.

### 5.1 PostNav Main Form

On starting PostNav the main form is opened as shown. Normally the form is presented with all fields empty.

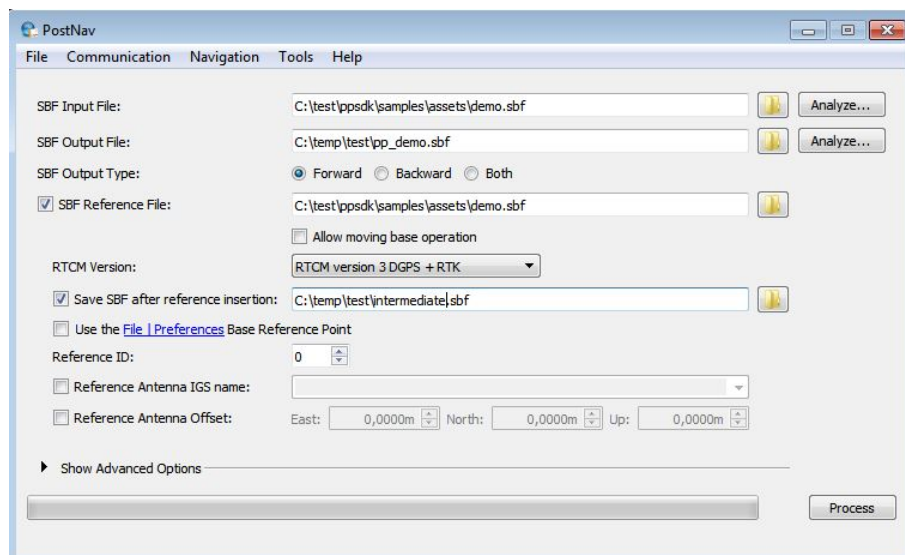


Figure 5-1: PostNav Main Form.

The only mandatory input fields are that for the input and output SBF files. The input file is the 'Rover' file to be processed. Pressing 'Process' will then produce the output file with all options set to their default values.

It is possible to choose either a Forward, a Backward or Both in the final solution. Forward will process the solution blocks one by one from start of time till end. Backward will however process the file starting from the last epoch found in the Rover file passed. Both will create 2 output files in the solution. The output file names will contain *.fw* or *.bw* in the solution appended to the original file name (before the *.sbf* extension).

Setting the checkbox SBF Reference File enables the other fields in the 'compact' view of the main form. The path and name of the reference file to be merged with the 'Rover' stream is entered and RTCM version to be produced in the merged stream selected. A merged SBF stream is a file that will contain the Raw Measurements from the Rover and the Differential Corrections inserted by the PPSDK Engine when using a Reference or Base file. If required, an integer Reference ID may be selected to provide a 'tag' in the output. It is possible to save the merged SBF stream to a file prior to putting it through the PPSDK Engine. Pressing 'Process' at this point opens and loads the 2 SBF streams and uses the API function `SSNSBFStream_insertReferenceStream` to merge the two input streams prior to invoking `SSNPPEngine_calculatePVT`.

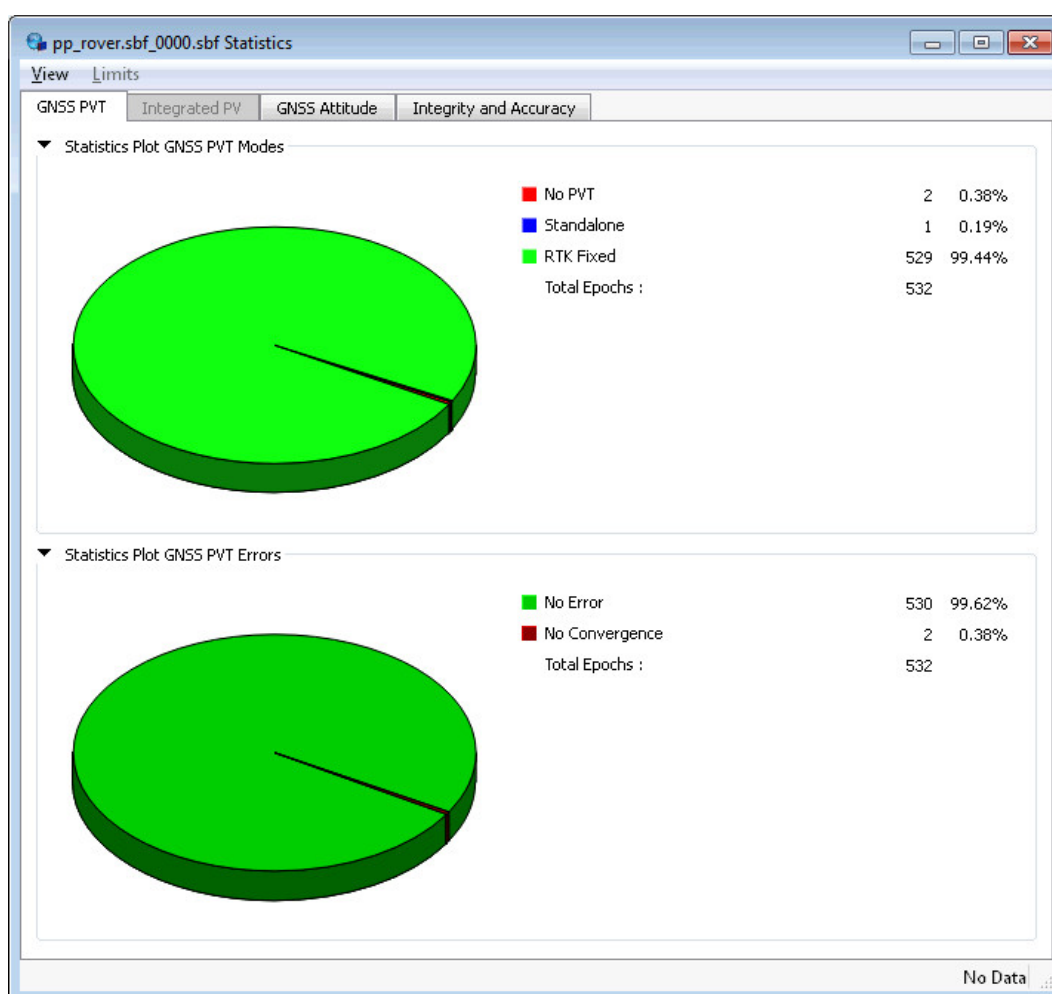
Clicking the Checkbox 'Use the File/Preferences Base Reference Point' will use the reference point entered via the menus 'File|Preferences|Reference' as the reference marker point. If not provided the marker position will be extracted from the Reference File.

It is also possible to specify the Reference antenna IGS name and offsets used on the Base Reference receiver. The antenna type string should be the exact string name of the antenna as used in NGS. Note that this field only applies for the Reference receiver antenna; the Rover antenna type can be specified via the menus 'Navigation/Engine Setup'. If not provided their value will be extracted from the Reference File. Note that if you check the 'Use the File/Preferences Base Reference Point' the offsets are no longer read from the Reference File, so you have to provide them manually.

Finally the 'Analyze...' buttons may be used to call the functions

**SSNSBFAnalyze\_getPVTModePercentages** and

**SSNSBFAnalyze\_getPVTErrorsPercentages** to display the form below.



**Figure 5-2: PostNav SBF Analysis.**

By clicking on the GNSS Attitude tab the statistics for Attitude computation can be displayed (if your license allows for this computation mode). The output file can also be fully analyzed by using SBF Analyzer provided within the RxTools package of Septentrio (downloadable from Septentrio NV/SA website).

## 5.2 PostNav Extended Main Form

Selecting 'Show Advanced Options' extends the main form allowing further control of the post processed solution.

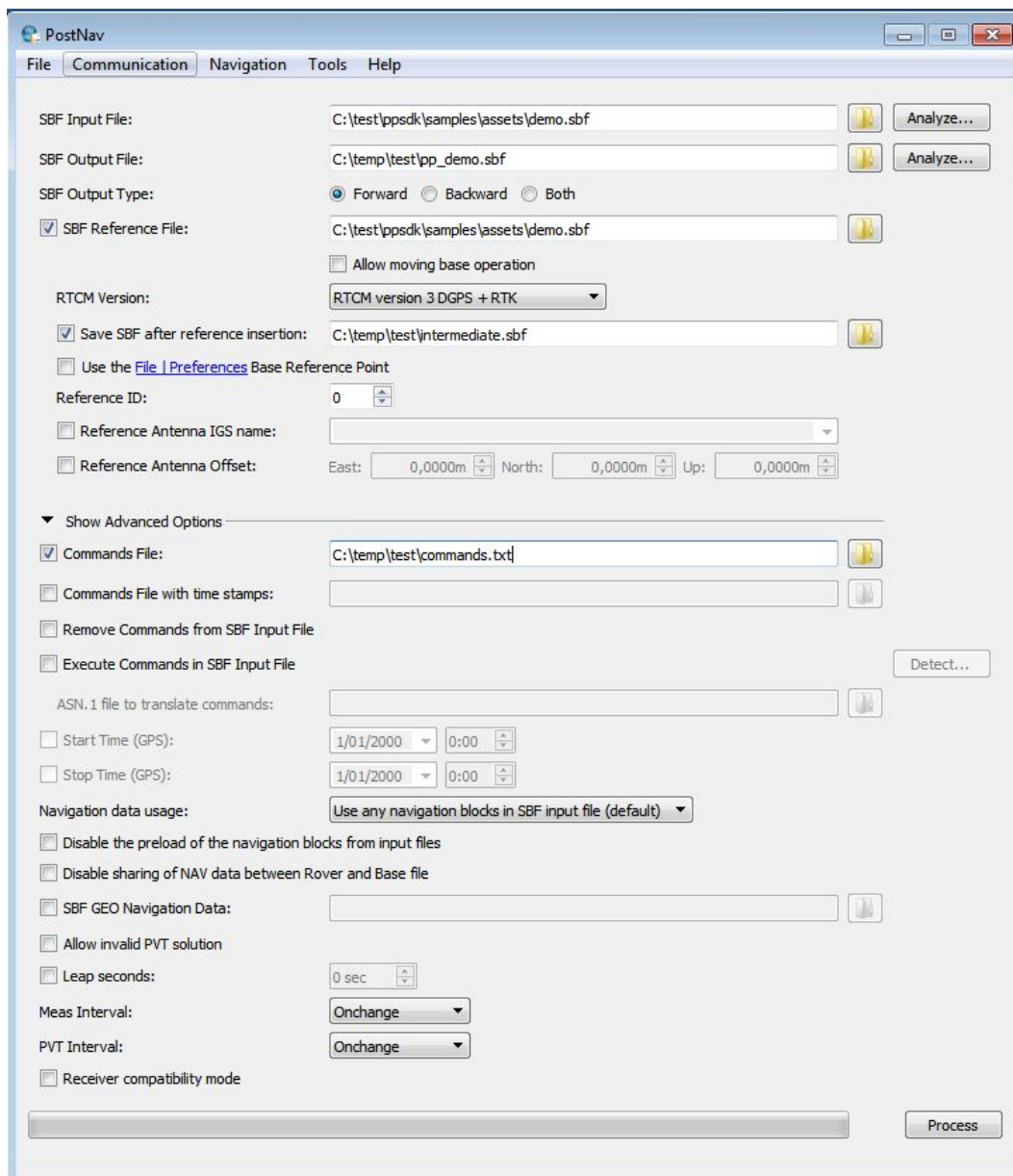


Figure 5-3: PostNav Extended Main Form

Checking 'Commands File' enables the name of a text file containing commands for the PPSDK Engine prior to calculating the solution. The function `SSNPPEngine_sendFileCommands` is used to process the commands contained in the file.

It is also possible to insert commands into the (merged) SBF input stream at any epoch using the 'Commands File with Time Stamps' checkbox.

This uses the API function `SSNSBFStream_insertFileCommandOverTime`.

Check 'Remove commands from SBF Input File' (using the **SSNSBFStream\_removeBlocks** function) to remove any command blocks from the **Rover** Stream.

Entering a start and stop time will crop (**SSNSBFStream\_cropGNSS**) the stream prior to processing by the PPSDK Engine. If two streams are to be merged the the cropping function will be applied to the resulting stream.

The user may determine which Navigation Blocks in the input file to use; Decoded, Raw or both types.

GEO Navigation data contained in an external SBF file may also be included to be used in the solution. Here the API function **SSNSBFStream\_merge** is used to add the GEO data to the previously combined streams.

If the stream to be used for the calculation contains commands it is possible to specify that they be executed and to link to an ASN.1 file to translate the commands if necessary.

It is possible to force the creation of an output file even if no solution could be calculated, this can happen when you do not have enough Measurement or Navigation data on your input SBF stream.

You can specify a window of post-processing by using the Start and Stop functions. This might be handy when working with large SBF files.

By default the PPSDK engine pre-loads the Navigation data from an SBF file before processing is done. While it is recommended to keep this option by default, in some occasions it might be handy to skip the pre-loading step for testing purposes.

By default the PPSDK engine will try to share the navigation data between the RoverSBF file and the RoverSBF when processing is done. This is quite handy specially when either the passed Rover or Base file lack proper Navigation data (it can be the case when the reference network does not provide Navigation data from RINEX downloadable files). While it is recommended to keep this option by default, in some occasions it might be handy skip the sharing of NAV data for testing purposes. Note that this option also takes some extra CPU.

## 5.3 PostNav File Menu Options

The file menu offers a number of items.

- Selection of general program preferences such as units, reference point and PostNav startup options;
- Display of a diagnostic report containing such things as PPSDK versions and permissions. See next figure as an example of some of the output.
- An option to save the PPSDK MIB description to a file (This can be used to create an ASN.1 file for use with the PPSDK and some tools such as sbf2cmd);
- An option to save the PostNav configuration to a file;
- An option to restore the PostNav configuration from a file;
- the ability to copy configurations (mainly to restore the Default configuration of the engine)



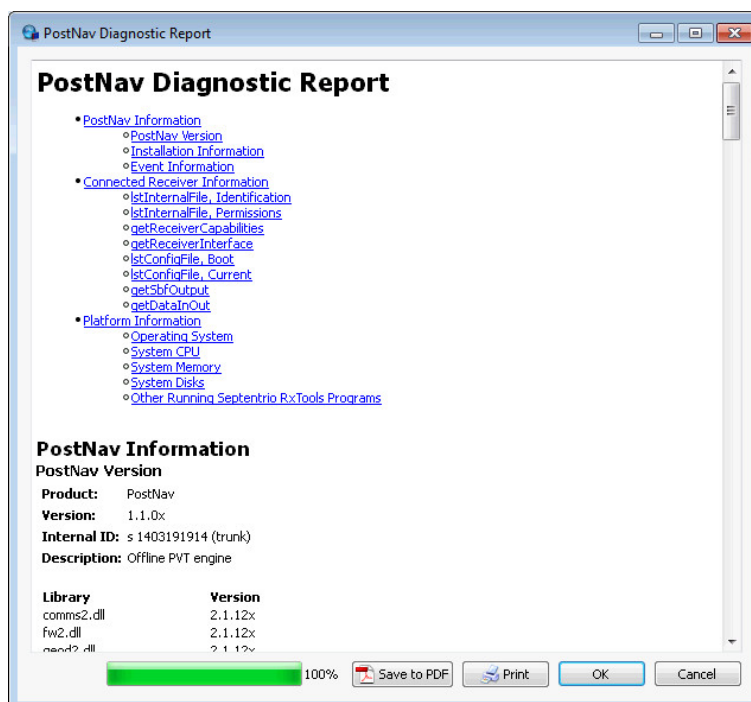


Figure 5-4: PostNav Diagnostics

## 5.4 PostNav Communication Menu Options

The 'Communication' menu offers three items.

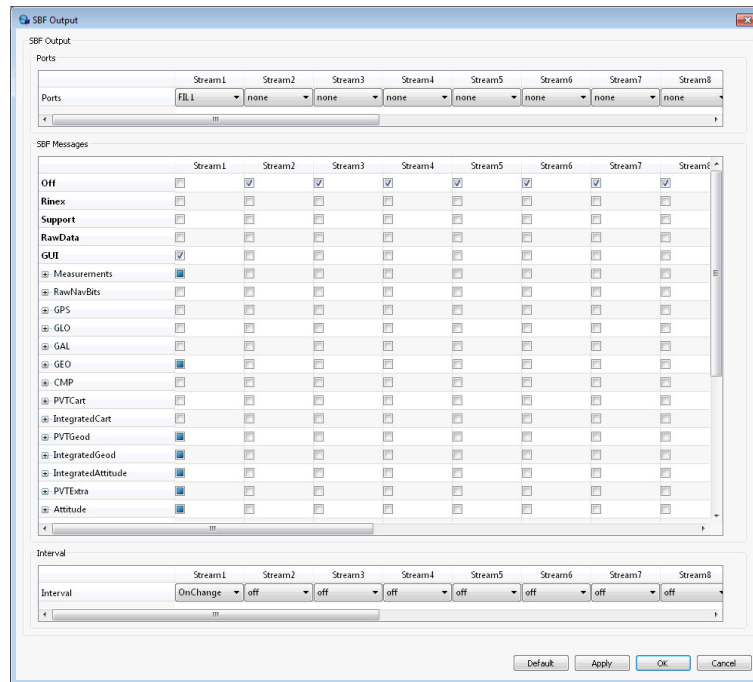
The input/output selection is currently reserved for future use.

The 'Output Settings|SBF Groups' allows you to define a set of pre-defined SBF blocks which can then be output in the SBF output file. However an alternative way to select the blocks is by directly choosing the SBF output by using the 'Output Settings|SBF Output' menu described below.

The 'Output Settings|SBF Output' provides an easy way to determine which blocks should be contained in the output file by using either predefined groups or individual settings.

This uses the `SSNPPEngine_sendAsciiCommand` function to set the output blocks.





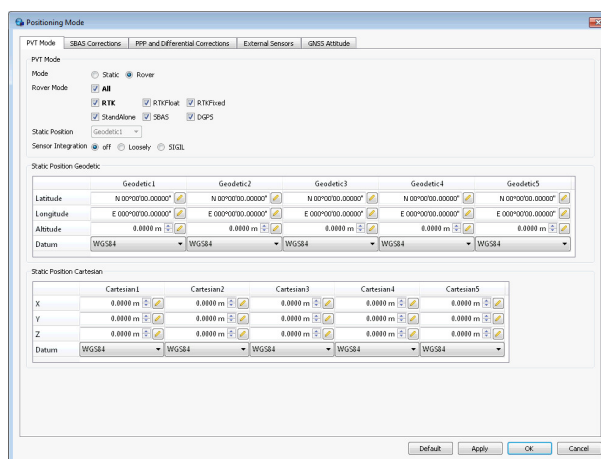
**Figure 5-5: PostNav SBF Output**

The 'Output Settings|SBF Output Once' does not have a direct use when working with PostNav, however it could be used in a file with commands over time (and therefore this menu option is available in the application). This would allow you to generate certain SBF output on specific moments of the post-processing.

'Input Settings|Differential Corrections' opens a form enabling the choice of which differential correction input format and messages in the merged or original file will be used in the computation of the solution. By default PostNav uses any DiffCorr format present in the Rover or intermediate file (if a merge is done). Note that you also have the option of selecting the RTCM format to be included in the intermediate or merged file (in the main screen) as such this menu option should properly match the selected RTCM format chosen in the main screen if changed or different from Default.

## 5.5 PostNav Navigation Menu Options

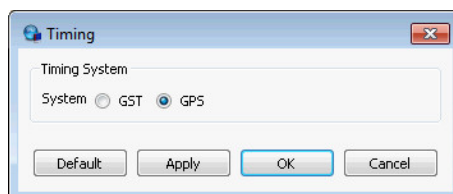
Selection of 'Navigation|Positioning Mode...' allows the main configuration of the GNSS engine computation such as the PVT mode to be chosen and or the SBAS and Differential Correction Usage to be set. If the PPSDK has a permitted option for INS then the system will show the different options for configuring the sensor in use. If the PPSDK has permitted option for Attitude then the system will show the different options for Attitude configuration.



**Figure 5-6: PostNav PVT Mode**

Using 'Engine Operation|Masks...' allows commands to be sent to the Post Processing Engine to set Elevation and Health Masks.

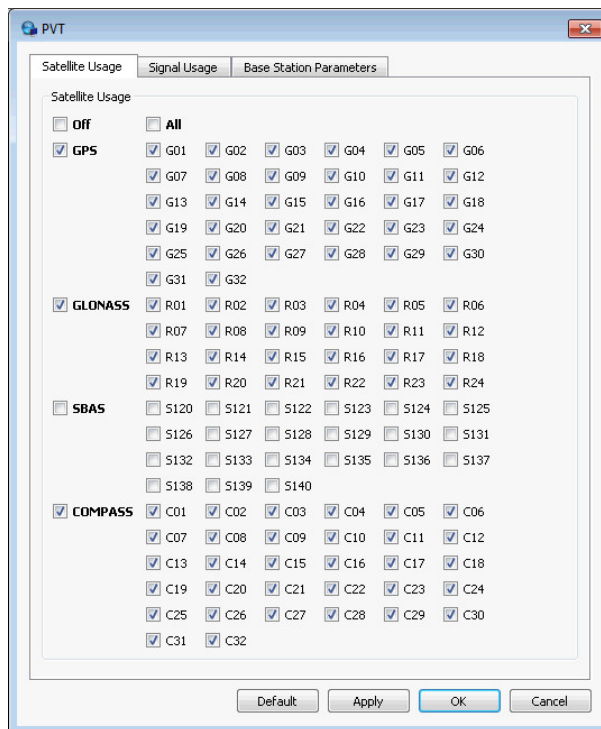
Using 'Engine Operation|Timing...' allows the timing system to be set to either GST or GPS (default).



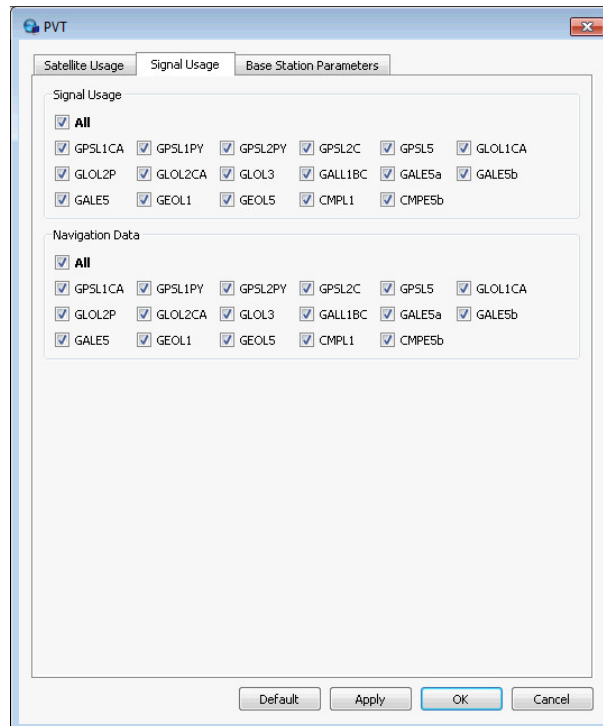
**Figure 5-7: PostNav Timing**

Using 'Advanced User Settings|Engine Setup...' allows to specify information about the Rover antenna used (or antennas when using GNSS Attitude). Note that the Base Antenna information is mainly set in the main screen of PostNav.

