GREIS

GNSS Receiver External Interface Specification

Reflects Firmware Version 4.6.00

Last revised: June 10, 2025

Table of Contents

PREFACE

Thank you for purchasing your JAVAD GNSS receiver. The materials available in this Reference Guide (the "Guide") have been prepared by JAVAD GNSS, Inc. for owners of JAVAD GNSS products. It is designed to assist owners with the use of the receiver and its use is subject to these terms and conditions (the "Terms and Conditions").

Terms and Conditions

PROFESSIONAL USE – JAVAD GNSS receivers are designed to be used by a professional. The user is expected to have a good knowledge and understanding of the user and safety instructions before operating, inspecting or adjusting. Always wear the required protectors (safety shoes, helmet, etc.) when operating the receiver.

DISCLAIMER OF WARRANTY – EXCEPT FOR ANY WARRANTIES IN THIS GUIDE OR A WARRANTY CARD ACCOMPANYING THE PRODUCT, THIS GUIDE AND THE RECEIVER ARE PROVIDED "AS-IS." THERE ARE NO OTHER WARRANTIES. JAVAD GNSS DISCLAIMS ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR USE OR PURPOSE. JAVAD GNSS AND ITS DISTRIBUTORS SHALL NOT BE LIABLE FOR TECHNICAL OR EDITORIAL ERRORS OR OMISSIONS CONTAINED HEREIN; NOR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM THE FURNISHING, PERFORMANCE OR USE OF THIS MATERIAL OR THE RECEIVER. SUCH DISCLAIMED DAMAGES INCLUDE BUT ARE NOT LIMITED TO LOSS OF TIME, LOSS OR DESTRUCTION OF DATA, LOSS OF PROFIT, SAVINGS OR REVENUE, OR LOSS OF THE PRODUCT'S USE. IN ADDITION, JAVAD GNSS IS NOT RESPONSIBLE OR LIABLE FOR DAMAGES OR COSTS INCURRED IN CONNECTION WITH OBTAINING SUBSTITUTE PRODUCTS OR SOFTWARE, CLAIMS BY OTHERS, INCONVENIENCE, OR ANY OTHER COSTS. IN ANY EVENT, JAVAD GNSS SHALL HAVE NO LIABILITY FOR DAMAGES OR OTHERWISE TO YOU OR ANY OTHER PERSON OR ENTITY IN EXCESS OF THE PURCHASE PRICE FOR THE RECEIVER.

LICENSE AGREEMENT – Use of any computer programs or software supplied by JAVAD GNSS or downloaded from a JAVAD GNSS website (the "Software") in connection with the receiver constitutes acceptance of these Terms and Conditions in this Guide and an agreement to abide by these Terms and Conditions. The user is granted a personal, non-exclusive, non-transferable license to use such Software under the terms

stated herein and in any case only with a single receiver or single computer. You may not assign or transfer the Software or this license without the express written consent of JAVAD GNSS. This license is effective until terminated. You may terminate the license at any time by destroying the Software and Guide. JAVAD GNSS may terminate the license if you fail to comply with any of the Terms or Conditions. You agree to destroy the Software and Guide upon termination of your use of the receiver. All ownership, copyright and other intellectual property rights in and to the Software belong to JAVAD GNSS. If these license terms are not acceptable, return any unused software and guide.

CONFIDENTIALITY – This guide, its contents and the Software (collectively, the "Confidential Information") are the confidential and proprietary information of JAVAD GNSS. You agree to treat JAVAD GNSS' Confidential Information with a degree of care no less stringent that the degree of care you would use in safeguarding your own most valuable trade secrets. Nothing in this paragraph shall restrict you from disclosing Confidential Information to your employees as may be necessary or appropriate to operate or care for the receiver. Such employees must also keep the Confidentiality Information confidential. In the event you become legally compelled to disclose any of the Confidential Information, you shall give JAVAD GNSS immediate notice so that it may seek a protective order or other appropriate remedy.

WEBSITE; OTHER STATEMENTS – No statement contained at the JAVAD GNSS website (or any other website) or in any other advertisements or JAVAD GNSS literature or made by an employee or independent contractor of JAVAD GNSS modifies these Terms and Conditions (including the Software license, warranty and limitation of liability).

SAFETY – Improper use of the receiver can lead to injury to persons or property and/or malfunction of the product. The receiver should only be repaired by authorized JAVAD GNSS warranty service centers.

MISCELLANEOUS – The above Terms and Conditions may be amended, modified, superseded, or canceled, at any time by JAVAD GNSS. The above Terms and Conditions will be governed by, and construed in accordance with, the laws of the State of California, without reference to conflict of laws.

INTRODUCTION

1.1 What is GREIS

GREIS is an interfacing language enabling user to effectively communicate with GNSS receivers by accessing all of their capabilities and functions.

GREIS represents a generic receiver language structure for the entire range of JAVAD GNSS hardware. This language structure is receiver-independent and open to future modification or expansion. GREIS is based on a unified approach allowing the user to control a JAVAD GNSS receiver using an appropriate set of *named objects*. Communication with these objects is achieved through predefined *commands* and *messages*. There are no specific constraints on the number or type of the receiver objects used.

1.2 How is GREIS Used

Any system communicating with the JAVAD GNSS receiver through one of its ports (serial, parallel, USB, Ethernet, etc.) will use GREIS commands and messages to accomplish the required task. A pair of typical applications where GREIS plays a very important role are, first, using hand-held controllers to communicate with the receivers during field operation in survey and RTK projects or, second, when downloading data from the receivers into desktop systems for further post processing. A post processing application itself doesn't use GREIS commands, but needs to be aware of GREIS messages to extract data from the data files.

One important feature of GREIS is that it can be effectively used both for the automatic and manual control of JAVAD GNSS receivers. For manual control, the user will enter necessary GREIS commands into the receiver through a terminal. This is easily achievable as GREIS is designed to be human-readable text interface. On the other hand, GREIS obeys rather strict rules that makes it easy to use by applications.

1.3 Lists

GREIS heavily utilizes a concept of *lists*. Lists are used both in the receiver input language and in the standard text messages.

Lists in GREIS are represented by a sequence of elements delimited by comma (`,`, ASCII code 44), and enclosed in braces (`{}`, ASCII codes 123 and 125):

```
{element1,element2,element3}
```

In turn, elements of a list may themselves be lists:

```
{e1,{ee21,ee22},e3}
```

Thus the above definition is recursive, so that lists of arbitrary nesting depth are allowed. Elements that are not lists are called *leaf elements*, or simply *leafs*. Elements of lists could be empty, in which case we say the element is *omitted*. For example, in the list below, second element is omitted:

```
{e1,,e3}
```

Spaces before and after delimiters are allowed and ignored.

If elements of a list all have the same substring (prefix) at the beginning, this substring could be moved out of the braces surrounding the list, e.g.,

```
elem{1,2,3}
```

is a shorter form of the

```
{elem1,elem2,elem3}
```

Elements could be enclosed into double-quotes (", ASCII code 34) that are stripped during parsing. Inside quoted element, special symbols (braces, commas, etc.) loose their role and are considered to be regular characters. Another use of quotes is to distinguish between “element is not specified” and “empty element specified” conditions. The former is denoted by simply omitting an element from the list, and the latter is denoted by putting pair of double-quotes between the commas. Quoting is also useful when one needs to have leading or trailing spaces in a string.

To put double-quote into element, quote this element and escape the double-quote inside with the backslash character (\, ASCII code 92). To put backslash by itself into quoted string, escape it with another backslash, for example:

Example:

```
"String with \"quotes\", backslash \\, and special characters, {}"
```

1.4 Objects

In the context of the model that GREIS is based on, a JAVAD GNSS receiver is identified with a set of *named objects*.

