Command Line Interface Reference Guide

Applicable to SSRC5 2.3.2

Command Line Interface Reference Guide

Applicable to SSRC5 2.3.2

Publication date September 23, 2011

Septentrio Satellite Navigation

Greenhill Campus, Interleuvenlaan 15G
B-3001 Leuven, Belgium

http://www.septentrio.com/support_request.htm
<support@septentrio.com>
+32 16 300 800
+32 16 221 640

Table of Contents

References

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

[2] *NMEA 0183, Standard for Interfacing Marine Electronic Devices, Version 2.30*, National Marine Electronics Association 1998

1. Introduction

1.1. Scope

This document describes the ASCII command-line interface supported by your receiver. The receiver also accepts binary commands conforming to the Septentrio proprietary SNMP' protocol. Please refer to the SNMP' Technical Note for a description of the SNMP' protocol. The Command and Log Reference Card provides a synopsis of all commands.

1.2. Typographical Conventions

abc	Commands to be entered by the user;
`abc`	Replies from the receiver;
abc	Command argument name.

2. Command Line Interface Outline

The receiver outputs a prompt when it is ready to accept a user command. The prompt is of the form:

```
CD>
```

where `CD` is the connection descriptor of the current connection (`COMx`, `USBx` or `IP1x`). For instance, if a user is connected to `COM1`, the prompt will be:

```
COM1>
```

There are four types of commands:

`set`-commands to change one or more configuration parameters;
`get`-commands to get the current value of one or more configuration parameters;
`exe`-commands to initiate some action;
`lst`-commands to retrieve the contents of internal files or list the commands.

Each **`set`**-command has its **`get`**- counterpart, but the opposite is not true. For instance, the **`setCOMSettings`** command has a corresponding **`getCOMSettings`**, but **`getReceiverCapabilities`** has no **`set`**- counterpart. Each **`exe`**-command also has its **`get`**- counterpart which can be used to retrieve the parameters of the last invocation of the command.

The prompt indicates the termination of the processing of a given command. When sending multiple commands to the receiver, it is necessary to wait for the prompt between each command.

2.1. Command Line Syntax

Each ASCII command line consists of a command name optionally followed by a list of arguments and terminated by <CR>, <LF> or <CR><LF> character(s) usually corresponding to pressing the "Enter" key on the keyboard.

To minimize typing effort when sending commands by hand, the command name can be replaced by its 3- or 4-character mnemonic. For instance, **`grc`** can be used instead of **`getReceiverCapabilities`**.

The receiver is case insensitive when interpreting a command line.

The maximum length of any ASCII command line is 2000 characters.

For commands requiring arguments, the comma "," must be used to separate the arguments from each other and from the command's name. Any number of spaces can be inserted before and after the comma.

Each argument of a **`set`**-command corresponds to a single configuration parameter in the receiver. Usually, each of these configuration parameters can be set independently of the others, so most of the **`set`**-command's arguments are optional. Optional arguments can be omitted but if omitted arguments are followed by non-omitted ones, a corresponding number of commas must be entered. Omitted arguments always keep their current value.

2.2. Command Replies

The reply to ASCII commands always starts with "$R" and ends with <CR><LF> followed by the prompt corresponding to the connection descriptor you are connected to.

The following types of replies are defined for ASCII commands:

- For comment lines (user input beginning with "#") or empty commands (just pressing "Enter"), the receiver replies with the prompt.

```
COM1> # This is a comment! <CR>
COM1>
```

- For invalid commands, the reply is an error message, always beginning with the keyword "$R?" followed by an error message. The different error messages are listed in Appendix A.

- For all valid **set**-, **get**- and **exe**-commands, the first line of the reply is an exact copy of the command as entered by the user, preceded with "$R:". One or more additional lines are printed depending on the command. These lines report the configuration of the receiver after execution of the command.

```
COM1> setCOMSettings, all, baud115200 <CR>
$R: setCOMSettings, all, baud115200
  COMSettings, COM1, baud115200, bits8, No, bit1, none
  COMSettings, COM2, baud115200, bits8, No, bit1, none
  COMSettings, COM3, baud115200, bits8, No, bit1, none
COM1>
```

  For commands which reset or halt the receiver (**exeResetReceiver** and **exePowerMode**), the reply is terminated by "STOP>" instead of the standard prompt, to indicate that no further command can be entered.

- For all valid **lst**-commands, the first line of the reply is an exact copy of the command as entered by the user, preceded with "$R;". The second line is a pseudo-prompt "---->" and the remaining of the reply is a succession of formatted blocks, each of them starting with "$--BLOCK".

ASCII replies to **set**-, **get**- and **exe**-commands, including the terminating prompt, are atomic: they cannot be broken by other messages from the receivers. For the **lst**-commands, the replies may consist of several atomic formatted blocks which can be interleaved with other output data. If more than one formatted block is output for a **lst**-command, each of the intermediate blocks is terminated with a pseudo-prompt "---->". The normal prompt will only be used to terminate the last formatted block of the reply so that one single prompt is always associated with one command.

2.3. Command Syntax-tables

All ASCII commands are listed in Chapter 3, *Command List*. Each command is introduced by a compact formal description of it called a "syntax-table". Syntax tables contain a complete list of arguments with their possible values and default settings when applicable.

The conventions used in syntax-tables are explained below by taking a fictitious **setCommandName** command as example. The syntax-table for that command is:

scn gcn	**setCommandName** **getCommandName**	**Cd** *Cd*	*Distance*	*Time*	*Message(120)*	*Mode*	*PRN*
		+COM1 +COM2 all	-20.00 ... 0.00 ... 20.00 m	1 ... 50 sec	Unknown	off on	none +G01 ... G32 GPS +S120 ... S138 +SBAS all

The associated **set**- and **get**-commands are always described in pairs, and the same holds for the associated **exe**- and **get**-commands. The command name and its equivalent 3- or 4-character mnemonic are printed in the first two columns. The list of arguments for the **set**- and **get**-commands is listed in the first and second row respectively. In our example, **setCommandName** can accepts up to 6 arguments and **getCommandName** only accepts one argument. Mandatory arguments are printed in bold face. Besides the mandatory arguments, at least one of the optional arguments must be provided in the command line.

The list of possible values for each argument is printed under each of them. Default values for optional arguments are underlined.

The fictitious command above contains all the possible argument types:

- *Cd* serves as an index for all following arguments. This can be noticed by the possibility to use this argument in the **get**-command. This argument is mandatory in the **set**-command. The accepted values are `COM1`, `COM2` and `all`, corresponding to the first or second serial ports, or to both of them respectively. The "+" sign before the first two values indicates that they can be combined to address both serial ports in the same command.

  Examples: `COM1, COM1+COM2, all` (which is actually an alias for `COM1+COM2`).

- *Distance* is a number between `-20` and `20` with a default value of `0`, and up to 2 decimal digits. An error is returned if more digits are provided. The "m" indicates that the value is expressed in meters. Note that this "m" should not be typed when entering the command.

  Examples: `20, 10.3, -2.34`

- *Time* is a number between `1` and `50`, with no decimal digit (i.e. this is an integer value). This value is expressed in seconds.

  Examples: `1, 10`

- *Message* is a string with a maximum length of 120 characters. The default value of that argument is "`Unknown`". When spaces must to be used, the string has to be put between quotes and these enclosing quotes are not considered part of the string. The list of allowed characters in strings is:

  ```
  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
  0123456789 !#%()*+-./:;<=>?[\\]^_`{|}~
  ```

  Example: "`Hello World!`"

- *Mode* is a range of individual values that cannot be combined (they are not preceded by a "+" sign). Either `off` or `on` can be selected for that argument and the default value is `on`.

  Example: `on`

- *PRN* is a range of values that can be combined together with the "+" sign. The default value `GPS` is an alias for `G01+G02+ ... +G32`, `SBAS` is an alias for `S120+ ... +S138` and `all` an

alias for GPS+SBAS. A "+" sign can be set before the argument to indicate to add the specified value(s) to the current list. If the value "none" is supported (which is the case in this example), a "-" sign can be set before the argument to remove the specified value(s) from the current list. It is possible to add or remove multiple values at once by "adding" or "subtracting" them with the "+" or "-" operator. However, "+" and "-" can never be combined in a single argument.

Examples: `G01+G02, +G03, GPS+S120, +G04+G05, -S122-S123, -GPS`

3. Command List

3.1. Receiver Administration Commands

lai	lstAntennaInfo	*Antenna*						
		Overview Main [Antenna List]						

Use this command with the argument *Antenna* set to `Overview` to get a list of all antenna names for which the receiver knows the phase center variation parameters (see Firmware User Manual for a discussion on the phase center variation).

Use this command with the argument *Antenna* set to one of the antenna names returned by **`lstAntennaInfo, Overview`** to retrieve the complete phase center variation parameters for that particular antenna. Do not forget to enclose the name between quotes if it contains whitespaces.

Using the values `Main` will return the phase center variation parameters corresponding to the main antenna type as specified in the command **`setAntennaOffset`**.

Examples

```
COM1> lai, Overview <CR>
$R; lai, Overview
<?xml version="1.0" encoding="ISO-8859-1" ?>
<AntennaInfo version="0.1">
    <Antenna ID="AERAT1675_29    NONE"/>
    <Antenna ID="AERAT2775_150   NONE"/>
    <Antenna ID="AERAT2775_159       "/>
    <Antenna ID="AERAT2775_159   SPKE"/>
    <Antenna ID="AERAT2775_160       "/>
    ...
    <Antenna ID="TRM_R8_GNSS         "/>
</AntennaInfo>
COM1>

COM1> lai, "AERAT2775_159   SPKE" <CR>
$R; lai,"AERAT2775_159   SPKE"
<?xml version="1.0" encoding="ISO-8859-1" ?>
<AntennaInfo version="0.1">
    <Antenna ID="AERAT2775_159   SPKE"/>
        <L1>
            <offset north="0.4" east="0.1" up="77.2"/>
            <phase elevation="90" value="0.0"/>
            <phase elevation="85" value="-0.2"/>
...
            <phase elevation=" 5" value="0.0"/>
            <phase elevation=" 0" value="0.0"/>
        </L2>
</AntennaInfo>
COM1>
```

help	lstCommandHelp	*Action (255)*						
		[CMD List]						

Use this command to retrieve a short description of the ASCII command-line interface.

When invoking this command with the `Overview` argument, the receiver returns the list of all supported **`set`**-, **`get`**- and **`exe`**-commands. The **`lstCommandHelp`** command can also be called with any supported **`set`**-, **`get`**- or **`exe`**-command (the full name or the mnemonic) as argument.

The reply to this command is free-formatted and subject to change in future versions of the receiver's software. This command is designed to be used by human users. When building software applications, it is recommended to use the formal **`lstMIBDescription`**.

Examples

```
COM1> help, Overview <CR>
$R; help, Overview
$-- BLOCK 1 / 0
MENU: communication
  GROUP: ioSelection
    sdio, setDataInOut
    gdio, getDataInOut
...
COM1>

COM1> help, getReceiverCapabilities <CR>
$R; help, getReceiverCapabilities
... Here comes a description of getReceiverCapabilities ...
COM1>

COM1> help, grc <CR>
$R; help, grc
... Here comes a description of getReceiverCapabilities ...
COM1>
```

lcf	lstConfigFile	*File*						
		Current Boot RxDefault User1 User2						

Use this command to list the contents of a configuration file. Configuration files keep the value of all the non-default user-selectable parameters, such as the elevation mask, the positioning mode, etc.

The `Current` file contains the current receiver configuration. The `Boot` file contains the configuration that is loaded at boot time. The `RxDefault` file may contain some receiver-specific default values that are different from the generic defaults defined in this document.

User-defined configuration files (`User1`, `User2`) can be used to store frequently-used user-specific configurations inside the receiver.

See also the related **`exeCopyConfigFile`** command.

Example

```
COM1> sem, tracking, 10 <CR>
$R: sem, tracking, 10
  ElevationMask, Tracking, 10
COM1> lcf, Current <CR>
$R; lcf, Current
$-- BLOCK 1 / 1
  setElevationMask, Tracking, 10
COM1>
```

eccf gccf	**exeCopyConfigFile** **getCopyConfigFile**	***Source***	***Target***					
		Current Boot User1 User2 RxDefault	Current Boot User1 User2					

Use this command to manage the configuration files. See the **`lstConfigFile`** command for a description of the different configuration files.

With this command, the user can copy configurations files into other configuration files. For instance, copying the `Current` file into the `Boot` file makes that the receiver will always boot in the current configuration.

Examples

To save the current configuration in the `Boot` file, use:

```
COM1> eccf, Current, Boot <CR>
$R: eccf, Current, Boot
  CopyConfigFile, Current, Boot
COM1>
```

To load the configuration stored in `User1`, use:

```
COM1> eccf, User1, Current <CR>
$R: eccf, User1, Current
  CopyConfigFile, User1, Current
COM1>
```

lif	lstInternalFile	*File*						
		Permissions Identification Debug Error SisError DiffCorrError						

Use this command to retrieve the contents of one of the receiver internal files.

The following table describes all files that can be retrieved from Septentrio receivers:

File	Description
`Permissions`	List of permitted options in your receiver.
`Identification`	Information about the different components being part of the receiver (e.g. serial number, firmware version, etc.).
`Debug`	Program flow information that can help Septentrio engineers to debug certain issues.
`Error`	Last internal error reports.
`SisError`	Last detected signal-in-space anomalies.
`DiffCorrError`	Last detected anomalies in the incoming differential correction streams.

Example

```
COM1> lif, Permissions <CR>
$R; lif, Permissions
---->
$-- BLOCK 1 / 1

... here follows the permission file ...

COM1>
```

lmd	lstMIBDescription	*File (255)*						
		[CMD List]						

Use this command to retrieve the ASN.1-compliant syntax of both the ASCII and SNMP' command interface. The name of the command refers to the MIB (Management Information Base), which holds the whole receiver configuration. There is a one-to-one relationship between the formal MIB description and the ASCII command-line interface for all **exe**-, **get**- and **set**-commands. See the SNMP' Technical Note for a detailed description of the Septentrio proprietary SNMP' protocol and MIB.

When the value `Overview` is used, the general syntax of the interface is returned. With the value `SBFTable`, the receiver will output the list of supported SBF blocks and whether they can be output at a user-selectable rate or not. The **lstMIBDescription** command can also be called with every supported **set**-, **get**- or **exe**-command (the full name or the mnemonic) as argument.

No formal description of the **lst**-commands can be retrieved with **lstMIBDescription**.

Examples

```
COM1> lmd, Overview <CR>
$R; lmd, Overview
... Here comes the generic command syntax ...
COM1>

COM1> lmd, grc <CR>
$R; lmd, grc
... Here comes the description of getReceiverCapabilities ...
COM1>
```

grc	getReceiverCapabilities							

Use this command to retrieve the so-called "capabilities" of your receiver. The first returned value is the list of supported antenna(s), followed by the list of supported signals, the list of available communication ports and the list of enabled features.