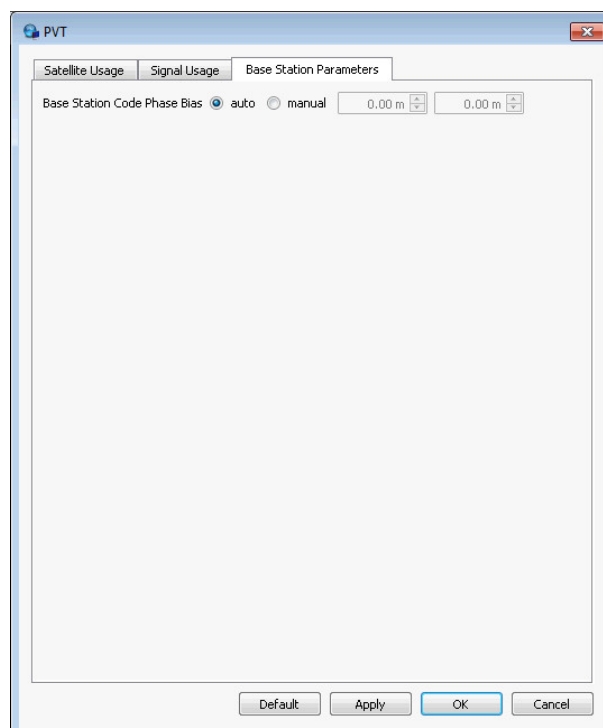
Using 'Advanced User Settings|PVT...' enables Satellite Usage and Signal Usage to be selected and transferred to the Post Processing Engine.



**Figure 5-8:** PostNav Satellite Usage



**Figure 5-9:** PostNav Signal Usage



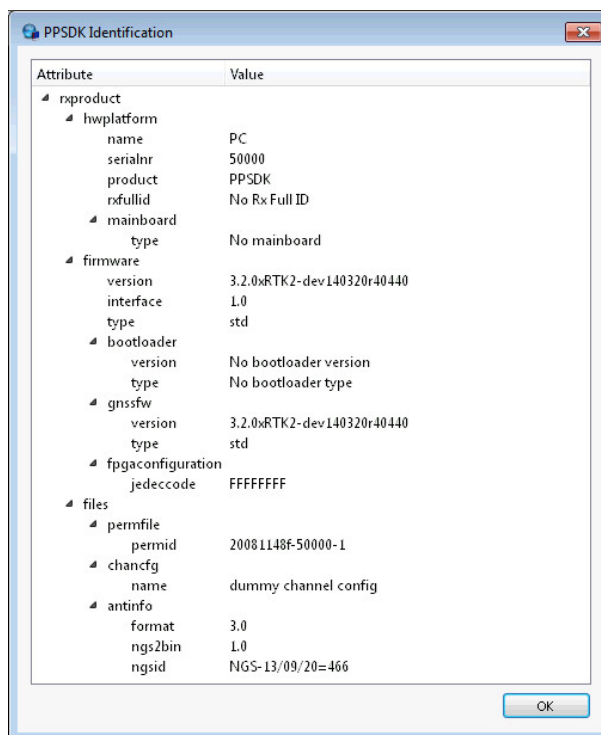
**Figure 5-10:** PostNav Base Station Parameters

## 5.6 PostNav Tools Menu Options

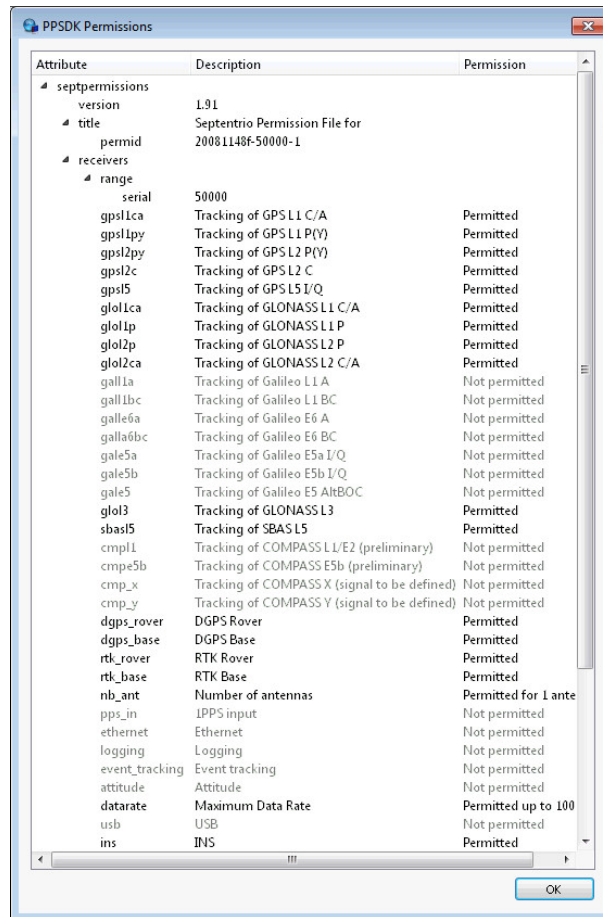
There is one option in this menu enabling RINEX files to be converted to SBF prior to processing. This option may be required if reference files can only be obtained in RINEX format. See Section 6 on page 31 for more details about RINEX Converter.

## 5.7 PostNav Help Menu Options

In the 'Help' menu the user may view the Identification, the PPSDK Permissions and the Permitted Capabilities as illustrated.

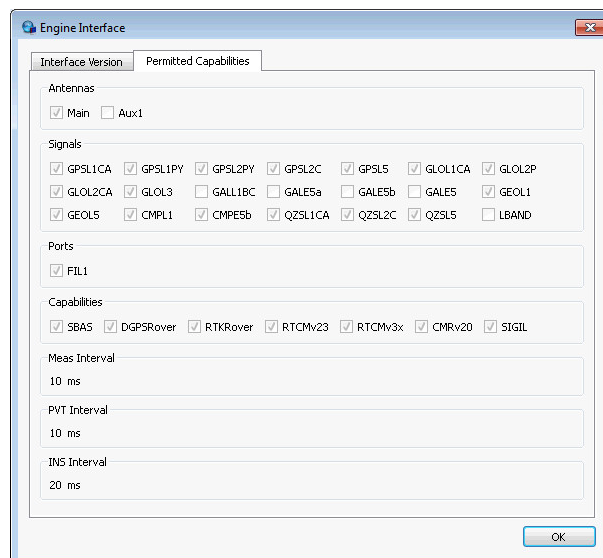


**Figure 5-11: PostNav Receiver ID**



Attribute	Description	Permission
septpermissions		
version	L91	
title	Septentrio Permission File for 20081148f-50000-1	
permid		
receivers		
range		
serial	50000	
gpsl1ca	Tracking of GPS L1 C/A	Permitted
gpsl1py	Tracking of GPS L1 P(Y)	Permitted
gpsl2py	Tracking of GPS L2 P(Y)	Permitted
gpsl2c	Tracking of GPS L2 C	Permitted
gpsl5	Tracking of GPS L5 I/Q	Permitted
glol1ca	Tracking of GLONASS L1 C/A	Permitted
glol1p	Tracking of GLONASS L1 P	Permitted
glol2p	Tracking of GLONASS L2 P	Permitted
glol2ca	Tracking of GLONASS L2 C/A	Permitted
gall1a	Tracking of Galileo L1 A	Not permitted
gall1bc	Tracking of Galileo L1 BC	Not permitted
galle6a	Tracking of Galileo E6 A	Not permitted
galle6bc	Tracking of Galileo E6 BC	Not permitted
gale5a	Tracking of Galileo E5a I/Q	Not permitted
gale5b	Tracking of Galileo E5b I/Q	Not permitted
gale5	Tracking of Galileo E5 AltBOC	Not permitted
glol3	Tracking of GLONASS L3	Permitted
sbas15	Tracking of SBAS L5	Permitted
cmpl1	Tracking of COMPASS L1/E2 (preliminary)	Not permitted
cmpe5b	Tracking of COMPASS E5b (preliminary)	Not permitted
cmp_x	Tracking of COMPASS X (signal to be defined)	Not permitted
cmp_y	Tracking of COMPASS Y (signal to be defined)	Not permitted
dgps_rover	DGPS Rover	Permitted
dgps_base	DGPS Base	Permitted
rtk_rover	RTK Rover	Permitted
rtk_base	RTK Base	Permitted
nb_ant	Number of antennas	Permitted for 1 ante
pps_in	PPS input	Not permitted
ethernet	Ethernet	Not permitted
logging	Logging	Not permitted
event_tracking	Event tracking	Not permitted
attitude	Attitude	Not permitted
datarate	Maximum Data Rate	Permitted up to 100
usb	USB	Not permitted
ins	INS	Permitted

Figure 5-12: PostNav Receiver Permissions



Interface Version	Permitted Capabilities
<b>Antennas</b> <input checked="" type="checkbox"/> Main <input type="checkbox"/> Aux1	
<b>Signals</b> <input checked="" type="checkbox"/> GPSL1CA <input checked="" type="checkbox"/> GPSL1PY <input checked="" type="checkbox"/> GPSL2PY <input checked="" type="checkbox"/> GPSL2C <input checked="" type="checkbox"/> GPSL5 <input checked="" type="checkbox"/> GLOL1CA <input checked="" type="checkbox"/> GLOL2P <input checked="" type="checkbox"/> GLOL2CA <input checked="" type="checkbox"/> GLOL3 <input type="checkbox"/> GALL1BC <input type="checkbox"/> GALE5a <input type="checkbox"/> GALE5b <input type="checkbox"/> GALE5 <input checked="" type="checkbox"/> GEOL1 <input checked="" type="checkbox"/> GEOL5 <input checked="" type="checkbox"/> CMPL1 <input checked="" type="checkbox"/> CMPE5b <input checked="" type="checkbox"/> QZSL1CA <input checked="" type="checkbox"/> QZSL2C <input checked="" type="checkbox"/> QZSL5 <input type="checkbox"/> LBAND	
<b>Ports</b> <input checked="" type="checkbox"/> FIL1	
<b>Capabilities</b> <input checked="" type="checkbox"/> SBAS <input checked="" type="checkbox"/> DGPSRover <input checked="" type="checkbox"/> RTKRover <input checked="" type="checkbox"/> RTCMv23 <input checked="" type="checkbox"/> RTCMv3x <input checked="" type="checkbox"/> CMRv20 <input checked="" type="checkbox"/> SIGIL	
<b>Meas Interval</b> 10 ms	
<b>PVT Interval</b> 10 ms	
<b>INS Interval</b> 20 ms	

Figure 5-13: PostNav Engine Interface, Permitted Capabilities

## 6 Septentrio PPSDK RINEX Converter Application

The PPSDK installation contains the RINEX Converter application, to be found in the 'applications\postnav' sub-directory of the main installation folder. It is recommended to make a shortcut to the executable file and place it in a convenient folder or on the desktop if RINEX Converter is to be used frequently.

RINEX Converter is a Graphical User Interface (GUI) Application that uses the PPSDK dll to convert RINEX files to SBF.

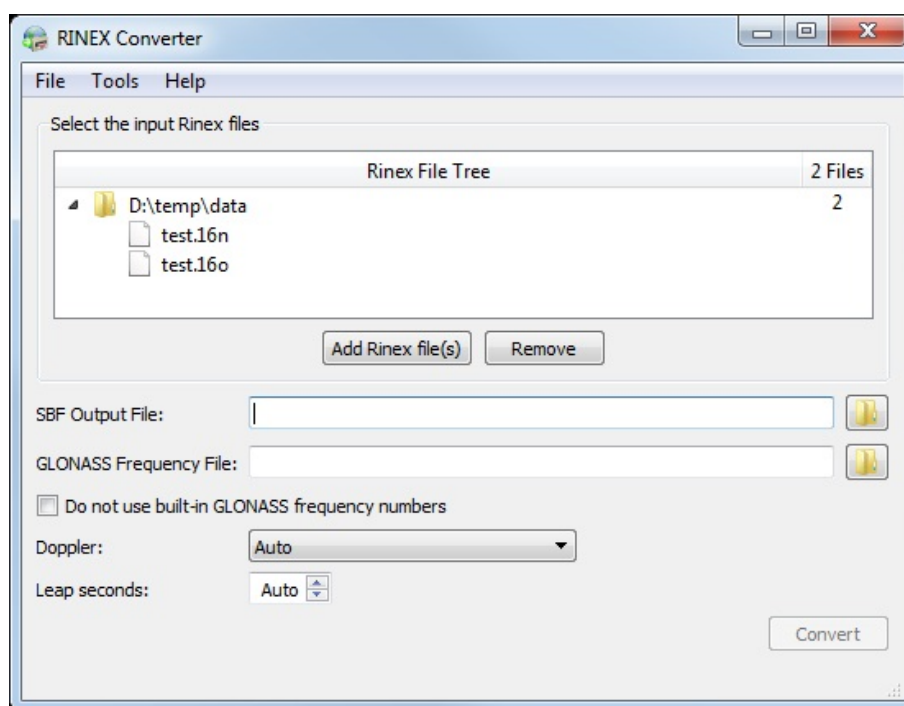


Figure 6-1: RINEX Converter

### 6.1 RINEX Converter Main Form

- The top part of the window (Select the input RINEX files) allows to create a list of the RINEX files to be converted to SBF. Clicking the Add RINEX File(s) button adds the given RINEX files to the ones to be processed. Removing a file from the ones to be processed is done by selecting the file in the top part and clicking the Remove button.
- In the SBF Output File field the SBF output file to which the selected RINEX file(s) must be converted must be specified.
- Optionally the GLONASS Frequency File can be set by specifying the file in the corresponding field. For more details about this file see Section 11.2.1 on page 71.
- In order to avoid using the backup GLONASS Frequency numbers check the Do not use built-in GLONASS Frequency numbers checkbox.
- The Doppler computation can be set to:
  - Auto: The Doppler is taken from the Dx observable in RINEX if available, or is computed from carrier phase otherwise. This is the default value.
  - Compute Doppler from carrier phase: The Doppler is always computed from differencing carrier phase measurements (Lx). The Dx observable in Rinex is discarded.
  - Use Doppler from Rinex: The Doppler is taken from the Dx observable in Rinex, and is set to NOT-VALID if there is no Dx observable.

- GLONASS (unlike GPS) system time is connected to UTC. Therefore GLONASS receivers need the current leap second value in order to function correctly. The `Leap Seconds` spinbox allows to specify this value. If set to `Auto` the best leap second value known by the RINEX Converter at the time when the PPSDK dll was built will be used.

## 6.2 RINEX Converter File Menu Options

The file menu offers a number of items.

- Selection of general program preferences such as the RINEX Converter start-up options;
- Display of a diagnostic report containing such things as RINEX Converter version;
- The ability to exit the application

## 6.3 RINEX Converter Tools Menu Options

This menu contains entries to open the other GUI tools of the package.

## 6.4 RINEX Converter Help Menu Options

The Help menu contains the following items:

- Event Viewer: allows to see events fired by the program;
- A link to the Septentrio NV/SA website;
- A link to the Septentrio NV/SA support website;
- The about box of RINEX Converter

# 7 How To...

This chapter contains step-by-step instructions to help you with typical tasks or information to help to understand the PPSDK. It does not provide a complete overview of the PPSDK's operations, but rather an introduction to different operation modes.

The following documents found in the doc manual subdirectory provide fuller descriptions of items discussed below.

- PPSDK Command Line Interface Reference Guide.
- PPSDK v4.1.8 Command And Log Reference Card.
- SBF Reference Guide.
- SNMP' Technical Note.

Depending on the terms of your particular license (see section 3), some of the features described here may not be supported.



## 7.1 Manage SBF Streams

Operations on SBF Streams are at the heart of the PPSDK. SBF files provide the input to the PPSDK Engine and the Engine will output an SBF File. SBF files contain much information and large SBF files can take time to process. If you are certain of the blocks required in the SBF Stream, only record those blocks required. If a large SBF file is to be used many times please consider using the provided tools such as `sbf2sbf` (section 11.8) to preprocess it.

On the input side the PPSDK provides a rich set of functions to process or condition SBF files for the PVT Calculation:



- Open, close, copy, validate and write an SBF Stream to a file.
- Find specific blocks by time or Block ID.
- Crop a stream between two timestamps.
- Filter an SBF Stream with only the more relevant SBF blocks.
- Find common epochs in two streams.
- Find missing epochs in a stream.
- Set a reference position into a stream or read the reference position.
- Insert commands into a stream.
- Merge two streams. For example base station reference into a rover stream.

On the output side the PPSDK Engine writes an SBF stream to a file. The blocks contained in the file are determined by commands sent to the PPSDK Engine prior to the PVT Calculation being invoked.

By default the PPSDK Engine outputs SBF blocks defined by the predefined group 'GUI' at the rate they change. This and other predefined groups are tabulated below and are used by for example the setSBFOutput command defined in the Command Line Interface Reference Guide.

Group	SBF Blocks
GUI	MeasEpoch, EndOfMeas, AuxAntPositions, AttEuler, GEOIGPMask, GEOIonoDelay, PVTGeodetic, PosCovGeodetic, VelCovGeodetic, DOP, PVTSatCartesian, PVTResiduals, RAIMStatistics, BaseLine, EndOfPVT, IntPVGeod, IntAttEuler, ReceiverTime, ExtEvent, DiffCorrIn, BaseStation, SatVisibility, ChannelStatus, ReceiverStatus, ReceiverSetup, Commands, Comment, ExtSensorMeas, IPStatus, QualityInd
Rinex	MeasEpoch, GEORawL1, GPSNav, GPSIon, GPSUtc, GLONav, GALNav, GALUtc, GALGstGps, GEONav, PVTGeodetic, ReceiverSetup, Comment, CMPNav
Support	MeasEpoch, MeasExtra, EndOfMeas, GPSRawCA, GPSRawL2C, GPSRawL5, GLORawCA, GALRawFNAV, GALRawINAV, GEORawL1, GEORawL5, GPSNav, GPSAlm, GPSIon, GPSUtc, GLONav, GLOAlm, GLOTime, AuxAntPositions, GALNav, GALAlm, GALIon, GALUtc, GALGstGps, AttEuler, GEONav, GEOAlm, BaseVectorGeod, PVTGeodetic, DOP, EndOfPVT, ExtEvent, DiffCorrIn, BaseStation, ChannelStatus, ReceiverStatus, ReceiverSetup, Commands, CMPRaw, IPStatus, QZSRawL1CA, QZSRawL2C, QZSRawL5, PVTSupport, IntSupport, CMPNav, QualityInd
RawData	MeasEpoch, MeasExtra, GPSRawCA, GPSRawL2C, GPSRawL5, GLORawCA, GALRawFNAV, GALRawINAV, GEORawL1, GEORawL5, GPSNav, GLONav, GALNav, GEONav, PVTGeodetic, DiffCorrIn, ReceiverSetup, Commands, Comment, CMPRaw, QZSRawL1CA, QZSRawL2C, QZSRawL5, CMPNav

**Table 7-1:** Predefined SBF Output Groups

Please refer to the delivered documentation for the definition of the various Groups.

## 7.1.1 SBF Input Requirements

Some blocks that serve as the main input of the PPSDK Engine are mandatory for the post-processing to work, the minimum requirement depends on the PVT mode or the feature to be used from the PPSDK Engine. It is recommended that when recording SBF data from a Septentrio receiver you use the `RawData` group for proper post-processing.

The following blocks are the minimum requirement for the PPSDK Engine in Standalone/SBAS mode:

- Measurement Blocks (`MeasEpoch` is mandatory, `MeasExtra` is optional but can improve your post-processing performance).
- Navigation Pages (`GPSRawCA`, `GPSRawL2C`, `GPSRawL5`, `GLORawCA`, `GALRawFNAV`, `GALRawINAV`, `GEORawL1`, `GEORawL5`, `CMPRaw`, `QZSRawL1CA`, `QZSRawL2C` and `QZSRawL5`).
- GeoCorrections for SBAS computation (`GEORawL1`)
- GPS Decoded Messages (optional to the Raw data).
- GLONASS Decoded Messages (optional to the Raw data). Galileo Decoded Messages (optional to the Raw data).
- SBAS Decoded Messages (optional to the Raw data).

Please refer to the delivered SBF documentation for the definition of the various blocks.

The following blocks are the minimum requirement for the PPSDK Engine to be able to post-process in DGPS or RTK mode:

- The same blocks above as for Standalone/SBAS mode.
- Differential Correction Messages (`DiffCorrIn`).

Note that if your Rover input file does not contain `DiffCorrIn` blocks then you can generate these blocks by using the merge or insertion of reference functionality of the PPSDK Engine (as such you can use RINEX files from any reference Station to generate `DiffCorrIn` blocks which can be used for Differential post-processing).

The following blocks are the minimum requirement for the PPSDK Engine to be able to post-process in PPP mode:

- The same blocks above as for Standalone/SBAS mode.
- Differential Correction Messages containing an RTCMV stream (`DiffCorrIn`).

The following blocks are the minimum requirement for the PPSDK Engine to be able to post-process in Moving Base mode:

- The same blocks above as for Standalone/SBAS mode.
- Differential Correction Messages (`DiffCorrIn`) coming from the moving base (or inserted from a Moving base file).

The following blocks are the minimum requirement for the PPSDK Engine to be able to post-process in INS mode:

- The same blocks above as for Standalone/SBAS mode.
- External Sensor measurements contained in the SBF block (`ExtSensorMeas`).
- External Sensor status and setup contained in the SBF blocks (`ExtSensorStatus` and `ExtSensorSetup`).

The following blocks are the minimum requirement for the PPSDK Engine to be able to post-process an Attitude solution:

- The same blocks above as for Standalone/SBAS mode.
- Note that the Measurement blocks must contain data of multiple antennas (`MeasEpoch`).

The following blocks are the minimum requirement for the PPSDK Engine to be able to post-process External Events:

- The same blocks above as for Standalone/SBAS mode.
- External Events from the original receiver (`ExtEvent`).

## 7.1.2 SBF Output

Some blocks that serve as the main input of the PPSDK Engine will not be output if they are not present in the input stream.

The following blocks are output only if a proper input file contains the corresponding SBF Input :

- Measurements Blocks.
- Navigation Pages.
- GEO Corrections.
- Differential Corrections Blocks (unless generated from a Base insertion process).
- External Event Blocks.
- Other Miscellaneous Blocks.

If `DiffCorrIn` blocks are specified for output these will be

- decoded from the input stream and encoded when available
- created by the PPSDK Engine when a Reference stream is merged with the input stream. In this case an option exists to remove `DiffCorrIn` blocks from the rover stream in the `SSNSBFStream_insertReferenceStream` API function.

The following, shows how to configure the Engine to output the `MeasEpoch` and `PVTCartesian` SBF blocks at 10 Hz and the `GPSNav` SBF block at its natural "OnChange" rate, i.e. when new GPS navigation data is available in the SBF File. These three blocks must be output to the FIL1 'connection' used to create the temporary output file for eventual writing to the disk.

1. Scheduling SBF blocks for output is done by defining so-called "SBF streams". Up to 10 SBF streams can be defined by the user. A stream consists of a set of SBF blocks that need to be output at a given rate to the output stream. Defining these SBF streams involves the `setSBFOutput` command (with PostNav you can do this via the 'Communication/Output Settings' menu):  

```
setSBFOutput, Stream1, FIL1, MeasEpoch+PVTCartesian, msec100 <CR>
setSBFOutput, Stream2, FIL1, GPSNav, OnChange <CR>
```
2. Closing the PVT Engine Handle will reset the output back to default when the PVT Engine is opened again.

## 7.2 Control the processing of Data

In the same way that the Septentrio NV/SA's receivers are controlled by sending ascii commands or SNMP data to them, the PPSDK will process these commands to enable fine control of the computation and input/output parameters.

The commands used by the PPSDK engine are detailed in the commands reference document - 'PPSDK Command Line Interface Reference Guide' provided with the PPSDK. This also documents the default values of each command, defining the input/output status of the PPSDK when no commands are entered.