Object is defined as a hardware or software entity of the receiver's that can be addressed, set, or queried. Hardware entities are commonly referred to as *devices*, whereas firmware objects are normally *files* and *parameters*. Receiver ports and memory modules are all good examples of devices. All devices, files and parameters are treated in a uniform way by GREIS. Every object has an associated set of attributes that can be accessed, defined, and/or changed through GREIS.

1.4.1 Object Identifiers

It has been already mentioned that a receiver is considered as a set of objects (devices, files, messages, parameters, etc.) in the context of the GREIS model. For the purposes of addressing the objects in the receiver commands, a unique identifier should be assigned to every object.

Objects in the receiver are logically organized into groups. A group itself is also an object and belongs to another group unless it is the root group. Thus all objects in the receiver are organized into a tree-like hierarchy starting at the single root group. This representation resembles the organization of files into directories (folders) that most computer users are familiar with.

In GREIS, object groups are represented as *lists* of corresponding object names. The object name is unique inside the list to which the object belongs. Globally unique object identifier is defined as all the object names on the path through the object tree from the root list to the object, delimited by the forward slash (/). The root list itself is identified by the single forward slash.

Examples of object identifiers are:

Example: The root group:

```
/
```

Example: Receiver electronic ID:

```
/par/rcv/id
```

Example: Serial Port A baud rate:

```
/par/dev/ser/a/rate
```

Example: Attributes (size and last modification time) of the file `NAME` (file attributes are different from object attributes discussed below):

```
/log/NAME
```

Example: NMEA GGA sentence:

```
/msg/nmea/GGA
```

All the objects have one or more attributes associated with them. Object attributes are identified by appending the `&` character and the attribute name to the object identifier. The primary attribute each object has is `value`. This attribute is always accessed implicitly by GREIS commands. Some of objects may have additional attributes, for example:

Example: Serial port A default baud rate:

```
/par/dev/ser/a/rate&def
```

Example: Contents of the file `NAME`:

```
/log/NAME&content
```

1.4.2 Object Types

Every object in the receiver has GREIS type associated with it. The type of an object defines its behavior with respect to GREIS commands. Specifically, the type defines which values the object can take and which particular commands are applicable to the object.

Refer to “Primary Object Types” on page 184 for detailed description of currently supported object types.

1.5 Periodic Output

An important role in the receiver operation plays its ability to periodically output some information, such as different kinds of measurements, calculated values, etc., according to specified schedule. GREIS defines a rich set of *messages* containing different types of information in different formats that are minimal units of output, and provides methods to request *periodic output* of any combination of the messages in any order to any of the supported media suitable for data output. Any supported medium suitable for data output is called *output stream* in GREIS.

For every output stream, receiver maintains a list of messages that are currently enabled to be output to the stream, called *output list*. The order in which messages are output, matches the order of messages in the output list. In addition, every message that is present in an output list has its own set of *scheduling parameters* associated with it. Scheduling parameters attached to a message in an output list define the schedule of output of this particular message into this particular output stream. GREIS provides three com-

mands, em, out, and dm, to allow for efficient manipulation of output lists and scheduling parameters.

Message scheduling parameters comprise four fields: period, phase, count, and flags, each of which plays different role in the output schedule definition. Below we will describe how exactly their values affect the output, but basically, the period specifies interval between outputs of the message; phase specifies time shift of the moments of output with respect to time moments when current time is multiple of period; the count, when greater than zero, limits the number of times the message will be output; whereas flags filed allows for some fine tuning of the output process.

1.5.1 Output Period and Phase

The period and phase fields of the message scheduling parameters are floating point values in the range [0…86400) seconds. Their exact meaning is described below.

Note: When the F_CHANGE bit is set in the flags field of the scheduling parameters, the phase field looses its usual role and becomes "forced output period" instead. See description of the F_CHANGE flag below for details.

The receiver has its internal time grid that is defined by the *receiver clock* and the value of the /par/raw/curmsint parameter that defines the *step* of receiver *internal epochs*. Receiver internal epochs occur when *receiver time* is multiple of the *step*. In turn, receiver time is defined as the value of receiver clock modulo one day (86400 seconds). Receiver scans the output lists only at internal receiver epochs, so that no output could be generated more frequently than that.

Taking into account the internal time grid, the *period* and *phase* variables define the time moments of the output of a message as follows: receiver will output the message only at the receiver times T_{out} simultaneously satisfying the following two equations:

$$\begin{cases} T_{out}(\mathrm{mod}\ \ period) = phase & (1) \\ T_{out} = N \cdot step & (2) \end{cases}$$

where N is integer number taking the values [0,1,2,…,(86400/step)-1].

The first equation defines the basic rule of messages output, and the second one imposes additional constraints related to the internal receiver epochs. Note that in the most usual case, when both *period* and *phase* are multiples of *step*, the second equation is satisfied automatically whenever the first equation is satisfied. Also note that if

$$86400\ (\mathrm{mod}\ period) \neq 0,$$

the actual interval between the last message sent before the day rollover and the first message after the day rollover will be different from the value of *period*.

Consider a couple of examples illustrating this mechanism:

Example: Suppose *period* is 10s, *phase* is 2.2s, and *step* is 0.2s. As T_{out}, according to the second equation, can take only values that are multiple of *step*, the left part of the first equation will take the following values: 0, 0.2, 0.4, …, 9.8, 0, …, from which only value 2.2 matches *phase*. These matches will occur, and the message will be output, every time T_{out} takes one of the following values: 2.2s, 12.2s, 22.2s, etc.

Example: Suppose *period* is 10s, *phase* is 2.2s, and *step* is 0.5s. The receiver will not output the message since the above pair of simultaneous equations is never satisfied.

Example: Suppose *phase* > *period*. The receiver won't output the message at all as the first equation will never be satisfied.

1.5.2 Output Count

The `count` field of the message scheduling parameters is an integer value in the range `[-256…32767)` and serves two different purposes:

1. When the `count` is `0`, unlimited number of messages will be output. When the `count` is greater than `0`, it defines how many times the message will be output. In this case the counter is decremented by `1` every time the message is output, and when it becomes `0`, the `F_DISABLED` bit is set in the flags field. The message scheduler doesn't output messages with `F_DISABLED` bit set.
2. When the `count` is set to a value in the range `[-256…-1]`, the output of the message is not suppressed, and the `count` field serves entirely different purpose. It enables wrapping of the message into special `[>>]` message before output (see "[>>] Wrapper" on page 132). The value of `count` is then used to set the `id` field in the generated `[>>]` message so that the `id` is numerically equal to `(-1 - count)`.

Note: The wrapping feature is useful, for example, for a server application that gets messages from receiver and forwards them to multiple clients. It can request wrapping of arbitrary messages into the `[>>]` messages with different identifiers, unwrap the received messages, and dispatch the data to particular client(s) based on the received `id`. Utilizing this feature, such an application doesn't need to be aware of any other data formats but the format of the `[>>]` message, and can use single channel of communication with the receiver to get and dispatch messages in different formats.

1.5.3 Output Flags

The `flags` field of the message scheduling parameters is a 16-bit wide bit-field. Each bit of this bit field is a separate flag and serves different purpose. The following is a list of the message scheduling flags.

Table 1-1. Message Scheduling Flags

Bit#	HEX	Name
0	0x0001	F_OUT
1	0x0002	F_CHANGE
2	0x0004	F_OUT_ON_ADD
3	0x0008	F_NOTENA
4	0x0010	F_FIX_PERIOD
5	0x0020	F_FIX_PHASE
6	0x0040	F_FIX_COUNT
7	0x0080	F_FIX_FLAGS
8	0x0100	reserved
9	0x0200	reserved
10	0x0400	reserved
11	0x0800	F_DISABLED
12–15	0xF000	reserved

Note: Field names are introduced here only for the purpose of referring to them in this manual. There is no way to use them in the GREIS commands.