The three values at the end of the reply line correspond to the default measurement interval, the default PVT interval and the default integrated INS/GNSS interval respectively. This is the interval at which the corresponding SBF blocks are output when the `OnChange` rate is selected with the **`setSBFOutput`** command. These values are expressed in milliseconds.

Each of the above-mentioned lists contain one or more of the elements in the tables below.

Antennas	Description
`Main`	The receiver's main antenna.

Signals	Description
`GPSL1CA`	GPS L1 C/A signal.
`GPSL2PY`	GPS L2 P(Y) signal.
`GPSL2C`	GPS L2 C signal.
`GLOL1CA`	GLONASS L1 C/A signal.
`GLOL2CA`	GLONASS L2 C/A signal.
`GALL1BC`	Galileo L1 BC signal.
`GEOL1`	GEO L1 C/A signal.

ComPorts	Description
`COM1`	RS232/TTL serial port 1.
`COM2`	RS232/TTL serial port 2.
`COM3`	RS232/TTL serial port 3.
`USB1`	Virtual serial port 1.
`USB2`	Virtual serial port 2.

Capabilities	Description
`SBAS`	Positioning with SBAS corrections.
`DGPSRover`	Positioning with DGPS corrections.
`DGPSBase`	Generation of DGPS corrections.
`RTKRover`	Positioning with RTK corrections.
`RTKBase`	Generation of RTK corrections.
`RTCMv23`	Generation/decoding of RTCM v2.3 corrections.
`RTCMv3x`	Generation/decoding of RTCM v3.x corrections.
`CMRv20`	Generation/decoding of CMR v2.0 corrections.
`xPPSOutput`	Generation of xPPS output signal.
`TimedEvent`	Accurate time mark of event signals.
`APME`	A-Posteriori Multipath Estimator.
`RAIM`	Receiver Autonomous Integrity Monitoring.

Example

```
COM1> grc <CR>
$R: grc
  ReceiverCapabilities, Main, GPSL1CA+GEOL1, COM1+COM2+COM3+COM4+ USB1+USB2,
APME+SBAS, 100, 100, 100
COM1>
```

gri	**getReceiverInterface**	*Item*						
		+RxName +SNMPLanguage +SNMPVersion all						

Use this command to retrieve the version of the receiver command-line interface. The reply to this command is a subset of the reply returned by the **`lstInternalFile, Identification`** command.

Example

```
COM1> gri <CR>
$R: gri
  ReceiverInterface, RxName, AsteRx1
  ReceiverInterface, SNMPLanguage, English
  ReceiverInterface, SNMPVersion, 20060308
COM1>
```

era gra	**exeRegisteredApplications** **getRegisteredApplications**	***Cd*** *Cd*	***Application (12)***					
		+COM1 +COM2 +COM3 +USB1 +USB2 all	Unknown					

Use these commands to define/inquire the name of the application that is currently using a given connection descriptor (*Cd*).

Registering an application name for a connection does not affect the receiver operation, and is done on a voluntary basis. Application registration can be useful to developers of external applications when more than one application is to communicate with the receiver concurrently. Whether or not this command is used, and the way it is used is up to the developers of external applications.

The RxControl graphical interface registers itself with this command, such that third party applications can know which connection RxControl is connected to.

Example

```
COM1> sra, com1, MyApp <CR>
$R: sra, com1, MyApp
  RegisteredApplications, COM1, "MyApp"
  RegisteredApplications, COM2, "Unknown"
  RegisteredApplications, COM3, "Unknown"
  RegisteredApplications, COM4, "Unknown"
  RegisteredApplications, USB1, "Unknown"
  RegisteredApplications, USB2, "Unknown"
COM1>
```

erst grst	**exeResetReceiver** **getResetReceiver**	***Level***	***EraseMemory***					
		Soft Hard Upgrade	none +Config +PVTData +SatData all					

Use this command to reset the receiver and to erase some previously stored data. The first argument specifies which level of reset you want to execute:

Level	Description
`Soft`	This is a reset of the receiver's firmware. After a few seconds, the receiver will restart operating in the same configuration as before the command was issued, unless the "`Config`" value is specified in the second argument.
`Hard`	This is similar to a power off/on sequence. After hardware reset, the receiver will use the configuration saved in the boot configuration file.
`Upgrade`	Set the receiver into upgrade mode. After a few seconds, the receiver is ready to accept an upgrade file (SUF format) from any of its connections.

The second argument specifies which part of the non-volatile memory should be erased during the reset. The following table contains the possible values for the *EraseMemory* argument:

EraseMemory	Description
`Config`	The receiver's configuration is reset to the factory default. The `Current` and `Boot` configuration files are erased (see the **`exeCopyConfigFile`** command). Note that the `User1` and `User2` configuration files are not erased: use the **`exeCopyConfigFile`** command instead.
`PVTData`	The latest computed PVT data stored in non-volatile memory is erased.
`SatData`	All satellite navigation data (ephemeris, almanac, ionosphere parameters, UTC, ...) stored in non-volatile memory are erased.

Before resetting, the receiver broadcasts a "`$TE ResetReceiver`" message to all active communication ports, to inform all users of the imminent reset.

After a reset, the user may have to adapt the communication settings of his/her terminal program as they may be reset to their default values.

Example

```
COM1> erst, Hard, none <CR>
$R: erst, Hard, none
  ResetReceiver, Hard, none
STOP>
$TE ResetReceiver Hard
STOP>
```

3.2. Tracking Configuration Commands

sac gac	**setAntennaConnector** **getAntennaConnector**	*MainAntenna*						
		auto Ext Int						

Use these commands to define/inquire where to take the RF signal from. These are the different options:

MainAntenna	Description
`auto`	Automatically switch to the `Ext` antenna when connected. Otherwise use `Int` antenna. Refer to the Hardware Manual for a description of the criterium used to detect the presence of an external antenna.
`Ext`	Force to take the RF signal from the `Ext` antenna input connector.
`Int`	Force to take the RF signal from the `Int` antenna input connector.

Example

```
COM1> sac, auto <CR>
$R: sac, auto
  AntennaConnector, auto
COM1>
```

sca gca	**setChannelAllocation** **getChannelAllocation**	***Channel*** *Channel*	*Satellite*	*Search*	*Doppler*	*Window*		
		+Ch01 ... Ch29 all	auto G01 ... G32 F01 ... F21 E01 ... E32 S120 ... S138	auto manual	-50000 ... 0 ... 50000 Hz	1 ... 16000 ... 100000 Hz		

Use these commands to define/inquire the satellite-to-channel allocation of the receiver.

The action of the **`setChannelAllocation`** command is to force the allocation of a particular satellite on the set of channels identified with the *Channel* argument, thereby overruling the automatic channel allocation performed by the receiver. It is possible to allocate the same satellite to more than one channel. If you assign a satellite to a given channel, any other channel that was automatically allocated to the same satellite will be stopped and will be reallocated.

The values `Gxx`, `Exx`, `Fxx`, `Sxxx` and `Cxx` for the *Satellite* argument represent GPS, Galileo, GLONASS, SBAS and COMPASS satellites respectively. For GLONASS, the frequency number (with an offset of 8) should be provided, and not the slot number (hence the "F"). Setting the *Satellite* argument to `auto` brings the channel back in auto-allocation mode.

The user can specify the Doppler window in which the receiver has to search for the satellite. This is done by setting the *Search* argument to `manual`. In that case, the *Doppler* and *Window* arguments can be provided: the receiver will search for the signal within an interval of *Window* Hz centred on *Doppler* Hz. The value to be provided in the *Doppler* argument is the expected Doppler at the GPS L1 carrier frequency (1575.42MHz). This value includes the geometric Doppler and the receiver and satellite frequency biases. Specifying a Doppler window can speed up the search process in some circumstances. A satellite already in tracking that falls outside of the prescribed window will remain in tracking.

If *Search* is set to `auto`, the receiver applies its usual search procedure, as it would do for auto-allocated satellites, and the *Doppler* and *Window* arguments are ignored.

Be aware that this command may disturb the normal operation of the receiver and is intended only for expert-level users.

Examples

```
COM1> sca, Ch05, G01 <CR>
$R: sca, Ch05, G01
  ChannelAllocation, Ch05, G01, auto, 0, 16000
COM1>

COM1> gca, Ch05 <CR>
$R: gca, Ch05
  ChannelAllocation, Ch05, G01, auto, 0, 16000
COM1>
```

gcc	**getChannelConfiguration**	*Channel*						
		+Ch01 ... Ch29 all						

Use this command to get the list of signals that a given channel can track.

Example

To display the different signals that the first channel can track, use:

```
COM1> gcc, Ch01 <CR>
$R: gcc, Ch01
  ChannelConfiguration, Ch01, GPSL1CA
COM1>
```

scm gcm	**setCN0Mask** **getCN0Mask**	***Signal*** *Signal*	*Mask*					
		+GPSL1CA +Reserved2 +GPSL2C +GLOL1CA +GLOL2CA +GALL1BC +GEOL1 all	0 ... 28 ... 60 dB-Hz					

Use these commands to define/inquire the carrier-to-noise ratio mask for the generation of measurements. The receiver does not generate measurements for those signals of which the C/N_0 is under the specified mask, and does not include these signals in the PVT computation. However, it continues to track these signals and to decode and use the navigation data as long as possible, regardless of the C/N_0 mask.

The mask can be set independently for each of the signal types supported by the receiver, except for the GPS P-code, of which the mask is fixed at 1 dB-Hz (this is because of the codeless tracking scheme needed for GPS P-code).

Examples

```
COM1> scm, GEOL1, 30 <CR>
$R: scm, GEOL1, 30
  CN0Mask, GEOL1, 30
COM1>

COM1> gcm, GEOL1 <CR>
$R: gcm, GEOL1
  CN0Mask, GEOL1, 30
COM1>
```

smm gmm	**setMultipathMitigation** **getMultipathMitigation**	*Code*	*Carrier*					
		off on	off on					

Use these commands to define/inquire whether multipath mitigation is enabled or not.

The arguments *Code* and *Carrier* enable or disable the A-Posteriori Multipath Estimator (APME) for the code and carrier phase measurements respectively. APME is a technique by which the receiver continuously estimates the multipath error and corrects the measurements accordingly.

This multipath estimation process slightly increases the thermal noise on the pseudoranges. However, this increase is more than compensated by the dramatic decrease of the multipath noise.

Note that APME is not applied to high chip-rate signals (e.g. GPS L5): these signals are by nature much more immune to multipath than the low chip-rate ones (e.g. GPS CA or Galileo L1BC). The increase in thermal noise solely affects those signals where APME is applied.

Examples

```
COM1> smm, on, off <CR>
$R: smm, on, off
  MultipathMitigation, on, off
COM1>

COM1> gmm <CR>
$R: gmm
  MultipathMitigation, on, off
COM1>
```

sst gst	**setSatelliteTracking** **getSatelliteTracking**	*Satellite*						
		none +G01 ... G32 +R01 ... R24 +E01 ... E32 +S120 ... S138 +GPS +GLONASS +GALILEO +SBAS all						

Use these commands to define/inquire which satellites are allowed to be tracked by the receiver. It is possible to enable or disable a single satellite (e.g. `G01` for GPS PRN1), or a whole constellation. `Gxx`, `Exx`, `Rxx`, `Sxxx` and `Cxx` refer to a GPS, Galileo, GLONASS, SBAS or COMPASS satellite respectively. GLONASS satellites must be referenced by their slot number (from 1 to 24) in this command.

A satellite which is disabled by this command is not considered anymore in the automatic channel allocation mechanism, but it can still be forced to a given channel, and tracked, using the **`setChannelAllocation`** command.

Tracking a satellite does not automatically mean that the satellite will be included in the PVT computation. The inclusion of a satellite in the PVT computation is controlled by the **`setSatelliteUsage`** command.

Examples

To only enable the tracking of GPS satellites, use:

```
COM1> sst, GPS <CR>
$R: sst, GPS
  SatelliteTracking, G01+G02+G03+G04+G05+G06+G07+G08+G09+G10+G11
+G12+G13+G14+G15+G16+G17+G18+G19+G20+G21+G22+G23+G24+G25+G26+G27
+G28+G29+G30+G31+G32
COM1>
```

To add all SBAS satellites in the list of satellites to be tracked, use:

```
COM1> sst, +SBAS <CR>
$R: sst, +SBAS
  SatelliteTracking, G01+G02+G03+G04+G05+G06+G07+G08+G09+G10+G11
+G12+G13+G14+G15+G16+G17+G18+G19+G20+G21+G22+G23+G24+G25+G26+G27
+G28+G29+G30+G31+G32+S120+S121+S123+S123+S124+S125+S126+S127+S128
+S129+S130+S131+S132+S133+S134+S135+S136+S137+S138
  COM1>
```

To remove SBAS PRN120 from the list of allowed satellites, use:

```
COM1> sst, -S120 <CR>
$R: sst, -S120
  SatelliteTracking, G01+G02+G03+G04+G05+G06+G07+G08+G09+G10+G11
+G12+G13+G14+G15+G16+G17+G18+G19+G20+G21+G22+G23+G24+G25+G26+G27
+G28+G29+G30+G31+G32+S121+S122+S123+S124+S125+S126+S127+S128+S129
+S130+S131+S132+S133+S134+S135+S136+S137+S138
  COM1>
```

snt gnt	**setSignalTracking** **getSignalTracking**	*Signal*						
		+GPSL1CA +GPSL2PY +GPSL2C +GLOL1CA +GLOL2CA +GALL1BC +GEOL1 all						

Use these commands to define/inquire which signals are allowed to be tracked by the receiver. By using this command, it is for instance possible to let a multi-frequency receiver emulate a single-frequency receiver.

Invoking this command causes all tracking loops to stop and restart.

Examples

To configure the receiver in a single-frequency L1 GPS+SBAS mode, use:

```
COM1> snt, GPSL1CA+GEOL1 <CR>
$R: snt, GPSL1CA+GEOL1
  SignalTracking, GPSL1CA+GEOL1
COM1>

COM1> gnt <CR>
$R: gnt
  SignalTracking, GPSL1CA+GEOL1
COM1>
```

ssi gsi	**setSmoothingInterval** **getSmoothingInterval**	***Signal*** *Signal*	*Interval*	*Alignment*				
		+GPSL1CA +GPSL2PY +GPSL2C +GLOL1CA +GLOL2CA +GALL1BC +GEOL1 all	0 ... 1000 sec	0 ... 1000 sec				

Use these commands to define/inquire the code measurement smoothing interval.

The *Interval* argument defines the length of the smoothing filter that is used to smooth the code measurements by the carrier phase measurements. It is possible to define a different interval for each signal type. If *Interval* is set to 0, the code measurements are not smoothed. The smoothing interval can vary from 1 to 1000 seconds.

To prevent transient effect to perturb the smoothing filter, smoothing is disabled during the first ten seconds of tracking, i.e. when the lock time is lower than 10s. Likewise, the smoothing effectively starts with a delay of 10 seconds after entering the **`setSmoothingInterval`** command.