### 7.2.1 Using ASCII Commands

If you use `pvtcalc` or `PostNav` then you can send commands by creating an ASCII file with these commands. Both applications allow you to specify the commands to be used. These applications use PPSDK engine in order to pass the necessary commands for post-processing. If you are programming an application on top of the PPSDK engine then there are two ways to use commands with the PPSDK:

- Insert the commands into the input sbf streams at any point using the **SSNSBFStream\_insertCommandOverTime** or **SSNSBFStream\_insertFileCommandOverTime** to insert the commands at any desired epoch in an SBF file (specified by using the GNSS time). They will then be executed at the time they appear in the stream.
- Use **SSNPPEngine\_sendAsciiCommand** or **SSNPPEngine\_sendFileCommands** to execute commands to be applied to the entire PVT Calculation.

Note that the PPENGINE option **SSNPPENGINE\_OPTIONS\_EXECMDS** must be set in order to execute commands in an SBF Stream. If the original SBF files contains unwanted command blocks they should first be removed by using one of the provided tools, or use **SSNSBFStream\_removeBlocks** function to filter the Command Blocks.

When sending ASCII commands to the PPSDK Engine either directly or as part of a 'command file' the results of processing the commands can be returned by the PPSDK as shown in the following example (error handling not shown).

```
/* Send ASCII commands used when computing a PVT */
error = SSNPPEngine_sendAsciiCommand(
    penginehandle,
    "spm, Rover, StandAlone+SBAS+DGPS+RTK",
    &size,
    NULL);

/* Allocated memory */
replystring = (char*)malloc(size);
memset(replystring, 0, size);

error = SSNPPEngine_sendAsciiCommand(
    penginehandle,
    "spm, Rover, StandAlone+SBAS+DGPS+RTK",
    &size,
    replystring);

/* Free allocated memory */
free(replystring);
replystring = NULL;
```

The 'double call' mechanism illustrated above works as follows.

The first call to **SSNPPEngine\_sendAsciiCommand** has the final parameter set to NULL. This causes the function to calculate the size of the buffer required to hold the response and return it in the 'size' parameter. Memory can then be allocated correctly for the reply string which is then populated by the second call to **SSNPPEngine\_sendAsciiCommand** using the `replystring` as the final parameter. This way any errors executing the commands can be returned to the user.

## 7.2.2 Using the SNMP' Interface

Just like the Septentrio NV/SA's receivers, the PPSDK engine can also be controlled by sending Septentrio NV/SA SNMP' data over a binary interface. This protocol is loosely based on Simple Network Management Protocol (SNMP). Each receiver and the PPSDK contain a Management Information Base (MIB) which has a one to one relationship with the ASCII command interface for all exe, get and set commands.

The PPSDK is capable of listing it's formal MIB description using the API function **SSNPPEngine\_getMIBDescription** an extract from which is provided below.

```
MIB Description:
GROUP: ioSelection
    sdio, setDataInOut
    gdio, getDataInOut
GROUP: ioOutput
```

```

SUB: subSBFGroups
    ssgp, setSBFGroups
    gsgp, getSBFGroups
SUB: subSBFOut
    sso , setSBFOutput
    gso , getSBFOutput
SUB: subSBFOutOnce
    esoc, exeSBFOnce
    gsoc, getSBFOnce
GROUP: ioInput
SUB: subDiffCorrIn
TAB: tabDiffCorrInRTCM2
    sr2u, setRTCMv2Usage
    gr2u, getRTCMv2Usage
    sr2c, setRTCMv2Compatibility
    gr2c, getRTCMv2Compatibility
    sus , setUltraStation
    gus , getUltraStation
TAB: tabDiffCorrInRTCM3
    sr3u, setRTCMv3Usage
    gr3u, getRTCMv3Usage
TAB: tabDiffCorrInCMR2
    sc2u, setCMRv2Usage
    gc2u, getCMRv2Usage
GROUP: navPosMode
TAB: tabPVTMode
    spm , setPVTMode
    gpm , getPVTMode
TAB: tabSBAS
    ssbc, setSBASCorrections
    gsbc, getSBASCorrections
    ...
    ...
GROUP: fileConfiguration
    eccf, exeCopyConfigFile
    gccf, getCopyConfigFile

```

A description of this interface is provided in the document - SNMP' Technical Note.pdf.

With regard to the PPSDK the API function **SSNPPEngine\_sendSnmpCommand** allows multiple commands and requests to be sent to the PPSDK Engine prior to calculating a solution.

## 7.2.3 Using pvtcalc or PostNav

Both the sample application pvtcalc (section 11.1) and the application PostNav (section 5) allow the user to define a text file containing lines of commands to be applied to a calculation. pvtcalc will also take a command file with time stamps so that the commands will be inserted into the SBF stream prior to the calculation being executed.

## 7.3 Check the Capabilities of your PPSDK

The capabilities of the PPSDK are defined by the contents of the license file. The capabilities depend on the current firmware version and the current set of permissions. Permissions are further explained in section 7.14.

For instance, to have the capability to use the GPS L2C signal, the following conditions must be met: the PPSDK version must support L2C, the GPS Constellation and the L2C signal must be enabled in the permission file and the input SBF file must contain the signal.

The command **getReceiverCapabilities(grc)** lists the capabilities of the PPSDK, taking into account the permissions. With PostNav this can be seen by using the 'Help/Engine Interface' menu.

## 7.4 Handle structures returned by the PPSDK API

A number of API calls in the PPSDK return data to the caller. In most cases the size of the data is known in advance, such as with particular SBF blocks. Here the user declares a variable of the correct type provided in the PPSDK header files and passes a pointer to the API call. In some instances the size of the returned data cannot be predicted. Here, the API function has to be called twice.

The first call is made with the returned structure set to NULL. The function returns the size of the buffer required to hold the data. A buffer can then be allocated and the API called for a second time with a pointer to the buffer. An example of this 'double call mechanism' is given in section 7.2.1.

## 7.5 Return progress indication to the user

Processing of large SBF files and the use of several functions such as file merge or insert reference stream can take a while to process. The PPSDK provides facilities for returning the progress of the computation to the user as a percentage complete indicator. Four basic API functions are provided to implement this and are duplicated in some of the API function groups such as the RINEX Decoder and Post Processing Engine Modules.

In short the programmer creates a function to handle the percentage completion of a particular PPSDK API module. This is then registered. The the API functions to be associated with the PPSDK module handle (SBF stream handle for example) are subscribed to the PPSDK. Now whenever an PPSDK API function is called that has been subscribed, the callback function is invoked as necessary. The callback function is given an indication of the function calling it and the percentage completion of that function.

The SBF Stream module calls are discussed below.

### 7.5.1 SSNSBFStream\_pcSetCallback

This function takes two parameters:

- The stream handle of the SBF stream being processed.
- The name of the callback function provided by the programmer.

This function registers the callback function which will be called automatically and periodically by the SBF Stream Modules as they are executed.

### 7.5.2 SSNSBFStream\_pcSetUserDataCallback

This function is equivalent to the previous one, except that it takes three parameters:

- The stream handle of the SBF stream being processed.
- The name of the callback function provided by the programmer.
- A void pointer to userdata that then is returned when the callback function is called.

This function registers the callback function which will be called automatically and periodically by the SBF Stream Modules as they are executed.

### 7.5.3 SSNSBFStream\_pcSubscribe

This function takes two parameters:

- The stream handle of the SBF stream being processed.
- A list of the SBF module functions that wil be invoke as a bitmap.

This function registers the API functions used to process the stream handle.

## 7.5.4 SSNSBFStream\_pcUnsubscribe

This function takes two parameters:

- The stream handle of the SBF stream being processed.
- The Identification of the function to be removed from the subscription list.

This function registers the API functions used to process the stream handle.

## 7.5.5 SSNSBFStream\_pclSubscribed

This function takes three parameters:

- The stream handle of the SBF stream being processed.
- The Identification (name) of the callback function.
- A boolean is returned indicating if the given function is subscribed.

This function checks whether a given function has been subscribed to report its progress using the previously selected progress callback function.

An example of a callback function and setup is given below.

```
static void
printProgressSbfStream(ssn_sbfstream_progresscb_flist_t fitem,
    float percentage)
{
    float current_percentage = 0.0f;
    float formatted_percentage = 0.0f;

    if (fitem == SSNSBFSTREAM_PROGRESSCB_FLIST_LOADFILE) {
        if (belongs == 2) {
            /* This item belongs to the insert reference stream! */
            current_percentage = 37.0f;
        } else {
            current_percentage = 98.0f;
        }
        formatted_percentage = current_percentage +
            ((percentage / 100.0f) * 2.0f);
    } else if (fitem == SSNSBFSTREAM_PROGRESSCB_FLIST_MERGE) {
        current_percentage = 41.0f;
        formatted_percentage = current_percentage +
            ((percentage / 100.0f) * 7.0f);
    } else if (fitem == SSNSBFSTREAM_PROGRESSCB_FLIST_COPYFILE) {
        current_percentage = 48.0f;
        formatted_percentage = current_percentage +
            ((percentage / 100.0f) * 2.5f);
    } else if (fitem == SSNSBFSTREAM_PROGRESSCB_FLIST_CROPGNSS) {
        current_percentage = 50.5f;
        formatted_percentage = current_percentage +
            ((percentage / 100.0f) * 3.5f);
    } else if (fitem ==
        SSNSBFSTREAM_PROGRESSCB_FLIST_TRANSLATECOMMANDS) {
        current_percentage = 54.0f;
        formatted_percentage = current_percentage +
            ((percentage / 100.0f) * 4.0f);
    }

    printProgress(formatted_percentage);
}
```

In order to register the above function the following call is made.

```
error = SSNSBFStream_pcSetCallback(sbfin,
    (ssn_sbfstream_progresscb_t)printProgressSbfStream);
if (SSNERROR_GETCODE(error) != SSNERROR_WARNING_OK) {
    printError(error); }
```

Finally the functions to be used need to be subscribed.

```
error = SSNSBFStream_pcSubscribe(
    sbfin,
    SSNSBFSTREAM_PROGRESSCB_FLIST_LOADFILE |
    SSNSBFSTREAM_PROGRESSCB_FLIST_COPYFILE |
    SSNSBFSTREAM_PROGRESSCB_FLIST_CROPGNSS |
    SSNSBFSTREAM_PROGRESSCB_FLIST_REMOVEBLOCKS |
    SSNSBFSTREAM_PROGRESSCB_FLIST_INSERTREFERENCESTREAM |
    SSNSBFSTREAM_PROGRESSCB_FLIST_WRITEFILE |
    SSNSBFSTREAM_PROGRESSCB_FLIST_MERGE |
    SSNSBFSTREAM_PROGRESSCB_FLIST_TRANSLATECOMMANDS);
if (SSNERROR_GETCODE(error) != SSNERROR_WARNING_OK) {
    printError(error); }
```

## 7.6 Using Satellite Constellations

Depending upon the permissions associated with your license the PPSDK Engine is capable of using GPS, GLONASS and Galileo constellations in order to produce the most accurate solution. Furthermore, individual satellites and signals may be included or excluded from the derivation of a solution.

Use the **SetSatelliteUsage** command to define which satellites are allowed to be included in the PVT computation. For example:

- To only use GPS measurements in the PVT computation, use **ssu, GPS <CR>**.
- To add the usage of SBAS measurements in the PVT, use **ssu, +SBAS <CR>**. Or use **ssu GPS+SBAS <CR>** originally.
- To remove the measurement of one satellite from the PVT, use **ssu, -S120 <CR>**.

Note that this command only affects the usage of range and doppler measurements within the PVT computation. Navigation data transmitted by the satellite such as the SBAS corrections are always used if applicable.

Use the **SetSignalUsage** ASCII command to define which signals are allowed to be included in the PVT computation. For example to force the receiver to only use the L1 GPS C/A measurements in the PVT solution, use 'snu, GPSL1CA'. If you are using PostNav this setting can be found in the 'Navigation|Advanced User Settings' menu option.

## 7.7 Merge Files

Prior to processing an SBF file using the PPSDK an API function is provided allowing contents of two SBF files to be merged into a single SBF stream. For each stream the blocks to be included may be determined using the SBF stream merge options. Please refer to the API documentation on the Doc subdirectory of the installation folder. With PostNav and with pvtcalc you can also merge files by specifying a Reference SBF file. Internally these applications will call the necessary API functions for merging. If you would like to obtain an intermediate merged file (before post-processing) then you can use the following:

- In PostNav use the option Save SBF after reference insertion.
- In pvtcalc use the cmd line option '-p' which will also create an intermediate merged file.



## 7.8 Obtain an Standalone PVT Solution

By default the PPSDK Engine will compute an RTKFixed solution when all inputs required are available. If no reference stream is inserted by using an SBF file (original or converted from RINEX) the Engine will then fall back to an SBAS or a StandAlone aided solution as all modes are enabled by default.

In order to force the computation of a standalone solution the SBAS mode should be disabled using the command: **setPVTMode, Rover, StandAlone <CR>**

The following SBF Blocks must be present in the input stream to be able to compute a solution:

- MeasEpoch containing the appropriate signals of the permitted constellations i.e GPS L1,L2 or L1 and L2.
- GPSNav or GLONav.

To set the PVT Mode via PostNav please use the 'Navigation|Positioning Mode' menu option. With pvtcalc, you will need to create a command ASCII file which can then be passed to the application via the '-c' command line option.

## 7.9 Obtain an SBAS PVT Solution

The PPSDK is by default configured to make optimal use of the wide-area corrections sent by these satellites. In case the Engine is not in its default configuration, you can reconfigure it as follows:

1. If you want to use the SBAS corrections to improve the PVT accuracy, you need to configure the PVT in SBAS mode. For instance, the following command instructs the engine to compute a PVT using the SBAS corrections when available, and to fall back to the standalone mode otherwise:  
**setPVTMode, Rover, StandAlone+SBAS <CR>**
2. Make sure that the troposphere model is as prescribed by the RTCA DO 229 standard<sup>1</sup>. This is the default setting, but in case the engine is not in its default configuration, you should use:  
**setTroposphereModel, MOPS, MOPS <CR>**
3. It is recommended to leave the ionospheric model selection to `auto`. In particular, using the Klobuchar model in SBAS mode will lead to degraded performance and is not recommended.  
**setIonosphereModel, auto <CR>**
4. By default, the engine selects the SBAS satellite with the most SBAS corrections available. It is possible to force the PPSDK Engine to select which SBAS satellite should provide the corrections to the PVT (and override the automatic selection by the receiver), and how to deal with subtleties of the SBAS navigation message. This is done by the **setSBASCorrections** command. For instance to only accept corrections from EGNOS PRN126, use:  
**setSBASCorrections, S126 <CR>**
5. Optionally, it is possible to include the range to SBAS satellites as an additional ranging source for the PVT. This is not done by default as the SBAS ephemeris accuracy is poor (100 m error). However to do so (in very specific cases), use:  
**setSatelliteUsage, +SBAS <CR>**  
To revert to the default setting where SBAS ranging is excluded from the PVT, use:  
**setSatelliteUsage, -SBAS <CR>**

To compute a fully SBAS-aided position, the Engine has to receive and decode the following information:

- Long term corrections (corrections to the satellite orbit and clock as specified in the GPS ephemerides) `GeoLongTermCorr`;
- Fast corrections (short term satellite clock error) `GeoFastCorr`;
- Vertical ionospheric delays over the SBAS ionosphere grid surrounding the receiver position, `GeoIonoDelay`.

In order to utilize the SBAS signals the PPSDK requires the following blocks to be included in the input SBF stream:

- GeoRaw and or GEORawL1 if option `SSNPENGINE_OPTIONS_USENAVRW` is selected;

<sup>1</sup> Minimum Operational Performance Standards for Global Positioning/Wide Area Augmentation System Airborne Equipment RTCA/DO-229C, November 28, 2001

Due to the structure and order of the SBAS messages it can take up to 2.5 minutes before the long-term and fast corrections are available in the SBF input stream and up to 5 minutes before the ionospheric grid is available. Hence it is normal that the Engine cannot yield an SBAS-aided position immediately after the lock on an SBAS satellite.

For more details on SBAS positioning refer to section 8.4.1.

To set the PVT Mode and other commands via PostNav please use the 'Navigation|Positioning Mode' menu option. With pvtcalc, you will need to create a command ASCII file which can then be passed to the application via the '-c' command line option.

## 7.10 Obtain DGPS or RTK using input from a base station to increase accuracy

In order to obtain high accuracy solutions, the position data obtained from a receiver data needs to be augmented with data from a static receiver at a known fixed position in order to perform differential corrections. In order to produce a differential solution using Septentrio NV/SA Receivers the 'rover' needs to obtain differential corrections from either a second receiver at a fixed location or from a provider of such services.

The PPSDK Engine can use the differential corrections contained in `DiffCorrIn` blocks as part of an input stream. Alternatively the PPSDK Engine is capable of creating `DiffCorrIn` blocks given an SBF stream containing the necessary data. The relationship between the differential correction messages and the PPSDK are given below:

- If the input stream from the rover contains `DiffCorrIn` blocks they will be used to produce a solution of the appropriate mode (refer to the following subsections on DGPS and RTK solutions). The PPSDK Engine will use `DiffCorrIn` blocks containing RTCM 2, RTCM 3 or CMR data.
- An input file may be taken from a base station in the form of an SBF file, or RINEX data obtained from a provider of differential correction data.  
Using the API call `SSNSBFStream_insertReferenceStream` the PPSDK Engine will create the required `DiffCorrIn` blocks from the satellite data contained in the merged stream. These `DiffCorrIn` blocks are then used to calculate the position data in the appropriate mode from the input stream. When merging a reference stream an option is available that will cause any `DiffCorrIn` blocks already present to be removed.
- Note that when the PPSDK Engine produces the `DiffCorrIn` blocks it can only output RTCM 2 or RTCM 3 (the default) messages.

The following subsections refer to the creation of `DiffCorrIn` blocks from an inserted reference stream. However, they are also applicable if the rover stream contains the appropriate `DiffCorrIn` blocks.

### 7.10.1 Obtain an DGPS PVT Solution

In order to compute a DGPS solution the PPSDK Engine requires a second input stream created by a receiver configured as a Base Station (or Reference). This stream can be an SBF or a RINEX file and should contain the information necessary to calculate a DGPS solution. The PPSDK SBF Stream Module contains the functions necessary to convert RINEX files to an SBF stream before insertion as a reference stream. To configure the PPSDK Engine in DGPS-rover mode, the following has to be done:

The PVT processing needs to be configured to use DGPS corrections if they are available. If they are not available, your best choice would be to fall back on a standalone PVT or SBAS solution. This can be configured by the following command:

**setPVTMode, Rover, StandAlone+SBAS+DGPS <CR>** Indeed if only a DGPS solution is desired it may be necessary to use the command to ensure that an RTK solution is not calculated as the PPSDK Engine will always attempt to calculate an RTK solution by default.

Prior to invoking the PVT Calculation API function the Rover and base streams need to be merged using the `SSNSBFStream_merge` API function. Any `DiffCorrIn` blocks in the Rover stream can be removed at this point by using the option `SSNSBFSTREAM_REFOPTION_REMOVEDIFFCORR`. By default these blocks are not removed.

Note that the PPSDK Engine requires at least RTCM 2.x message 1 to function in DGPS rover mode. If no message 3 (or 23 and 24) are available, the receiver assumes the same tropospheric and ionospheric corrections at the base and the rover. If message 3 is provided (or 23 and 24), differential tropospheric and ionospheric corrections can be computed, which can significantly improve the solution, especially in cases where large height differences exist between base and rover.

RTCM 3.x data can be used to provide a DGPS solution if at least messages 1004 and 1006 are available.

Please refer to section 12 RTCM Overview for details.

To set the PVT Mode via PostNav please use the 'Navigation|Positioning Mode' menu option. With pvtcalc, you will need to create a command ASCII file which can then be passed to the application via the '-c' command line option.

For more details on DGPS refer to section 8.4.2.

## 7.10.2 Obtain an RTK PVT Solution

Similarly to the DGPS mode the PPSDK Engine can recompute the required RTCM messages in order to provide an RTK solution. Then RTK requires that the Reference Stream contains at least the minimum information to be able to encode RTCM2.x Messages into the `DiffCorrIn` blocks created.

The following minimum combinations of RTCM messages are required for the PPSDK Engine to produce an RTK solution.

### RTCM 2.x Messages

- 3, 18, 19, 22;
- 18, 19, 23, 24;
- 20, 21, 23, 24

### RTCM 3.x Messages

- 1004, 1006;
- 1004, 1006, 1012 to include GLONASS;

The input blocks required to obtain these messages are detailed in Section 12, RTCM Overview.

Message 23 and 1007 are recommended should you want to compensate for the Base or Reference antenna phase center which is dependent on the antenna type used. When converting RINEX files to SBF files the RINEX converter will generate a single `ReceiverSetup` block which will be then used for the `DiffCorrIn` insertion (message 1007). Note that the Antenna type for the Rover can be changed by using the PPSDK settings command `setAntennaOffset`. Both antennas need to be known by the PPSDK (see 7.13). More information about the antenna effects can also be found at 8.4.3.7.

To configure the PPSDK Engine in RTK-rover mode, the following has to be done:

1. The PVT processing needs to be configured to use RTK data. This can be configured by the following command:  
**`setPVTMode, Rover, StandAlone+SBAS+RTK <CR>`**. StandAlone and SBAS are included to provide fallback positions.
2. The PPSDK Engine has to be commanded to use the appropriate data in the Reference Station SBF Stream. For instance, to tell the PPSDK Engine to compute RTCM 3.x messages use:  
**`setDataInOut, F11, RTCMv3 <CR>`**  
 By default the PPSDK Engine will produce and use RTCM 3.x data.  
 To go back to the standalone operation, type:  
**`setPVTMode, Rover, StandAlone <CR>`**

For some types of VRS networks (for instance networks with long baselines between the base stations), optimal performance is obtained by forcing the type to VRS using the PPSDK settings command `setNetworkRTKConfig`:  
**`setNetworkRTKConfig, VRS <CR>`**

To set the PVT Mode via PostNav please use the 'Navigation|Positioning Mode' menu option. With pvtcalc, you will need to create a command ASCII file which can then be passed to the application via the '-c' command line option.

Please refer to section 8.4.3 for further details on the RTK positioning mode.

## 7.11 Use backwards post-processing

With more recent versions of the PPSDK it is possible to post-process a solution in backwards mode. Combining the results of forwards and backwards processing in urban conditions improves the position accuracy and reduces the jumps caused by multipath in the GNSS data. It helps to give more consistent results in difficult conditions.

The current version of the PPSDK does not yet support the automatic combining or smoothing of the final solution however it already allows you to output the forward and backward solutions in 2 separate files which can then be used for choosing/analyzing the best epochs for your application. You can also use SBF Stream classes to parse for the best results.

To enable backward processing you can use the '-b' option of pvtcalc (command line application). Via PostNav this is possible by selecting in the main screen the 'Backward' option. Using the option 'Both' in PostNav or '-B' in pvtcalc will make sure that both a forward and a backward solutions are created (output file names will be appended with .fw.sbf or .bw.sbf).

Via the API functions of the PPSDK, the backward option `SSNPPENGINE_OPTIONS_PROCESS_BACKWARDS` needs to be passed to the `SSNPPEngine_calculatePVT` function.

The backwards processing requires a first pre-processing step in forward mode which allows the PPSDK to use the best corrections present in the input SBF stream. When a forward post-processing is run this database will be created next to the output file defined in PPSDK with file extension '.db' (this step is called serialization and uses the API option `SSNPPENGINE_OPTIONS_SERIALIZE`). When the backward run is called then this file needs to exist so it can be read for choosing the best DiffCorr corrections available (this step is called de-serialization and uses the API option `SSNPPENGINE_OPTIONS_DESERIALIZE`). It is recommended to check the pvtcalc C source code to understand how to use the backward mechanism within the PPSDK API. Note that the database would be invalid should you use different settings when running in backward than from backward.