`F_OUT` - If this flag is set, the first messages after invocation of the corresponding command will be output at the internal receiver epoch closest to the command execution time no matter what is specified by the `period` scheduling parameter.

`F_CHANGE` - If this flag is set, the corresponding message will be output only if the message data have changed since the last output of the message to the given output stream. Receiver checks whether the message data have changed only at the moments defined by the equations (1),(2) where *phase* variable is set to zero, and *period* variable is set to the value of `period` field. The message scheduling parameter `phase`, which loses its original function in this case, now plays the role of a *forced output period.* “Forced output” means that the corresponding message will be output whether its contents will have changed or not at the time moments defined by the equations (1),(2) where *period* variable is set to the value of the `phase` field, and *phase* variable is set to zero. If the field `phase` is zero, then the receiver performs no forced output so that the corresponding message will be output only on condition that its data have changed.

F_OUT_ON_ADD – If this flag is set, then the first message will be output immediately after executing the corresponding em or out command. This flag is ignored for majority of messages[1].

F_NOTENA – If this flag is set for a message in an output list, the F_DISABLED flag for this message won't be cleared when the message is enabled, and therefore its output will remain suspended. For example, this flag is used in order not to output some of the messages from the default set of messages when the user changes output period on the fly, without first disabling the output.

F_FIX_PERIOD, F_FIX_PHASE, F_FIX_COUNT, F_FIX_PERIOD – Being set to 1 in a scheduling parameters, prevent changes to corresponding field(s) of this scheduling parameters through em and out commands.

F_DISABLED – Is not explicitly programmable by the user. When one enables a message with a positive count, then, after this message has been output count times, the message scheduler sets this flag to 1. This flag is cleared to 0 when the message is re-enabled, unless F_NOTENA flag is set for this message.

1. Currently only two GREIS messages, [JP] and [MF], honor this flag.

RECEIVER INPUT LANGUAGE

This chapter describes the syntax and semantics of the receiver input language. We begin with some examples to give the reader a feeling of the language, then turn to detailed syntax definition, and then describe all the defined commands along with their semantics.

2.1 Language Examples

Here are a few examples of real statements receiver understands along with receiver replies. You will find more examples of using particular commands in corresponding subsections. The input to the receiver is marked with the ⇒ character, while receiver output is marked with the ⇐ character:

Example: Ask receiver to print its electronic ID. Receiver generates the reply message shown:

```
⇒ print,/par/rcv/id<CR>
⇐ RE00C QP01234TR45<CR><LF>
```

Example: Ask receiver to set the baud rate of its serial port A to 9600. Receiver successfully executes the command and doesn't generate any reply.

```
⇒ set,/par/dev/ser/a/rate,9600<LF>
```

Example: Use the same command as in the previous example, but force receiver to generate reply by means of using the *statement identifier*.

```
⇒ %set_rate%set,/par/dev/ser/a/rate,9600<LF>
⇐ RE00A%set_rate%<CR><LF>
```

Example: Try to set too high baud rate. Receiver replies with the error message even though we used no statement identifier.

```
⇒ set,/par/dev/ser/a/rate,1000000<LF>
⇐ ER016{4,value out of range}<CR><LF>
```

●

Note: Receiver always puts its normal and error replies into two standard messages, [RE] and [ER], respectively. For more information on the format of GREIS messages, refer to "General Format of Messages" on page 64. The [RE] and [ER] messages themselves are described in "Interactive Messages" on page 129.

2.2 Language Syntax

GREIS defines *lines* of ASCII characters of arbitrary length[1], delimited by either carriage-return (<CR>, ASCII decimal code 13), or line-feed (<LF>, ASCII decimal code 10) characters, to be the top-level syntax elements of the language. Empty lines are allowed and ignored in GREIS. As a consequence, a line could be delimited by any combination of <CR> and/or <LF> characters. It allows GREIS to seamlessly support Windows™, Mac™, and UNIX™ line ending conventions.

Receiver input language is *case-sensitive*. It means that, for example, strings `GREIS`, `greis`, and `gReIs`, being different strings, are indeed considered as such by the receiver.

The number sign (`#`, ASCII code 35) is the comment introduction character. Receiver ignores everything starting from this character up to the end of the line.

After comment (if any) is stripped from the line, receiver removes leading and trailing spaces, and then breaks the line into *statements*. Statements are delimited with semicolon (`;`, ASCII code 59), or with two ampersands (`&&`, ASCII codes 38), or with two vertical bars (`||`, ASCII codes 124). Statements in a line are then executed in order, from left to right. If statement that ends in `&&` delimiter produces an error, the rest of statements in the line are not executed. If statement that ends in `||` delimiter executes successfully, the rest of statements in the line are not executed. Statement that ends in semicolon never stops execution of the sequence of statements. Note that the end of line is by itself statement terminator, so you don't need to put one of explicit statement delimiters at the end of the line.

The format of a statement is as follows:

```
[%ID%][COMMAND][@CS]
```

where square brackets denote optional fields, and any number of whitespaces is allowed before and after every field. Such whitespaces are ignored, except for the purpose of checksum calculation, see below. The fields are:

`%ID%` – statement identifier, where `ID` denotes arbitrary string, possibly empty. The identifier, if present, is copied unchanged by the receiver into the response message for the statement. Any statement with an identifier will always generate a response from the receiver. A statement that contains only an identifier is also allowed; in such a case, the receiver will just generate a response message.

`COMMAND` – a (possibly empty) *list* where the first element is called *command name*. It denotes the action to be performed. The rest of elements (if any) are command

1. Current GREIS implementation in the receivers supports lines of up to 256 characters in length.

arguments. Braces that surround command list could be omitted. Refer to "Lists" on page 19 for the syntax of lists.

`@CS` – checksum, where `CS` is 8-bit checksum formatted as 2-byte hexadecimal number. Before executing a statement with checksum, the receiver will compare the input checksum `CS` against that computed by the firmware and will refuse to execute the statement should these checksums mismatch. Checksum is computed starting with the statement's first non-blank character until and including the `@` character. See "Computing Checksums" on page 577 for details.

Statement identifier, `%ID%`, serves the following purposes:

1. Forces receiver response to the command.
2. Allows to send multiple commands with different identifiers to the receiver without waiting for response for every command, then receive the responses and tell which response corresponds to which command.
3. Helps to establish synchronization with the receiver by allowing to check that particular receiver response corresponds to particular command, and not to some other command issued before or after.

A list called *options* could be appended to any element of the `COMMAND` after the colon (`:`, ASCII code 58). If options list comprises single element, the surrounding braces could be omitted. Options list appended to a list propagates to every element of the list, though the options explicitly appended to an element of the list take precedence over propagated options. For example,

```
{e1,{e2:{o1,,o3},e3}}:{o4,o5}
```

is equivalent to:

```
{e1:{o4,o5},{e2:{o1,o5,o3},e3:{o4,o5}}}
```

Note also how missed `o2` option allows `o5` option to propagate to the list of options for `e2` element.

The number and the meaning of arguments and options in the command depends on particular command action and is defined in the description of every receiver command. In addition, if command description specifies some options, but some or all of them are missed in the statement, the default values for the missed options are substituted. The default values for options are also defined in the description of every receiver command.