Code smoothing allows reducing the pseudoranges noise and multipath. It has no influence on the carrier phase and Doppler measurements. The smoothing filter has an incremental effect; the noise of the filtered pseudoranges will decrease over time and reach its minimum after *Interval* seconds. For some applications, it may be necessary to wait until this transient effect is over before including the measurement in the PVT computation. This is the purpose of the *Alignment* argument. If *Alignment* is not set to 0, the first set of measurements during *Alignment*+10 seconds will not be used in the PVT computation (though it will still be available in the `MeasEpoch` SBF block). Logically, the effective amount of *Alignment* is never larger than *Interval*, even if the user sets it to a larger value.

Examples

```
COM1> ssi, GPSL1CA, 300 <CR>
$R: ssi, GPSL1CA, 300
  SmoothingInterval, GPSL1CA, 300, 0
COM1>

COM1> gsi, GPSL1CA <CR>
$R: gsi, GPSL1CA
  SmoothingInterval, GPSL1CA, 300, 0
COM1>
```

stlp gtlp	**setTrackingLoopParameters** **getTrackingLoopParameters**	***Signal*** *Signal*	*DLLBandwidth*	*PLLBandwidth*	*MaxTpDLL*	*MaxTpPLL*	*Adaptive*	
		+GPSL1CA +Reserved2 +GPSL2C +GLOL1CA +GLOL2CA +GALL1BC +GEOL1 all	0.01 ... 0.25 ... 5.00 Hz	1 ... 15 ... 100 Hz	1 ... 100 ... 500 msec	1 ... 10 ... 200 msec	off on	

Use these commands to define/inquire the tracking loop parameters for each individual signal type.

The *DLLBandwidth* and *PLLBandwidth* arguments define the single-sided DLL and PLL noise bandwidth, in Hz.

The *MaxTpDLL* argument defines the maximum DLL pre-detection time, in millisecond. The actual pre-detection time applied by the receiver (*TpDLL*) depends on the presence of a pilot component. For signals having a pilot component (e.g. GPS L2C), *TpDLL* = *MaxTpDLL*. For signals without pilot component (e.g. GPS L1CA), *TpDLL* is the largest divider of the symbol duration smaller than or equal to *MaxTpDLL*.

The *MaxTpPLL* argument defines the maximal PLL pre-detection time, in millisecond. The actual pre-detection time in the receiver (*TpPLL*) is computed in the same way as indicated for the *MaxTpDLL* argument.

Setting the *Adaptive* argument to `on` allows the receiver to dynamically change the loop parameters in order to optimize performance in specific conditions.

After entering this command, all active tracking loops stop and restart with the new settings.

This command should only be used by expert users who understand the consequences of modifying the default values. In some circumstances, changing the tracking parameters may result in the impossibility for the receiver to track a specific signal, or may significantly increase the processor load. It is recommended that the product of *TpPLL* (in milliseconds) and *PLLBandwidth* (in Hz) be kept between 100 and 200.

Note that decreasing the predetection times increases the load on the processor.

Example

```
COM1> stlp, GPSL1CA, 0.20, 12 <CR>
$R: stlp, GPSL1CA, 0.20, 12
  TrackingLoopParameters, GPSL1CA, 0.20, 12, 100, 10
COM1>
```

3.3. Navigation Configuration Commands

sal gal	**setAntennaLocation** **getAntennaLocation**	***Antenna*** *Antenna*	*Mode*	*DeltaX*	*DeltaY*	*DeltaZ*		
		+Base all	auto manual	-1000.0000 ... 0.0000 ... 1000.0000 m	-1000.0000 ... 0.0000 ... 1000.0000 m	-1000.0000 ... 0.0000 ... 1000.0000 m		

Use this command to define/inquire the relative location of the antennas in the vehicle reference frame in the context of attitude determination.

The attitude of a vehicle (more precisely the heading and pitch angles) can be determined from the orientation of the baseline between two antennas attached to the vehicle. These two antennas can be connected to two receivers configured in moving-base RTK operation (moving-base attitude) .

In moving-base attitude, **`setAntennaLocation`** should be invoked at the rover receiver with the *Antenna* argument set to `Base` to specify the relative position of the base antenna with respect to the rover antenna.

In `auto` mode, the receiver determines the attitude angles assuming that the baseline between the antenna ARPs is parallel to the longitudinal axis of the vehicle, and that the base antenna is in front of the rover antenna (i.e. towards the direction of movement). The length of the baseline is automatically computed by the receiver, and the baseline may be flexible. The *DeltaX*, *DeltaY* and *DeltaZ* arguments are ignored in `auto` mode.

In `manual` mode, the user can specify the exact position of the base antenna with respect to the rover antenna in the vehicle reference frame. That reference frame has its X axis pointing to the front of the vehicle, the Y axis pointing to the right, and the Z axis pointing down. Selecting `manual` mode implies that the baseline is rigid. The *DeltaX*, *DeltaY* and *DeltaZ* coordinates are ARP-to-ARP.

Example

In the case of moving-base attitude determination, if the moving-base antenna is located one meter to the left of the rover antenna, and 10 cm below, you should use:

```
COM1> sal, Base, manual, 0, -1, 0.1 <CR>
$R: sal, Base, manual, 0, 1, 0.1
    AntennaLocation, Base, manual, 0.0000, -1.0000, 0.1000
COM1>
```

sao gao	**setAntennaOffset** **getAntennaOffset**	***Antenna*** *Antenna*	*DeltaE*	*DeltaN*	*DeltaU*	*Type (20)*	*SerialNr (20)*	*SetupID*
		+Main all	-1000.0000 ... 0.0000 ... 1000.0000 m	-1000.0000 ... 0.0000 ... 1000.0000 m	-1000.0000 ... 0.0000 ... 1000.0000 m	Unknown	Unknown	0 ... 255

Use these commands to define/inquire the parameters that are associated with the antenna connected to your receiver.

The arguments *DeltaE*, *DeltaN* and *DeltaU* are the offsets of the antenna reference point (ARP) with respect to the marker, in the East, North and Up (ENU) directions respectively, expressed in meters. All absolute positions reported by the receiver are marker positions, obtained by subtracting this offset from the ARP. The purpose is to take into account the fact that the antenna may not be located directly on the surveying point of interest.

Use the argument *Type* to specify the type of your antenna. For best positional accuracy, it is recommended to select a type from the list returned by the command **`lstAntennaInfo, Overview`**. This is the list of antennas for which the receiver can compensate for phase center variation (see Firmware User Manual for details). If *Type* does not match any entry in the list returned by **`lstAntennaInfo, Overview`**, the receiver will assume that the phase center variation is zero at all elevations and frequency bands, and the position will not be as accurate. If the antenna name contains whitespaces, it has to be enclosed in quotes. For proper name matching, it is important to keep the exact same number of whitespaces and the same case as the name returned by **`lstAntennaInfo, Overview`**.

The argument *SerialNr* is the serial number of your particular antenna. It may contain letters as well as digits (do not forget to enclose the string in quotes if it contains whitespaces).

The argument *SetupID* is the antenna setup ID as defined in the RTCM standard. It is a parameter for use by the service provider to indicate the particular reference station-antenna combination. The number should be increased whenever a change occurs at the station that affects the antenna phase center variations. Setting *SetupID* to zero means that the values of a standard model type calibration should be used. The value entered for this argument is used to set the setup ID field in the message type 23 of RTCM2.3, and in message types 1007 and 1008 of RTCM3. It has otherwise no effect on the receiver operation.

Examples

If you are using an antenna type "AERAT2775_159 SPKE", serial number 5684, of which the ARP is located 0.1 meters eastward of and 0.2 meters above the marker of interest, set the offset as follows:

```
COM1> sao, Main, 0.1, 0.0, 0.2, "AERAT2775_159   SPKE", 5684 <CR>
$R: sao, Main, 0.1, 0.0, 0.2, "AERAT2775_159   SPKE", 5684
    AntennaOffset, Main, 0.1000, 0.0000, 0.2000, "AERAT2775_159   SPKE",
"5684", 0
COM1>

COM1> gao, Main <CR>
$R: gao, Main
    AntennaOffset, Main, 0.1000, 0.0000, 0.2000, "AERAT2775_159   SPKE",
"5684", 0
COM1>
```

sdca gdca	**setDiffCorrMaxAge** **getDiffCorrMaxAge**	*DGPSCorr*	*RTKCorr*					
		0.0 ... 120.0 ... 3600.0 sec	0.0 ... 20.0 ... 3600.0 sec					

Use these commands to define/inquire the maximum age acceptable for a given differential correction type. A correction is applied only if its age (aka latency) is under the timeout specified with this command and if it is also under the timeout specified with the *MaxAge* argument of the **`setDiffCorrUsage`** command. In other words, the command **`setDiffCorrUsage`** sets a global maximum timeout value, while the command **`setDiffCorrMaxAge`** can force shorter timeout values for certain correction types.

The argument *DGPSCorr* defines the timeout of the range corrections when the PVT is computed in DGPS mode.

The argument *RTKCorr* defines the timeout of the base station code and carrier phase measurements when the PVT is computed in RTK mode.

If the timeout is set to 0, the receiver will never apply the corresponding correction.

Note that this command does not apply to the corrections transmitted by SBAS satellites. For these corrections, the receiver always applies the timeout values prescribed in the DO229 standard.

Example

```
COM1> sdca, 10 <CR>
$R: sdca, 10
  DiffCorrMaxAge, 10.0, 20.0, 300.0, 300.0
COM1>
```

sdcu gdcu	**setDiffCorrUsage** **getDiffCorrUsage**	*Mode*	*MaxAge*	*BaseSelection*	*BaseID*	*MovingBase*	*MaxBase*	*MaxBaseline*
		LowLatency	0.1 ... 3600.0 sec	auto manual	0 ... 4095	off on	2 ... 5 ... 10	0 ... 2500000 m

Use these commands to define/inquire the usage of incoming differential corrections in DGPS or RTK rover mode.

The *Mode* argument defines the type of differential solution that will be computed by the receiver. If `LowLatency` is selected, the PVT is computed at the moment local measurements of the receiver are available. At that time, measurements from the base receiver for the same epoch are not yet available due to latency in the transmission, and the PVT is computed with measurements from two different epochs.

The *MaxAge* argument defines the maximum age of the differential corrections to be considered valid. *MaxAge* applies to all types of corrections (DGPS, RTK, satellite orbit, etc), except for those received from a SBAS satellite. See also the command **`setDiffCorrMaxAge`** to set different maximum ages for different correction types.

The *BaseSelection* argument defines how the receiver should select the base station(s) to be used. If `auto` is selected and the receiver is in DGPS-rover mode, it will use all available base stations. If `auto` is selected and the receiver is in RTK-rover mode, it will automatically select the nearest base station. If `manual` is selected, the receiver will only use the corrections from the base station defined by the *BaseID* argument (in both DGPS and RTK modes).

The *MovingBase* argument defines whether the base station is static or moving.

MaxBase sets the maximum number of base stations to include in the PVT solution in multi-base DGNSS mode.

MaxBaseline sets the maximum baseline length: base stations located beyond the maximum baseline length are excluded from the PVT.

Examples

```
COM1> sdcu, , 5 <CR>
$R: sdcu, , 5
  DiffCorrUsage, LowLatency, 5.0, auto, 0, off, 20, 20000000
COM1>

COM1> gdcu <CR>
$R: gdcu
  DiffCorrUsage, LowLatency, 5.0, auto, 0, off, 20, 20000000
COM1>
```

sem gem	**setElevationMask** **getElevationMask**	***Engine*** *Engine*	*Mask*					
		+Tracking +PVT all	-90 ... 0 ... 90 deg					

Use these commands to set or get the elevation mask in degrees. There are two masks defined: a tracking mask and a PVT mask.

Satellites under the tracking elevation mask are not tracked, and therefore there is no measurement, nor navigation data available from them. The tracking elevation mask does not apply to SBAS satellites: SBAS satellites are generally used to supply corrections and it is undesirable to make the availability of SBAS corrections dependent on the satellite elevation. The tracking elevation mask does not apply to satellites that are manually assigned with the **`setChannelAllocation`** command.

Satellite under the PVT mask are not included in the PVT solution, though they still provide measurements and their navigation data is still decoded and used. The PVT elevation mask do apply to the SBAS satellites: the ranges to SBAS satellites under the elevation mask are not used in the PVT, but the SBAS corrections are still decoded and potentially used in the PVT.

Although possible, it does not make sense to select a higher elevation mask for the tracking than for the PVT, as, obviously, a satellite which is not tracked cannot be included in the PVT.

The mask can be negative to allow the receiver to track satellites below the horizon. This can happen in case the receiver is located at high altitudes or if the signal is refracted through the atmosphere.

Examples

```
COM1> sem, PVT, 10 <CR>
$R: sem, PVT, 10
  ElevationMask, PVT, 10
COM1>

COM1> gem <CR>
$R: gem
  ElevationMask, Tracking, 0
  ElevationMask, PVT, 10
COM1>
```

sfr gfr	**setFixReliability** **getFixReliability**	***Engine*** *Engine*	*SearchVolume*	*Ratio*				
		+RTK all	0.001 ... 0.200 ... 10.000	1.00 ... 4.40 ... 20.00				

Use these commands to define/inquire the criteria that control the ambiguity fixing process of the RTK and/or attitude-determination engines.

The ambiguity fixing algorithm searches for the most likely integer carrier phase ambiguity set within the prescribed *SearchVolume*.

All integer ambiguity vectors contained in the search volume are sorted according to their distance to the float ambiguities. The candidate with the lowest value will be selected as the most likely ambiguity set. The likelihood of this candidate with respect to other candidates is characterized by the ratio between the best and the second-best candidate. If this ratio is lower than the prescribed threshold (*Ratio*, the third argument), the candidate is rejected and the ambiguity search will restart at the next epoch.

Lowering *SearchVolume* and increasing *Ratio* will increase the time to fix but also decrease the probability of fixing incorrect ambiguities.

Examples

```
COM1> sfr, RTK, 0.2 <CR>
$R: sfr, RTK, 0.2
  FixReliability, RTK, 0.200, 4.40
COM1>

COM1> gfr, RTK <CR>
$R: gfr, RTK
  FixReliability, RTK, 0.200, 4.40
COM1>
```

sgd ggd	**setGeodeticDatum** **getGeodeticDatum**	*Datum*						
		WGS84						

Use these commands to define/inquire the geodetic datum that the receiver uses to output position information. All positions computed by the receiver will be expressed in the selected datum.

The following geodetic datums are defined:

Datum	Description
WGS84	Datum used by the GPS constellation.

Example

```
COM1> sgd, WGS84 <CR>
$R: sgd, WGS84
  GeodeticDatum, WGS84
COM1>
```

sgu ggu	**setGeoidUndulation** **getGeoidUndulation**	*Mode*	*Undulation*					
		auto manual	-250.0 ... 0.0 ... 250.0 m					

Use these commands to define/inquire the geoid undulation at the receiver position. The geoid undulation specifies the local difference between the geoid, which is a reference equipotential surface of the Earth gravity field, and the ellipsoid associated with the datum selected in the **`setGeodeticDatum`** command.