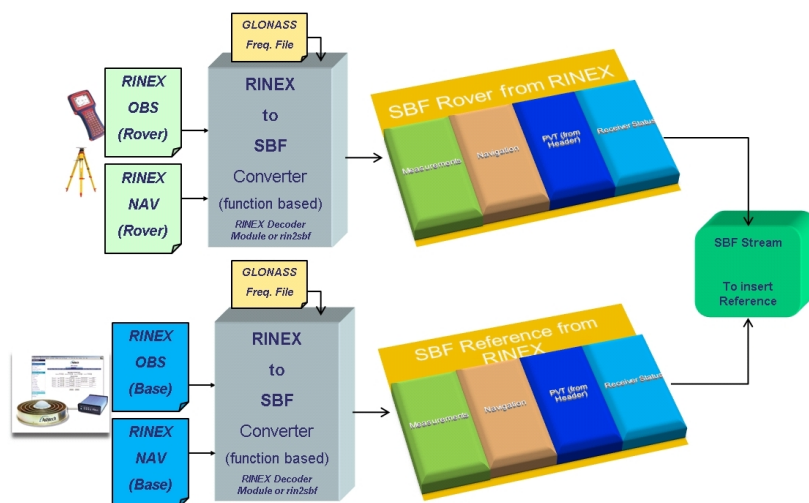
## 7.12 Use RINEX files with the PPSDK

It is possible to use the PPSDK to process files obtained from non Septentrio receivers when they are encoded in RINEX 2.x or RINEX 3.x format. Reference Station data providers (for example FLEPOS in Belgium) often make historical position data available in RINEX Format.

The PPSDK API provides functions to import and convert RINEX files:

- The function `SSNRNXDec_addRinexFile` adds a RINEX file to a list of files to be imported. For each required constellation a the RINEX Observables and Nav files should be added.
- `SSNRNXDec_listRinexFiles` can be used to list the files added.
- Finally `SSNRNXDec_createSBF` creates an SBF stream and populates it with data from the RINEX files.

Where possible obtain RINEX observables containing the L1 and L2 dopplers for better accuracy. A limitation of RINEX is its lack of data content in comparison to other formats like SBF.



**Figure 7-1: Rinx Decoder**

The PPSDK also delivers RINEX Converter (GUI application) and the rin2sbf application (command line tool). You can directly use these applications for making RINEX conversions for your post-processing.

Please note that RTCM 3.x messages require Doppler measurements in the input. As such you RINEX files without Doppler can only be used if you force the Doppler computation when converting from RINEX or if you use RTCM 2.x messages for the insertion. The computation offered by the RINEX converter is not recommended for RINEX files with a large interval rate or for kinematic files.

## 7.13 Use a different NGS Antenna file

For high-precision positioning, the GNSS measurements need to be corrected in such a way that they all refer to a common and stable point in space. That point is referred to as the antenna reference point (ARP) (See 8.4.3.7).

The PPSDK installer by default delivers the latest antenna file from the National Geodetic Survey (NGS). You may always use a new antenna definition file by converting a file in NGS format to a bin format which can be then used by the PPSDK. Please see to see section 11.11 to understand how to convert an NGS antenna file. Once converted you will need to modify the environment variable which defines the location of the NGS file used in the PPSDK (see section 2.2.3 and 8.4.3.7).

## 7.14 Check the Permission File

A permission file is a special file that enables the optional features (such as GLONASS, Galileo, RTK, ...) of your PPSDK. The permission file is associated with the license number contained in the dongle delivered with the PPSDK.

The permission file is stored in a binary file on your PC, and can be checked with the command **getReceiverCapabilities** and with the command **lstInternalFile, Permissions**. This could be included in a command text file and passed as a parameter to pvtcalc. However the easiest way is to run Post-Nav and then select PPSDK Permissions from the Help Menu. All permissions and their values are then listed; however it is strongly recommended to also take a look at the 'Help/Engine Interface' screen where you will find the Permitted Capabilities of the PPSDK.

To enable new options, the user can order a new permission file from Septentrio, and install it on his/her PC using the license setup.

## 7.15 Use the API to Analyze SBF Files

Before using an SBF file with the PPSDK it will sometimes be necessary to analyse the file in order to determine if it already contains a solution, if there were any errors and which satellites were tracked and used.

One method is to use the SBF Analyzer or SBF Converter Windows GUI application provided with RxTools package. This once can be downloaded freely from Septentrio web site. Alternatively, the SBF Analyze Module API contains some basic functions to return statistics related to an SBF file:

- Retrieve the mode and error percentages in an SBF file.
- List the tracked and used satellites.
- Determine if an individual satellite was tracked or used in the PVT Computation.

Refer to the provided HTML API documentation for details.

## 7.16 Use with other programming languages

The PPSDK libraries are 'C' libraries. In order to use them with languages other than 'C' or to use a .Net compiler it necessary to use the Dynamic Libraries and create an import class in order to access the functions in the Dynamic Link Library. It is important to mention that Septentrio does not guarantee proper functioning of the PPSDK with other languages other than 'C' and 'C++', however limited testing has been performed with languages such as 'c#' where the PPSDK has proven to worked properly.

The sample application `ListMissingEpochs` (provided with the PPSDK) is written in c# and contains the required import classes as an example. The `SbfStream` class is reproduced here.

```
using System.Runtime.InteropServices;
using System.Text;
using SbfStreamHandle = PpSdkLib.SsnHandle;
using SSN_ERROR = System.UInt32;
using SSN_SBFID = System.UInt16;
using SsnSdkHandle = PpSdkLib.SsnHandle;

namespace PpSdkLib
{
    public class SbfStream
    {
        public enum SbfStreamReadOptions
        {
            SBFSTREAM_OPEN_READONLY,
            SBFSTREAM_OPEN_READWRITE
        }

        [DllImport("ppsdk.dll")]
        public extern static
        SSN_ERROR SSNSBFStream_open(
            SsnSdkHandle ssnsdkhandle,
            ref SbfStreamHandle sbfstreamhandle);

        [DllImport("ppsdk.dll")]
        public extern static
        SSN_ERROR SSNSBFStream_loadFile(
            SbfStreamHandle sbfstreamhandle,
            StringBuilder filename,
            SbfStreamReadOptions openoptions)
    }
}
```

```
[DllImport("ppsdk.dll")]
public extern static
SSN_ERROR SSNSBFStream_close(
SbfStreamHandle sbfstreamhandle);

[DllImport("ppsdk.dll")]
public extern static
SSN_ERROR SSNSBFStream_rewind(
SbfStreamHandle sbfstreamhandle);

[DllImport("ppsdk.dll")]
public extern static
SSN_ERROR SSNSBFStream_getCommonEpochInterval(
SbfStreamHandle sbfstreamhandle,
SSN\_SBFID sbfid,
ref double interval);

[DllImport("ppsdk.dll")]
public extern static
SSN_ERROR SSNSBFStream_getNextMissingEpoch(
SbfStreamHandle sbfstreamhandle,
SSN\_SBFID sbfid,
double interval,
ref double gnsstime);

[DllImport("ppsdk.dll")]
public extern static
SSN_ERROR SSNSBFStream_writeToFile(
SbfStreamHandle sbfstreamhandle,
StringBuilder filename);
public SbfStream()
{
// Nothing to do in here
}
}
}
```

## 7.17 Create the Debug Log File whilst using the PPSDK

The PPSDK will automatically write any warning and error events to a log file during processing. The user needs to perform no action other than to determine the path to and file name of the log file. This is held in an environment variable created during the PPSDK installation see 2.2.4. The file path provided by the installation may be overwritten by editing the environment variable. This file is important when submitting problems to the Septentrio Support team since it will help in understanding any possible internal errors of the Engine.

## 7.18 Interpret errors returned by the PPSDK

All API functions return an error code indicating success or failure. It is recommended to always evaluate the return code before proceeding further in an application. The application should then free allocated data structures and close any open handles before returning the error to the user.

The error code is contained in a 32bit integer designed to be very similar to WIN32\_HRESULT and has the following structure.

	<b>S</b>	<b>Module</b>	<b>Submodule</b>	<b>G</b>	<b>Code</b>
Byte:	33	3333332222	2222111111	1	100000000
Count:	10	9876543210	9876543210	9	876543210

**Table 7-2:** ssn\_error\_t structure

The fields contained in the error code are interpreted as follows.

- S - severity indicates success(0)/fail(1).
- Module - indicates the module that generated the error.
- Submodule - indicates the submodule that generated the error.
- G - a code indicating if the error is general to all modules; Private(0), General(1).
- Code - indicates the ID of the error.

The Modules, Submodules and Error Codes are defined below.

- 0 - Unknown (also used for success).
- 1 - General.
- 2 - Post Processing Engine.
- 3 - SBF stream module.
- 4 - RINEX Decoder.
- 5 - License mechanism.
- 6 - SSN Handle.
- 7 - SSN License mechanism.
- 8 - SSN Error.
- 9 - SSN SNMP Handle.
- 10 - SBF Analyze.
- 11 - SSN SDK.
- 12 - Post-processing computation.

**Table 7-4:** Module Codes

- 0 - Unknown (also used for success).
- 1 - General.
- 2 - SBF stream module.
- 3 - RINEX Decoder.
- 4 - Post Processing Engine.
- 5 - RINEX 2.x decoder.
- 6 - RINEX 3.x decoder.
- 7 - SSN License mechanism.
- 8 - SSN Handle.
- 9 - SBF Analyze.
- 10 - SSN License PP-SDK.
- 11 - SSN License dummy.
- 12 - SSN error.
- 13 - SSN SNMP.
- 14 - SBF Analyze.
- 15 - SSN SDK.
- 16 - SSN SDK PPSDK.
- 17 - PPE Compute module (internal usage only).

**Table 7-5:** Submodule Codes



The error code itself is further divided into warnings and errors.

### Warnings

- 0 - Success.
- 1 - False success.
- 2 - Object already present.
- 3 - End of stream reached.
- 4 - End of file reached.
- 5 - Invalid SBF block.
- 6 - No DGPS data could be created.
- 7 - Given time stamp was out of SBF range.
- 8 - Reached the end of a list.
- 9 - The given SBF stream is empty.
- 10 - One or more trace errors were seen inside algorithm.

**Table 7-6:** Warning Codes

## Errors

- 11 - Unexpected event inside algorithm
- 12 - Functionality not yet implemented
- 13 - Invalid argument passed to the function
- 14 - An unexpected nullpointer has been passed to the function
- 15 - At least one parameter is out of range
- 16 - The buffer presented to this function is too small
- 17 - Passed string argument is empty
- 18 - Allocation problem (out of memory?)
- 19 - Object busy
- 20 - Object not present
- 21 - Invalid license
- 22 - No license information files not found.
- 23 - No dongle found.
- 24 - Could not initialize license object
- 25 - Demo period ended
- 26 - Invalid handle
- 27 - The specified file or directory name is not valid
- 28 - Object is read only -> probably in use
- 29 - Error opening file
- 30 - Error closing file
- 31 - Error reading file
- 32 - Error writing file
- 33 - Error seeking file
- 34 - Error while removing file.
- 35 - File was opened read only.
- 36 - Object is in the wrong state
- 37 - Invalid RINEX file
- 38 - No END OF HEADER line found
- 39 - Invalid epoch flag
- 40 - Invalid SBF file
- 41 - Invalid ASCII command
- 42 - Invalid SNMP command
- 43 - Invalid timestamp
- 44 - Invalid rate specified.
- 45 - Invalid SBF block
- 46 - Invalid SBF ID specified
- 47 - Only one instance is allowed
- 48 - Not enough GLONASS frequency numbers.
- 49 - Not enough data to compute a PVT
- 50 - Not enough data to compute RTCM3 GPS corrections (possible lack of Doppler Measurements)
- 51 - No info files path found.
- 52 - File path of info files exceeds 256 chars.
- 53 - The specified SBF id has no flex rate.
- 54 - ASN.1 parse error
- 55 - General SNMP error
- 56 - No position data
- 57 - No valid permissions file found
- 58 - Permissions not for this hardware (incorrect serial number)
- 59 - The hardware platform ID's do not match
- 60 - The specified stream is not empty

**Table 7-7:** Error Codes

### Errors

- 61 - A directory issue (does not exist or invalid permissions)
- 62 - Could not initialize NeQuick.
- 63 - The requested SBF block could not be found.
- 64 - PPECompute module failed to initialize.
- 65 - PPECompute module PVT computation failed.
- 66 - PPECompute module NavMsg decoding failed.
- 67 - PPECompute module Meas decoding failed.
- 68 - PPECompute module failed to update.
- 69 - PVT computation failed.
- 70 - RTCM encoding failed.
- 71 - Failed to append an SBF block.
- 72 - PPECompute module PVA computation failed.
- 73 - PPECompute module Ext Meas for INS decoding failed.
- 74 - Invalid settings combination for ELC
- 75 - Number of warning and error codes in this enum

**Table 7-8:** Error Codes

Please use the Module, sub-module or Error definitions described in the API reference instead of the return value for assuring future forward or backward compatibility.



Refer to the provided HTML API documentation for details or for an updated list of these errors or module names.

## 8 Receiver (PPSDK Engine) Operation Details

This Chapter describes the key processes implemented in the Septentrio NV/SA receivers. It provides useful background information which will help the user to understand similar processes in the PPSDK. Not all commands given in this section are applicable to the PPSDK and are marked as such. Please consider that you should think of the PPSDK engine whenever the 'receiver' term is used in this section.



### 8.1 GNSS Constellation and Signal/Satellite Usage

The receiver automatically allocates satellites to tracking channels up to the limit of the number of channels. It is possible to override this automatic channel allocation by forcing a satellite to a given channel by using the **setChannelAllocation** command. Also, a subset of satellites or a whole constellation can be disabled with the **setSatelliteTracking** command.

For each satellite, the receiver tries to track all signal types enabled with the **setSignalTracking** command. For example, if that command enables the GPSL1CA, GPSL2PY and GLOL1CA signals, GPS satellites will be tracked in dual-frequency mode (GPSL1CA and GPSL2PY) and GLONASS satellites will be tracked in single-frequency mode (GLOL1CA only). It is a good practice to only enable those signal types that are needed for your application to avoid wasting tracking channels.

### 8.2 Generation of Measurements

For each tracked GNSS signal, the receiver generates a "measurement set", mainly consisting of the following observables:

- a pseudorange in meters;
- a carrier phase in cycles;
- a Doppler in Hertz;

- a carrier-to-noise ratio in dB-Hz.

All data in a measurement set, and all measurement sets are taken at the same time, which is referred to as the "measurement epoch". All the measurement sets taken at a given measurement epoch are output in a `MeasEpoch` SBF block.

Several commands affect the way the receiver produces and outputs measurements:

- The `setHealthMask` command can be used to filter out measurements from unhealthy satellites: these measurements will not be used by the PVT algorithm, nor will they be included in the `MeasEpoch` SBF block.
- To further reduce the code measurement noise, the receiver can be ordered to smooth the pseudorange by the carrier phase. This technique, sometimes referred to as a "Hatch filtering", allows to reduce the pseudorange noise and multipath. It is controlled by the `setSmoothingInterval` command and is disabled by default.
- The `setMultipathMitigation` command can be used to enable or disable the mitigation of multipath errors in the pseudorange. It is enabled by default.

For advanced applications or in-depth signal analysis, the `MeasExtra` SBF block contains various additional data complementing the `MeasEpoch` SBF block. Among other things, this block reports the multipath correction applied to the pseudorange (allowing one to recompute the original pseudorange), and the observable variances.

## 8.2.1 Pilot vs. Data Component

Most modern GNSS signals consist of two components: a so-called pilot component and a data component. For such signals, the measurements are based on the pilot component for optimal performance. More specifically, the table below indicates which signal component is used for all signals having a pilot and data component.

Signal	Signal component being used for measurement generation
Galileo L1	L1-C (for GIOVE satellites, L1-B is used instead)
Galileo E6	E6-C
Galileo E5a	E5a-Q
Galileo E5b	E5b-Q
Galileo E5AltBOC	E5AltBOC-Q
GPS L2C	L2C-L
GPS L5	L5-Q

## 8.3 Time Management

All time tags in the receiver refer to the receiver time scale. The receiver is designed in such a way that the receiver time is kept as close as possible to the selected GNSS system time (GPS or Galileo as prescribed by the `setTimingSystem` command). Internally, the receiver time is kept in two counters: the time-of-week counter in integer milliseconds (TOW) and the week number counter (WNC). WNC counts the number of complete weeks elapsed since January 6, 1980 (even if the selected GNSS system time is Galileo). The TOW and WNC counters are reported in all SBF blocks.

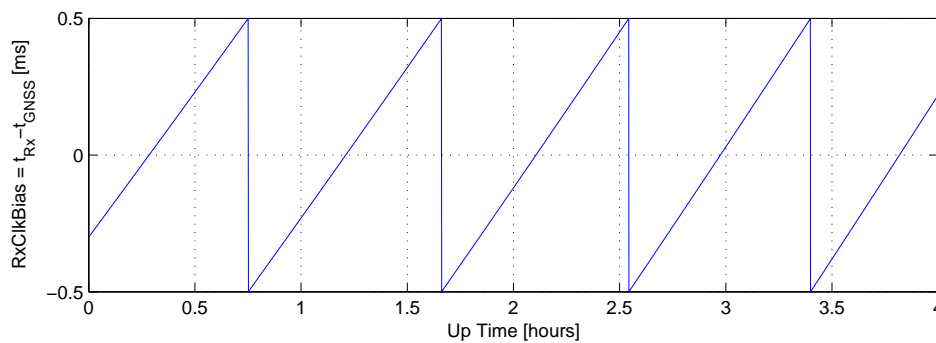
The synchronization of TOW and WNC with the GNSS system time involves the following steps:

- Upon powering up the receiver, TOW and WNC are assumed unknown, and set to a "Do-Not-Use value" in the SBF blocks.
- The transmission time-of-week and week number are coded in the GPS or Galileo navigation messages:
  - As soon as the first time-of-week is decoded from the GPS or Galileo Signal In Space (SIS), the TOW counter is initialized to within 20 ms of GNSS system time and starts counting. This is also the time when the receiver starts generating measurements.
  - As soon as the week number is decoded from the GPS or Galileo SIS (which can be either simultaneously with the time-of-week, or several seconds later), the WNC counter is set and starts counting.

- After the first position and time fix has been computed (for which measurements from at least 4 satellites are required), TOW is set to within K milliseconds of GNSS time. This is done by introducing a jump of an integer number of milliseconds in the TOW counter. K is the maximal allowed offset between the receiver time and GNSS time, and is set by the **setClockSyncThreshold** command (by default, K=0.5ms). This initial clock synchronization leads to a simultaneous jump in all the pseudorange and carrier phase measurements.

The level to which the receiver time is synchronized with the GNSS system time is given by three status bits (TOWSET, WNSET and FINETIME) available both in the *ReceiverTime* SBF block and the *ReceiverStatus* SBF block.

The receiver time unavoidably drifts relative to the GNSS time. The receiver continuously monitors the bias between its internal clock and GNSS time: this is the clock bias term computed in the PVT solution, as provided in the *RxClockBias* field of the *PVTCartesian* and *PVTGeodetic* SBF blocks. A clock jump of an integer number of milliseconds is imposed on the internal clock each time the clock bias exceeds K milliseconds by an absolute value. This typically results in a saw-tooth profile similar to that shown in Figure 8-1. In this example, K=0.5ms and each time the clock bias becomes greater than 0.5ms, a jump of 1ms is applied.



**Figure 8-1:** Example of the evolution of the receiver time offset with respect to the GNSS time if K=0.5.

When a receiver clock jump occurs, all measurements jump simultaneously. For example, a clock jump of 1ms will cause all the pseudoranges to jump by  $0.001s \times \text{velocity\_of\_light} = 299792.458m$ . The jump is applied on both the pseudoranges and the carrier phase measurements, and hence will not be seen on a code-minus-phase plot.

## 8.4 Computation of Position, Velocity, and Time (PVT Solution)

The receiver computes the position and velocity of its antenna, and the time offset of the receiver based on the pseudoranges, the Doppler measurements and, if applicable, the differential corrections.

The availability of the PVT depends on:

- the number of available pseudoranges and Doppler measurements, equal to the number of tracked satellites, or a subset of them as specified by the **setSatelliteUsage** command;
- the number of valid sets of broadcast ephemerides, which are needed to compute the position, velocity, and clock bias for each tracked satellite;
- the number of valid sets of fast and long-term SBAS corrections and their age in the case of SBAS-aided positioning;
- the number of valid differential corrections and their age in the case of DGPS/RTK positioning.

A position fix requires a minimum of 4 tracked satellites with associated ephemerides. When only 3 satellites are available or in case of bad satellite geometry (large DOP), the receiver will compute a 2D position fix assuming that the ellipsoidal height is the same as for the latest 3D fix. The mode of position fix is reported by the *Mode* field in the PVT-related SBF blocks. If less than 3 satellites are available, the receiver does not compute a position.

When a PVT solution is not available, PVT-related SBF blocks are still output with all the numeric fields set to Do-Not-Use values, and with the `Error` field set to indicate the source of the problem.

The accuracy of the PVT depends on:

- The signal level: measurements with a  $C/N_0$  of 32 dB-Hz will exhibit considerably more noise than measurements with a  $C/N_0$  of 52 dB-Hz. Hence it is recommended to use a high quality antenna.
- The geometry of the satellite constellation expressed in the DOP values: these values indicate the ratio of positional errors to range errors and are computed on the basis of the error propagation theory. When the DOP is high, the accuracy of positioning will be low.
- The number of available satellites: the more satellites are available, the lower the DOP. Measurement redundancy also enables better outlier detection.
- Multipath errors on the pseudorange measurements: multipath errors can be largely attenuated by enabling the APME multipath mitigation method (see `setMultipathMitigation`) and/or using code smoothing (see `setSmoothingInterval`).
- The PVT mode as set by the `setPVTMode` command: the user can select between the following modes, listed in the order of increasing accuracy: standalone, SBAS, DGPS and RTK.
- The data available to compute ionospheric delays (see `setIonosphereModel`).
- The choice of the dynamics model: if the dynamics parameter set by the `setReceiverDynamics` command does not correspond to the actual dynamics of the receiver platform, the position estimation will be sub-optimal.

The a-posteriori accuracy estimate of the computed position is reported in the variance-covariance matrix, which comes in the `PosCovCartesian` and `PosCovGeodetic` SBF blocks. This accuracy estimate is based on the assumed measurement noise model and may differ from actual errors due to many external factors, most of all multipath.

By default, the pseudoranges from the geostationary SBAS satellites are not used in the PVT solution due to the lower quality of the SBAS ephemerides and pseudoranges. However, for applications where satellite availability is expected to be low, it could be beneficial to allow their use in the PVT computation. This can be done by using the `setSatelliteUsage` command.

## 8.4.1 SBAS Positioning

SBAS, which stands for 'Space-Based Augmentation System', enables differential operation over a large area with associated integrity information. System errors are computed from a dataset recorded over a continental area and disseminated via a geostationary satellite. The operation of SBAS is documented in the RTCA DO 229 standard. SBAS improves over DGPS corrections, in that it provides system corrections (ionosphere corrections and ephemeris long-term corrections) next to range corrections (the "fast corrections" in the DO 229 terminology).

The receiver provides an SBAS-aided position when it has sufficient satellites with at least fast and long-term corrections. The corrections are used as long as their applicability has not timed out. During the time-out interval the receiver applies correction degradation using the information received in message type (MT) 07 and 10.

The receiver will attempt to optimise the selection of the SBAS correction provider based on the number of corrections available. For example when it has only 4 corrections from EGNOS but 8 corrections from WAAS the receiver will use the WAAS satellite even though it may be located in the EGNOS service area.