For reference, below is the table comprising all the character sequences that have special meaning in the receiver input language:

Table 2-1. Input Language Special Characters

Characters	Decimal ASCII code	Meaning
<LF>	10	line separator
<CR>	13	line separator
#	35	beginning of comment mark
;	59	statements separator
&&	38	statements *and* separator
||	124	statements *or* separator
%	37	statement identifier mark
@	64	checksum mark
{	123	beginning of list mark
}	125	end of list mark
,	44	list elements separator
:	58	options mark
"	34	quotation mark
\	92	escape

2.3 Commands

In this section we describe all the commands defined in GREIS. Syntax and semantics specifications of every command are accompanied by explanatory examples. For detailed description of objects used as arguments in the examples, please refer to Chapter 4 on page 181.

2.3.1 set

Name

`set` - set value of an object.

Synopsis

Format: `set,object,value`
Options: `none`

Arguments

`object` - the target object identifier. If `object` does not begin with "/", then "`/par/`" prefix is automatically inserted before the `object` prior to executing the command.

`value` - the value to be assigned to the target object. The range of allowed values as well as semantics of the assignment depends on the type of the object and is specified later in this manual for every supported object.

Options

None.

Description

This command assigns `value` to the `object`. No response is generated unless there is an error or response is forced by the statement identifier.

Examples

Example: Set baud rate of serial port C to 115200. Either of:

```
⇒ set,/par/dev/ser/c/rate,115200
⇒ set,dev/ser/c/rate,115200
```

Example: Set baud rate of serial port A to 9600 and force reply:

```
⇒ %%set,dev/ser/a/rate,9600
⇐ RE002%%
```

●

2.3.2 print

Name

`print` - print value of an object.

Synopsis

Format: `print,object`
Options: `{names}`

Arguments

`object` - the object identifier of the object to be output. If `object` does not begin with "/", then "`/par/`" prefix is automatically inserted before the `object` prior to executing the command.

Options

Table 2-2. `print` options summary

Name	Type	Values	Default
names	boolean	`on,off`	`off`

`names` - if `off`, output only object values. When `on`, output object names in addition to object values in the format `NAME=VALUE`.

Description

This command prints value of the `object`, optionally prefixing the value with the name of corresponding object. The response is always generated, and more than one [RE] message could be generated in response to a single `print` command.

The value of an object of type *list* is printed as a list of values for every object in the list. This is applied recursively until leaf objects are reached, so printing an object of non-leaf type effectively outputs entire sub-tree starting from the specified `object`. In case of printing of lists, multiple [RE] messages could be generated. However, splitting of the output may occur only immediately after list separator characters.

Examples

Example: Print current period of the internal receiver time grid. Either of:

```
⇒ print,/par/raw/curmsint
⇐ RE004 100
⇒ print,raw/curmsint
⇐ RE004 100
```

Example: Print current period of the internal receiver time grid along with the object name. Either of:

```
⇒ print,/par/raw/curmsint:on
⇐ RE015/par/raw/curmsint=100
⇒ print,raw/curmsint:on
⇐ RE015/par/raw/curmsint=100
```

Example: Print receiver version information:

```
⇒ print,rcv/ver
⇐ RE028{"2.5 Sep,13,2006 p2",0,71,MGGDT_5,none,
⇐ RE00D {none,none}}
```

Example: Print receiver version information along with corresponding names:

```
⇒ print,rcv/ver:on
⇐ RE043/par/rcv/ver={main="2.5 Sep,13,2006 p2",boot=0,hw=71,board=MGGDT_5,
⇐ RE00C modem=none,
⇐ RE017 pow={fw=none,hw=none}}
```

Example: Print all the messages enabled for output to serial port B along with their scheduling parameters:

```
⇒ print,out/dev/ser/b:on
⇐ RE02D/par/out/dev/ser/b={jps/RT={1.00,0.00,0,0x0},
⇐ RE01A jps/SI={1.00,0.00,0,0x0},
⇐ RE01A jps/rc={1.00,0.00,0,0x0},
⇐ RE01A jps/ET={1.00,0.00,0,0x0},
⇐ RE01D nmea/GGA={10.00,5.00,0,0x0}}
```

2.3.3 list

Name

`list` - list contents of an object.

Synopsis

Format: `list[,object]`
Options: `none`

Arguments

`object` - the object identifier of the object to be output. If `object` is omitted, `/log` is assumed. If `object` does not begin with "/", then "`/log/`" prefix is automatically inserted before the `object` prior to executing the command.

Options

None.

Description

This command outputs names of every member of the `object`. The response is always generated, and more than one [RE] message could be generated in response to a single `list` command.

If the `object` specified is not of type *list*, empty [RE] message is generated. If the object specified is a *list,* the list of names of every object in the list is printed. This is applied recursively until leaf objects are reached, so listing an object of non-leaf type effectively outputs entire sub-tree starting from the specified `object`. In case of printing of lists, multiple [RE] messages could be generated. However, splitting of the output may occur only immediately after list separator characters.

Examples

Example: Empty reply for listing of a non-list object:

```
⇒ list,/par/rcv/ver/main
⇐ RE000
```

Example: Error reply for listing of non-existing object:

```
⇒ list,/does_not_exist
⇐ ER018{2,,wrong 1st parameter}
```

Example: Obtain a list of existing log-files. Either of

```
⇒ list,/log
⇒ list
```

will produce the same output, e.g.:

```
⇐ RE013{log1127a,log1127b}
```

Example: List all standard GREIS messages supported by the receiver:

```
⇒ list,/msg/jps
⇐ RE03D{JP,MF,PM,EV,XA,XB,ZA,ZB,YA,YB,RT,RD,ST,LT,BP,TO,DO,OO,UO,GT,
⇐ RE040 NT,GO,NO,TT,PT,SI,NN,EL,AZ,SS,FC,RC,rc,PC,pc,CP,cp,DC,CC,cc,EC,
⇐ RE040 CE,TC,R1,P1,1R,1P,r1,p1,1r,1p,D1,C1,c1,E1,1E,F1,R2,P2,2R,2P,r2,
⇐ RE040 p2,2r,2p,D2,C2,c2,E2,2E,F2,ID,PV,PO,PG,VE,VG,DP,SG,BI,SE,SM,PS,
⇐ RE040 GE,NE,GA,NA,WE,WA,WO,GS,NS,rE,rM,rV,rT,TM,MP,TR,MS,DL,TX,SP,SV,
⇐ RE031 RP,RK,BL,AP,AB,re,ha,GD,LD,RM,RS,IO,NP,LH,EE,ET}
```

Example: List all the messages in the default set of messages:

```
⇒ list,/msg/def
⇐ RE040{jps/JP,jps/MF,jps/PM,jps/EV,jps/XA,jps/XB,jps/RT,jps/RD,jps/SI,
⇐ RE040 jps/NN,jps/EL,jps/FC,jps/RC,jps/DC,jps/EC,jps/TC,jps/CP,jps/1R,
⇐ RE040 jps/1P,jps/2R,jps/2P,jps/E1,jps/D2,jps/E2,jps/SS,jps/SE,jps/PV,
⇐ RE040 jps/ST,jps/DP,jps/TO,jps/DO,jps/UO,jps/IO,jps/GE,jps/NE,jps/GA,
⇐ RE01D jps/NA,jps/WE,jps/WA,jps/WO}
```

2.3.4 em & out

Name

em, out - enable periodic output of messages.

Synopsis

Format: em,[target],messages
Format: out,[target],messages
Options: {period, phase, count, flags}

Arguments

target - any output stream or message set. If no target is specified, the current terminal, /cur/term, is assumed.

messages - the list (either with or without surrounding braces) of message names and/or message set names to be enabled. If some of the specified names do not begin with "/", then "/msg/" prefix is automatically inserted before such names prior to executing the command.