If *Mode* is set to `auto`, the receiver uses its internal geodetic model, and the *Undulation* argument is ignored.

The geoid undulation is included in the `PVTCartesian` and the `PVTGeodetic` SBF blocks and in the NMEA position messages.

Examples

```
COM1> sgu, manual, 25.3 <CR>
$R: sgu, manual, 25.3
  GeoidUndulation, manual, 25.3
COM1>

COM1> ggu <CR>
$R: ggu
  GeoidUndulation, manual, 25.3
COM1>
```

sga gga	**setGNSSAttitude** **getGNSSAttitude**	*Source*						
		none MovingBase						

Use this command to define/inquire the way GNSS-based attitude is computed.

The attitude of a vehicle (more precisely the heading and pitch angles) can be determined from the orientation of the baseline between two antennas attached to the vehicle.

The *Source* argument specifies how to compute the GNSS-based attitude:

Source	Description
`none`	GNSS attitude computation is disabled.
`MovingBase`	Attitude is computed from the baseline between antennas connected to two receivers configured in moving-base RTK operation (moving-base attitude).

Example

```
COM1> sga, MovingBase <CR>
$R: sga, MovingBase
    GNSSAttitude, MovingBase, Fixed
COM1>
```

shm ghm	**setHealthMask** **getHealthMask**	***Engine*** *Engine*	*Mask*					
		+Tracking +PVT all	off on					

Use these commands to define/inquire whether measurements should be produced for unhealthy satellite signals, and whether these measurements should be included in the PVT solution. All satellite signals of which the health is unknown to the receiver (because the health information has not been decoded yet) are considered unhealthy.

If *Mask* is `on` for the `Tracking` engine, no measurements are generated for unhealthy signals, but these signals will remain internally tracked and their navigation data will be decoded and processed. This is to ensure immediate reaction in the event that the signal would become healthy again.

If *Mask* is `on` for the `PVT` engine, measurements from unhealthy signals are not included in the PVT. Setting this mask to `off` must be done with caution: including a non-healthy signal in the PVT computation may lead to unpredictable behaviour of the receiver.

Although possible, it does not make sense to enable unhealthy satellites for the PVT if they are disabled for tracking.

Examples

To track unhealthy satellites/signals, use:

```
COM1> shm, Tracking, off <CR>
$R: shm, Tracking, off
  HealthMask, Tracking, off
COM1>

COM1> ghm <CR>
$R: ghm
  HealthMask, Tracking, off
  HealthMask, PVT, on
COM1>
```

sim gim	**setIonosphereModel** **getIonosphereModel**	*Model*						
		auto off Klobuchar SBAS MultiFreq						

Use these commands to define/inquire the type of model used to correct ionospheric errors in the PVT computation. The following models are available:

Model	Description
`auto`	With this selection, the receiver will, based on the available information, automatically select the best model on a satellite to satellite basis.
`off`	The receiver will not correct measurements for the ionospheric delay. This may be desirable if the receiver is connected to a GNSS signal simulator.
`Klobuchar`	This model uses the parameters as transmitted by the GPS satellites to compute the ionospheric delays.
`SBAS`	This model complies with the DO229 standard [1]: it uses the near real-time ionospheric delays transmitted by the SBAS satellites in MT18 and MT26. If no such message has been received, the `Klobuchar` model is selected automatically.
`MultiFreq`	This model uses a combination of measurements on different carriers to accurately estimate ionospheric delays. It requires the availability of at least dual-frequency measurements.

Unless the model is set to `auto`, the receiver uses the same model for all satellites, e.g. if the `Klobuchar` model is requested, the Klobuchar parameters transmitted by GPS satellites are used for all tracked satellites, regardless of their constellation.

If not enough data is available to apply the prescribed model to a given satellite (for instance if only single-frequency measurements are available and the model is set to `MultiFreq`), the satellite in question will be discarded from the PVT. Under most circumstances, it is recommended to leave the model to `auto`.

Examples

To disable the compensation for ionospheric delays, use:

```
COM1> sim, off <CR>
$R: sim, off
  IonosphereModel, off
COM1>

COM1> gim <CR>
$R: gim
  IonosphereModel, off
COM1>
```

smv gmv	**setMagneticVariance** **getMagneticVariance**	*Mode*	*Variance*					
		manual	-180.0 ... 0.0 ... 180.0 deg					

Use these commands to define/inquire the magnetic variance at the current position. The magnetic variance specifies the local offset of the direction to the magnetic North with respect to the geographic North.

Note that the magnetic variance is used solely in the generation of NMEA messages.

Examples

```
COM1> smv, manual, 1.1 <CR>
$R: smv, manual, 1.1
  MagneticVariance, manual, 1.1
COM1>

COM1> gmv <CR>
$R: gmv
  MagneticVariance, manual, 1.1
COM1>
```

snrc gnrc	**setNetworkRTKConfig** **getNetworkRTKConfig**	*NetworkType*						
		auto VRS						

Use these commands to define/inquire the type of the RTK network providing the differential corrections.

In most cases, it is recommended to leave the *Type* argument to `auto` to let the receiver autodetect the network type. For some types of VRS networks (especially for those having long baselines between the base stations), optimal performance is obtained by forcing the type to VRS.

Example

```
COM1> snrc, VRS <CR>
$R: snrc, VRS
NetworkRTKConfig, VRS
COM1>
```

spm gpm	**setPVTMode** **getPVTMode**	*Mode*	*RoverMode*	*StaticPosition*				
		Static Rover	+StandAlone +SBAS +DGPS +RTKFloat +RTKFixed +RTK all	auto Geodetic1 Geodetic2 Geodetic3 Geodetic4 Geodetic5 Cartesian1 Cartesian2 Cartesian3 Cartesian4 Cartesian5				

Use these commands to define/inquire the PVT mode of the receiver. The argument *Mode* specifies the general positioning mode. If `Rover` is selected, the receiver will assume that the receiver is moving and compute the best PVT allowed by the *RoverMode* argument. If `Static` is selected, the receiver will assume that it is fixed and will use the position defined by the *StaticPosition* argument.

The argument *RoverMode* specifies the allowed PVT modes when the receiver is operating in `Rover` mode. Different modes can combined with the "+" operator. Refer to the "Operation Details" chapter of the Firmware User Manual for a description of the PVT modes. The value `RTK` is an alias for `RTKFloat+RTKFixed`. When more than one mode is enabled in *RoverMode*, the receiver automatically selects the mode that provides the most accurate solution with the available data.

The position provided in the *StaticPosition* argument must be defined with the **`setStaticPosCartesian`** or the **`setStaticPosGeodetic`** commands. If the value `auto` is selected for this argument, the receiver will wait till a reliable PVT solution is available and it will use that solution as fixed position.

Examples

To configure an RTK rover using RTCM v2 corrections from COM2, use the following sequence:

```
COM1> sdio, COM2, RTCMv2 <CR>
$R: sdio, COM2, RTCMv2
  DataIntOut, COM2, RTCMv2, SBF+NMEA
COM1> spm, Rover, StandAlone+RTK <CR>
$R: spm, Rover, StandAlone+RTK
  PVTMode, Rover, StandAlone+RTK, auto, off
COM1>
```

To set up a fixed base station at a known location, use the following:

```
COM1> sspg, Geodetic1, 50.5209, 4.4245, 113.3 <CR>
$R: sspg, Geodetic1, 50.5209, 4.4245, 113.3
  StaticPosGeodetic, Geodetic1, 50.52090000, 4.42450000, 113.3000
COM1> spm, Static, , Geodetic1 <CR>
$R: spm, Static, , Geodetic1
  PVTMode, Static, StandAlone+RTK, Geodetic1, off
COM1>
```

srl grl	**setRAIMLevels** **getRAIMLevels**	*Mode*	*Pfa*	*Pmd*	*Reliability*			
		off on	-12 ... -4 ... -1	-12 ... -4 ... -1	-12 ... -3 ... -1			

Use these commands to define/inquire the parameters of the Receiver Autonomous Integrity Monitoring (RAIM) algorithm. Refer to the Firmware User Manual for a description of RAIM in your receiver.

The *Mode* argument acts as an on/off switch: it determines whether RAIM is active or not. If *Mode* is set to `off`, the test statistics are still computed and the results are available in the `RAIMStatistics` SBF block. However, the outcome of the tests has no effect on the positional output: there is no attempt to remove outliers from the PVT.

The *Pfa* argument sets the probability of false alarm of the w-test used in the "identification" step of the RAIM algorithm. Increasing this parameter increases the integrity but may reduce the availability of the positional solution.

The *Pmd* argument sets the probability of missed detection, which the receiver uses to compute the Minimal Detectable Bias and hence the XERL values.

The *Reliability* argument sets the probability of false alarm of the Overall Model test used in the "detection" step of the RAIM algorithm.

The value to be provided in the *Pfa*, *Pmd* and *Reliability* arguments are the base-10 logarithms of the desired probabilities. For instance, if you want a probability of false alarm of 1e-6, you have to set the *Pfa* argument to -6.

Examples

To configure the receiver outlier detection with a probability of 0.01% that a false alarm will be raised (type I error), a probability of 0.01% that an outlier will be missed (type II error) and an Overall Model reliability of 99.99%, use:

```
COM1> srl, on, -4, -4, -4 <CR>
$R: srl, on, -4, -4, -4
  RAIMLevels, on, -4, -4, -4
COM1>
```

To disable the outlier detection, use:

```
COM1> srl, off <CR>
$R: srl, off
  RAIMLevels, off, -4, -4, -4
COM1>
```

srd grd	**setReceiverDynamics** **getReceiverDynamics**	*Level*	*Motion*					
		Max High Moderate Low	Static Quasistatic Pedestrian Automotive Unlimited					

Use these commands to set/inquire the type of the dynamics the GNSS antenna is subjected to. The receiver adapts internal parameters for optimal performance for the selected type of dynamics.

The *Level* argument indicates the level of short-term acceleration and jerk the antenna is subjected to. That argument has effect on the navigation Kalman filter and on the tracking loops (only if the loops are in adaptive mode, see the **`setTrackingLoopParameters`** command). Setting this argument to `low` will reduce the noise on the GNSS measurements and PVT at the expense of filtering out rapid dynamic changes. Setting it to `high` will allow more dynamics to be visible at the expense of noise. The `max` level is primarily meant for test purposes: in that level, the Kalman filter is disabled and the receiver computes epoch-by-epoch independent PVT solutions.

Note that rapid displacements such as the ones caused by shocks, drops, oscillations or vibrations lead to high jerk values, even if the amplitude of the motion is not larger than a few centimeters. It is recommended to set *Level* to `high` for antennas subjected to that type of displacements.

The *Motion* argument defines the type of motion the antenna is subjected to.

Motion	Description
`Static`	Fixed base and reference stations.
`Quasistatic`	Low speed, limited area motion typical of surveying applications.
`Pedestrian`	Low speed (<7m/s) motion. E.g. pedestrians, low-speed land vehicles, ...
`Automotive`	Medium speed (<50m/s) motion. E.g. passenger cars, rail vehicles, ...
`Unlimited`	Unconstrained motion.

Example

```
COM1> srd, Max <CR>
$R: srd, Max
  ReceiverDynamics, Max, Automotive
COM1>
```

ernf grnf	**exeResetNavFilter** **getResetNavFilter**	***Level***						
		+PVT +AmbRTK all						

Use this command to reset the different navigation filters in the receiver. The user can reset each navigation filter independently or together with the value `all`.

The following values for *Level* are defined:

Level	Description
PVT	Reset the whole PVT filter such that all previous positioning information is discarded, including the RTK ambiguities and the INS/GNSS integration filter when applicable.
AmbRTK	Only reset the ambiguities used in RTK positioning to float status.

Example

```
COM1> ernf, PVT <CR>
$R: ernf, PVT
  ResetNavFilter, PVT
COM1>
```

ssu gsu	**setSatelliteUsage** **getSatelliteUsage**	*Satellite*						
		none +G01 ... G32 +R01 ... R24 +E01 ... E32 +S120 ... S138 +GPS +GLONASS +GALILEO +SBAS all						

Use these commands to define/inquire which satellites are allowed to be included in the PVT computation. It is possible to enable or disable a single satellite (e.g. `G01` for GPS PRN1), or a whole constellation. `Gxx`, `Exx`, `Rxx` and `Sxxx` refer to a GPS, Galileo, GLONASS and SBAS satellite respectively. GLONASS satellites must be referenced by their slot number (from 1 to 24) in this command.

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.

Examples

To only use GPS measurements in the PVT computation, use:

```
COM1> ssu, GPS <CR>
$R: ssu, GPS
  SatelliteUsage, G01+G02+G03+G04+G05+G06+G07+G08+G09+G10+G11
+G12+G13+G14+G15+G16+G17+G18+G19+G20+G21+G22+G23+G24+G25+G26
+G27+G28+G29+G30+G31+G32
COM1>
```

To add the usage of SBAS measurements in the PVT, use:

```
COM1> ssu, +SBAS <CR>
$R: ssu, +SBAS
  SatelliteUsage, G01+G02+G03+G04+G05+G06+G07+G08+G09+G10+G11
+G12+G13+G14+G15+G16+G17+G18+G19+G20+G21+G22+G23+G24+G25+G26
+G27+G28+G29+G30+G31+G32+S120+S121+S122+S123+S124+S125+S126
+S127+S128+S129+S130+S131+S132+S133+S134+S135+S136+S137+S138
COM1>
```

To remove the measurement of one satellite from the PVT, use:

```
COM1> ssu, -S120 <CR>
$R: ssu, -S120
  SatelliteUsage, G01+G02+G03+G04+G05+G06+G07+G08+G09+G10+G11
+G12+G13+G14+G15+G16+G17+G18+G19+G20+G21+G22+G23+G24+G25+G26
+G27+G28+G29+G30+G31+G32+S121+S122+S123+S124+S125+S126+S127
+S128+S129+S130+S131+S132+S133+S134+S135+S136+S137+S138
COM1>
```

ssbc gsbc	**setSBASCorrections** **getSBASCorrections**	*Satellite*	*SISMode*	*NavMode*	*DO229Version*			
		auto EGNOS WAAS MSAS S120 ... S138	Test Operational	EnRoute PrecApp	auto DO229C			

Use these commands to define/inquire the details on the usage of SBAS data in the PVT computation. This command does not define whether SBAS corrections are to be used or not in the PVT (this is done by the **`setPVTMode`** command), but it specifies how these corrections should be used.

The *Satellite* argument defines the provider of SBAS corrections, being either an individual satellite or a satellite system. If `EGNOS`, `WAAS` or `MSAS` is selected, the receiver restricts the automatic selection of a satellite to those that are part of the EGNOS, WAAS or MSAS system. When `auto` is selected for the *Satellite* argument, the receiver will automatically select a satellite on the basis of the location of the receiver and on the availability of SBAS corrections.