The PVT propagates the correction variances into a Horizontal Protection Level (HPL) and a Vertical Protection Level (VPL). These protection levels indicate the expected user error with an integrity of  $10^{-7}$ . Note that these protection levels only refer to the signal-in-space errors. Local effects such as severe multipath are not considered into the HPL/VPL computation.

If the service provider transmits MT27 and MT28, the receiver can detect when it is located outside the service area and adjust the PVT accuracy accordingly. Without these messages the receiver has no means of knowing the extent of the service area.

The DO 229 standard defines two operation modes for SBAS positioning: en-route and precision approach. As the integrity requirements for final approach are significantly higher, the HPL/VPL values in this mode are higher and, more importantly, the time-out interval of the corrections is shorter, which can lower the availability of a position. The default operation of the receiver is en-route, and the user has the choice to select final approach using the `setSBASCorrections` command.

An SBAS provider can transmit MT00 to reset the data transmission in case of severe errors. However, this message is also transmitted for test purposes. For proper operation during a test phase (such as ESTB), it is recommended to ignore the MT00, which can be done using the **setSBASCorrections** command.

The **GEOCorrections** SBF block contains all the corrections and their variances as used in the PVT computation. This block allows for a detailed analysis of the SBAS PVT computation in the receiver.

## 8.4.2 DGPS Positioning (Single and Multi-Base)

Differential GPS (DGPS) reduces the effect of GNSS system errors by the use of range corrections. GNSS system errors such as orbit and atmospheric errors are highly correlated within an area of several kilometres. This can be exploited by computing the pseudorange errors with respect to one or more known locations and by transmitting these errors to nearby users. The receiver can be configured as a DGPS rover, in which it accepts range corrections, or as base in which it computes range corrections.

Local errors at base stations, such as multipath, will propagate into the rover position. Hence a high quality antenna should be used and care should be taken in the choice of the location of the base station(s). Furthermore any error in the base coordinates will translate in the rover position.

To work in DGPS rover mode, the receiver requires the reception of differential corrections. The format of these corrections is standardized in RTCM.

Note that the receiver takes the  $\tau_{gd}$  parameter transmitted by the GPS satellites into account during the computation of the pseudorange corrections, as prescribed in v2.2 and v2.3 of the RTCM standard. The RTCM standard version 2.1 is ambiguous in this respect: it does neither prescribe nor discourage the use of  $\tau_{gd}$ . The receiver can be configured in both modes using the command **setRTCMv2Compatibility**.

If the received RTCM stream contains corrections from multiple base stations, the receiver will compute a multi-base DGPS solution, unless the user has forced the usage of a particular base station with the command **setDiffCorrUsage**. Be aware that multi-base DGPS can quickly overload the receiver processor if the number of base stations is large.

## 8.4.3 RTK Positioning

RTK, which stands for "Real-Time Kinematic", is a carrier phase positioning method where the carrier phase ambiguities are estimated in a kinematic mode: it does not require static initialization.

To work in RTK mode, the receiver requires the reception of RTK messages. Both the RTCM and the CMR message formats are supported. The base station providing these RTK messages can be either static or moving. Multiple-base RTK is not supported: by default, the receiver selects the nearest base station if more than one base station is available.

In RTK mode, the absolute position is reported in the **PVTCartesian** or **PVTGeodetic** SBF blocks, and the baseline vector is reported in the **BaseVectorCart** and **BaseVectorGeod** SBF blocks.

### 8.4.3.1 Pseudorange versus carrier phase: ambiguity

Pseudoranges typically have a thermal noise in the decimetre range. The resulting position accuracy is in the metre range if multipath and orbit errors are taken into account. On the other hand, the phase measurements from the carrier signal are very precise, with a millimetre-level precision.

However, phase measurements are by nature ambiguous. Consider the dial of a clock as an analogue: if only the big hand would be available on the dial we would only know how many minutes have gone by. Only by counting the hour crossovers every 60 minutes we could gain the knowledge of the current hour. GPS carrier phase measurements behave in the same way: we only know the current phase but do not know the total number of wavelengths which make up the range to the satellite: the carrier phase contains an ambiguity. To actually use the carrier phase measurement as a satellite range, this ambiguity has to be resolved.

Summing up, pseudorange measurements are low accuracy absolute ranges to GPS satellites, while carrier phase measurements are high precision relative ranges to satellites. By estimating the ambiguity, the carrier phase measurements are turned into high-accuracy satellite ranges, and the low accuracy pseudoranges are not needed for positioning.

### 8.4.3.2 Carrier Phase Positioning

To use the high accuracy of the carrier phase measurements, error sources such as broadcast ephemeris errors, satellite clock errors and atmospheric delay must be eliminated as much as possible. This is achieved by performing differential positioning: by differencing the phase measurements with those of a receiver at a nearby location. The common errors are eliminated and the position can be accurately estimated with respect to this base station. This requires two receivers which are connected by a data link. One receiver (the base) is located at a known location and transmits its position and measurements to another receiver (the rover) which is placed at the location of interest. Standardized data format for this measurement exchange are RTCM 2.2 and higher or CMR. Thanks to this standardization, measurements from publicly available reference stations can also be used, eliminating the need for a second receiver. The distance between the roving receiver and the reference station will be the driving factor to make the choice between a dedicated and a public base station: as the baseline length increases, the common errors will start to decorrelate.

Due to the differential nature of phase positioning, the unknown ambiguities of phase measurements become integer. This is the key to the accuracy of carrier phase positioning: if the exact integer value of the ambiguity is known, phase measurements can be used as highly accurate satellite ranges. If the ambiguity cannot be estimated as an integer, the ambiguity will absorb errors that did not completely cancel in the differential application, such as multipath.

### 8.4.3.3 Integer Ambiguities (RTK-fixed)

Under normal circumstances the receiver will compute the integer ambiguities within several seconds and yield an RTK-fixed solution with centimetre-level accuracy. The less accurate pseudorange measurements will not be used. As long as no cycle slips or loss-of-lock events occurs, the carrier phase position is readily available.

RTK with fixed ambiguities is also commonly referred to as phase positioning using 'On-The-Fly' (OTF) ambiguity fixing. The RTK positioning engine of the receiver uses the LAMBDA method<sup>2</sup> developed at Delft University, department of Geodesy.

### 8.4.3.4 Floating Ambiguities (RTK-float)

When data availability is low (no L2 data or low number of satellites) or when the data are not of sufficient quality (high multipath), the receiver will not fix the carrier phase ambiguities to their integer value, but will keep them floating. At the start of the RTK-float convergence process, the position accuracy is equal to that of code-based DGPS. Over the course of several minutes the positional accuracy will converge from several decimetres to several centimetres as the floating ambiguities become more accurate.

### 8.4.3.5 Moving Base

In RTK, the base station does not necessarily need to be static. In some applications, one is interested in the relative positioning of two moving vehicles. In that case, both base and rover receivers are mounted on moving platforms and the RTK engine computes the baseline between them. If both base and rover receivers are mounted on the same vehicle, the baseline can be used to determine the orientation of the vehicle. If accurate absolute positioning is required in addition to relative positioning, the moving base receiver can operate in RTK mode and get RTK correction from a fixed base station.

With the command `setDiffCorrUsage`, the rover receiver must be informed that the base is moving. The baseline coordinates and orientation is contained in the `BaseVectorCart` and `BaseVectorGeod` SBF blocks.

<sup>2</sup> Teunissen, P.J.G., and C.C.J.M. Tiberius (1994) Integer least-squares estimation of the GPS phase ambiguities. Proceedings of International Symposium on Kinematic Systems in Geodesy, Geomatics and Navigation KIS'94, Banff, Canada, August 30-September 2, pp. 221-231.



Due to delays in the generation and transmission of the RTK data (base station position and measurements) from the base to the rover, the RTK data has a certain "age" when received by the rover. When operating with a moving base station, the RTK engine is of the "low-latency" type. This means that, when the rover computes its RTK position at time  $t_0$ , it extrapolates the most recently received RTK data from the base to time  $t_0$ . The accuracy of this extrapolation, and hence the accuracy of the final RTK solution, degrades with the age of the RTK data. Therefore it is essential that the base sends its position and measurements at a sufficient rate.

The default rate of 1 Hz is adequate in the case of a static base station, but is generally too low for a moving base with a non-constant velocity. For its extrapolation, the rover assumes a constant velocity of the base. If the base is subject to an acceleration  $a$ , the extrapolation error for an age  $\Delta t$  is given by  $a\Delta t^2/2$ . Even for moderate values of acceleration, it is apparent that the error will rapidly grow (e.g. it is 50 cm for an acceleration of 0.1g and an age of 1 second). In moving base operation, it is therefore recommended to set the RTK data rate to its maximum allowed value of 10 Hz.

Not only the RTK data rate, but also the communication link latency is important. Especially in moving base, it is essential to have a low-latency communication link between base and rover. To avoid old data to corrupt the RTK solution, the rover discards any RTK data of which the age exceeds a prescribed threshold (see the **setDiffCorrUsage** command). The default threshold value is 20 seconds. For moving base, it is recommended to reduce this value to 5 seconds.

### 8.4.3.6 Datum Transformation

RTK coordinates are expressed in the same reference frame as the one in which the base station coordinates are expressed. For example, if the base station coordinates relate to the ETRF frame, the RTK coordinates reported in the `PVTCartesian` and `PVTGeodetic` SBF blocks will also relate to ETRF.

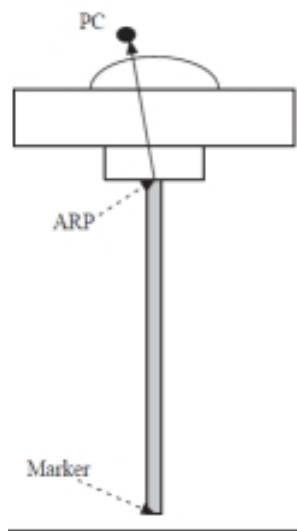
However, in many cases, RTK users must present their results in the coordinates of local datums instead of global or regional datums. If your RTK service provider transmits coordinate transformation parameters in RTCM v3.x message types 1021 to 1023, the receiver can compute coordinates in the applicable local datum. Local-datum coordinates are reported in the `PosLocal` SBF block, while the `PVTCartesian` and `PVTGeodetic` SBF blocks always contain untransformed coordinates.

The following conditions must be met for the receiver to provide a valid position in the `PosLocal` SBF block:

- the receiver must be in RTK positioning mode;
- the usage of RTCM v3.x MT1021-1023 must be enabled by the command **setRTCMv3Usage** (these messages are enabled by default);
- complete datum transformation parameters must have been received from the RTCM stream;
- the position must be in the area of validity of the transformation parameters.

### 8.4.3.7 Antenna Effects

To achieve the highest precision in RTK operations, it is essential to take antenna effects into account.



**Figure 8-2: Antenna mount.**

The GNSS measurements (pseudoranges and carrier phases observables) refer to a theoretical point in space called the phase center (noted PC in figure 8-2). The position of this point is dependent on the elevation of the satellite and on the frequency band. It varies with time and it is different for L1 and L2. The phase center variation can reach a few centimeters.

If no correction is applied, the computed position refers to an "average" phase center with no easy link with the antenna physical element. This average phase center fluctuates with time and cannot be used for accurate millimeter-level positioning.

For high-precision positioning, the GNSS measurements need to be corrected in such a way that they all refer to a common and stable point in space. That point is referred to as the antenna reference point (ARP). For convenience, it is usually selected at the center of the bottom surface of the antenna. The National Geodetic Survey has calibrated the offset from the PC to the ARP as a function of the elevation and of the frequency band for a large number of geodetic-grade antennas. NGS publishes calibration tables that can be downloaded from the following URL:

<http://www.ngs.noaa.gov/ANTCAL/index.shtml>.

The antenna naming convention in such table is the one adopted by the IGS Central Bureau.

The receiver has a copy of the absolute calibration table in its non-volatile memory. In the PPSDK this table can be upgraded following the standard procedure as described in section 7.13. Please check the manual of your Septentrio receiver to understand how to upgrade the antenna info file on the receiver itself. To let the receiver compensate for the phase center variations and compute the ARP position, the user must specify the type of his/her antenna using the **setAntennaOffset** command. If the antenna is not specified, or the antenna type is not present in the antenna calibration file, the receiver cannot make the distinction between phase center and ARP, and the position accuracy is slightly degraded, especially in the height component.

The point to be positioned is the "marker" (see figure 8-2). The offset between the ARP and the marker is a function of the antenna monumentation. It must be measured by the user and specified with the **setAntennaOffset** command.

The absolute position reported in the **PVTCartesian** and **PVTGeodetic** SBF blocks is always the marker position.

The base-to-rover baseline coordinates in the **BaseVectorCart** and **BaseVectorGeod** SBF blocks is from ARP to ARP unless the receiver is not able to properly compensate for the phase center variation at base or rover. Details on this is to be found in the description of these blocks in the SBF Reference Guide.

### 8.4.3.8 Practical Considerations

The reasons for possible low accuracy or availability of the RTK position are:

- Multipath;
- Ionosphere decorrelation;
- Loss-of-lock;
- L2 availability;
- RTCM/CMR availability.

To ensure high accuracy and availability, care must be taken that the above error sources have as little impact as possible. This can be achieved by using survey-grade antennas and choosing a suitable location for the base station with an unobstructed view of the sky. Since low-elevation satellites are more prone to loss-of-lock and multipath, it is also recommended to use an elevation mask of 10 degrees.

The availability of fixed ambiguities increases significantly with the use of L2 carrier phase measurements. When in single-frequency operation, it is advised to force the receiver to remain in RTK-float mode, using the `setPVTMode` command.

## 8.4.4 Precise Point Positioning

Precise Point Positioning (PPP) provides high accuracy positioning without the need for a local base station. PPP uses precise satellite orbit and clock corrections computed by a global network of reference stations and broadcast in real time by geostationary satellites in the L band.

PPP provides centimeter-level position accuracy, but suffers from a relatively long convergence time that can reach 15 to 20 minutes depending on the local multipath environment.

### 8.4.4.1 PPP Seeding

PPP provides centimeter-level position accuracy, but suffers from a relatively long convergence time that can reach 15 to 20 minutes depending on the local multipath environment. The convergence time can be dramatically reduced by feeding the known position into the PPP engine. This process is referred to as PPP seeding, and the position fed into the PPP engine is called the PPP seed. The receiver supports two seeding modes:

**Manual Seeding:** in manual seeding, the user provides the accurate marker position (see section 8.4.3.7 for a definition of the marker position) to the PPP engine by using the `exePPPSetSeedGeod` command.

**Automatic Seeding:** the receiver can be configured to automatically seed the PPP engine from its current DPGS or RTK position. Automatic seeding is configured with the `setPPPAutoSeed` command.

### 8.4.4.2 PPP Datum Offset

By default, PPP positions are expressed in the ITRFxx reference frame (xx depending on the PPP service provider), and the PPP seed must relate to the ITRFxx frame as well. You can command the receiver to apply a datum offset compensation by using the command `setPPPDatumOffset`. In this case, both the PPP position output and the seed are corrected accordingly. Providing the offset between the datum used by your local RTK provider and ITRFxx is mandatory to allow a correct automatic seeding from RTK.

Note that the datum offset may change slowly in time, and needs to be regularly updated to avoid errors.

### 8.4.4.3 Tide Corrections

Since PPP is based on global satellite corrections, the PPP position would be sensitive to earth tide variations if no correction were applied. The receiver applies a tide correction based on the Sinko Earth tide model<sup>3</sup>. All positions reported in the `PVTCartesian`, `PVTGeodetic` and `PosCart` SBF blocks are always tide-corrected.

<sup>3</sup> Sinko, J., A Compact Earth Tides Algorithm for WADGPS. Proceedings of ION GPS-95, Palm Springs, California, September 12-15, 1995, pp. 35-44.

## 8.5 INS/GNSS Integration

GNSS receivers compute their position by tracking signals from GNSS satellites. The position is absolute and accurate, but requires constant sky visibility.

An Inertial Navigation System (INS) consists of an Inertial Measurement Unit (IMU) and a processing unit to continuously compute a relative position solution based on the sensed motion. In order to provide an absolute position, the INS needs to be initialized with absolute position information from an external device such as a GNSS receiver. The accuracy of the relative position solution provided by the INS degrades with time due to IMU measurement errors, such that periodic re-initialization with the absolute position is required.

Integrating INS and GNSS offers several advantages:

- The receiver is capable of providing precise positioning in shadowed environments where GNSS-only would fail.
- Due to the large bandwidth of the IMU, the receiver is able to sense high dynamics and to output the position and velocity information at a higher rate.
- Besides accelerometers, the IMU contains three orthogonal gyroscopes to measure the angular-rate. This allows to provide the vehicle attitude.

The current version of the firmware supports the MTi IMU from Xsens.

The integrated INS/GNSS position, velocity and attitude are reported in the following SBF blocks: `IntPVCart`, `IntPVGeod` and `IntAttEuler`. The attitude angles are defined in appendix A. The maximum update rate of these blocks is receiver- and permission-dependent and can be inquired by the command **getReceiverCapabilities**. The unprocessed IMU measurements (accelerations and angular rates as provided by the sensor) are reported in the `ExtSensorMeas` SBF block.

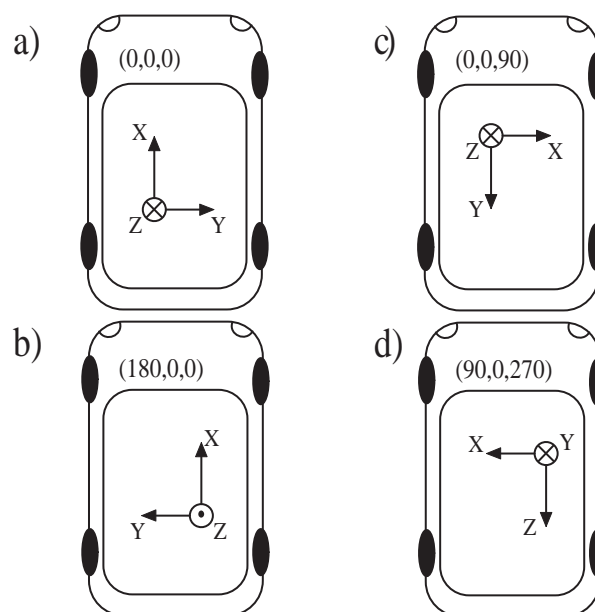
### 8.5.1 Calibration

#### 8.5.1.1 IMU Sensor Orientation

The direction of the X, Y and Z axes of the IMU sensor are marked on the sensor enclosure. Note that an axis pointing towards you is represented as  $\odot$ , while an axis pointing away from you is represented as  $\otimes$ .

By default, the INS/GNSS integration filter assumes that the IMU is mounted horizontally, upside up and with the X axis marked on the IMU enclosure pointing to the direction of travel (i.e. to the front of the vehicle). If it is not possible to mount the IMU in this default orientation, the orientation angles ( $\theta_x, \theta_y, \theta_z$ ) need to be specified with the command **setExtSensorCalibration**.

$\theta_x, \theta_y$  and  $\theta_z$  are to be interpreted as follows. If you start from the nominal axes orientation depicted in figure 8-3-a) and you first apply a right-handed (clockwise) rotation through  $\theta_z$  degrees about the Z axis, and then a right-handed rotation through  $\theta_y$  degrees about the rotated Y axis, and finally a right-handed rotation through  $\theta_x$  degrees about the twice-rotated X axis, the resulting axes must have the same orientation as the axes marked on the IMU when it is mounted on your vehicle. As an illustration, some examples of ( $\theta_x, \theta_y, \theta_z$ ) triplets for four different IMU orientations are shown in figure 8-3.

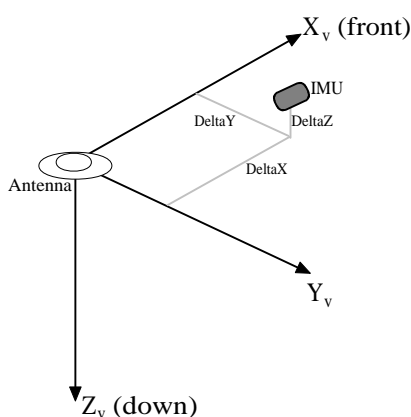


**Figure 8-3:** Example of values of  $(\theta_x, \theta_y, \theta_z)$  for different IMU orientations on a car. X, Y and Z refer to the axes as marked on the IMU enclosure.

### 8.5.1.2 Lever Arm

The lever-arm is the vector from the GNSS antenna reference point (ARP) to the IMU reference point. On the MTi IMU, the reference point is the origin of the X and Y axes drawn on the sensor enclosure.

For optimal INS/GNSS integration, the effect of the lever arm must be compensated for. The user must provide the relative position ( $\Delta X$ ,  $\Delta Y$  and  $\Delta Z$ ) of the IMU with respect to the antenna with the command **setExtSensorCalibration**.  $\Delta X$ ,  $\Delta Y$  and  $\Delta Z$  are resolved in the vehicle reference frame (see appendix A for a description of the reference frames). An example is depicted in figure 8-4.



**Figure 8-4:** Example of antenna/IMU relative position. In this example,  $\Delta X$  and  $\Delta Y$  are positive, while  $\Delta Z$  is negative.

An accuracy of a few centimeters in the lever arm coordinates is sufficient.

## 8.5.2 Alignment

The alignment is the process of determining the initial attitude of the vehicle with respect to the local-level East-North-Up reference frame. See appendix A for a definition of the attitude angles.

During alignment, no INS/GNSS integrated solution is available.

### 8.5.2.1 Static Coarse Alignment

The first phase of the alignment process is a static coarse alignment, where initial values of the pitch and roll are determined while the heading remains unknown.

Static coarse alignment takes 20 seconds after the INS/GNSS integration has been enabled with the `setPVTMode` command and valid PVT and IMU data are available. During that phase, the `Error` field in the integrated INS/GNSS SBF blocks is set to code 23 "static alignment ongoing". It is necessary for the vehicle to remain static during that time for accurate determination of the initial roll and pitch angles.

### 8.5.2.2 In-Motion Alignment

The second phase of the alignment aims at determining the initial heading and requires the receiver to move.

When the static coarse alignment is finished, the `Error` field in the integrated INS/GNSS SBF blocks switches to code 28 "in-motion alignment ongoing". This error code stays till the vehicle starts moving. When motion is detected, the heading is initialized based on the course-over-ground azimuth, alignment is complete and the receiver starts outputting INS/GNSS integrated positions.

Note that the initial heading determination assumes that the vehicle is moving forward. If the initial motion is in the backwards direction, the initial heading will be biased by 180 degree. This situation will typically last a few seconds and is flagged in the `Info` field of the `IntAttEuler` SBF block.

## 8.5.3 Zero-Velocity Update (ZUPT)

A zero-velocity update (ZUPT) uses the motion constraint that the vehicle is static to limit the error growth of the integrated solution.

By default, the receiver automatically detects that it is static using the GNSS velocity. When no GNSS solution is available, or in demanding environments with significant GNSS signal deterioration, the GNSS ZUPT detection may not be sufficiently reliable or not even possible. To further improve performance in these situations, the receiver is able to use an external zero-velocity signal. Many vehicles have such signal available.

The button pin and the CTS lines of COM2 or COM3 can be reconfigured as external zero-velocity input using the command `setExtZUPTSource`. Please refer to the Hardware Manual to check the availability and position of these pins on your particular receiver.

The state of the external zero-velocity signal is reported in the `ExtSensorMeas` SBF block. If GNSS is available and the receiver detects a discrepancy between the GNSS zero-velocity indication and the external zero-velocity indication, the `EXTSENSORERROR` bit of the `ExtError` field of the `ReceiverStatus` SBF block is set.

When connecting the zero-velocity wire of your vehicle to one of the input pins of the receiver, make sure that the electrical level matches.