Options

Table 2-3. em and out options summary

Name	Type	Values	Default
period	float	[0…86400)	-
phase	float	[0…86400)	-
count	integer	[-256…32767]	0 for em 1 for out
flags	integer	[0…0xFFFF]	-

period, phase, count, flags - message scheduling parameters.

Description

These commands enable periodic output of the specified messages into the target, enforcing the message scheduling parameters to be those specified by options. No response is generated unless there is an error, or response is forced by the statement identifier.

The em and out commands are the same except the default value of the count option is set to 0 for em, and 1 for out. The out command is just a more convenient way to request

one-time output of message(s). We will speak only about `em` in this description though everything applies to the `out` as well.

The description below expects the reader is familiar with the material in the section "Periodic Output" on page 22.

For every output stream, there is corresponding *output list* of messages[1,2] that are currently enabled to be output to the given stream. When a `message` passed as argument to `em` command is not currently in the output list, the `em` command appends specified message to the *end* of the list. When a `message` passed to `em` command is already in the output list, the `em` command just changes this message's scheduling parameters and doesn't modify message's position inside the list.

Note: As the `em` command merges the specified messages to the output list, it's often a good idea to use `dm` command to clear the output list for the given stream before issuing `em` commands.

The `em` command processes the `messages` list one message at a time, from left to right, and from the first message of message set to the last message of message set. Should it encounter a name that doesn't correspond to any supported receiver message or message set, it remembers there was an error during execution, but doesn't stop processing of the `messages` list. This way all messages from the `messages` list that could be enabled will be enabled, and only single error will be reported when one or more of the specified messages can't be enabled.

When the `em` command processes a message at hand, the final operating message scheduling parameters in the corresponding output list of messages are calculated taking into account multiple sources of information about scheduling parameters, specifically:

1. Values explicitly specified in the `options` of the `em` command.
2. The default values of options of `em` command.
3. Scheduling parameters specified for the given message as part of the corresponding message set. These are taken into account only when enabling a message by specifying message set, not an individual message.
4. Current scheduling parameters of the message in the corresponding output list (if any).
5. Default scheduling parameters specified for the given message as part of the corresponding message group.

The above sources of parameters are listed in the order of their precedence, the first one having the highest precedence, and are applied individually to each of the four scheduling parameters. Therefore, values from (1) override values from (2), the resulting value

1. For a stream `NAME`, corresponding output list is called `/par/out/NAME`
2. Current firmware has arbitrary limit for maximum number of messages in an output list set to 49.

overrides value from (3), etc. However, if some of the F_FIX_PERIOD, F_FIX_PHASE, F_FIX_COUNT, or F_FIX_FLAGS bits are set in the flags field of the next source, corresponding fields of this next source will not be overridden.

Examples

Example: Enable one time output of NMEA GGA message to the current terminal:

```
⇒ em,,nmea/GGA:{,,1}
```

The same as above, but using out instead of em:

```
⇒ out,,nmea/GGA
```

Example: Enable the output of the default set of messages to the current log-file A using the default output parameters. Either of:

```
⇒ em,/cur/file/a,/msg/def
⇒ em,/cur/file/a,def
```

Example: Enable output of the default set of messages to the current log-file A every 10 seconds For the other output parameters, their default values will be used:

```
⇒ em,/cur/file/a,def:10
```

Example: Enable output of the default set of messages to the current terminal using default output parameters. Either of:

```
⇒ em,/cur/term,/msg/def
⇒ em,,/msg/def
⇒ em,,def
```

Example: Enable output of GREIS messages [~~](RT) and [RD] to the current terminal. Either of:

```
⇒ em,,/msg/jps/RT,/msg/jps/RD
⇒ em,,jps/{RT,RD}
```

Example: Enable output of NMEA messages GGA and ZDA to the current terminal every 20 seconds:

```
⇒ em,,nmea/{GGA,ZDA}:20
```

Example: Enable output of messages [SI], [EL] and [AZ] to serial port A. Set scheduling parameters for [SI] so that interval between any two subsequent [SI] messages will be equal to 10 seconds, if they coincide, and 1 second otherwise; output only the first fifty [SI] messages. In addition, the receiver, set output interval to 2 seconds for [EL] and [AZ] messages:

```
⇒ em,/dev/ser/a,jps/{SI:{1,10,50,0x2},EL,AZ}:2
```

Example: Enable output of RTCM 2.x message types 1 and 31 to serial port B with output interval 3 seconds, and RTCM 2.x message types 18, 19, 3, 22 to port C with output interval 1 second for types 18 and 19; and 10 seconds for types 3 and 22:

```
⇒ em,/dev/ser/b,rtcm/{1,31}:3; em,/dev/ser/c,rtcm/{18:1,19:1,22,3}:10
```

Example: Customize the default set of messages to only contain NMEA ZDA and GGA:

```
⇒ dm,/msg/def
⇒ em,/msg/def,/msg/nmea/{ZDA,GGA}
```

2.3.5 dm

Name

`dm` - disable periodic output of messages.

Synopsis

Format: `dm[,[target][,messages]]`
Options: `none`

Arguments

`target` - any output stream or message set. If no target is specified, the current terminal, `/cur/term`, is assumed. If some of the specified names do not begin with "/", then "`/msg/`" prefix is automatically inserted before such names prior to executing the command.

`messages` - the list of messages to be disabled, either with or without surrounding braces, or any message group or message set. If no messages are specified, all periodic output to the target is disabled.

Options

None.

Description

This command disables periodic output of the specified `messages` into the object `target`. No response is generated unless there is an error, or response is forced by the statement identifier.

If no messages are specified, all the periodic output to the `target` is disabled.

If the `target` is a current log-file and no messages are specified, all the output to the file is disabled, the file is closed, and corresponding current log-file is set to none.

If a message is specified in the `messages` list that is not currently enabled to be output to the given `target`, no corresponding error is generated by the `dm` command. Though this condition doesn't disable other possible errors from being reported.

Examples

Example: Disable all of the messages being output into the current log-file A and close the file:

⇒ `dm,/cur/file/a`

Example: Disable all the periodic output into the current terminal. Either of:

⇒ dm,/cur/term

⇒ dm

Example: Disable output of GREIS message [~~](RT) into the serial port B:

⇒ dm,/dev/ser/b,/msg/jps/RT

Example: Disable output of the GREIS message [DO] into the current log-file B:

⇒ dm,/cur/file/b,/msg/jps/DO

Example: Remove GREIS message [PM] from the default set of messages:

⇒ dm,/msg/def,/msg/jps/PM

Example: Disable output of all NMEA messages to the current terminal:

⇒ dm,/cur/term,/msg/nmea

Example: Disable output of the NMEA messages GGA and ZDA into the current terminal. Either of:

⇒ dm,/cur/term,/msg/nmea/GGA,/msg/nmea/ZDA

⇒ dm,,/msg/nmea/GGA,/msg/nmea/ZDA

⇒ dm,,nmea/GGA,nmea/ZDA

⇒ dm,,nmea/{GGA,ZDA}

2.3.6 init

Name

`init` - initialize objects.

Synopsis

Format: `init,object[/]`
Options: `none`

Arguments

`object` - the object to be initialized.

`/` - if present and the `object` is of type *list*, initialize all the contained objects instead of the `object` itself.

Options

None.

Description

This command initializes specified objects. No response is generated unless there is an error, or response is forced by the statement identifier.

The exact semantics of initialization depends on the object being initialized, but in general could be considered as turning an object to its "default" or "clean" state. For example, for parameters it means setting their values to corresponding defaults, for the file-storage device it means re-formatting the underlying medium, etc.