The *SISMode* argument defines the interpretation of a "Do Not Use for Safety Applications" message. When set to `Operational`, the receiver will discard all SBAS corrections received from a satellite upon reception of a MT00 from that satellite. Note that MT02 content encoded in a MT00 message will be interpreted by the receiver as a MT02 message: only MT00 with all '0' symbols will be interpreted as a true "Do Not Use for Safety Applications". When the argument *SISMode* is set to `Test`, the receiver will ignore the reception of a "Do Not Use for Safety Applications" message. This provides the possibility to use a signal from a SBAS system in test mode.

The DO 229 standard, which has its origin in aviation, makes a distinction between two positioning applications: en-route and precision approach. The choice between both applications influences the length of the interval during which the SBAS corrections are valid. During a precision approach the validity of the data is much shorter. The receiver can operate in both modes, which is controlled by the *NavMode* argument.

The *DO229Version* argument can be used to specify which version of the DO 229 standard to conform to.

Example

To force the receiver to use corrections from PRN 122 and ignore message MT00:

```
COM1> ssbc, S122, Test <CR>
$R: ssbc, S122, Test
  SBASCorrections, S122, Test, EnRoute, auto
COM1>
```

snu gnu	**setSignalUsage** **getSignalUsage**	*Signal*	*NavData*					
		+GPSL1CA +GPSL2PY +GPSL2C +GLOL1CA +GLOL2CA +GALL1BC +GEOL1 all	+GPSL1CA +GPSL2PY +GPSL2C +GLOL1CA +GLOL2CA +GALL1BC +GEOL1 all					

Use these commands to define/inquire which signals are allowed to be included in the PVT computation. This command has a similar role as **`setSignalTracking`**, but its effect is limited to the PVT engine. Removing an entry from the argument *Signal* will disable the usage of the corresponding range, phase & Doppler measurements in the PVT computation. Removing an entry from the argument *NavData* will disable the usage of the corresponding navigation information (ephemeris, ionosphere parameters ...).

Example

To force the receiver to only use the L1 GPS C/A measurements and navigation information in the PVT solution, use:

```
COM1> snu, GPSL1CA, GPSL1CA <CR>
$R: snu, GPSL1CA, GPSL1CA
  SignalUsage, GPSL1CA, GPSL1CA
COM1>
```

sspc gspc	**setStaticPosCartesian** **getStaticPosCartesian**	***Position*** *Position*	*X*	*Y*	*Z*	*Datum*		
		+Cartesian1 +Cartesian2 +Cartesian3 +Cartesian4 +Cartesian5 all	-8000000.0000 ... 0.0000 ... 8000000.0000 m	-8000000.0000 ... 0.0000 ... 8000000.0000 m	-8000000.0000 ... 0.0000 ... 8000000.0000 m	WGS84		

Use these commands to define/inquire a set of Cartesian coordinates. This command should be used in conjunction with the **setPVTMode** command to specify a base station position. The cartesian coordinates in the *X*, *Y* and *Z* arguments must refer to the antenna reference point (ARP), and not to the marker.

If the *Datum* argument is set to a different value than specified in the **setGeodeticDatum** command, the receiver will automatically convert the given position.

Example

To set up a static base station in Cartesian coordinates, use the following sequence:

```
COM1> sspc, Cartesian1, 4019952.028, 331452.954, 4924307.458 <CR>
$R: sspc, Cartesian1, 4019952.028, 331452.954, 4924307.458
  StaticPosCartesian, Cartesian1, 4019952.0280, 331452.9540, 4924307.4580,
WGS84
COM1> spm, Static, , Cartesian1 <CR>
$R: spm, Static, , Cartesian1
  PVTMode, Static, StandAlone+SBAS+DGPS+RTKFloat+RTKFixed, Cartesian1
COM1>
```

sspg gspg	**setStaticPosGeodetic** **getStaticPosGeodetic**	***Position*** *Position*	*Latitude*	*Longitude*	*Altitude*	*Datum*		
		+Geodetic1 +Geodetic2 +Geodetic3 +Geodetic4 +Geodetic5 all	-90.000000000 ... 0.000000000 ... 90.000000000 deg	-180.000000000 ... 0.000000000 ... 180.000000000 deg	-1000.0000 ... 0.0000 ... 30000.0000 m	WGS84		

Use these commands to define/inquire a set of geodetic coordinates. This command should be used in conjunction with the **`setPVTMode`** command to specify a base station position. The geodetic coordinates in the *Latitude*, *Longitude* and *Altitude* arguments must refer to the antenna reference point (ARP), and not to the marker.

If the *Datum* argument is set to a different value than specified in the **`setGeodeticDatum`** command, the receiver will automatically convert the given position.

Example

To set up a static base station in geodetic coordinates, use the following sequence:

```
COM1> sspg, Geodetic1, 50.86696443, 4.71347657, 114.880 <CR>
$R: sspg, Geodetic1, 50.86696443, 4.71347657, 114.880
  StaticPosGeodetic, Geodetic1, 50.86696443, 4.71347657, 114.8800, WGS84
COM1> spm, Static, , Geodetic1 <CR>
$R: spm, Static, , Geodetic1
  PVTMode, Static, StandAlone+SBAS+DGPS+RTKFloat+RTKFixed, Geodetic1
COM1>
```

sts gts	**setTimingSystem** **getTimingSystem**	*System*						
		GST GPS						

Use these commands to define/inquire the time system in which the receiver should operate. Galileo System Time (GST) is only supported on Galileo-enabled receivers. The selected *System* time will be used as reference time for clock bias computation.

Note that at least one satellite of the selected system (GPS or Galileo) must be visible and tracked by the receiver. Otherwise no PVT will be computed.

Examples

```
COM1> sts, GPS <CR>
$R: sts, GPS
  TimingSystem, GPS
COM1>

COM1> gts <CR>
$R: gts
  TimingSystem, GPS
COM1>
```

stm gtm	**setTroposphereModel** **getTroposphereModel**	*ZenithModel*	*MappingModel*					
		off Saastamoinen MOPS	Niell MOPS					

Use these commands to define/inquire the type of model used to correct tropospheric errors in the PVT computation.

The *ZenithModel* parameter indicates which model the receiver uses to compute the dry and wet delays for radio signals at 90 degree elevation. The modelled zenith tropospheric delay depends on assumptions for the local air total pressure, the water vapour pressure and the mean temperature. The following zenith models are defined:

ZenithModel	Description
`off`	The measurements will not be corrected for the troposphere delay. This may be desirable if the receiver is connected to a GNSS signal simulator.
`Saastamoinen`	Saastamoinen, J. (1973). "Contributions to the theory of atmospheric refraction". In three parts. Bulletin Géodésique, No 105, pp. 279-298; No 106, pp. 383-397; No. 107, pp. 13-34.
`MOPS`	Minimum Operational Performance Standards for Global Positioning/Wide Area Augmentation System Airborne Equipment RTCA/DO-229C, November 28, 2001.

The `Saastamoinen` model uses user-provided values of air temparature, total air pressure referenced to the Mean Sea Level and relative humidity (see **`setTroposphereParameters`** command) and estimates actual values adjusted to the receiver height.

The MOPS model neglects the user-provided values and instead assumes a seasonal model for all the climatic parameters. Local tropospheric conditions are estimated based on the coordinates and time of the year.

The use of the `Saastamoinen` model can be recommended if external information on temperature, pressure, humidity is available. Otherwise it is advisable to rely on climate models.

The zenith delay is mapped to the current elevation for each satellite using the requested *MappingModel*. The following mapping models are defined:

MappingModel	Description
`Niell`	Niell, A.E. (1996). Global Mapping Functions for the atmosphere delay at radio wavelengths, Journal of Geophysical Research, Vol. 101, No. B2, pp. 3227-3246.
`MOPS`	Minimum Operational Performance Standards for Global Positioning/Wide Area Augmentation System Airborne Equipment RTCA/DO-229C, November 28, 2001.

Examples

```
COM1> stm, MOPS, MOPS <CR>
$R: stm, MOPS, MOPS
  TroposhereModel, MOPS, MOPS
COM1>
```

```
COM1> gtm <CR>
$R: gtm
  TroposhereModel, MOPS, MOPS
COM1>
```

stp gtp	**setTroposphereParameters** **getTroposphereParameters**	*Temperature*	*Pressure*	*Humidity*				
		-100.0 ... 15.0 ... 100.0 degC	800.00 ... 1013.25 ... 1500.00 hPa	0 ... 50 ... 100 %				

Use these commands to define/inquire the climate parameters to be used when the zenith troposphere is estimated using the Saastamoinen model (see the **`setTroposphereModel`** command).

The troposphere model assumes the climate parameters to be valid for a receiver located at the Mean Sea Level (MSL). If you want to use your receiver with a weather station, you have to convert the measured *Temperature*, *Pressure* and *Humidity* to MSL.

Examples

```
COM1> stp, 25 <CR>
$R: stp, 25
  TroposhereParameters, 25.0, 1013.25, 50
COM1>

COM1> gtp <CR>
$R: gtp
  TroposhereParameters, 25.0, 1013.25, 50
COM1>
```

3.4. Receiver Operation Commands

scst gcst	**setClockSyncThreshold** **getClockSyncThreshold**	*Threshold*						
		ClockSteering usec500 msec1 msec2 msec3 msec4 msec5						

Use these commands to define/inquire the maximum allowed offset between the receiver internal clock and the system time defined by the **`setTimingSystem`** command.

If the argument `ClockSteering` is selected, the receiver internal clock is continuously steered to the system time to within a couple of nanoseconds.

If any other argument is selected, the internal clock is left free running, and synchronization with the system time is done through regular millisecond clock jumps. More specifically, when the receiver detects that the time offset is larger than *Threshold*, it initiates a clock jump of an integer number of milliseconds to re-synchronise its internal clock with the system time. These clock jumps have no influence on the generation of the xPPS pulses: the xPPS pulses are always maintained within a few nanoseconds from the requested time, regardless of the value of the *Threshold* argument.

Please refer to the Firmware User Manual for a more detailed description of the time keeping in your receiver.

Example

To enable clock steering, use:

```
COM1> scst, clocksteering <CR>
$R: scst, clocksteering
  ClockSyncThreshold, ClockSteering
COM1>
```

sep gep	**setEventParameters** **getEventParameters**	***Event*** *Event*	*Polarity*					
		+EventA all	Low2High High2Low					

Use these commands to define/inquire the polarity of the electrical transition on which the receiver will react on its `Event` input(s). The polarity of each event pin can be set individually or simultaneously by using the value `all` for the *Event* argument.

Example

```
COM1> sep, EventA, High2Low <CR>
$R: sep, EventA, High2Low
  EventParameters, EventA, High2Low
COM1>
```

slm glm	**setLEDMode** **getLEDMode**	*GPLED*						
		DIFFCORLED PVTLED TRACKLED						

Use these commands to define/inquire the blinking mode of the General Purpose LED (GPLED).

The different LED blinking modes are described in the Hardware Manual of your receiver.

Example

```
COM1> slm, DIFFCORLED <CR>
$R: slm, DIFFCORLED
  LEDMode, DIFFCORLED
COM1>
```

spps gpps	**setPPSParameters** **getPPSParameters**	*Interval*	*Polarity*	*Delay*	*TimeScale*	*MaxSyncAge*		
		off msec100 msec200 msec500 sec1 sec2 sec5 sec10	Low2High High2Low	-1000000.00 ... 0.00 ... 1000000.00 nsec	TimeSys UTC RxClock GLONASS	1 ... 60 ... 3600 sec		

Use these commands to define/inquire the interval, the polarity, the delay and time scale of the x-pulse-per-second (xPPS) output. Please refer to the Firmware User Manual for a detailed description of the xPPS functionality.

The *Interval* argument specifies the time interval between the pulses. A special value "`off`" is defined to disable the xPPS signal.

The *Polarity* argument defines the polarity of the xPPS signal.

The *Delay* argument can be used to compensate for the overall signal delays in the system (including antenna, antenna cable and xPPS cable). Setting *Delay* to a higher value causes the xPPS pulse to be generated earlier. For example, if the antenna cable is replaced by a longer one, the overall signal delay could be increased by, say, 20 nsec. If *Delay* is left unchanged, the xPPS pulse will come 20 nsec too late. To re-synchronize the xPPS pulse, *Delay* has to be increased by 20 nsec.

By default, the xPPS pulses are aligned with the satellite time system (*TimeSys*) that is defined by the **`setTimingSystem`** command. Using the *TimeScale* argument, it is also possible to align the xPPS pulse with the local receiver time (`RxClock`), with GLONASS time or with UTC.

When *TimeScale* is set to anything else than `RxClock`, the accuracy of the position of the xPPS pulse depends on the age of the last PVT computation. During PVT outages (due for instance to signal blockage), the xPPS position is extrapolated on the basis of the last available PVT information, and may start to drift. To avoid large biases, the receiver stops outputting the xPPS pulse when the last PVT is older than the age specified in the *MaxSyncAge* argument. *MaxSyncAge* is ignored when *TimeScale* is set to `RxClock`.

Examples

```
COM1> spps, sec1, , 23.4 <CR>
$R: spps, sec1, , 23.4
  PPSParameters, sec1, Low2High, 23.40, TimeSys, 60
COM1>

COM1> gpps <CR>
$R: gpps
  PPSParameters, sec1, Low2High, 23.40, TimeSys, 60
COM1>
```

3.5. Session Settings Commands

smp gmp	**setMarkerParameters** **getMarkerParameters**	*MarkerName (60)*	*MarkerNumber (20)*	*MarkerType (20)*				
		SEPT	Unknown	Unknown				

Use these commands to define/inquire the name, number and type of the monument marker.