## 8.6 Receiver Autonomous Integrity Monitoring (RAIM)

The receiver features RAIM to ensure the integrity of the computed position solution, provided that sufficient satellites are available. The RAIM algorithm consists of three steps: detection, identification and adaptation, or shortly "D-I-A"<sup>4</sup>:

- Detection : an overall model statistical test is performed to assess whether an integrity problem has occurred;
- Identification : statistical  $w$ -tests are performed on each individual measurement to assess whether it should be marked as an outlier;
- Adaptation : measurements marked as an outlier are removed from the position computation to restore the integrity of the position solution. This step is only applied if outliers have been detected in the detection step.

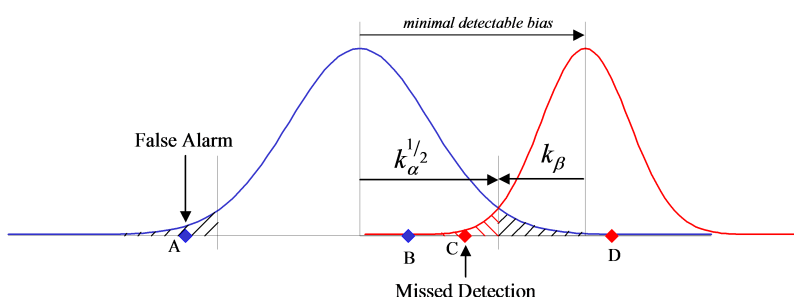
If an integrity loss is detected in the first step, the RAIM module attempts to recover from the integrity failure by removing the responsible measurement(s) identified in the second step. As a consequence, the RAIM module will generally increase the continuity of integrity. A loss-of-integrity-flag is raised if insufficient measurements remain after outlier removal (after several D-I-A steps), or if the overall model statistical test fails while no outliers can be identified. In the latter case the "sum of squared residuals too large" error is reported in the PVT related SBF blocks.

The statistical tests assume an a-priori model of the measurement error probability distribution. As such, these tests can have the four classical outcomes in hypothesis testing, as shown in the table below (the letters A, B, C and D refer to the samples in Figure 8-5):

	<i>no outlier</i>	<i>outlier present</i>
<i>outlier detected</i>	False Alarm (type I error) A	Correct D
<i>no outlier detected</i>	Correct B	Missed Detection (type II error) C

The RAIM module makes a correct decision in two cases: an outlier present in the data is indeed detected, and no outlier is detected when none is present. However, when no outlier is present and the RAIM module declares an outlier is present, a false alarm is triggered. When an outlier remains undetected, a missed detection occurs.

The probability computations are based on the assumption that the residuals are distributed as a Normal distribution (central if there is no outlier, and non-central if there is one), as illustrated in Figure 8-5.



**Figure 8-5:** Statistical test outcomes.

<sup>4</sup> Baarda, W., A Testing Procedure For Use in Geodetic Networks, Netherlands Geodetic Commission, Publ. On Geodesy, Vol.2, no. 5, 1968

Samples corresponding to the four test outcomes are represented in Figure 8-5: samples A and B are from the unbiased measurement distribution, while samples C and D are from a biased measurement distribution corresponding to an outlier. Since sample A is larger than the test threshold, it will be incorrectly flagged as an outlier (false alarm). Sample C is not detected as an outlier although it is part of the biased distribution (missed detection). The acceptable probability of false alarm and the probability of missed detection for the application must be determined and provided to the receiver. This is the purpose of the **setRAIMLevels** command.

## 8.6.1 Integrity Algorithm

Two kinds of statistical tests are performed: the detection step uses an *overall model* test to evaluate the integrity of the position solution as a whole, and the identification step uses the *w*-test (also known as "datasnooping") to evaluate the integrity of individual measurements. Depending on the positioning mode, the overall model test is computed for range, range-rate and/or phase measurements simultaneously, while the *w*-test is computed for each range, range rate and/or phase measurement individually. Both the overall model and the *w*-tests are of the *Generalized Likelihood Ratio Test* type.

The overall model test uses the weighted sum of the squared residuals as test statistic. This test statistic is distributed as a  $\chi^2$  distribution with  $r$  degrees of freedom, where  $r$  is the redundancy number equal to the number of satellites used in the position computation minus 4. The test reads:

$$\sigma^2 = \bar{e}^T Q_y \bar{e} > \chi_\alpha^2(r, 0)$$

where:

- $\sigma^2$  is the overall model test statistic;
- $\bar{e}$  is the vector of residuals;
- $Q_y$  is the variance-covariance matrix of the measurements;
- $\chi_\alpha^2(r, 0)$  is the test threshold yielding a probability  $\alpha$  of false alarm.

The probability of false alarm of the overall model test is selectable by the user with the *ModelReliability* argument of the **setRAIMLevels** command.

If the overall model test statistic is lower than the test threshold, the test is passed and the integrity is guaranteed under the statistical assumptions specified by the **setRAIMLevels** command.

If the overall model test statistic is higher than the threshold, the test is rejected. In this case, the identification step will attempt to identify the measurement responsible for the rejection using the *w*-test discussed below. After removal of the responsible outlier(s), the overall model test statistic is recomputed to verify the integrity of the solution without the outlier present. This iterative process continues until either the overall model test along with the associated *w*-tests are accepted, or until the *w*-tests for each individual measurement are accepted with a rejected overall model test. In the latter case an integrity loss is declared; in the former case integrity is available. Note that under extreme circumstances the interactive D-I-A process can also halt due to insufficient available measurements for testing, after removal of outliers. In this case the "too many outliers" error is reported in the PVT related SBF blocks.

For the evaluation of the *w*-test statistic, the following inequality is verified:

$$-k_\alpha^{1/2} < w_i = \frac{e_i}{\sigma_{e_i}} < +k_\alpha^{1/2}$$

where:

- $w_i$  is the *w*-test statistic for the  $i$ th satellite;
- $e_i$  is the residual for the  $i$ th satellite;
- $\sigma_{e_i}$  is the standard deviation of the residual for the  $i$ th satellite;
- $k_\alpha^{1/2}$  is the test threshold yielding a probability  $\alpha$  of false alarm.



The probability of false alarm of the  $w$ -test is selectable by the user with the  $Pfa$  argument of the **setRAIMLevels** command.

The test threshold is computed by the receiver with the assumption that the  $w$ -test statistic is distributed as a Normal distribution. For instance, if  $Pfa$  is set to 10%, residuals larger than 1.64 sigma are flagged as outliers. If  $Pfa$  is 0.01% the threshold will be 3.89.

## 8.6.2 Internal and External Reliability Levels

To assess the impact of undetected measurement errors on the computed position, the Minimum Detectable Bias (MDB) in the range domain is computed and propagated to the position domain.

The MDB describes the internal reliability of the corresponding  $w$ -test. It is a measure of the range error that can be detected with a given probability of missed detection. It is computed as follows for each satellite (neglecting the probability that the biased measurement falls on the left-hand side of the non-biased distribution shown in Figure 8-5):

$$MDB_i = \sigma_{y_i} \left( \frac{\lambda_0}{1 - \frac{\sigma_{\hat{y}_i}^2}{\sigma_{y_i}^2}} \right)^{1/2}$$

where:

- $\sigma_{y_i}$  is the standard deviation of the range measurement of the  $i$ th satellite;
- $\sigma_{\hat{y}_i}$  is the standard deviation of the estimator for the (measured) range of the  $i$ th satellite;
- $\lambda_0$  is the non-centrality parameter, which depends upon the probability of false alarm of the  $w$ -test and the probability of missed detection.

The user can select the probability of missed detection acceptable for his/her application with the  $Pmd$  argument of the **setRAIMLevels** command.

The external reliability is defined as the influence of a model error of size MDB on the user position. It is computed by propagating the MDB for each satellite to the position domain, taking the satellite geometry into account. The receiver computes a distinct external reliability level (XERL) for the horizontal and the vertical components (referred to as HERL and VERL respectively). These values should be compared to the alarm threshold of your specific application in order to verify if the position solution is adequate for that application.

Detailed results of the RAIM algorithm are available in the **RAIMStatistics** and the **PVTResidual** SBF blocks and in the GBS NMEA message.

## 9 Distribution

The following chapter explains how the PPSDK should be integrated into your application and what has to be taken into account when distributing your application.

### 9.1 Libraries

The PPSDK allows you to use either a *Dynamic Link Library* or a *Static Library*. The Static Library is used to embed the functionality PPSDK into your application or library directly and does not have to be shipped to the customer separately from your application. Building your application to use the Dynamic Link Library will reduce the size of your executable but the PPSDK dll has to be installed next to your application or library and therefore has to be provided to the user.

The following libraries (found in the library subdirectory of the PPSDK installation directory) are provided. In the dynamic and static subdirectories libraries are provided for use with Visual Studio C++ 2005, 2008 and 2010.

The dynamic subdirectories contain the files ppsdk.lib to be linked to the application and the distributable file ppsdk.dll.

The static directories contain the ppsdk.lib files for linking into the application. Two versions are provided in subdirectories MD and MT for Multithreaded Dynamic Linking or Multithreaded Static Linking respectively to the microsoft runtime.

It is important to use the correct compiled version of the library to avoid errors on the system. As such an application built with Visual Studio C++ 2005 may not properly work if you link it to the Visual Studio C++ 2008 library.



## 9.2 Environment Variables

To be able to use the PPSDK functionality the environment variables mentioned in chapter 2.2 must be set, either at runtime or by the installation process of your application.

# 10 Septentrio PPSDK Sample Applications

The PPSDK installation contains the source code for a number of applications illustrating the use of the PPSDK.

- ComputePVT;
- GetMIBInfo;
- ListBlocks;
- SendCommand;
- ListMissingEpochs;

The first four are Windows Console applications and the last is a Windows GUI application programmed in C#. The following subsections give an overview of each sample application. Some of the samples use the double call mechanism described earlier in section 7.2.1.

The tools are to be found in the 'samples\sources\win32' subdirectory under the installation path.

## 10.1 ComputePVT

This PPSDK sample is a simple application to compute a PVT from a single SBF file. The following functionality is used:

- Open a PPSDK license handle (necessary when a PVT is to be computed);
- Open a SBFStream handle and load the SBF File to the stream;
- Open the PPSDK Engine;
- Send a simple command to the PPSDK Engine to determine the output block Ids ;
- Invoke the PPSDK Engine using the '**SSNPPEngine\_calculatePVT**' function;
- Write the resulting output SBF file to disk;
- Close the PPSDK Engine;
- Close the SBFStream Handles;
- Close the SSN Handle.

## 10.2 GetMIBInfo

The PPSDK **GetMIBInfo** sample demonstrates how to retrieve the MIB description of the PPSDK.

The **GetMIBInfo** sample starts by opening an SSN License and a PP Engine handle. If all went well the API function to retrieve the MIB description is called. This function uses the double-call mechanism described in section 7.2.1.

After the MIB description has been displayed all opened handles are closed again.

## 10.3 ListBlocks

This sample takes an SBF file as it's input and demonstrates how to list the contents of an SBF file and print out the GNSS time stamp and ID of the blocks.

An SSN License and SBF Stream handle are created. Since the SBF Stream module does not require a valid SSN License handle an unlicensed handle can be created. Also note that we open the SBF file with the Read Only option. Since we don't need to make any modifications to the SBF file this will speed up the process.

The SBF Stream module has several functions to get the next SBF block present in an SBF file. Make sure that enough memory is allocated to store all possible SBF blocks in the VoidBlock\_t structure.

For every valid SBF block the GNSS time stamp is calculated and is printed to the screen together with the SBF ID.

Finally all opened handles are closed.

## 10.4 SendCommand

The PPSDK SendComand sample shows how to send a simple text command to the PPSDK Engine and outputs the result.

The sample application starts by opening a valid SSN License handle and a PPSDK Engine handle.

After the handles have been successfully created the ASCII command is sent. Since the function to send an ASCII command uses the so double-call mechanism it has to be called twice. The first time it is called the amount of bytes needed to store the reply message is calculated and returned. After enough memory is allocated the function is called a second time and will execute the command and return a meaningful reply message. It is recommended to set buffersize to the size of the allocated memory before the second call.

Finally all opened handles are closed.

The sample produces the following output.

```
Reply: $R: getReceiverInterface, RxName
ReceiverInterface, RxName, "Post Processing SDK"
```

## 10.5 ListMissingEpochs

This sample is written in Visual C# to show the use of import libraries enabling the use of the PPSDK in a .net environment. It uses the PPSDK API function **SSNSBFStream\_getNextMissingEpoch** to check for possible missing SBF blocks at certain epochs.

The required handles have to be opened before the PPSDK API can be used. After the handles have been opened successfully the most common delta is computed for a certain type of SBF block using the **SSNSBFStream\_getCommonEpochInterval** API function .

This delta value is used to estimate the interval at which an SBF block of the given type should occur and a loop is used to search for the next epoch where no block of the given Id is present.

Finally all opened handles are closed.

## 11 Septentrio Tools

The PPSDK installation contains a number of executable software tools that provide the user with the following possibilities:

- Recalculation of a PVT Solution given an SBF input files and optionally an SBF files from a Base Station;
- Conversion of an SBF file to RINEX 2.x or RINEX 3.x format;
- Conversion of a set of RINEX files in either 2.x or 3.x format to an SBF File;
- Listing of the contents of an SBF File in ASCII format;
- Listing of the contents of an SBF File in Septentrio Text Format (STF);
- Listing of commands contained in an SBF File;
- Manipulation of an SBF files such as block filtering and cropping;
- Conversion of an SBF file to GPX format;
- Conversion of an SBF file to KML format for Google Earth visualization;
- Listing of the block types contained in an SBF File.

The following subsections give an overview of each tool. Detailed usage can be obtained by executing the appropriate tools from a command window without any parameters. The tools are to be found in the 'tools\binary\win32' subdirectory under the installation path.

### 11.1 pvtcalc

The PPSDK **pvtcalc** tool is a Windows Console Application that goes beyond the simple CalculatePVT sample and takes a number of parameters in order to produce a PVT from merged SBF files:

- Two SBF files may be input. The program assuming that the main file is from a 'Rover' and the second optional SBF file is from a reference file 'e.g. Base Station';
- Options are available to execute or ignore any commands to be found in the main SBF file. Indeed new commands may be executed at the start of PVT processing or inserted into the input stream at a given time stamp;
- An ASN.1 file may be specified to translate commands in the main SBF stream. This option is handy when you want to re-use the commands embedded within the original SBF file;
- Navigation data usage in the input SBF stream may be specified and GEO navigation data used if required;
- The input SBF files may be cropped between two GPS times;
- The user may specify the RTCM version to be used when inserting a reference stream;
- Finally a reference position may be specified otherwise the Engine will take the position from the first PVT block it finds in the main input SBF file;
- The resulting output will contain the reprocessed PVT solution and any blocks the user requested to be transferred from the main input SBF stream.

Invoking **pvtcalc** without argument prints the list of options and their usage. The parameters taken by **pvtcalc** are given in the table below:

Argument	Value	Description
-f	input file	(Mandatory) Input SBF File
-o	output file	Output SBF File
-c	commands file	ASCII file with commands
-e	commands file	ASCII file with commands to be inserted at a given time stamp
-t	asn_1 file	ASN.1 file used to translate commands in SBF stream.
-g	geo_nav file	SBF file with GEO navigation data
-r	reference file	Reference SBF stream
-m		Allow moving base operation (change merge behavior)
-p	intermediate file	Intermediate output file for input SBF after reference and/or command insertion
-i	reference ID	Reference ID
-S	start time	Start cropping from (GPS time in seconds)
-E	end time	End cropping at (GPS time in seconds)
-d	temp directory	Directory where to store temporary files
-N	Nav Type	Navigation data usage: 0: Use any navigation blocks in SBF input file (default) 1: Use only decoded Navigation blocks in SBF input file 2: Use only raw Navigation blocks in SBF input file
-n		Disable pre-loading of all navigation blocks from input files
-s		Disable sharing of NAV data between Rover and Base file. In this case only the navigation file embedded in the corresponding SBF file will be used
-C		Compatibility mode with receiver.
-R	rtcm version	RTCM version to be used when inserting a reference SBF stream: 0: RTCM version 2 (All messages) 1: RTCM version 2 DGPS+RTK (default) 2: RTCM version 3 (All messages) 3: RTCM version 3 DGPS+RTK
-X	reference position X	The reference position X value, if not provided the position is read from the reference file.
-Y	reference position Y	The reference position Y value, if not provided the position is read from the reference file.
-Z	reference position Z	The reference position Z value, if not provided the position is read from the reference file.
-A	AntType	The IGS name of the reference (base) antenna. The rover antenna type can be set through the ascii commands and not through this option
-l	antenna offset E	The North component of the base antenna offset. If not provided and the reference position is not set (-X, -Y, -Z) the value is read from the reference file. The rover antenna offset can be set through the ascii commands and not through this option.

Table 11-1: pvtcalc Arguments

Argument	Value	Description
-J	antenna offset N	The East component of the base antenna offset. If not provided and the reference position is not set (-X, -Y, -Z) the value is read from the reference file. The rover antenna offset can be set through the ascii commands and not through this option.
-K	antenna offset U	The Up component of the base antenna offset. If not provided and the reference position is not set (-X, -Y, -Z) the value is read from the reference file. The rover antenna offset can be set through the ascii commands and not through this option.
-L	leapseconds	User defined number of Leap seconds for the Reference and Rover processing (will avoid PPSDK prediction and will overwrite leapseconds found in UTC blocks). If set to -128 then Leap seconds will be used only if UTC blocks are present in the SBF file.
-u		Remove commands in SBF file
-x		Execute commands in SBF file
-a		Allow invalid PVT solution
-b		Process input file backwards
-B		Process Both fw and bw (adding '.fw.sbf' and '.bw.sbf' to -o file
-M	measint	Use an specific Measurement interval. The default is 'on-change' (which uses the rate of measurements in the input file): 10: 10 milli-seconds. 20: 20 milli-seconds. 50: 50 milli-seconds. 100: 100 milli-seconds. 200: 200 milli-seconds 500: 500 milli-seconds 1000: 1 second. 2000: 2 seconds. 3000: 3 seconds. 4000: 4 seconds. 5000: 5 seconds. 6000: 6 seconds. 10000: 10 seconds. 15000: 15 seconds. 20000: 20 seconds. 30000: 30 seconds.
-P	pvtint	Use an specific PVT computation interval. The default is 'onchange' (which uses the rate of measurements in the input file): Same options as -M (see above)
-v		Verbose mode
-V		Program version
-D		Checks for the presence of a dongle and in case of a limited license for the number of runs left.

Table 11-2: pvtcalc Arguments (part 2)

## 11.2 rin2sbf

The PPSDK **rin2sbf** tool is a Windows Console Application that takes a number of related RINEX (2.1 or 3.0) files and converts them to an SBF file. An Observation and a GPS navigation file are required with optionally GLONASS, Galileo and SBAS navigation files and SBAS Broadcast data. Start and end GPS times may be specified. GLONASS frequency number may be input contained in a file.

Invoking **rin2sbf** without argument prints the list of options and their usage. The parameters taken by **rin2sbf** are given in the table below:

Argument	Value	Description
-o	output file	(Mandatory) Output SBF File
-f	input file	(Input RINEX File
-d	temp directory	Directory where to store temporary files
-g	GLONASS Frequency file	GLONASS frequency file
-l	leap second	Leap seconds (from 0 to 128), if not specified then an internal stored list of leap seconds will be used
-S	start time	Crop start time from (GPS time in seconds)
-E	end time	Crop end time at (GPS time in seconds)
-N		Never compute the Doppler from the carrier phase. By default, the Doppler in the SBF file is taken from the RINEX file if present, or is computed from differencing the carrier phase if there is no Dx observable in the RINEX file.
-D		Always compute the Doppler from the carrier phase, ignoring the Dx observable. See also the -N option.
-b		Do not use Backup frequency number list for GLONASS
-h		Print help message
-v		Verbose mode
-V		Program version

**Table 11-3:** rin2sbf Arguments

Note: The RINEX observation file is required, the RINEX navigation file should be included if possible. Up to 48 RINEX files can be used.

### 11.2.1 GLONASS Frequency File

The conversion of GLONASS observation data requires the presence of GLONASS frequency numbers. These can be found in a GLONASS RINEX navigation file or can be passed as an external file when using the -g switch. The RINEX to SBF converter also has a backup list of known frequency numbers which are used when no frequency number is known for a given Space Vehicle ID (SVID). If this backup list is not to be used, turn it off by using the -b switch.

If passed as an external file, it should be a text file containing one frequency number per line:

SVID=FREQ\_NBR

Example:

1=1

2=-4

5=1

6=-4

Note: Please make sure that you do not use zeros as prefixes for either the SVID or the FREQ\_NBR.

## 11.2.2 Glonass Leap Seconds

GLONASS (unlike GPS) system time is connected to UTC. Therefore GLONASS receivers need the current leap second value in order to function correctly. The leap second is provided as a parameter to the conversion. If no value is provided the function will use the leap second value current at the time the function was built.



## 11.3 sbf2asc

The PPSDK **sbf2asc** tool is a Windows Console Application that lists the contents of the blocks in an SBF file in ASCII format. **sbf2asc** was mainly created as a sample application to assist users in developing their own conversion tools. For converting SBF data into ASCII or Text format, we recommend to use the more flexible **bin2asc**.

Invoking **sbf2asc** without argument prints the list of options and their usage. The possible options for **sbf2asc** are given in the table below:

Argument	Value	Description
-f	input file	(Mandatory) Input SBF File
-o	output file	Name of the ASCII File. (if not provided, measasc.dat is used)
-m		Include contents of the (Short)MeasEpoch blocks
-p		Include contents of the PVTCartesian blocks
-g		Include contents of the PVTGeodetic blocks
-c		Include contents of the PUTCov blocks
-d		Include contents of the DOP blocks
-a		Include contents of the AttitudeEuler blocks
-s		Include contents of the AttitudeCovEuler blocks
-u		Include contents of the AuxPos blocks
-t		Include contents of the ReceiverStatus blocks
-x		Include contents of the ExtEvent blocks
-n		Include contents of the BaseStation blocks
-l		Include contents of the BaseLine blocks
-k		Include contents of the BaseLink blocks
-h		Include contents of the GPSAlm blocks
-j		Include contents of the ExtSensorMeasurements blocks
-b	start epoch	Time of first epoch to insert in the file Format: yyyy-mm-dd_hh:mm:ss.sss or hh:mm:ss.sss.
-e	end epoch	Last epoch to insert in the file Format: yyyy-mm-dd_hh:mm:ss.sss or hh:mm:ss.sss
-i	interval	Decimation interval in seconds
-E		Exclude blocks where time stamp is invalid
-v		Verbose mode, progress displayed
-V		Display the sbf2asc version

**Table 11-4:** sbf2asc Arguments

The output of **sb2asc** is a text file containing columns of data. The first column identifies the format and contents of each row as follows:

<b>1-255</b>	the row contains data from a (Short)MeasEpoch block
<b>0</b>	the row contains data from a PVTCar block
<b>-1</b>	the row contains data from a PVTGeo block
<b>-2</b>	the row contains data from a PUTCov block
<b>-3</b>	the row contains data from a DOP block
<b>-4</b>	the row contains data from a AttitudeEuler block
<b>-5</b>	the row contains data from a AttitudeCovEuler block
<b>-6</b>	the row contains data from a ExtEvent block
<b>-7</b>	the row contains data from a ReceiverStatus block
<b>-8</b>	the row contains data from a BaseStation block
<b>-9</b>	the row contains data from a BaseLine block
<b>-10</b>	the row contains data from a BaseLink block
<b>-11</b>	the row contains data from a GPSAlm block
<b>-12</b>	the row contains data from a AuxPos block

**Table 11-5:** sb2asc Row Identifier

Then for each further column the data is to be interpreted as in the tables below.