Note: Initializing some of objects will result in receiver reboot. This is currently the case for initialization of receiver non-volatile memory (`/dev/nvm/a`).

Note: Though it may change in the future, current implementation of this generic command in the receivers is rather limited. In fact only initialization of objects that are found in the examples below is currently supported.

Examples

Example: Clear NVRAM and reboot receiver. All the data stored in the NVRAM (almanacs, ephemeris, etc.) will be lost, all the parameters will be set to their default values after reboot:

⇒ `init,/dev/nvm/a`

Example: Clear ephemeris:

⇒ `init,/eph/`

Example: Set all the receiver parameters to their default values:

⇒ `init,/par/`

Example: Set all WLAN parameters to their default values. Reboot of the unit is required for the changes to take effect:

⇒ `init,/par/net/wlan/`

Example: Initialize the file system (i.e., reformat the underlying medium). All files stored in the receiver will be lost:

⇒ `init,/dev/blk/a`

Example: Initialize all the message sets to their default values:

⇒ `init,/msg/`

2.3.7 create

Name

`create` - create a new object.

Synopsis

Format: `create[,object]`
Options: `{log}`

Arguments

`object` - object identifier of the object to be created. If `object` does not begin with "/", then "`/log/`" prefix is automatically inserted before the `object` prior to executing the command. If omitted, then creation of a file is assumed and an unique file name is automatically generated.

Options

Table 2-4. `create` options summary

Name	Type	Values	Default
`log`	string	`a,b,…`	`a`

`log` - the log-file the created file is to be assigned to. The log-file selected is `/cur/log/X`, where `X` is the value of the option[1].

Description

This command creates a new object. No response is generated unless there is an error, or response is forced by the statement identifier.

Both the location in the tree and the type of the created object are defined by the `object` argument.

Two kinds of objects could be created:

1. Files. A new file is created whenever the object identifier specifies an object in a `/log` sub-tree, or when the `object` argument is omitted.
2. Message specifiers. A new message specifier is created whenever the object identifier specifies an object in a message set (e.g., `/msg/def`).

1. Current firmware supports either one or two simultaneous log-files depending on particular receiver.

Creating Files

When creating files, the `object` argument is either omitted or has a format `/log/NAME`, where `NAME` is the name of the file to be created, and `/log/` is optional. In the former case receiver will automatically select an unique name for the file. In the latter case the `NAME` specified should be a string of up to 31 characters and should contain neither spaces nor the following characters: “`,{}()@&"/\`”.

If the file `/log/NAME` already exists, the `create` command will fail and produce an error message. As a consequence, there is no way to clobber some of existing files with the `create` command.

After a new file is successfully created, it’s assigned to one of the current log-files depending on the value of the `log_file` option. If corresponding log-file already points to another file when `create` is executed, the old log-file will be closed and the output will continue into the new file without any interruption.

Creating Message Specifiers

When adding messages to a message set, the `object` argument has a format `/msg/SET/GROUP/MSG`, where `SET` is the name of the message set where new message should be created, `GROUP` is the name of the group the message belongs to, and `MSG` is the name of the message itself (e.g., `/msg/def/nmea/GGA`, or `/msg/jps/rtk/min/jps/ET`).

The message scheduling parameters will be copied from those defined for given message in the message group. Use `set` command to customize the scheduling parameters if required.

Examples

Creating Files

Example: Create a new file with an automatically generated name and assign it to the current log-file A (`/cur/file/a`). Either of:

```
⇒ create
⇒ create,:a
```

Example: Create a new log-file with the name “my_file”. Either of:

```
⇒ create,/log/my_file:a
⇒ create,my_file
```

Example: Create files “file1” and “file2”, and assign them to `/cur/file/a` and `/cur/file/b`:

```
⇒ create,file1:a; create,file2:b
```

●

Creating Message Specifiers

Example: Add `/msg/jps/ET` messages to the default set of messages:

```
⇒ create,/msg/def/jps/ET
```

Example: Add NMEA GGA message to the default set of messages and force its `period` and `phase` to be always 10 and 5, respectively, no matter what values for them will be specified in a `em` or `out` command:

```
⇒ create,/msg/def/nmea/GGA
⇒ set,/msg/def/nmea/GGA,{10,5,,0x30}
```

●

2.3.8 remove

Name

`remove` - remove an object.

Synopsis

Format: `remove,object[/]`
Options: `none`

Arguments

`object` - object identifier of the object to be removed. If `object` does not begin with "/", then "`/log/`" prefix is automatically inserted before the `object` prior to executing the command.

`/` - if present and the `object` is of type *list*, remove all the object contents instead of the `object` itself.

Options

None.

Description

This command removes (deletes) an existing object. No response is generated unless there is an error, or response is forced by the statement identifier.

If there is no object specified by `object`, or if the object can't be removed, an error is generated.

Two kinds of objects could be removed:

1. Files. If the file is one of current log-files, the command will fail and error message will be generated.
2. Message specifiers from message sets.

Examples

Example: Remove the log-file with the name "NAME". Either of:

```
⇒ remove,/log/NAME
⇒ remove,NAME
```

Example: Remove all log-files:

```
⇒ remove,/log/
```

Example: Remove GREIS standard [GA] message from the default set of messages:

⇒ `remove,/msg/def/jps/GA`

Example: Remove all the messages from the default set of messages:

⇒ `remove,/msg/def/`

Example: Remove all the messages from the minimal set of standard GREIS messages suitable for RTK:

⇒ `remove,/msg/rtk/jps/min/`

●

2.3.9 event

Name

`event` - generate free-form event.

Synopsis

Format: `event,string`
Options: `none`

Arguments

`string` - an arbitrary[1] string comprising up to 63 characters.

Options

None.

Description

This command generates a free-form event. No response is generated unless there is an error, or response is forced by the statement identifier.

The given `string` along with the time of receiving the `event` command is stored in the receiver in the special event buffer[2]. The contents of this buffer is output to all the output streams where the standard GREIS message [==](EV) (described on page 132) is enabled.

The free-form event mechanism is intended for the control programs to forward arbitrary text information to post-processing applications without interpreting this information in the receiver. The receiver firmware's core never generates free-form events on its own, nor does it somehow interpret the information sent through the `event` commands.

Note: All of the strings starting with the underscore character (ASCII 0x5F) are reserved for JAVAD GNSS applications. Care should be taken that such strings are not used with the `event` commands unless you can't accomplish your task otherwise or intend to cooperate with some JAVAD GNSS software. In the latter case please refer to detailed description of free-form events reserved for JAVAD GNSS applications in the *"Frame Format for Free-Form Events"* guide, available from *http://www.javad.com*.

Example: Generate a free-form event containing the string "Info1":

⇒ `event,Info1`

1. Recall that if a string contains any of the characters reserved for the receiver input language, you should enclose this string in double quotes.
2. The current firmware provides a buffer large enough to store up to sixteen 64 byte free-form events.

Example: Generate a free-form event containing reserved characters:

```
⇒ event,"EVENT{DATA,SENT}"
```

Example: Generate free-form event reserved for JAVAD GNSS application software (this event notifies post-processing application about change of dynamics):

```
⇒ event,"_DYN=STATIC"
```

Example: Generate a free-form with empty string:

```
⇒ event,""
```

Example: Generate a few free-form events and get back the [==](EV) messages (in the contents of [==] messages non-printable bytes are replaced with dots in the example):

```
⇒ em,,jps/EV
⇒ %accepted% event,"some string"
⇐ RE00A%accepted%
⇐ ==011.....some_string.
⇒ %1% event,1; %2% event,2
⇐ RE003%1%
⇐ RE003%2%
⇐ ==007.....1.
⇐ ==007.....2.
⇒ dm,,jps/EV
```

2.3.10 get

Name

`get` - start retrieving of file contents using DTP[1].