The set of allowed characters for the *MarkerName* argument is:

```
_0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

When generating a RINEX observation file with the **sbf2rin** utility, the marker name, number and type are reflected in the header section and a "new site occupation" event is inserted between observation records each time the marker name or number is changed with this command.

Examples

```
COM1> smp, TestMarker, , GEODETIC <CR>
$R: smp, TestMarker, , GEODETIC
    MarkerParameters, "TestMarker", "Unknown", "GEODETIC"
COM1>

COM1> gmp <CR>
$R: gmp
    MarkerParameters, "TestMarker", "Unknown", "GEODETIC"
COM1>
```

soc goc	**setObserverComment** **getObserverComment**	*Comment (120)*						
		Unknown						

Use these commands to define/inquire the content of the `Comment` SBF block.

Examples

```
COM1> soc, "Data taken with choke ring antenna" <CR>
$R: soc, "Data taken with choke ring antenna"
    ObserverComment, "Data taken with choke ring antenna"
COM1>

COM1> goc <CR>
$R: goc
    ObserverComment, "Data taken with choke ring antenna"
COM1>
```

sop gop	**setObserverParameters** **getObserverParameters**	*Observer (20)*	*Agency (40)*					
		Unknown	Unknown					

Use these commands to define/inquire the observer name or ID, and his/her agency. These parameters are copied in the `ReceiverSetup` SBF block and in the header of RINEX observation files.

The length of the arguments complies with the RINEX format definition.

Examples

```
COM1> sop, TestObserver, TestAgency <CR>
$R: sop, TestObserver, TestAgency
    ObserverParameters, "TestObserver", "TestAgency"
COM1>

COM1> gop <CR>
$R: gop
    ObserverParameters, "TestObserver", "TestAgency"
COM1>
```

3.6. Input/Output Commands

scs gcs	**setCOMSettings** **getCOMSettings**	***Cd*** *Cd*	*Rate*	*DataBits*	*Parity*	*StopBits*	*FlowControl*	
		+COM1 +COM2 +COM3 all	baud1200 baud2400 baud4800 baud9600 baud19200 baud38400 baud57600 baud115200 baud230400 baud460800	bits8	No	bit1	none	

Use these commands to define/inquire the communication settings of the receiver's COM ports. By default, all COM ports are set to a baud rate of 115200 baud, using 8 data-bits, no parity, 1 stop-bit and no flow control.

In some housed products (e.g. PolaRx3 family), the number of COM ports configurable by this command is larger than the number of COM ports available to the user. The extra COM ports are used for internal purposes, and their settings should not be modified. Please refer to the Hardware Manual of your product for further details.

When modifying the settings of the current connection, make sure to also modify the settings of your terminal emulation program accordingly.

Examples

```
COM1> scs, COM1, baud19200 <CR>
$R: scs, COM1, baud19200
  COMSettings, COM1, baud19200, bits8, No, bit1, none
COM1>

COM1> gcs, COM1 <CR>
$R: gcs, COM1
  COMSettings, COM1, baud19200, bits8, No, bit1, none
COM1>
```

sdio gdio	**setDataInOut** **getDataInOut**	***Cd*** *Cd*	*Input*	*Output*	*(Show)*			
		+COM1 +COM2 +COM3 +USB1 +USB2 all	none CMD RTCMv2 RTCMv3 CMRv2 DC1 DC2 ASCIIIN	none +RTCMv2 +RTCMv3 +CMRv2 +SBF +NMEA +ASCIIDisplay +DC1 +DC2	(off) (on)			

Use these commands to define/inquire the type of data that the receiver should accept/send on a given connection descriptor (*Cd*).

The *Input* argument is used to tell the receiver how to interpret incoming bytes on the connection *Cd*. If the default value `CMD` is selected, the connection can be used to enter ASCII or SNMP' commands. If anything else is selected, the connection is blocked for command input. There are two ways to re-enable the command input on a blocked connection. The first way is to reconfigure the blocked connection by entering the command **`setDataInOut`** from another connection. The second way is to send a succession of ten "S" characters to the blocked connection within a time interval shorter than 5 seconds, and then wait for at least 10 seconds before sending any other character to that connection.

The *Output* argument is used to select the types of data allowed as output. The receiver supports outputting different data types on the same connection. The `ASCIIDisplay` is a textual report of the tracking and PVT status at a fixed rate of 1Hz. It can be used to get a quick overview of the receiver operation.

`DC1` and `DC2` represent two internal pipes that can be used to create a daisy-chain. Set the *Input* argument to `DCi` to connect the input of pipe *i* to the specified connection. Set the *Output* argument to `DCi` to connect the output of pipe *i* to the specified connection.

The following data types are not available in all receiver models. If they appear in the command syntax table on top of this page, they are available in your particular receiver. `MTI` represents the data stream coming from an MTi IMU sensor. `MMQ` is for the MMQ sensor from Systron Donner. `RTCMV` is the LBAS1-proprietary differential correction stream decoded from L-band.

After the *Cd*, *Input* and *Output* arguments, an extra read-only *Show* argument will be returned in the command reply. This last argument can take the value `on` or `off`, depending on whether the connection descriptor is open or not.

Examples

```
COM1> sdio, COM2, RTCMv2 <CR>
$R: sdio, COM2, RTCMv2
  DataInOut, COM2, RTCMv3, SBF+NMEA, (on)
COM1>
```

To setup a two-way daisy-chain between COM1 and COM2:

```
COM1> sdio, COM1, DC1, DC2 <CR>
$R: sdio, COM1, DC1, DC2
  DataInOut, COM1, DC1, DC2, (on)
COM1> sdio, COM2, DC2, DC1 <CR>
$R: sdio, COM2, DC2, DC1
  DataInOut, COM2, DC2, DC1, (on)
```

```
COM1>
```

eecm gecm	**exeEchoMessage** **getEchoMessage**	***Cd***	***Message (242)***	***EndOfLine***				
		COM1 COM2 COM3 USB1 USB2	A:Unknown	none +CR +LF all				

Use this command to send a message to one of the connections of the receiver.

The *Message* argument defines the message that should be sent on the *Cd* port. If the given message starts with "`A:`", the remainder of the message is considered an ASCII string that will be forwarded without changes to the requested connection. If the given message starts with "`H:`", the remainder of the message is considered a hexadecimal representation of a succession of bytes to be sent to the requested connection. In this case, the string should be a succession of 2-character hexadecimal values separated by a single whitespace.

Make sure to enclose the string in quotes if it contains whitespaces. The maximum length of the *Message* argument (including the `A:` or `H:` prefix) is 242 characters.

The *EndOfLine* argument defines which end-of-line character should be sent after the message. That argument is ignored when the *Message* argument starts with `H:`.

To send a message at a regular interval instead of once, use the command **`setPeriodicEcho`**.

Examples

To send the string "Hello world!" to COM2, use:

```
COM1> eecm, COM2, "A:Hello world!" <CR>
$R: eecm, COM2, "A:Hello world!"
  EchoMessage, COM2, "A:Hello world!", none
COM1>
```

To send the same string, the following command can also be used:

```
COM1> eecm, COM2, "H:48 65 6C 6C 6F 20 77 6F 72 6C 64 21" <CR>
$R: eecm, COM2, "H:48 65 6C 6C 6F 20 77 6F 72 6C 64 21"
  EchoMessage, COM2, "H:48 65 6C 6C 6F 20 77 6F 72 6C 64 21", none
COM1>
```

enoc gnoc	**exeNMEAOnce** **getNMEAOnce**	***Cd***	***Messages***					
		COM1 COM2 COM3 USB1 USB2	+ALM +DTM +GBS +GGA +GLL +GNS +GRS +GSA +GST +GSV +HDT +RMC +ROT +VTG +ZDA +HRP +LLQ +RBP +RBV +RBD					

Use this command to output a set of NMEA messages [2] on a given connection. This command differs from the related **`setNMEAOutput`** command in that it instructs the receiver to output the specified messages only once, instead of at regular intervals.

The *Cd* argument defines the connection descriptor on which the message(s) should be output and the *Messages* argument defines the list of messages that should be output. The HRP, RBP, RBD and RBV messages are Septentrio proprietary and are described in the Firmware User Manual.

Please make sure that the connection specified by *Cd* is configured to allow NMEA output (this is the default for all connections). See the **`setDataInOut`** command.

Example

To output the receiver position on COM1, use:

```
COM1> enoc, COM1, GGA <CR>
$R: enoc, COM1, GGA
  NMEAOnce, COM1, GGA
COM1>
```

sno gno	**setNMEAOutput** **getNMEAOutput**	***Stream*** *Stream*	*Cd*	*Messages*	*Interval*			
		+Stream1 ... Stream10 all	none COM1 COM2 COM3 USB1 USB2	none +ALM +DTM +GBS +GGA +GLL +GNS +GRS +GSA +GST +GSV +HDT +RMC +ROT +VTG +ZDA +HRP +LLQ +RBP +RBV +RBD	off OnChange msec10 msec20 msec40 msec50 msec100 msec200 msec500 sec1 sec2 sec5 sec10 sec15 sec30 sec60 min2 min5 min10 min15 min30 min60			

Use this command to output a set of NMEA messages [2] on a given connection at a regular interval. The *Cd* argument defines the connection descriptor on which the message(s) should be output and the *Messages* argument defines the list of messages that should be output. The HRP, RBP, RBD and RBV messages are Septentrio proprietary and are described in the Firmware User Manual.

This command is the counterpart of the **setSBFOutput** command for NMEA sentences. Please refer to the description of that command for a description of the arguments.

Examples

To output GGA at 1Hz and RMC at 10Hz on COM1, use:

```
COM1> sno, Stream1, COM1, GGA, sec1 <CR>
$R: sno, Stream1, COM1, GGA, sec1
  NMEAOutput, Stream1, COM1, GGA, sec1
COM1> sno, Stream2, COM1, RMC, msec100 <CR>
$R: sno, Stream2, COM1, RMC, msec100
  NMEAOutput, Stream2, COM1, RMC, msec100
COM1> sdio, COM1, , +NMEA <CR>
$R: sdio, COM1, , +NMEA
  DataInOut, COM1, CMD, SBF+NMEA (on)
COM1>
```

To get the list of NMEA messages currently output, use:

```
COM1> gno <CR>
$R: gno
  NMEAOutput, Stream1, COM1, GGA, sec1
  NMEAOutput, Stream2, COM1, RMC, msec100
  NMEAOutput, Stream3, none, none, off
  NMEAOutput, Stream4, none, none, off
  NMEAOutput, Stream5, none, none, off
  NMEAOutput, Stream6, none, none, off
  NMEAOutput, Stream7, none, none, off
```

```
  NMEAOutput, Stream8, none, none, off
  NMEAOutput, Stream9, none, none, off
  NMEAOutput, Stream10, none, none, off
COM1>
```

snp gnp	**setNMEAPrecision** **getNMEAPrecision**	*NrExtraDigits*						
		0 ... 3						

Use these commands to define/inquire the number of extra digits in the latitude, longitude and altitude reported in NMEA sentences.

By default (*NrExtraDigits* is 0), latitude and longitude are reported in degrees with 5 decimal digit, and altitude is reported in meters with 2 decimal digit. These default numbers of digits lead to a centimeter-level resolution of the position. To represent RTK positions with their full precision, it is recommended to set *NrExtraDigits* to 2.

It is important to note that increasing the number of digits (setting *NrExtraDigits* to a non-zero value) may cause the NMEA standard to be broken, as the total number of characters in a sentence may end up exceeding the prescribed limit of 82. This is why it is not done by default.

Examples

```
COM1> snp, 2 <CR>
$R: snp, 2
  NMEAPrecision, 2
COM1>

COM1> gnp <CR>
$R: gnp
  NMEAPrecision, 2
COM1>
```

snti gnti	**setNMEATalkerID** **getNMEATalkerID**	*TalkerID*						
		GP GN						

Use these commands to define/inquire the "Device Talker" for NMEA sentences. The device talker allows users to identify the type of equipment from which the NMEA sentence was issued.

This command has no effect on the ALM, GGA, GNS, GSV and ZDA sentences. For ALM, GGA and ZDA, the Device Talker is always set to `GP`. For GNS and GSV, it is set to `GP`, `GN` or `GL` depending on the contents, as per the NMEA standard.

Example

```
COM1> snti, GN <CR>
$R: snti, GN
  NMEATalkerID, GN
COM1>
```

spe gpe	**setPeriodicEcho** **getPeriodicEcho**	***Cd*** *Cd*	*Message (242)*	*Interval*				
		+COM1 +COM2 +COM3 +USB1 +USB2 all	A:Unknown	once off msec100 msec200 msec500 sec1 sec2 sec5 sec10 sec15 sec30 sec60 min2 min5 min10 min15 min30 min60				

Use this command to periodically send a message to one of the connections of the receiver.

The *Message* argument defines the message that should be sent on the *Cd* port. If the given message starts with "A:", the remainder of the message is considered an ASCII string that will be forwarded to the requested connection. All occurrences of the %%CR character sequence are replaced by a single carriage return character (ASCII code 13d) and all occurrences of the %%LF character sequence are replaced by a single line feed character (ASCII code 10d). If the *Message* argument starts with "H:", the remainder of the message is considered a hexadecimal representation of a succession of bytes to be sent to the requested connection. In this case, the string should be a succession of 2-character hexadecimal values separated by a single whitespace.

Make sure to enclose the string in quotes if it contains whitespaces. The maximum length of the *Message* argument (including the A: or H: prefix) is 242 characters.

The *Interval* argument defines the interval at which the message should be sent.

To send a message only once, set *Interval* to once. The only difference with the command **exeEchoMessage** is that **exeEchoMessage** cannot be stored in the boot configuration file, while **setPeriodicEcho** can. This can be used to output a message once at each reset or reboot. The third example below shows how to do this.

Examples

To send the string "Hello!<CR><LF>" to COM2 every minute, use:

```
COM1> spe, COM2, "A:Hello!%%CR%%LF", sec60 <CR>
$R: spe, COM2, "A:Hello!%%CR%%LF", sec60
  PeriodicEcho, COM2, "A:Hello!%%CR%%LF", sec60
COM1>
```

The same can be achieved with the following command:

```
COM1> spe, COM2, "H:48 65 6C 6C 6F 21 0D 0A", sec60 <CR>
$R: spe, COM2, "H:48 65 6C 6C 6F 21 0D 0A", sec60
  PeriodicEcho, COM2, "H:48 65 6C 6C 6F 21 0D 0A", sec60
COM1>
```

To let the receiver output the string "Hello!<CR><LF>" to COM2 at each reset, use the following command sequence:

```
COM1> spe, COM2, "A:Hello!%%CR%%LF", once <CR>
$R: spe, COM2, "A:Hello!%%CR%%LF", once
  PeriodicEcho, COM2, "A:Hello!%%CR%%LF", once
COM1> eccf, Current, Boot <CR>
$R: eccf, Current, Boot
  CopyConfigFile, Current, Boot
COM1>
```

ssgp gsgp	**setSBFGroups** **getSBFGroups**	***Group*** *Group*	*Messages*					
		+Group1 +Group2 +Group3 +Group4 all	none [SBF List] +Measurements +RawNavBits +GPS +GLO +GAL +GEO +PVTCart +PVTGeod +PVTExtra +Attitude +Time +Events +DiffCorr +Status +Rinex +Support +RawData +GUI					

Use these commands to define/inquire user-defined groups of SBF blocks that can be re-used in the **exeSBFOnce** and the **setSBFOutput** commands. The purpose of defining groups is to ease the typing effort when the same set of SBF blocks are to be addressed regularly.

The list of supported SBF blocks [SBF List] is to be found in Section 3.10, “SBF List”.

A number of predefined groups of SBF blocks are available (such as `Measurements` or `RawNavBits`). See the command **setSBFOutput** for a description of these predefined groups.

Example

To output the messages `MeasEpoch`, `PVTCartesian` and `DOP` as one group on COM1 at a rate of 1Hz, you could use the following sequence of commands:

```
COM1> ssgp, Group1, MeasEpoch+PVTCartesian+DOP <CR>
$R: ssgp, Group1, MeasEpoch+PVTCartesian+DOP
  SBFGroups, Group1, MeasEpoch+PVTCartesian+DOP
COM1> sso, Stream1, COM1, Group1, sec1 <CR>
$R: sso, Stream1, COM1, Group1, sec1
  SBFOutput, Stream1, COM1, Group1, sec1
COM1> sdio, COM1, , +SBF <CR>
$R: sdio, COM1, , +SBF
  DataInOut, COM1, CMD, SBF+NMEA, (on)
COM1>
```

esoc gsoc	**exeSBFOnce** **getSBFOnce**	***Cd***	***Messages***					
		COM1 COM2 COM3 USB1 USB2	[SBF List] +Measurements +GPS +GLO +GAL +GEO +PVTCart +PVTGeod +PVTExtra +Attitude +Time +Status +UserGroups +Rinex +Support +RawData +GUI					

Use this command to output a set of SBF blocks on a given connection. This command differs from the related **`setSBFOutput`** command in that it instructs the receiver to output the specified SBF blocks only once, instead of at regular intervals.

The *Cd* argument defines the connection descriptor on which the message(s) should be output and the *Messages* argument defines the list of messages that should be output. The list of SBF blocks [SBF List] is to be found in Section 3.10, “SBF List”. Only a subset of SBF blocks can be sent with the **`exeSBFOnce`** command: refer to the SBF Reference Guide for a list of them.

Make sure that the connection specified by *Cd* is configured to allow SBF output (this is the default for all connections). See also the **`setDataInOut`** command.

Predefined groups of SBF blocks (such as `Measurements` or `PVTCart`) can be addressed in the *Messages* argument. These groups are defined in the table below.

Alias	Description
`Measurements`	+MeasEpoch+MeasExtra+IQCorr+EndOfMeas
`GPS`	+GPSNav+GPSAlm+GPSIon+GPSUtc
`GLO`	+GLONav+GLOAlm+GLOTime
`GAL`	+GALNav+GALAlm+GALIon+GALUtc+GALGstGps
`GEO`	+GEONav+GEOAlm
`PVTCart`	+PVTCartesian+PosCovCartesian+VelCovCartesian+BaseVectorCart
`PVTGeod`	+PVTGeodetic+PosCovGeodetic+VelCovGeodetic+BaseVectorGeod
`PVTExtra`	+DOP+PVTSatCartesian+PVTResiduals+RAIMStatistics+GEOCorrections+BaseLine+EndOfPVT
`Attitude`	+AttEuler+AttCovEuler+EndOfAtt
`Time`	+ReceiverTime
`Status`	+SatVisibility+ChannelStatus+ReceiverStatus+InputLink+OutputLink
`UserGroups`	+Group1+Group2+Group3+Group4
`Rinex`	+MeasEpoch+GPSNav+GPSIon+GPSUtc+GLONav+GALNav+GALUtc+GALGstGps+GEONav+PVTGeodetic+ReceiverSetup+Comment
`Support`	+MeasEpoch+MeasExtra+EndOfMeas+GPSNav+GPSAlm+GPSIon+GPSUtc+GLONav+GLOAlm+GLOTime+GAL-

Alias	Description
	Nav+GALAlm+GALIon+GALUtc+GALGstGps+GEONav+GEOAlm+PVTGeodetic+BaseVectorGeod+AttEuler+DOP+EndOfPVT+ChannelStatus+ReceiverStatus+InputLink+OutputLink+ReceiverSetup+Commands
RawData	+MeasEpoch+MeasExtra+GPSNav+GLONav+GALNav+GEONav+PVTGeodetic+ReceiverSetup+Commands+Comment
GUI	+MeasEpoch+PVTGeodetic+PosCovGeodetic+VelCovGeodetic+DOP+PVTSatCartesian+PVTResiduals+RAIMStatistics+BaseLine+AttEuler+ReceiverTime+ReceiverStatus+InputLink+OutputLink+ReceiverSetup+Comment

Example

To output the next `MeasEpoch` block, use:

```
COM1> esoc, COM1, MeasEpoch <CR>
$R: esoc, COM1, MeasEpoch
  SBFOnce, COM1, MeasEpoch
COM1>
```

sso gso	**setSBFOutput** **getSBFOutput**	***Stream*** *Stream*	*Cd*	*Messages*	*Interval*			
		+Stream1 ... Stream10 +Res1 +Res2 +Res3 +Res4 all	none COM1 COM2 COM3 USB1 USB2	none [SBF List] +Measurements +RawNavBits +GPS +GLO +GAL +GEO +PVTCart +PVTGeod +PVTExtra +Attitude +Time +Event +DiffCorr +Status +UserGroups +Rinex +Support +RawData +GUI	off OnChange msec10 msec20 msec40 msec50 msec100 msec200 msec500 sec1 sec2 sec5 sec10 sec15 sec30 sec60 min2 min5 min10 min15 min30 min60			

Use this command to output a set of SBF blocks on a given connection at a regular interval.

A *Stream* is defined as a list of messages that should be output with the same interval on one connection descriptor (*Cd*). In other words, one *Stream* is associated with one *Cd* and one *Interval*, and contains a list of SBF blocks defined by the *Messages* argument.

The list of supported SBF blocks [SBF List] is to be found in Section 3.10, “SBF List”.

Predefined groups of SBF blocks (such as `Measurements` or `RawNavBits`) can be addressed in the *Messages* argument. These groups are defined in the table below.

Alias	Description
`Measurements`	+MeasEpoch+MeasExtra+IQCorr+EndOfMeas
`RawNavBits`	+GPSRawCA+GPSRawL2C+GLORawCA+GAL-RawINAV+GEORawL1
`GPS`	+GPSNav+GPSAlm+GPSIon+GPSUtc
`GLO`	+GLONav+GLOAlm+GLOTime
`GAL`	+GALNav+GALAlm+GALIon+GALUtc+GALGstGps
`GEO`	+GEOMT00+GEOPRNMask+GEOFastCorr+GEOIntegri-ty+GEOFastCorrDegr+GEONav+GEODegrFactors+GEONet-workTime+GEOAlm+GEOIGPMask+GEOLongTerm-Corr+GEOIonoDelay+GEOServiceLevel+GEOClockEphCov-Matrix
`PVTCart`	+PVTCartesian+PosCovCartesian+VelCovCartesian+BaseVec-torCart
`PVTGeod`	+PVTGeodetic+PosCovGeodetic+VelCovGeodetic+BaseVec-torGeod
`PVTExtra`	+DOP+PVTSatCartesian+PVTResiduals+RAIMStatis-tics+GEOCorrections+BaseLine+EndOfPVT
`Attitude`	+AttEuler+AttCovEuler+EndOfAtt
`Time`	+ReceiverTime+xPPSOffset
`Event`	+ExtEvent+ExtEventPVTCartesian+ExtEventPVTGeodetic
`DiffCorr`	+DiffCorrIn+BaseStation

Alias	Description
Status	+SatVisibility+ChannelStatus+ReceiverStatus+InputLink+OutputLink
UserGroups	+Group1+Group2+Group3+Group4
Rinex	+MeasEpoch+GEORawL1+GPSNav+GPSIon+GPSUtc+GLONav+GALNav+GALUtc+GALGstGps+GEONav+PVTGeodetic+ReceiverSetup+Comment
Support	+MeasEpoch+MeasExtra+EndOfMeas+GPSRawCA+GPSRawL2C+GLORawCA+GALRawINAV+GEORawL1+GPSNav+GPSAlm+GPSIon+GPSUtc+GLONav+GLOAlm+GLOTime+GALNav+GALAlm+GALIon+GALUtc+GALGstGps+GEONav+GEOAlm+PVTGeodetic+BaseVectorGeod+AttEuler+DOP+EndOfPVT+ExtEvent+DiffCorrIn+BaseStation+ChannelStatus+ReceiverStatus+InputLink+OutputLink+ReceiverSetup+Commands
RawData	+MeasEpoch+MeasExtra+GPSRawCA+GPSRawL2C+GLORawCA+GALRawINAV+GEORawL1+GPSNav+GLONav+GALNav+GEONav+PVTGeodetic+DiffCorrIn+ReceiverSetup+Commands+Comment
GUI	+MeasEpoch+PVTGeodetic+PosCovGeodetic+VelCovGeodetic+DOP+PVTSatCartesian+PVTResiduals+RAIMStatistics+BaseLine+AttEuler+ReceiverTime+BaseStation+ReceiverStatus+InputLink+OutputLink+ReceiverSetup+Comment

The *Interval* argument defines the rate at which the SBF blocks specified in the *Messages* argument are output. If set to `off`, the SBF blocks are disabled. If set to `OnChange`, the SBF blocks are output at their natural renewal rate. Please refer to the "Output Rate" section of the SBF Reference Guide for further details. If a specific interval is specified (e.g. `sec1` corresponds to an interval of 1 second), the SBF blocks are decimated from their renewal rate to the specified interval. Some blocks can only be output at their renewal rate (e.g. the `GPSNav` block). For these blocks, the receiver ignores any decimation interval and always assumes `OnChange`. The list of those blocks can be found in the SBF Reference Guide.

Please make sure that the connection specified by *Cd* is configured to allow SBF output (this is the default for all connections). See the **`setDataInOut`** command.

`Res1` to `Res4` are reserved values of *Stream* for Septentrio's GUIs. If you are never using Septentrio tools to control your receiver, you are free to use these entries as any other *Stream*. It is worth mentioning that these streams are not saved in the configuration files and, as a consequence, they will always be reset at boot time. For most users, it is not recommended to use these streams.

Examples

To output the `MeasEpoch` block at 1Hz and the `PVTCartesian` block at 10Hz on COM1, use the following sequence:

```
COM1> sso, Stream1, COM1, MeasEpoch, sec1 <CR>
$R: sso, Stream1, COM1, MeasEpoch, sec1
  SBFOutput, Stream1, COM1, MeasEpoch, sec1
COM1> sso, Stream2, COM1, PVTCartesian, msec100 <CR>
$R: sso, Stream2, COM1, PVTCartesian, msec100
  SBFOutput, Stream2, COM1, PVTCartesian, msec100
```

```
COM1> sdio, COM1, , +SBF <CR>
$R: sdio, COM1, , +SBF
  DataInOut, COM1, CMD, SBF+NMEA, (on)
COM1>
```

To get the list of SBF blocks currently output, use:

```
COM1> gso <CR>
$R: gso
  SBFOutput, Stream1, COM1, MeasEpoch, sec1
  SBFOutput, Stream2, COM1, PVTCartesian, msec100
  SBFOutput, Stream3, none, none, off
  SBFOutput, Stream4, none, none, off
  SBFOutput, Stream5, none, none, off
  SBFOutput, Stream6, none, none, off
  SBFOutput, Stream7, none, none, off
  SBFOutput, Stream8, none, none, off
  SBFOutput, Stream9, none, none, off
  SBFOutput, Stream10, none, none, off
  SBFOutput, Res1, none, none, off
  SBFOutput, Res2, none, none, off
  SBFOutput, Res3, none, none, off
  SBFOutput, Res4, none, none, off
COM1>
```

3.7. RTCM v2.x Commands

sr2c gr2c	**setRTCMv2Compatibility** **getRTCMv2Compatibility**	*PRCType*	*GLOToD*					
		Standard GroupDelay	Tk Tb					

Use these commands to define/inquire the compatibility of the RTCM 2.x input correction stream. This command applies to rover receivers only and should be used in case the available base station correction stream is not fully compatible with the latest version of the RTCM 2.x standard.

The argument *PRCType* is used to handle a difference in the interpretation of DGPS corrections between the version 2.0 of the RTCM standard and later versions. If the base station is sending RTCM Message Type 1 based on version 2.0, the value `GroupDelay` must be selected to have a correct usage of incoming corrections.

The argument *GLOToD* specifies how to interpret the time-of-day field in the differential GLONASS correction message (MT31). Select `Tb` to be compatible with RTCM version up to 2.2, and select `Tk` to be compatible with RTCM 2.3 and later.

Example

To make to rover receiver compatible with a base station sending RTCM 2.2 corrections, use:

```
COM1> sr2c, , Tb <CR>
$R: sr2c, , Tb
  RTCMv2Compatibility, Standard, Tb
COM1>
```

sr2f gr2f	**setRTCMv2Formatting** **getRTCMv2Formatting**	*ReferenceID*						
		0 ... 1023						

Use these commands to define/inquire the reference station ID assigned to the receiver when operating in base station mode. The reference station ID is transmitted in the first word of each outgoing RTCM v2.x message.

Examples

```
COM1> sr2f, 345 <CR>
$R: sr2f, 345
  RTCMv2Formatting, 345
COM1>

COM1> gr2f <CR>
$R: gr2f
  RTCMv2Formatting, 345
COM1>
```

sr2i gr2i	**setRTCMv2Interval** **getRTCMv2Interval**	***Message*** *Message*	*ZCount*					
		+RTCM1 +RTCM3 +RTCM9 +RTCM16 +RTCM22 +RTCM23|24 all	1 ... 2 ... 1000					

Use these commands to define/inquire at which interval the RTCM v2.x messages specified in the *Message* argument should be generated. The related **setRTCMv2IntervalObs** command must be used to specify the interval of some RTK-related messages such as messages 18 and 19.

The interval for every message is given in the *ZCount* argument, in units of 0.6 seconds. For example, to generate a message every 6 seconds, *ZCount* should be set to 10.

The intervals specified with this command are not connection-specific: all the connections which output a given RTCM v2.x message will output it with the same interval.

Note that this command only defines the interval of RTCM messages. To make the receiver actually output these messages, use the **setRTCMv2Output** and **setDataInOut** commands.

Examples

```
COM1> sr2i, RTCM22, 15 <CR>
$R: sr2i, RTCM22, 15
  RTCMv2Interval, RTCM22, 15
COM1>

COM1> gr2i <CR>
$R: gr2i
  RTCMv2Interval, RTCM1, 2
  RTCMv2Interval, RTCM3, 2
  RTCMv2Interval, RTCM16, 2
  RTCMv2Interval, RTCM22, 15
  RTCMv2Interval, RTCM23|24, 2
COM1>
```

sr2b gr2b	**setRTCMv2IntervalObs** **getRTCMv2IntervalObs**	***Message*** *Message*	*Interval*					
		+RTCM18|19 +RTCM20|21 all	1 ... 600 sec					

Use these commands to define/inquire at which interval the RTCM v2.x messages specified in the *Message* argument should be generated. The related **`setRTCMv2Interval`** command must be used to specify the interval of other supported RCTCM v2.x messages.

The intervals specified with this command are not connection-specific: all the connections which output a given RTCM v2.x message will output it with the same interval.

Note that this command only defines the interval of RTCM messages. To make the receiver actually output these messages, use the **`setRTCMv2Output`** and **`setDataInOut`** commands.

Examples

```
COM1> sr2b, RTCM20|21, 2 <CR>
$R: sr2b, RTCM20|21, 2
  RTCMv2IntervalObs, RTCM20|21, 2
COM1>

COM1> gr2b <CR>
$R: gr2b
  RTCMv2IntervalObs, RTCM18|19, 1
  RTCMv2IntervalObs, RTCM20|21, 2
COM1>
```

sr2m gr2m	**setRTCMv2Message16** **getRTCMv2Message16**	*Message (90)*						
		Unknown						

Use these commands to define/inquire the string that will be transmitted in the RTCM v2.x message 16. The argument *Message* can contain up to 90 characters.

Note that this command only defines the content of message 16. To make the receiver actually output this message, use the **`setRTCMv2Output`** and **`setDataInOut`** commands.

Example

To send the string "Hello" in message 16 over COM2 at the default interval, use the following sequence:

```
COM1> sr2m, Hello <CR>
$R: sr2m, Hello
  RTCMv2Message16, "Hello"
COM1> sr2o, COM2, RTCM16 <CR>
$R: sr2o, COM2, RTCM16
  RTCMv2Output, COM2, RTCM16
COM1> sdio, COM2, , RTCMv2 <CR>
$R: sdio, COM2, , RTCMv2
  DataInOut, COM2, CMD, RTCMv2
COM1>
```

sr2o gr2o	**setRTCMv2Output** **getRTCMv2Output**	***Cd*** *Cd*	***Messages***					
		+COM1 +COM2 +COM3 +USB1 +USB2 all	none +RTCM1 +RTCM3 +RTCM9 +RTCM16 +RTCM18|19 +RTCM20|21 +RTCM22 +RTCM23|24 +DGPS +RTK all					

Use these commands to define/inquire which RTCM v2.x messages are enabled for output on a given connection descriptor (*Cd*). The *Messages* argument specifies the RTCM message types to be enabled. Some pairs of messages are always enabled together, such as messages 18 and 19. `DGPS` is an alias for "`RTCM1+RTCM3`" and `RTK` is an alias for "`RTCM3+RTCM18|19+RTCM22`".

Please make sure that the connection specified by *Cd* is configured to allow RTCMv2 output, which can be done with the **`setDataInOut`** command. The interval at which each message is output is to be specified with the **`setRTCMv2Interval`** or the **`setRTCMv2IntervalObs`** command.

Example

To enable RTCM v2.x messages 3, 18, 19 and 22 on COM2, use the following sequence:

```
COM1> sr2o, COM2, RTCM3+RTCM18|19+RTCM22 <CR>
$R: sr2o, COM2, RTCM3+RTCM18|19+RTCM22
  RTCMv2Output, COM2, RTCM3+RTCM18|19+RTCM22
COM1> sdio, COM2, , RTCMv2 <CR>
$R: sdio, COM2, , RTCMv2
  DataInOut, COM2, CMD, RTCMv2
COM1>
```

sr2u gr2u	**setRTCMv2Usage** **getRTCMv2Usage**	*MsgUsage*						
		none +RTCM1 +RTCM3 +RTCM9 +RTCM15 +RTCM18|19 +RTCM20|21 +RTCM22 +RTCM23|24 +RTCM31 +RTCM32 +RTCM59 all						

Use this command to restrict the list of incoming RTCM v2.x messages that the receiver is allowed to use in its differential PVT computation.

Example

To only accept RTCM1 and RTCM3 corrections from the base station 1011, use the following sequence:

```
COM1> sr2u, RTCM1+RTCM3 <CR>
$R: sr2u, RTCM1+RTCM3
  RTCMv2Usage, RTCM1+RTCM3
COM1> sdcu, , , manual, 1011 <CR>
$R: sdcu, , , manual, 1011
  DiffCorrUsage, LowLatency, 3600.0, manual, 1011, 20, 20000000
COM1>
```

3.8. RTCM v3.x Commands

sr3f gr3f	**setRTCMv3Formatting** **getRTCMv3Formatting**	*ReferenceID*						
		0 ... 4095						

Use these commands to define/inquire the reference station ID assigned to the receiver when operating in base station mode. The reference station ID is transmitted in the header of each outgoing RTCM v3.x message.

Examples

```
COM1> sr3f, 345 <CR>
$R: sr3f, 345
  RTCMv3Formatting, 345
COM1>

COM1> gr3f <CR>
$R: gr3f
  RTCMv3Formatting, 345
COM1>
```

sr3i gr3i	**setRTCMv3Interval** **getRTCMv3Interval**	***Message*** *Message*	*Interval*					
		+RTCM1001|2 +RTCM1003|4 +RTCM1005|6 +RTCM1007|8 +RTCM1009|10 +RTCM1011|12 +RTCM1013 +RTCM1033 all	0.1 ... 1.0 ... 600.0 sec					

Use these commands to define/inquire at which interval RTCM v3.x messages should be generated.

The intervals specified with this command are not connection-specific: all the connections which output a given RTCM v3.x message will output it with the same interval.

Note that this command only defines the interval of RTCM messages. To make the receiver actually output these messages, use the **`setRTCMv3Output`** and **`setDataInOut`** commands.

Example

```
COM1> sr3i, RTCM1001|2, 2 <CR>
$R: sr3i, RTCM1001|2, 2
  RTCMv3Interval, RTCM1001|2, 2
COM1>
```

sr3o gr3o	**setRTCMv3Output** **getRTCMv3Output**	***Cd*** *Cd*	*Messages*					
		+COM1 +COM2 +COM3 +USB1 +USB2 all	none +RTCM1001 +RTCM1002 +RTCM1003 +RTCM1004 +RTCM1005 +RTCM1006 +RTCM1007 +RTCM1008 +RTCM1009 +RTCM1010 +RTCM1011 +RTCM1012 +RTCM1013 +RTCM1033 all					

Use these commands to define/inquire which RTCM v3.x messages are enabled for output on a given connection descriptor (*Cd*). The *Messages* argument specifies the RTCM message types to be enabled.

Please make sure that the connection specified by *Cd* is configured to allow RTCMv3 output, which can be done with the **`setDataInOut`** command. The interval at which each message is output is to be specified with the **`setRTCMv3Interval`** command.

Example

To enable RTCM v3.x messages 1001, 1002, 1005 and 1006 on COM2, use the following sequence:

```
COM1> sr3o, COM2, RTCM1001+RTCM1002+RTCM1005+RTCM1006 <CR>
$R: sr3o, COM2, RTCM1001+RTCM1002+RTCM1005+RTCM1006
  RTCMv3Output, COM2, RTCM1001+RTCM1002+RTCM1005+RTCM1006
COM1> sdio, COM2, , RTCMv3 <CR>
$R: sdio, COM2, , RTCMv3
  DataInOut, COM2, CMD, RTCMv3
COM1>
```

sr3u gr3u	**setRTCMv3Usage** **getRTCMv3Usage**	*MsgUsage*						
		none +RTCM1001 ... RTCM1012 +RTCM1033 all						

Use this command to restrict the list of incoming RTCM v3.x messages that the receiver is allowed to use in its differential PVT computation.

Example

To only accept RTCM1001 and RTCM1002 corrections from the base station 1011, use the following sequence:

```
COM1> sr3u, RTCM1001+RTCM1002 <CR>
$R: sr3u, RTCM1001+RTCM1002
  RTCMv3Usage, RTCM1001+RTCM1002
COM1> sdcu, , , manual, 1011 <CR>
$R: sdcu, , , manual, 1011
  DiffCorrUsage, LowLatency, 3600.0, manual, 1011, 20, 20000000
COM1>
```

3.9. CMR v2.0 Commands

sc2f gc2f	**setCMRv2Formatting** **getCMRv2Formatting**	*ReferenceID*						
		0 ... 31						

Use these commands to define/inquire the reference station ID assigned to the receiver when operating in base station mode. The reference station ID is transmitted in the header of each outgoing CMR v2.0 message.

Examples

```
COM1> sc2f, 12 <CR>
$R: sc2f, 12
  CMRv2Formatting, 12
COM1>

COM1> gc2f <CR>
$R: gc2f
  CMRv2Formatting, 12
COM1>
```

sc2i gc2i	**setCMRv2Interval** **getCMRv2Interval**	***Message*** *Message*	*Interval*					
		+CMR0 +CMR1 +CMR2 +CMR3 all	0.1 ... 1.0 ... 600.0 sec					

Use these commands to define/inquire at which interval CMR v2.0 messages should be generated.

The intervals specified with this command are not connection-specific: all the connections which output a given CMR v2.0 message will output it with the same interval.

Note that this command only defines the interval of CMR messages. To make the receiver actually output these messages, use the **`setCMRv2Output`** and **`setDataInOut`** commands.

Examples

```
COM1> sc2i, CMR0, 2 <CR>
$R: sc2i, CMR0, 2
  CMRv2Interval, CMR0, 2
COM1>

COM1> gc2i <CR>
$R: gc2i CMRv2Interval, CMR0, 2
  CMRv2Interval, CMR1, 1 CMRv2Interval, CMR2, 1
COM1>
```

sc2m gc2m	**setCMRv2Message2** **getCMRv2Message2**	*ShortID (8)*	*LongID (50)*	*COGO (16)*				
		Unknown	Unknown	Unknown				

Use these commands to define/inquire the strings that will be transmitted in the CMR v2.0 message 2.

The argument *ShortID* is the short station ID. It can contain up to 8 characters in compliance with the CMR standard. If less than 8 characters are defined, the string will be right justified and padded with spaces.

The argument *LongID* is the long station ID. It can contain up to 50 characters in compliance with the CMR standard. If less than 50 characters are defined, the string will be right justified and padded with spaces. Some CMR implementations use the character "@" in the long name. As this character is not allowed in a command argument, the character "%" should be used instead. The receiver will automatically replace all occurrences of "%" in *LongID* with "@" when CMR2 message is output.

The argument *COGO* is the COGO code. It can contain up to 16 characters in compliance with the CMR standard. If less than 16 characters are defined, the string will be right justified and padded with spaces.

Note that this command only defines the contents of message 2. To make the receiver actually output this message, use the **`setCMRv2Output`** and **`setDataInOut`** commands.

Example

To send the string "Hello" as short station ID and send CMR2 messages through COM2, use the following sequence:

```
COM1> sc2m, Hello <CR>
$R: sc2m, Hello
  CMRv2Message2, "Hello", "Unknown", "Unknown"
COM1> sc2o, COM2, CMR2 <CR>
$R: sc2o, COM2, CMR2
  CMRv2Output, COM2, CMR2
COM1> sdio, COM2, , CMRv2 <CR>
$R: sdio, COM2, , CMRv2
  DataInOut, COM2, CMD, CMRv2
COM1>
```

sc2o gc2o	**setCMRv2Output** **getCMRv2Output**	***Cd*** *Cd*	*Messages*					
		+COM1 +COM2 +COM3 +USB1 +USB2 all	none +CMR0 +CMR1 +CMR2 +CMR3 all					

Use these commands to define/inquire which CMR v2.0 messages are enabled for output on a given connection descriptor (*Cd*). The *Messages* argument specifies the CMR message types to be enabled.

Please make sure that the connection specified by *Cd* is configured to allow CMRv2 output, which can be done with the **`setDataInOut`** command. The interval at which each message is output is to be specified with the **`setCMRv2Interval`** command.

Example

To enable CMR v2.0 message 0 on COM2, use the following sequence:

```
COM1> sc2o, COM2, CMR0 <CR>
$R: sc2o, COM2, CMR0
  CMRv2Output, COM2, CMR0
COM1> sdio, COM2, , CMRv2 <CR>
$R: sdio, COM2, , CMRv2
  DataInOut, COM2, CMD, CMRv2
COM1>
```

sc2u gc2u	**setCMRv2Usage** **getCMRv2Usage**	*MsgUsage*						
		none +CMR0 +CMR1 +CMR3 +CMR0p +CMR0w all						

Use this command to restrict the list of incoming CMR v2.0 messages that the receiver is allowed to use in its differential PVT computation. `CMR0p` and `CMR0w` refer to the CMR+ and CMR-W variants respectively.

Example

To only accept CMR0 from the base station 12, use the following sequence:

```
COM1> sc2u, CMR0 <CR>
$R: sc2u, CMR0
  CMRv2Usage, CMR0
COM1> sdcu, , , manual, 12 <CR>
$R: sdcu, , , manual, 12
  DiffCorrUsage, LowLatency, 3600.0, manual, 12, 20, 20000000
COM1>
```

3.10. SBF List

ASCIIIn
AttCovEuler
AttEuler
BaseLine
BaseStation
BaseVectorCart
BaseVectorGeod
ChannelStatus
Commands
Comment
DiffCorrIn
DOP
EndOfAtt
EndOfMeas
EndOfPVT
ExtEvent
ExtEventPVTCartesian
ExtEventPVTGeodetic
GALAlm
GALGstGps
GALIon
GALNav
GALRawINAV
GALUtc
GEOAlm
GEOClockEphCovMatrix
GEOCorrections
GEODegrFactors
GEOFastCorr
GEOFastCorrDegr
GEOIGPMask
GEOIntegrity
GEOIonoDelay
GEOLongTermCorr
GEOMT00
GEONav
GEONetworkTime
GEOPRNMask
GEORawL1
GEOServiceLevel
GLOAlm
GLONav
GLORawCA
GLOTime
GPSAlm
GPSIon
GPSNav
GPSRawCA
GPSRawL2C
GPSUtc
Group1
Group2
Group3
Group4
InputLink
IQCorr
MeasEpoch
MeasExtra
OutputLink
PosCart
PosCovCartesian
PosCovGeodetic
PVTCartesian
PVTGeodetic
PVTResiduals
PVTSatCartesian
RAIMStatistics
ReceiverSetup
ReceiverStatus
ReceiverTime
SatVisibility
VelCovCartesian
VelCovGeodetic
xPPSOffset

Appendix A. Error Messages

The following table lists the possible ASCII error messages and their meaning.

Error message	Description
Invalid command!	Syntax error or unsupported command.
Argument 'xxx' can't be omitted!	Omission of a mandatory argument.
At least one non tabular argument needed!	Omission of a mandatory argument.
Argument 'xxx' is invalid!	Value out of range, or too many decimal digits.
Argument 'xxx' could not be handled!	Argument impossible to parse or invalid combination of values.
ASCII commands between prompts were discarded!	Argument impossible to parse or invalid combination of values.

Appendix B. List of Acronyms

APME	A-Posteriori Multipath Estimator
ARP	Antenna Reference Point
C/A code	Coarse/Acquisition code
C/N0	Carrier-to-Noise ratio
<CR>	Carriage Return (ASCII code 13)
DGPS	Differential GPS
DHCP	Dynamic Host Configuration Protocol
DLL	Delay Locked Loop
EGNOS	European Geostationary Navigation Overlay Service
GNSS	Global Navigation Satellite System
GP	General Purpose
GPS	Global Positioning System
INS	Inertial Navigation System
IMU	Inertial Measurement Unit
LBAS	L-Band Augmentation Service
<LF>	Line Feed (ASCII code 10)
MT	Message Type
NMEA	National Marine Electronics Association
PRN	Pseudorandom Noise
PPS	Pulse(s) per Second
PVT	Position, Velocity and Time solution
P(Y) code	Precision code, or the Anti-Spoofing encrypted version
RAIM	Receiver Autonomous Integrity Monitoring
RINEX	Receiver Independent Exchange format
RTCM	Radio Technical Commission For Maritime Services
RTK	Real Time Kinematic
SBAS	Space-Based Augmentation System
SBF	Septentrio Binary Format
SIS	Signal In Space
WAAS	Wide Area Augmentation System

Index of Commands

A

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T