<b>Col1</b>	PRN identifier (from 1 to 255). For GLONASS, PRN is 45+FreqNumber
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	CA pseudorange in meters, or -20000000000 if not available
<b>Col4</b>	L1 carrier phase in cycles, or -20000000000 if not available
<b>Col5</b>	CA C/N0 in dB-Hz, or -3276.8 if not available
<b>Col6</b>	P1 pseudorange in meters, or -20000000000 if not available
<b>Col7</b>	P2 pseudorange in meters, or -20000000000 if not available
<b>Col8</b>	L2 carrier phase in cycles, or -20000000000 if not available
<b>Col9</b>	P1 C/N0 in dB-Hz, or -3276.8 if not available
<b>Col10</b>	P2 C/N0 in dB-Hz, or -3276.8 if not available
<b>Col11</b>	Receiver Channel
<b>Col12</b>	Lock time in seconds
<b>Col13</b>	L1 Doppler in Hz, or -214748.365 if not available
<b>Col14</b>	L2 Doppler in Hz, or -214748.365 if not available
<b>Col15</b>	CA Multipath correction in meters, or 0 if unknown or not applicable
<b>Col16</b>	P2 Multipath correction in meters, or 0 if unknown or not applicable

**Table 11-6:** sbf2asc (Short)MeasEpoch block

<b>Col1</b>	0
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	X in meters, or -20000000000 if not available
<b>Col4</b>	Y in meters, or -20000000000 if not available
<b>Col5</b>	Z in meters, or -20000000000 if not available
<b>Col6</b>	Vx in m/s, or -20000000000 if not available
<b>Col7</b>	Vy in m/s, or -20000000000 if not available
<b>Col8</b>	Vz in m/s, or -20000000000 if not available
<b>Col9</b>	RxCkBias in seconds, or -20000000000 if not available
<b>Col10</b>	RxClockDrift in seconds/seconds, or -20000000000 if not available
<b>Col11</b>	NbrSV
<b>Col12</b>	PVT Mode field
<b>Col13</b>	MeanCorrAge in 1/100 seconds, or 65535 if not available
<b>Col14</b>	PVT Error
<b>Col15</b>	COG

**Table 11-7:** sbf2asc PVTCartesian block

<b>Col1</b>	-1
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	Latitude in radians, or -20000000000 if not available
<b>Col4</b>	Longitude in radians, or -20000000000 if not available
<b>Col5</b>	Ellipsoidal height in meters, or -20000000000 if not available
<b>Col6</b>	Geodetic Ondulation, or -20000000000 if not available
<b>Col7</b>	Vn in m/s, or -20000000000 if not available
<b>Col8</b>	Ve in m/s, or -20000000000 if not available
<b>Col9</b>	Vu in m/s, or -20000000000 if not available
<b>Col10</b>	Clock bias in seconds, or -20000000000 if not available
<b>Col11</b>	Clock drift in seconds/seconds, or -20000000000 if not available
<b>Col12</b>	NbrSV
<b>Col13</b>	PVT Mode field
<b>Col14</b>	MeanCorrAge in 1/100 seconds, or 65535 if not available
<b>Col15</b>	PVT Error
<b>Col16</b>	COG

**Table 11-8:** sbf2asc PVTGeodetic block

<b>Col1</b>	-2
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	Covariance xx
<b>Col4</b>	Covariance yy
<b>Col5</b>	Covariance zz
<b>Col6</b>	Covariance tt

**Table 11-9:** sbf2asc PVTCov block

<b>Col1</b>	-3
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	PDOP value, or NA if PDOP not available
<b>Col4</b>	TDOP value, or NA if TDOP not available
<b>Col5</b>	HDOP value, or NA if HDOP not available
<b>Col6</b>	VDOP value, or NA if VDOP not available
<b>Col7</b>	HPL value in meters, or NA if not available
<b>Col8</b>	VPL value in meters, or NA if not available
<b>Col9</b>	NbrSV

**Table 11-10:** sbf2asc PVTDOP block

<b>Col1</b>	-4
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	Heading in degree
<b>Col4</b>	Pitch in degree
<b>Col5</b>	Roll in degree
<b>Col6</b>	Error flag for attitude solution
<b>Col7</b>	Mode used to compute attitude solution
<b>Col8</b>	NbrSV

**Table 11-11:** sbf2asc AttitudeEuler block

<b>Col1</b>	-5
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	Covariance HeadingHeading
<b>Col4</b>	Covariance PitchPitch
<b>Col5</b>	Covariance RollRoll
<b>Col6</b>	Error flag for attitude solution

**Table 11-12:** sbf2asc AttitudeCovEuler block

<b>Col1</b>	-6
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	Source (1 = GPIN1, 2 = GPIN2)
<b>Col4</b>	Counter used to indicate the number of events that have occurred from the source (Col3)

**Table 11-13:** sbf2asc ExtEvent block

<b>Col1</b>	-7
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	CPU-Load in percentage
<b>Col4</b>	Uptime in seconds
<b>Col5</b>	RxStatus field (HEX)

**Table 11-14:** sbf2asc ReceiverStatus block

<b>Col1</b>	-8
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	Base Station ID
<b>Col4</b>	Base type
<b>Col5</b>	Source
<b>Col6</b>	X_L1 Phase center
<b>Col7</b>	Y_L1 Phase center
<b>Col8</b>	Z_L1 Phase center

**Table 11-15:** sbf2asc BaseStation block

<b>Col1</b>	-9
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	Base Station ID
<b>Col4</b>	East
<b>Col5</b>	North
<b>Col6</b>	Up

**Table 11-16:** sbf2asc BaseLine block

<b>Col1</b>	-10
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	Number of Bytes Received
<b>Col4</b>	Number of Bytes Accepted
<b>Col5</b>	Number of Messages Received
<b>Col6</b>	Number of Messages Accepted
<b>Col7</b>	Age of last message

**Table 11-17:** sbf2asc BaseLink block

<b>Col1</b>	-11
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	PRN
<b>Col4</b>	Eccentricity
<b>Col5</b>	Almanac reference time of week
<b>Col6</b>	Inclination angle at reference time, relative to $i_0 = 3$ semi-circles
<b>Col7</b>	Rate of right ascension
<b>Col8</b>	Square root of the semi-major axis
<b>Col9</b>	Longitude of ascending node of orbit plane at weekly epoch
<b>Col10</b>	Argument of perigee
<b>Col11</b>	SV Clock Drift
<b>Col12</b>	SC Clock Bias
<b>Col13</b>	PVT Mode field
<b>Col14</b>	Almanac reference week, to which $t_{oa}$ is referenced
<b>Col15</b>	Health on 8 bits from the almanac page
<b>Col16</b>	Health summary on 6 bits

**Table 11-18:** sbf2asc GPSAlm block



<b>Col1</b>	-12
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	Antenna ID
<b>Col4</b>	Delta East
<b>Col5</b>	Delta North
<b>Col6</b>	Delta Up
<b>Col7</b>	Number of Satellites
<b>Col8</b>	Error
<b>Col9</b>	Ambiguity Type

**Table 11-19:** sbf2asc AuxPos block

<b>Col1</b>	-13
<b>Col2</b>	time (GPS second since Jan 06, 1980)
<b>Col3</b>	SensorID
<b>Col4</b>	Type
<b>Col5</b>	X
<b>Col6</b>	Y
<b>Col7</b>	Z

**Table 11-20:** sbf2asc ExtSensorMeas block

## 11.4 sbf2cmd

The PPSDK **sbf2cmd** tool is a Windows Console Application. It converts all commands found in an SBF file into plain text format.

Invoking **sbf2cmd** without argument prints the list of options and their usage. The possible options for **sbf2cmd** are given in the table below:

Argument	Value	Description
-f	input file	(Mandatory)Input SBF File
-o	output file	Name of the output ASCII file (if not provided,'commands.txt' is used)
-m	mib file	Name of the ASN.1 file containing the MIB description . The MIB can be downloaded from the receiver .
-b	start epoch	Time of first epoch to insert in the output file. Format:yyyy-mm-dd_hh:mm:ss or hh:mm:ss
-e	end epoch	Time of last epoch to insert in the output file. Format:yyyy-mm-dd_hh:mm:ss or hh:mm:ss
-i	Interval	Decimation interval in seconds
-E		Exclude blocks where time stamp is invalid
-v		Verbose mode, progress displayed
-V		Program version

**Table 11-21:** sbf2cmd Arguments

An example of sbf2cmd output is given.

```
1475,210460.14,exeSBFOnce,
1475,210460.14,exeSBFOnce, ,GPSNav+GEONav+ReceiverSetup+Commands+Comment
1475,210483.82,setSBFOutput, Res1,
1475,210486.22,setSBFOutput,Res1,,MeasEpoch+MeasExtra+Comment
```

## 11.5 sbf2gpx

The PPSDK **sbf2gpx** tool is a Windows Console Application that is used to convert SBF files to GPX format. GPS eXchange Format is used to exchange GPS data between software applications and devices as an XML schema.

Invoking **sbf2gpx** without argument prints the list of options and their usage. The possible options for **sbf2gpx** are given in the table below:

Argument	Value	Description
-f	input file	(Mandatory)Input SBF File
-o	output file	name of the GPX file. (if not provided, SBF file name is used plus gpx extension)
-x		convert to standard GPX format (default)
-a		make use of PVTGeodetic and PVTCartesian blocks
-g		make use of PVTGeodetic blocks
-c		make use of PVTCartesian blocks
-m		include Waypoints on change of PVT mode
-r		include Waypoints on PVT error
-l		print the detection of PVT blocks(g=geodetic, c=cartesian)
-b	start epoch	Time of first epoch to insert in the output file. Format: yyyy-mm-dd_hh:mm:ss or hh:mm:ss
-e	end epoch	Time of last epoch to insert in the output file. Format: yyyy-mm-dd_hh:mm:ss or hh:mm:ss
-i	Interval	Decimation interval in seconds
-E		Exclude blocks where time stamp is invalid
-v		Verbose mode, progress displayed
-V		Program version

**Table 11-22:** sbf2gpx Arguments

## 11.6 sbf2kml

The PPSDK **sbf2kml** tool is a Windows Console Application that converts SBF files to KML 2.0 format.

Keyhole Markup Language is an XML-based notation for detailing geographic annotation and visualization on Web-based maps and three-dimensional Earth browsers. KML was originally developed for use with Google Earth.

Invoking **sbf2kml** without argument prints the list of options and their usage. The possible options for **sbf2kml** are given in the table below:

Argument	Value	Description
-f	input file	(Mandatory)Input SBF File
-o	output file	name of the KML file. (if not provided, SBF file name is used plus kml extension)
-k		Convert to KML format (DEFAULT)
-t		Select the mode for the Track Mode: (DEFAULT) P = PVT Track using PVTGeodetic and PVTCartesian blocks g = PVT Track using PVTGeodetic blocks c = PVT Track using PVTCartesian blocks l = PVT Track using IntPVTGeodetic and IntPVTCartesian blocks j = PVT Track using IntPVTGeodetic r = PVT Track using IntPVTCartesian a = Attitude Track with Attitude mode using AttEuler blocks t = Attitude Track with Attitude mode using IntAttEuler blocks s = Satellite Survey with PVT-Tracking using Channel Status blocks (use with -n option)
-m		Include different colored PVT Tracks on change of PVT/Attitude mode
-c		RGB (Red, Blue, Green) Color values expressed in hexadecimal notation to be used for the PVT/Attitude Tracks in case where -m option is not used. The order of expression is rrggbb. e.g. -c FF0000 which would give a red color
-u		Include Waypoints on the change of PVT, Attitude or Satellite-Survey modes.
-r		Include Waypoints on PVT/Attitude/Satellite-Survey error
-A	NoOfEpochs	Include Attitude model on Attitude solution every NoOfEpochs epochs
-x	External Events	Include External Events (on PVT, Attitude or Satellite Survey Tracks): 0 = do not include External Events (DEFAULT) 1 = add Waypoint in Event 2 = add 3D-Model in Event using Att info (Attitude tracks) 3 = add Waypoint and Model using Att info (Attitude tracks) Note that only up to 5000 events can be output

Argument	Value	Description
-d		Show Baseline between Rover and Reference Stations. This requires the presence of BaseStation blocks.
-p		Include satellite tracks on sky: 0 = Do not add sat. tracks (DEFAULT) 1 = Show only sat. tracks on sky 2 = Show sat. tracks connected to position on earth 3 = add Waypoint and Model using Att info (Attitude tracks)
-n		Satellite to be included in the Satellite tracks or in the Satellite Survey tracks. (DEFAULT) a = All satellites are shown g = Only GPS satellites r = Only GLONASS satellites e = Only GALILEO satellites s = Only SBAS satellites PRN = The numeric value of the specific satellite to be shown
-h		Use any of the following Altitude modes on KML output: 1 = clampToGround 2 = relativeToGround (DEFAULT) 3 = absolute
-w	LineWidth	Width of the line track (from 0.0 to 4.0) DEFAULT=1.0
-s	Model Scale	Scale of 3D model in 3D ExtEvent (from 1 to 10) DEFAULT=1
-b	start epoch	Time of first epoch to insert in the output file. Format: yyyy-mm-dd_hh:mm:ss or hh:mm:ss
-e	end epoch	Time of last epoch to insert in the output file. Format: yyyy-mm-dd_hh:mm:ss or hh:mm:ss
-i	Interval	Decimation interval in seconds
-E		Exclude blocks where time stamp is invalid
-l		Print detection of blocks(g=geodetic, c=cartesian, t=attitude)
-v		Verbose mode, progress displayed
-V		Program version

**Table 11-24:** sbf2kml Arguments

## 11.7 sbf2rin

The PPSDK installation contains the **sbf2rin** utility software. **sbf2rin** converts a binary SBF file to the widely used RINEX ASCII format. RINEX v2.10, v2.11 and v3.02 are supported.

The following RINEX file types can be generated:

- Observation file (extension '.yyO');
- GPS navigation file (extension '.yyN');
- GLONASS navigation file (extension '.yyG');
- Galileo navigation file (extension '.yyL');
- SBAS navigation file (extension '.yyH');
- BeiDou navigation file (non-standard extension '.yyI');
- SBAS broadcast data (extension '.yyB');
- Meteo file (extension '.yyM').

In order to generate a RINEX file, the following procedure is recommended:

1. Use the **setAntennaOffset**, **setMarkerParameters** and **setObserverParameters** commands to specify the contents of the **ReceiverSetup** SBF block. The contents of this block is used to generate the RINEX observation header.

The receiver has to be instructed to output the SBF blocks needed for the generation of the RINEX file. The necessary SBF blocks depend on the type of RINEX file:

file type	Mandatory and optional SBF blocks
Observation 'O'	MeasEpoch (mandatory) PVTCartesian or PVTGeodetic (optional: if not available, the "APPROX POSITION XYZ" line will be absent from the RINEX header) ReceiverSetup (optional: if not available, a default header will be generated, with most fields replaced by "unknown") Comment (optional: if available, user comments can be inserted in the RINEX file).
GPS Navigation 'N'	GPSNav (mandatory) GPSIon (optional: needed only if the header should contain the alpha and beta Klobuchar parameters) GPSUtc (optional: needed only if the header should contain UTC related data).
GLO Navigation 'G'	GLONav (mandatory) GPSUtc or GALUtc (mandatory : without at least one GPSUtc or GALUtc block in the file, <b>sbf2rin</b> is unable to generate a GLONASS navigation file).
Galileo Navigation 'L'	GALNav (mandatory) GALIon (optional) GALUtc (optional)
SBAS Navigation 'H'	GEONav (mandatory)
BeiDou Navigation 'I'	CMPNav (mandatory)
SBAS Broadcast 'B'	GEORawL1 (mandatory)
Meteo file 'M'	ASCIIIn (mandatory)

2. Use RxControl or any suitable communication program to log the raw bytes coming from the receiver. Make sure that no character translation is applied by your logging program. Let's call the log file **LOG.SBF**. It is possible that **LOG.SBF** does not only contain SBF blocks, since the receiver may output other data in between two SBF blocks (replies to user commands, NMEA sentences). This is not a problem: the SBF header allows identifying the SBF blocks in the raw stream from the receiver.
3. The command below generates a RINEX v2.11 observation file (default) from the file **LOG.SBF**:  
**sbf2rin -f LOG.SBF <CR>**  
Note that the size of the SBF file must not exceed 2GBytes.

Invoking **sbf2rin** without argument prints the list of options and their usage:

```
sbf2rin -f input_file [-o output_file] [-i interval]
                    [-b startepoch] [-e endepoch] [-n type] [-MET] [-s] [-D]
                    [-v] [-R3] [-R210] [-x systems] [-a antenna] [-V]
```

-f input\_file (mandatory) Name of the SBF file.  
 -o output\_file Name of the RINEX file.  
 If not provided, the RINEX convention is applied (ssssdddf.yyt). With the "-o copy" option, the name of the RINEX file is a copy of the name of the SBF file, with the last character being set to O, N, G or L according to the RINEX file type.  
 With the "-o copybase" option, the name of the RINEX file is a copy of the name of the SBF file, with the last 3 characters being set to yyt (2-digit year and type) according to the RINEX convention.  
 -R3 Generate a RINEX version 3.02 file instead of version 2.11.  
 -R210 Generate a RINEX version 2.10 file instead of version 2.11.  
 -i interval Interval in the RINEX obs and meteo file, in seconds (by default, the interval is the same as in the SBF file).  
 -b startepoch Time of first epoch to insert in the RINEX file.  
 Format: yyyy-mm-dd\_hh:mm:ss or hh:mm:ss.  
 -e endepoch Last epoch to insert in the RINEX file  
 Format: yyyy-mm-dd\_hh:mm:ss or hh:mm:ss.  
 -s Add the Sx obs types for the SNRs in dB-Hz.  
 -c Allow comments in the RINEX file (from the Comment block) (only applicable for RINEX v2.11 and v2.10)  
 -C commentstr Add the specified comment string to the RINEX obs header. The comment string must not be longer than 240 characters. Enclose the string between quotes if it contains whitespaces.  
 -D Add the Dx obs types for the Doppler in Hz.  
 -x systems Exclude one or more satellite systems from the obs file. systems may be G (GPS), R (Glonass), E (Galileo), S (SBAS), C (Compass/BeiDou), J (QZSS) or any combination thereof. For instance, -xERSC produces a GPS-only observation file.  
 -n type Generate a RINEX navigation file (default is observation). type may be N for GPS, G for GLONASS, E for Galileo (v3 only), H for GEO, I for Compass/BeiDou (v3 only) or B for broadcast SBAS.  
 -MET Generate a RINEX meteo file.  
 -a antenna Convert data from the specified antenna (antenna is 1, 2 or 3). The default is 1, corresponding to the main antenna.  
 -ma Insert a "start moving" event right after the header if the RINEX file contains kinematic data.  
 -mf Force inserting a "start moving" event right after the header.  
 -S Automatically increase the file sequence character in the output file name. This is useful when converting several SBF files collected on the same day and on the same marker. For each file to be converted, first call sbf2rin to make the .O file, then call it again with the option -nN (if needed), then again with the option -nG (if needed), then with the option -nE, and finally with the option -nI. When the .O, .N, .G, .L and .I files are ready from the first SBF file, repeat the same sequence for the second SBF file to be converted, and so forth.  
 The "-S" option has no effect if the "-o" option is used.  
 -v Run in verbose mode.  
 -V Display the sbf2rin version.

## 11.8 sbf2sbf

The PPSDK **sbf2sbf** tool is a Windows Console Application that used to preprocess an SBF file. Blocks may be included or excluded in a number of ways. The file may be cropped and invalid blocks discarded. This application is different from the others in that the options (except the input file option) may be entered multiple times and in any order. The input file will then be processed in the order of the arguments.

Invoking **sbf2sbf** without argument prints the list of options and their usage. The parameters taken by **sbf2sbf** are given in the table below:



Argument	Value	Description
-f	input file	(Mandatory)Input SBF File (Stream 1)
-d		Directory to store temporary files
-l		Insert 'END OF' SBF blocks
-C	Crop Option	Crop option: 0 None (DEFAULT) 1 Discard invalid SBF data 2 Discard navigation applicability
-Q	crop times	Crop the SBF stream. Expected format is 'S:E' (GNSS time). Always provide the ':' character. When cropping from the start, use: ':E'; when cropping to the end use 'S:' (to be used before -o option)
-r		Use relative sample time (to be used before -s option)
-s	sampletime	Sample the SBF stream at a given interval (ms)(to be used before -o option)
-M		Merge option for Stream 1 (to be used before -m option) (categories like in SBF ICD): 0 Include all blocks (DEFAULT) 1 Discard navigation applicability (all nav blocks will be included) 2 Include all measurement blocks 3 Include all navigation pages 4 Include all GPS decoded messages 5 Include all GLONASS decoded messages 6 Include all Galileo decoded messages 7 Include all SBAS decoded messages 8 Include all PVT blocks 9 Include all attitude blocks 10 Include all receiver time blocks 11 Include all external event blocks 12 Include all differentiation corrections blocks 13 Include all status blocks 14 Include all miscellaneous blocks 15 Include all external sensor measurement blocks 16 Include all integrated PVT blocks 17 Include all TUR blocks Note that if a merge option is used then none of the default options are kept. Therefore you will need to issue them as well. E.g. -M 3 -M 1
-N	mergeoption	Merge option for Stream 2 (to be used before -m option) (categories as in SBF ICD): The same option values as the -M option are used!

Argument	Value	Description
-m		SBF file (Stream 2) to merge into Input SBF file (Stream 1) (to be used before -o option)
-e		ASCII file with commands and timestamps to be inserted (to be used before -o option)
-R	blockID	Remove blocks with ID (to be used before -o option). You can also pass multiple blocks comma separated: e.g. -R 5891,5893,5894,4004,4026 (spaces in between are not accepted)
-F	blockID	Remove all blocks except with ID (to be used before the -o option). This option should be called once per SBF stream. As such you cannot pass multiple blocks comma separated (since the option will immediately filter the stream to one type of blocks)
-o	output file	Output SBF file (will output the current SBF stream state depending on the order of occurrence passed in the command line)
-b		Check block validity on the Input file and report it on the command console.
-c	blockID	Check missing epochs of block type on the Input file and report it on a text file named 'infile'.missing.txt
-h		Show help
-V		Program version

**Table 11-26:** sbf2sbf Arguments

Note: The 'sbf2sbf' console application processes the options in the given order and any option (except '-f') can be called multiple times in any given order. Note, however, that the options will be processed in the given order (including the -o option). Make sure that the merge option or other options are called before.

Example 1: 'sbf2sbf -f test.sbf -R 4027 -o out\_1.sbf -l -o out\_2.sbf'

First, the SBF file 'test.sbf' will be loaded and prepared. Next, all blocks with ID '4027' will be removed from the stream and the result will be stored in the SBF file 'out\_1.sbf'.

Afterwards the '-l' option will insert all the 'END OF' SBF blocks into the SBF stream and the result will be stored in the 'out\_2.sbf' SBF file.

Example 2: 'sbf2sbf -f file1.sbf -C -Q 894021979:894022015 -M 1 -m file2.sbf -o out\_1.sbf'

First, the SBF file 'file1.sbf' will be loaded and prepared. Next, the option to discard all Invalid SBF data from 'file1.sbf' file is used so while cropping between the GNSS time 894021979 and 894022015. Afterwards the '-M' option will be used in order to Discard navigation applicability while doing the merging with file file2.sbf. Finally the merged file will be output into file 'out\_2.sbf'. Note how the -C is passed before -Q and -M is passed before -m; this is needed so that the right options are used before the respective crop and merge actions.

## 11.9 sbf2stf

The PPSDK **sbf2stf** tool is a Windows Console Application that displays the contents of an SBF file in a proprietary Septentrio Text format. A separate text file is created for each SBF block type.

Note that no future releases to **sbf2stf** will be made and users are recommended to implement **bin2asc** where possible.