Synopsis

Format: `get,object[,offset]`
Options: `{timeout,block_size,period,phase,attempts}`

Arguments

`object` - object identifier of the file to be retrieved. If `object` does not begin with "/", then "`/log/`" prefix is automatically inserted before the `object` prior to executing the command. If the `object` does not exist or can't be retrieved, an error message is generated.

`offset` - offset in bytes from the beginning of the file at which to start retrieving. If omitted, 0 is assumed.

Options

Table 2-5. `get` options summary

Name	Type	Values	Default
`timeout`	integer	`[0…86400]`, seconds	`10`
`block_size`	integer	`[1…16384`[1]`]`	`512`
`period`	float	`[0…86400)`, seconds	`0`
`phase`	float	`[0…86400)`, seconds	`0`
`attempts`	integer	`[-257…100]`	`10`

1. 2048 for receivers that don't support TCP or USB.

`timeout` - the timeout for DTP.

`block_size` - the size of a DTP data block.

`period` - the output period for filtering (see below).

`phase` - the output phase for filtering (see below).

`attempts` - different meaning depending on the range, as follows:

1. See "Data Transfer Protocol" on page 578.

`[1…100]` – maximum number of attempts DTP transmitter will take to send single block. When set to 1, special *streaming* mode is activated (see below).

`0` – rather than starting DTP, output raw contents of the object.

`[-256…-1]` – rather than starting DTP, output the contents of the object wrapped into [>>] messages.

`-257` – rather than starting DTP, output the contents of the object wrapped into [RE] messages.

Description

This command starts retrieving of a file into the host computer using the Data Transfer Protocol (DTP) or raw output format. No response is generated unless there is an error, or response is forced by the statement identifier.

When in DTP mode, after the `get` command succeeds, the *DTP transmitter* is started on the receiver and waits for *DTP receiver* to be run on the host. Therefore, to actually retrieve any data, one needs *DTP receiver* implementation on the host.

The optional `offset` argument allows host to implement support for resuming of interrupted data transfer. Note that seeking to a large offset may require rather long time to perform in the receiver. To correctly implement resumption in the host software, force receiver response to the `get` command using *statement identifier* and wait for the reply from the receiver before running DTP on the host. This method takes advantage of the fact that receiver replies to the `get` command *after* the seek is performed.

When the `attempts` option is set to `1`, the *DTP transmitter* will be put into so-called *streaming* mode. In this mode, after receiving the first `NACK` from the *DTP receiver*, the *DTP transmitter* will stream data blocks without waiting for `ACK`s from the *DTP receiver*, and the *transmitter* will immediately abort data transfer should `NACK` be received. This approach allows significantly faster data transfer over reliable connections having high latencies (such as TCP) or relatively high direction switch overhead (such as USB). Correctly implemented receiving part of the protocol does not require any special care to support this method.

When the `period` option is non-zero special *filtering* mode is activated. For example, it allows to download 1Hz data from a file that was written using 10Hz update rate. Specifically, the receiver will send the data only for the epochs where receiver time modulo one day (T_r) satisfies the following equation:

$$T_r \{\text{mod } \texttt{period}\} = \texttt{phase}$$

To achieve this, receiver parses the contents of the file and filters-out some of the messages. Note that implementation of resumption of interrupted download is very hard if

not impossible in this case due to the fact that the host has no idea at what offset of the receiver file the download has been interrupted.

Any of the types of transfer could be aborted by data receiving end by sending any DTP error symbol (e.g., ASCII '#').

When transferring data in [RE] messages, the value of block_size will determine maximum size of data payload for every [RE] message (limited also by the size of internal firmware buffer). As usual, every [RE] message will be started with the command ID (if any).

When transferring data in [>>] messages, the value of the attempts option will determine the id field of the [>>] messages as follows:

id = -1 - attempts

and the value of "block_size" will determine maximum size of data payload for every [>>] message (limited also by the size of internal firmware buffer).

The next byte after id (the first byte of the data field) in the [>>] message will then be sequence character starting with ASCII symbol 0 and being incremented modulo 64 for every message, resulting in the sequence of ASCII symbols from 0 to o, inclusive:

```
seq = 0
loop { seq_char = '0' + (seq++ % 64) }
```

The sequence character allows receiving end to detect loss of [>>] message(s) in the sequence.

Then the object data payload of up to block_size bytes will follow, and then the check sum, according to the format of [>>] message.

Successful output in the wrapped mode will always be finalized by [>>] message with no data payload, to allow receiving end to reliably determine the end of transfer.

Examples

Example: Start retrieving the contents of the file NAME using DTP. Either of:

⇒ get,/log/NAME

⇒ get,NAME

Example: Start retrieving the contents of the file NAME starting from byte number 3870034 (counting bytes from zero). Expect rather long time to pass between the command and the reply:

⇒ %%get,NAME,3870034
⇐ RE002%%

Example: Start retrieving the contents of the file `my_logfile` starting from byte 3000 using time-out 50 seconds and block size of 8192 bytes:

⇒ `get,my_logfile:{50,8192},3000`

Example: Start retrieving the contents of the file `NAME` filtering out epochs so that the resulting retrieved file would be 0.1Hz data:

⇒ `get,NAME:{,,10}`

Example: Start retrieving the contents of the file `NAME` using *streaming* mode (`attempts` option set to 1):

⇒ `get,NAME:{,,,,1}`

Example: Send contents of the file `NAME` wrapped into `[>>]` messages with `id`

`61` (being ASCII symbol '='), using up to 128 bytes of data per message:

⇒ `get,NAME:{,128,,,-62}`

Example: Send contents of the file `NAME` wrapped into `[RE]` messages using up

to 190 bytes of data per message, prepended by `%MY_ID%`:

⇒ `%MY_ID%get,NAME:{,190,,,-257}`

2.3.11 put

Name

`put` - start file uploading using DTP[1].

Synopsis

Format: `put,object[,offset]`
Options: `{timeout, block_size}`

Arguments

`object` - object identifier of the file to write data to. If `object` does not begin with "/", then "`/log/`" prefix is automatically inserted before the `object` prior to executing the command.

`offset` - offset in bytes from the beginning of the file at which to start writing. If omitted, 0 is assumed.

Options

Table 2-6. `put` options summary

Name	Type	Values	Default
`timeout`	integer	`[0…86400]`, seconds	`10`
`block_size`	integer	`[1…16384`[1]`]`	`512`

1. 2048 for receivers that don't support TCP or USB.

`timeout` - the timeout for DTP.

`block_size` - the size of a DTP data block.

Description

This command starts uploading of data from host computer into a file in the receiver using the Data Transfer Protocol (DTP). No response is generated unless there is an error, or response is forced by the statement identifier.

After the `put` command succeeds, the *DTP receiver* is started on the receiver and waits for *DTP transmitter* to be run on the host. Therefore, to actually upload any data, one needs *DTP transmitter* implementation on the host.

1. See "Data Transfer Protocol" on page 578.

The optional `offset` argument allows host to implement support for resuming of interrupted data transfer. A non-zero `offset` value allows host to request appending data to the end of an existing file of matching size.

If `offset` is 0 and the file `object` doesn't exist, receiver will try to create and open for writing a new file with the name defined by `object`. In this case the command will fail if there already exist a file with given name.

If offset is greater than 0, and there is a file `object`, and the file size is equal to the value of `offset`, then the `put` command will open the file `object` for append. In this case the command will fail if there is no existing file with given name or if the size of the existing file doesn't match those specified by `offset`.