Invoking **sbf2stf** without argument prints the list of options and their usage. The possible options for **sbf2stf** are given in the table below:

Argument	Value	Description
-f	input file	(Mandatory)Input SBF File
-p	output path	Output directory, default is the same as input
-m	msg1,msg2,...	Messages (Block Names) to be decoded. If not provided, all messages will be converted to STF files
-l		List all supported messages
-d	delimiter	Field delimiter, default is comma
-n	donotuse	Value for donotuse fields, default is empty
-x		Show headers in each one of the output files
-t		Show title columns for each of the output files
-b	start epoch	Time of first epoch to insert in the output file. Format: yyyy-mm-dd_hh:mm:ss or hh:mm:ss
-e	end epoch	Time of last epoch to insert in the output file. Format: yyyy-mm-dd_hh:mm:ss or hh:mm:ss
-i	Interval	Decimation interval in seconds
-E		Exclude blocks where time stamp is invalid
-v		Verbose mode, progress displayed
-V		Program version

**Table 11-27:** sbf2stf Arguments

## 11.10 sbfblocks

The PPSDK **sbfblocks** tool is a Windows Console Application that lists the individual SBF blocks in a file along with their time stamp.

Invoking **sbfblocks** without argument prints the list of options and their usage. The possible options for **sbfblocks** are given in the table below:

Argument	Value	Description
-f	input file	(Mandatory)Input SBF File
-o	output file	name of the text file with block info (if not provided, SBF file name is used plus .blocks.txt extension)
-l	detail	Show blocks over time instead of only Summary of results (tab separated). S: Show only Summary of blocks (DEFAULT). T: Show only Description over time of blocks. B: Show both Description over time of blocks and Summary
-h	hide details	When Description over time of blocks is enabled, extra decoding of blocks such as DiffCorr and Comment is hidden. (By default details are printed out)
-b	start epoch	Time of first epoch to insert in the output file. Format: yyyy-mm-dd_hh:mm:ss or hh:mm:ss
-e	end epoch	Time of last epoch to insert in the output file. Format: yyyy-mm-dd_hh:mm:ss or hh:mm:ss
-i	Interval	Decimation interval in seconds
-E		Exclude blocks where time stamp is invalid
-v		Verbose mode, progress displayed
-V		Program version

**Table 11-28:** sbfblocks Arguments

## 11.11 ngs2bin

For high-precision positioning, the GNSS measurements need to be corrected in such a way that they all refer to a common and stable point in space. That point is referred to as the Antenna Reference Point (ARP). For convenience, it is usually selected at the center of the bottom surface of the antenna. The National Geodetic Survey has calibrated the offset from the PC to the ARP as a function of the elevation and of the frequency band for a large number of geodetic-grade antennas. NGS publishes calibration tables that can be downloaded from the following URL:

<http://www.ngs.noaa.gov/ANTCAL/index.shtml>

The antenna naming convention in such table is the one adopted by the IGS Central Bureau.

The PPSDK **ngs2bin** tool is a Windows Console Application that converts NGS Antenna definition files into a binary file which can be used by the PPSDK.

Invoking **ngs2bin** without argument prints the list of options and their usage. The parameters taken by **ngs2bin** are given in the table below:

Argument	Value	Description
	input file	(Mandatory) Input NGS format file
	output file	(Mandatory) name of the Binary file to be generated
-V		Program version

**Table 11-29:** ngs2bin Arguments

## 12 RTCM Overview

The following tables provide a short overview of selected RTCM messages. For a full description of these messages, please refer to the respective standard.

The PPSDK does not reproduce all the messages given below when creating RTCM messages from a reference SBF file, however it supports all these messages should they come from an input SBF file within the `DiffCorrIn` SBF block (possibly recorded from a Septentrio receiver). Blocks `PVTCartesian` or `PVTGeodetic` is required for all messages. Navigation data from the used constellations is also needed for the corresponding messages. CMR messages are only used by the PPSDK should they come from a real time receiver (the insertion of a reference does not allow the generation of CMR messages).

CMR Message	Message Name
0	Observables
1	Reference Station Coordinates
2	Reference Station Description
3	GLONASS Observables

When performing an RTCM2 insertion with a reference file then only the following messages are supported (1, 3, 18, 19, 22, 23 and 24). Messages 23 and 24 are created using information of the `ReceiverSetup` present in the reference SBF file.

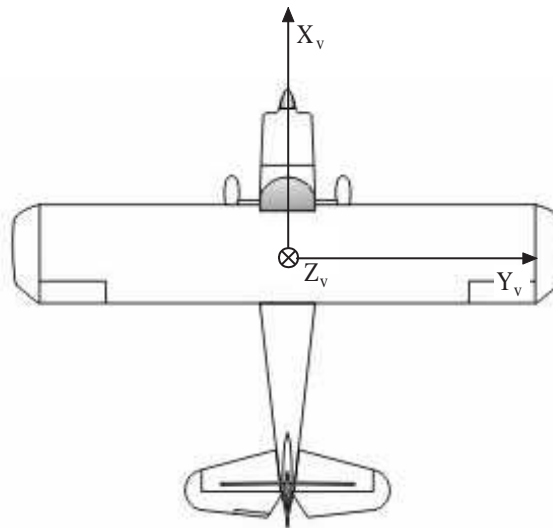
When performing an RTCM3 insertion with a reference file then only the following messages are supported (1003, 1004, 1005, 1006, 1007, 1009, 1010, 1011, 1012, 1014, 1015, 1016, 1021, 1022 and 1023).

For RTCM3 insertions the PPSDK requires the reference SBF stream to contain `MeasEpoch` and `PVTCartesian` or `PVTGeodetic`. The `MeasEpoch` block must contain data from the appropriate constellations. For example message 1012 requires data from the GLONASS satellites. Message 1007 is created using information of the `ReceiverSetup` present in the reference SBF file.

RTCM 2.x Message	Message Name
1	Differential GPS Corrections
3	GPS Reference Station Parameters
9	GPS Partial Correction Set
16	GPS Special Message
18	RTK Uncorrected Carrier Phases
19	RTK Uncorrected Pseudoranges
20	RTK Carrier Phase Corrections
21	RTK/Hi-Accuracy Pseudorange Corrections
22	Extended Reference Station Parameters
23	Antenne Type Definition Record
24	Antenna Reference Point (ARP)
31	Differential GLONASS Corrections
32	GLONASS Reference Station Parameters
59	Proprietary Message

## A Attitude Angles

The attitude of the vehicle is defined as the angles between the vehicle reference frame and the local-level reference frame (defined by the East, North and Up directions). The vehicle reference frame is defined as follows. It is attached to the vehicle and has its X axis pointing along the longitudinal vehicle axis, the Y axis pointing towards the vehicle starboard (right) side and the Z axis pointing down, as illustrated in figure A-1.



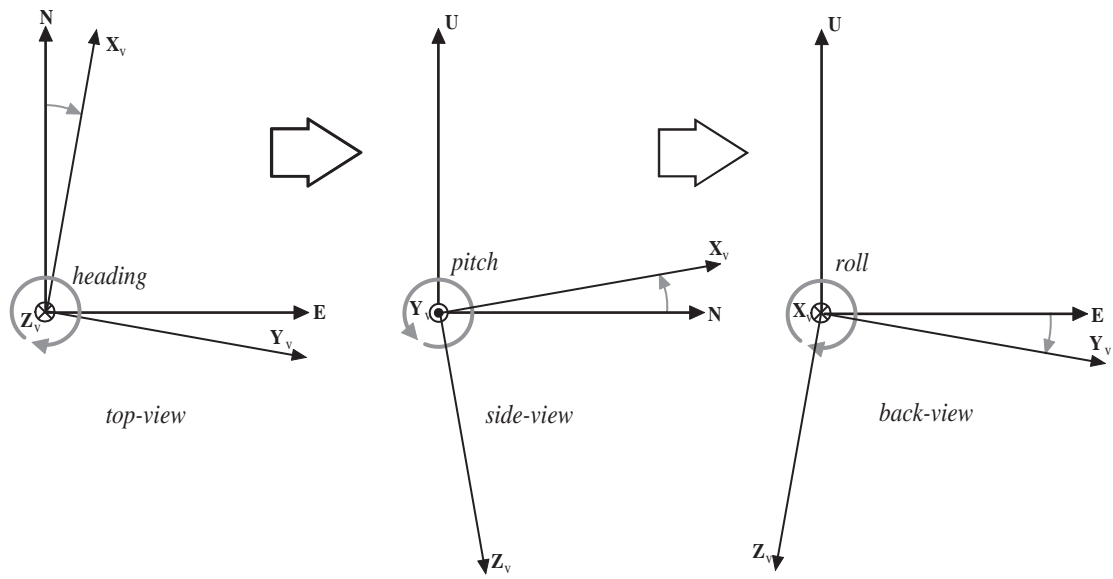
**Figure A-1:** Vehicle reference frame.

Septentrio receivers express the vehicle attitude in Euler angles using the heading-pitch-roll rotation sequence. More specifically, Euler angles are defined as successive rotations of the vehicle frame (X, Y, Z axes) relative to the local-level East-North-Up reference frame. The rotation sequence is shown in figure A-2. The heading ( $\psi$ ) of the vehicle is defined as the right-handed rotation of the vehicle about the Z axis ( $0^\circ \leq \psi \leq 360^\circ$ ). The pitch ( $\theta$ ) of the vehicle is defined as the right-handed rotation about the vehicle Y axis ( $-90^\circ \leq \theta \leq 90^\circ$ ). The roll ( $\phi$ ) of the vehicle is defined as the right-handed rotation about the vehicle X axis ( $-180^\circ \leq \phi \leq 180^\circ$ ).

RTCM 3.x Message	Message Name
1001	L1-Only GPS RTK Observables
1002	Extended L1-Only GPS RTK Observables
1003	L1&L2 GPS RTK Observables
1004	Extended L1&L2 GPS RTK Observables
1005	Stationary RTK Reference Station ARP
1006	Stationary RTK Reference Station ARP with Antenna Height
1007	Antenna Descriptor
1008	Antenna Descriptor and Serial Number
1009	L1-Only GLONASS RTK Observables
1010	Extended L1-Only GLONASS RTK Observables
1011	L1&L2 GLONASS RTK Observables
1012	Extended L1&L2 GLONASS RTK Observables
1013	System Parameters
1015	GPS Ionospheric Correction Differences
1016	GPS Geometric Correction Differences
1017	GPS Combined Geometric and Ionospheric Correction Differences
1021	Helmert / Abridged Molodenski Transformation Parameters
1022	Molodenski-Badekas Transformation Parameters
1023	Residuals, Ellipsoidal Grid Representation
1033	Receiver and Antenna Descriptors
1037	GLONASS Ionospheric Correction Differences
1038	GLONASS Geometric Correction Differences
1039	GLONASS Combined Geometric and Ionospheric Correction Differences

Starting from the situation where X points to the North, Y to the East and Z down, the following successive rotations define the attitude of the vehicle. Note that the order of the rotations is important.

1. Rotate through angle  $\psi$  about Z axis;
2. Rotate through angle  $\theta$  about new Y axis;
3. Rotate through angle  $\phi$  about new X axis;



**Figure A-2:** Euler angle sequence.



## Glossary

### - A -

- API** An **A**pplication **P**rogram **I**nterface is a set of routines, protocols and tools for building software applications.
- ARP** **A**ntenna **R**eference **P**oint.
- ASCII** The **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange is a standard seven-bit code. ASCII was established to achieve compatibility between various types of data processing equipment. The standard ASCII character set consists of 128 decimal numbers ranging from 0...127 assigned to letters, numbers, punctuation marks, and the most common special characters. The Extended ASCII Character Set also consists of another 128 decimal numbers and ranges from 128...255 representing additional special, mathematical, graphic, and foreign characters.
- ASN.1** **A**bstract **S**yntax **N**otation **O**ne is a standard way to describe a message (a unit of application data) that can be sent or received in a network. ASN.1 is divided into two parts: (1) the rules of syntax for describing the contents of a message in terms of data type and content sequence or structure and (2) how you actually encode each data item in a message.

### - B -

- BeiDou** The **BeiDou** navigation system is a global satellite navigation system being developed by China. The name **Compass** has been replaced by **BeiDou**.

### - C -

- C/A** The **C**oarse/**A**cquisition code is a pseudo random noise (PRN) binary code consisting of 1,023 elements, or chips, that repeats itself every millisecond. The term pseudo-random indicates that the code is apparently random although it has been generated by means of a known process, hence the repeatability.
- CMR** The **C**ompact **M**easurement **R**ecord format contains packet framing and message types for raw L1 and L2 carrier phase and pseudorange data, plus reference station location and description messages.
- COG** **C**ourse **O**ver **G**round.

### - D -

- DGPS** **D**ifferential **G**PS can be used when you have two receivers not too far from each other. The errors due to the satellite clock, the satellite orbit, the ionosphere, the troposphere and SA affect both receivers in the same way and with the same magnitude. If we know the exact location of one receiver, we could use that information to calculate errors in the measurement and then report these errors (or correction values) to the other receiver, so that it can compensate for them.

- DiffCorr** Differential **C**orrections from either a real time receiver or from an **SBF** file.
- DLL** A **DLL** is Microsoft's implementation of the shared library concept in the Microsoft Windows and OS/2 operating systems. These libraries usually have the file extension ".dll".
- DOP** The **D**ilution **O**f **P**recision measures the relative degradation of the accuracy of the navigation solution based on the constellation geometry. The reported value can be multiplied by the uncertainty in the range measurements (assumed to be the same for all transmitters) to provide the uncertainty in the navigation solution.

## - E -

- EGNOS** The **E**uropean **G**eostationary **N**avigation **O**verlay **S**ystem is the European SBAS system developed by ESA, European Commission and Eurocontrol. Its service zone is the European continental airspace.

## - G -

- Galileo** The **Galileo positioning system**, referred to simply as **Galileo**, is a European Global Navigation Satellite System, built by the European Satellite Navigation Industries for the European Union (EU) and European Space Agency (ESA) as an alternative to the United States operated Global Positioning System (GPS) and the Russian GLONASS. Galileo is tasked with multiple objectives including the following: to provide a higher precision to all users than is currently available through GPS or GLONASS, to improve availability of positioning services at higher latitudes, and to provide an independent positioning system upon which European nations can rely even in times of war or political disagreement.
- GEO** Geostationary **E**arth **O**rbiter.
- GLONASS** The Russian **G**lobal **O**rbiting **N**avigation **S**atellite **S**ystem is a satellite based radionavigation system which enables unlimited number of users to make all-weather 3D positioning, velocity measuring and timing anywhere in the world or near-Earth space.
- GNSS** A **G**lobal **N**avigation **S**atellite **S**ystem is a system of satellites that provides autonomous geo-spatial positioning with global coverage.
- GPS** **G**lobal **P**ositioning **S**ystem (also NAVSTAR GPS) is a satellite navigation system owned by the Department of Defence of the United States of America and designed to provide instantaneous position, velocity and time information almost anywhere on the globe at any time, and in any weather. NAVSTAR GPS stands for the NAVigation Satellite Timing And Ranging Global Positioning System.
- GPX** **G**PS **e**Xchange Format is an XML schema designed for transferring GPS data between software applications. It can be used to describe waypoints, tracks, and routes.
- GUI** The **G**raphical **U**ser **I**nterface gives the user a graphical way for controlling and viewing the information of the receiver.

## - H -

<b>HDOP</b>	<b>H</b> orizontal <b>D</b> ilution <b>O</b> f <b>P</b> recision is a measure of the uncertainty of the navigation solution in the local horizontal plane.
<b>HERL</b>	The <b>H</b> orizontal <b>E</b> xternal <b>R</b> eliability <b>L</b> evel for the position used in <b>RAIM</b> statistics.
<b>HPL</b>	<b>H</b> orizontal <b>P</b> rotection <b>L</b> evel.

## - I -

<b>IGS</b>	The <b>I</b> nternational <b>G</b> PS <b>S</b> ervice provides GPS orbits, tracking data, and other high-quality GPS data and data products on line in near real time to meet the objectives of a wide range of scientific and engineering applications and studies.
<b>INS</b>	<b>I</b> nertial <b>N</b> avigation <b>S</b> ystem.

## - K -

<b>KML</b>	<b>K</b> ML is a file format used to display geographic data in an earth browser, such as Google Earth, Google Maps, and Google Maps for mobile. A <b>K</b> ML file is processed in much the same way that <b>HTML</b> (and <b>XML</b> ) files are processed by web browsers. Like <b>HTML</b> , <b>K</b> ML has a tag-based structure with names and attributes used for specific display purposes. Thus, <b>Google Earth</b> and <b>Maps</b> act as browsers for <b>K</b> ML files.
------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## - L -

<b>L1</b>	GPS satellites transmit two L-band signals which can be used for positioning purposes. The signals, which are generated from a standard frequency of 10.23 MHz, are L1 at 1575.42 MHz and L2 at 1227.60 MHz and are often called the carriers. The frequencies are generated from the fundamental satellite clock frequency of $f_0 = 10.23$ MHz.
<b>L2</b>	GPS satellites transmit two L-band signals which can be used for positioning purposes. The signals, which are generated from a standard frequency of 10.23 MHz, are L1 at 1575.42 MHz and L2 at 1227.60 MHz and are often called the carriers. The frequencies are generated from the fundamental satellite clock frequency of $f_0 = 10.23$ MHz.
<b>L2C</b>	The <b>L2</b> <b>C</b> ivil signal to be transmitted by modernized IIR (IIR-M) and subsequent GPS satellites.

## - M -

<b>MDB</b>	<b>M</b> inimum <b>D</b> etectable <b>B</b> ias based on probability of missed detection set by the user.
------------	-----------------------------------------------------------------------------------------------------------

**MIB** a **MIB** is a type of database used to manage the devices in a communications network. The **MIB** contains information on the commands and on the target's objects (controllable entities or potential sources of status information).

## – N –

**NGS** National **G**eodetic **S**urvey.

**NMEA** The **N**ational **M**arine **E**lectronics **A**ssociation has developed a standard to permit ready and satisfactory data communication between electronic marine instruments, navigation equipment and communications equipment when interconnected via an appropriate interface. The standard implemented by the Please add \renewcommand{\receiver}{xxx} to your master tex file is the NMEA 0183, version 2.30.

## – P –

**PDOP** Position **D**ilution **O**f **P**recision is the geometric DOP parameter.

**PPP** Precise **P**oint **P**ositioning provides high accuracy positioning without the need for a local base station. PPP uses precise satellite orbit and clock corrections computed by a global network of reference stations and broadcast in real time by geostationary satellites in the L band.

**PRN** The **P**seudo **R**andom **N**oise refers to a code that is is apparently random although it has been generated by means of a known process, hence the repeatability of the code indicate by the prefix *pseudo random*. Each GNSS satellite has its PRN number.

**PVT** Position, **V**elocity and **T**ime, meaning that the navigation solution computes the current position, velocity and time clock bias of the receiver.

## – R –

**RAIM** The **R**eceiver **A**utonomous **I**ntegrity **M**onitoring is a technology developed to assess the integrity of GPS signals in a GPS receiver system. It is of special importance in safety-critical GPS applications, such as in aviation or marine applications. **RAIM** ensures the integrity of the computed position solution, provided that sufficient satellites are available. The **RAIM** algorithm consists in three steps: detection, identification and adaptation, or shortly "**D-I-A**".

**RGB** Red **G**reen **B**lue color model.

**RINEX** The **R**eceiver **I**Ndependent **E**Xchange format is data format independent of receiver type. RINEX can be seen as a standard exchange format for GPS data.

**RTCA** Radio **T**echnical **C**ommission for **A**eronautics.

**RTCM** Radio **T**echnical **C**ommission for **M**aritime **S**ervices. The committee NO. 104 of the RTCM recommended a standard for exchange of data for Differential GPS service. The standard addresses both code-based and carrier-phase based positioning.

**RTK** GPS **Real-Time Kinematic** is a high-precision surveying method. RTK is based on differential carrier-phase pruning with either float or integer phase ambiguities. RTK requires a real-time data link to transmit correction data from the base station to the rover.

## – S –

**SBAS** A **Space-Based Augmentation System** is a regional augmentation systems for GPS and/or GLONASS. An SBAS system is based on a networked ground segment and navigation payloads on-board of geostationary satellites whose main purpose is to provide higher position accuracies, better availability and continuity of service and integrity messages to the users of space based navigation systems. Currently existing SBASs are based on DO229 data exchange standard.

**SBF** The **Septentrio Binary Format** is a data format used by the Please add \renew-command{\receiver}{xxx} to your master tex file. It arranges the data in so-called SBF blocks, identified by block IDs. The benefit of SBF is compactness : large quantity of information with a high level of detail can be transmitted over a low-bandwidth serial connection. This format should be your first choice if you wish to receive detailed information from the receiver.

**SDK** A **Software Development Kit** is a software library designed to help develop applications and utilities. It includes one or more [short]APIs, sample applications and extensive documentation.

**SIS** **Signal In Space**.

**SNMP** **SNMP** is used by network management systems to monitor network-attached devices for conditions that warrant administrative attention. It consists of a set of standards for network management, including an Application Layer protocol, a database schema, and a set of data objects. **SNMP** exposes management data in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried and sometimes set by managing applications.

**STF** **Septentrio Text Format** where similar SBF blocks are grouped per file.

**SVID** **Space Vehicle ID**.

## – T –

**TDOP** **Time Dilution Of Precision** is a measure of the uncertainty of the navigation solution in the time determination.

**TOW** GPS time is transmitted by a combination of the current Week Number and the **Time Of Week**. The TOW represents the number of seconds into the week ranging from [0 ... 604800[ seconds and is counted from midnight Saturday/Sunday on the GPS time scale.

**TUR** The **Test User Receiver** is developed by Septentrio in contract to ESA with the purpose to validate the Galileo system. It consists of a Main Receiver Unit (MRU), an antenna and a control PC to run the TUR Monitoring and Control software.

## - C -

**UTC** Coordinated **U**niversal **T**ime is a time scale that couples Greenwich Mean Time, which is based solely on the Earth's inconsistent rotation rate, with highly accurate atomic time. When atomic time and Earth time approach a one second difference, a leap second is calculated into UTC. UTC was devised on January 1st, 1972 and is coordinated in Paris by the International Bureau of Weights and Measures (BIPM). For most practical purposes associated with the Radio Regulations, UTC is equivalent to mean solar time at the prime meridian (0° longitude), formerly expressed in Greenwich Mean Time (GMT). The maintenance by BIPM includes cooperation among various national laboratories around the world. The full definition of UTC is contained in CCIR Recommendation 460-4. The GPS system time is different from the UTC time by a whole number of leap seconds (15 at the time of this writing).

## - V -

**VDOP** Vertical **D**ilution **O**f **P**recision is a measure of the uncertainty of the navigation solution in the vertical direction.

**VERL** The **V**ertical **E**xternal **R**eliability **L**evel for the position used in **RAIM** statistics.

**VPL** Vertical **P**rotection **L**evel.

**VRS** Virtual **R**eference **S**tation.

## - W -

**WAAS** The **W**ide **A**rea **A**ugmentation **S**ystem is the American SBAS system developed by the FAA. WAAS is designed to improve the accuracy and ensure the integrity of information coming from GPS satellites.

**WNc** GPS time is transmitted by a combination of the current **W**eek **N**umber and the Time Of Week. The week number represents the number of weeks elapsed since the introduction of the GPS time scale on January, 6<sup>th</sup> 1980.

## - X -

**XERL** The **EXt**ernal **R**eliability **L**evels give the opportunity to introduce a more stringent application-specific integrity criterion. The positional solution is deemed as passed an application-level integrity test if the **XERLs** are within user-defined (and application-dependent) alarm limits. This comparison (and the definition of alarm limits as well) takes place in a user application and is outside of the receiver scope.

## - E -

**XML** **Ex**tensible **M**arkup **L**anguage.