Examples

Example: Start data uploading to a fresh file "NAME" using DTP. Either of:

```
⇒ put,/log/NAME
⇒ put,NAME
```

Example: Start uploading data and append them to existing file "NAME". Use default DTP timeout and DTP block size 4096 bytes. Get the size of the file before starting the upload (note that the file size is required on host anyway so that it can skip this number of bytes from its source data file):

```
⇒ print,/log/NAME&size
⇐ RE008 3870034
⇒ put,/log/NAME:{,4096},3870034
```

Example: Start data uploading to a fresh file "my_logfile" using timeout 50 seconds and block size of 8192 bytes:

```
⇒ put,my_logfile:{50,8192}
```

2.3.12 fld

Name

`fld` - firmware loading.

Synopsis

Format: `fld,id,object`
Options: `{timeout, block_size}`

Arguments

`id` - string containing the receiver electronic ID[1]. If specified ID does not correspond to the actual electronic ID of the receiver, the command will fail and produce error message.

`object` - object identifier of the source of the firmware to be loaded. Either the name of receiver file, or the name of an input port. When it's the name of input port, either `/cur/term` or actual name of the current port should be given, otherwise error will be reported.

Options

Table 2-7. `fld` options summary

Name	Type	Values	Default
`timeout`	integer	`[0…86400]`, seconds	`10`
`block_size`	integer	`[1…16384`[1]`]`	`512`

1. 2048 for receivers that don't support TCP or USB.

`timeout` - the timeout for DTP.

`block_size` - the size of a DTP data block.

Description

This command loads firmware from specified `object` into receiver and then resets the receiver. No response is generated unless there is an error, or response is forced by the statement identifier.

1. The ID could be obtained using `print,/par/rcv/id` command.

Warning: *Should a power failure or fatal interruption of firmware transfer through a port occur during the loading, the receiver may go into a semi-working state where only firmware loading through RS-232 ports using "power-on capture" method is possible.*

If the `object` designates an existing file[1], the receiver will first check whether the file contains valid firmware for the receiver (it takes a number of seconds to complete). If the check succeeds, the receiver will load the firmware and then perform self-reset. Note that the reply to the command (if any) will be sent after the check is performed but before firmware loading begins. The `timeout` and `block_size` options are ignored in this case.

If `object` designates an input stream, the command will send the reply (if any) and then start *DTP receiver* that will wait for *DTP transmitter* to be run on the host. Therefore, to actually upload the firmware, one needs *DTP transmitter* implementation on the host. Self reset (reboot) will be performed by the receiver after the loading successfully completes or is interrupted.

Examples

Example: Load firmware from the file "firmware.ldp" into receiver with electronic ID `123456789AB`. Expect a few seconds to pass between sending the command and receiving reply, while receiver checks the file for firmware validity:

```
⇒ %%fld,123456789AB,/log/firmware.ldp
⇐ RE002%%
```

Example: Start firmware uploading from the USB port using block size 16384 bytes and timeout 20 seconds. Obtain electronic ID before issuing the command:

```
⇒ print,rcv/id
⇐ RE00C 8PZFM10IL8G
⇒ fld,8PZFM10IL8G,/dev/usb/a:{20,16384}
```

1. It is expected that the file containing the firmware is uploaded to the receiver in advance, e.g., using the `put` command.

Receiver Messages

This chapter describes general format of GREIS standard messages as well as particular formats of all the predefined messages. Besides the GREIS standard messages, receiver supports quite a few messages of different formats, such as NMEA or BINEX. The formats of those "foreign" messages are described at the end of this chapter.

3.1 Conventions

3.1.1 Format Specifications

To describe some format as a sequence of bytes[1] in a compact form, we define formats for a few primary field types and then use notation close to those used in the C programming language to build definitions of more complex formats:

```
struct NAME {LENGTH} {
  TYPE FIELD[COUNT];  // DESCRIPTION
  …
  TYPE FIELD[COUNT];  // DESCRIPTION
};
```

where:

`NAME` – the name assigned to this format. It could be used in other format definitions as the `TYPE` of a field.

`LENGTH` – the length in bytes of entire sequence. For a fixed length format, it is a number, for a variable length message, it may be either an arithmetic expression depending on some other variable parameters or just the string `var`.

`TYPE FIELD[COUNT]` – field descriptor. It describes a sequence of `COUNT` elements of the same `TYPE` which is assigned the name `FIELD`. The `TYPE` could be either one of the primary field types described below, or a `NAME` of another format. When `[COUNT]` is absent, the field consists of exactly one element. When `COUNT` is absent (i.e., there are only empty square brackets, `[]`), it means that the field consists of unspecified number of elements.

1. In the context of this chapter, "byte" means 8-bit entity. Least significant bit of a byte has index zero.

DESCRIPTION – description of the field along with its measurement units and allowed range of values, where appropriate. Measurement units are surrounded by square brackets.

The following primary field types are defined:

Table 3-1. Primary Field Types

Type Name	Meaning	Length in Bytes
a1	ASCII character	1
i1	signed integer	1
i2	signed integer	2
i4	signed integer	4
u1	unsigned integer	1
u2	unsigned integer	2
u4	unsigned integer	4
f4	IEEE-754 single precision floating point	4
f8	IEEE-754 double precision floating point	8
str	zero-terminated sequence of ASCII characters	variable

To entirely define particular format, we also have to specify bytes order in the primary non-aggregate fields that are multi-byte (i2, i4, u2, u4, f4, f8). For GREIS messages this order is defined by the [MF] message, see "[MF] Messages Format" on page 74 for details.

Using the above definitions it's possible to (recursively) expand any format specification to corresponding sequence of bytes. For example, the format

```
struct Example {9} {
  u1 n1;
  f4 n2;
  i2 n3[2];
};
```

expands to the following sequence of bytes assuming least significant byte first (LSB) order:

```
n1[0](0),
n2[0](0),n2[0](1),n2[0](2),n2[0](3),
n3[0](0),n3[0](1),n3[1](0),n3[1](1)
```

and to the following sequence of bytes assuming most significant byte first (MSB) order:

```
n1[0](0),
n2[0](3)n2[0](2)n2[0](1)n2[0](0)
n3[0](1)n3[0](0)n3[1](1)n3[1](0)
```

where `x[i](j)` designates `j`-th byte (byte #0 being least significant one) of an `i`-th element of the field `x`.

3.1.2 Special Values

For binary messages, some of their integer and floating point fields may contain special values, which are used instead of actual data when no data for the field are available. Binary fields for which checking for special values is required during data extraction are marked with the exclamation mark, "!" in the first column of the field definition.

The following table defines special values for various data field types:

Table 3-2. Special Values for Fields

Field Type	Special Value	HEX Representation
i1	127	7F
u1	255	FF
i2	32767	7FFF
u2	65535	FFFF
i4	2147483647	7FFF_FFFF
u4	4294967295	FFFF_FFFF
f4	quiet NaN	7FC0_0000
f8	quiet NaN	7FF8_0000_0000_0000

3.2 Standard Message Stream

Standard GREIS message stream is a sequence of at most two kinds of messages, GREIS standard messages, and non-standard text messages.

Most important and widely used kind of messages is a rich set of GREIS standard messages. Their general format is carefully designed to allow for both binary and text mes-

sages, and to make it possible for applications to efficiently skip the messages the application doesn't know about or is not interested in.

Support for non-standard text messages, that should still adhere to the format defined for them in this manual, makes it possible to mix GREIS standard messages with messages of some other formats in the standard GREIS data stream. An example of such a format are NMEA messages.

Non-standard text messages of a special case, the messages that contain only ASCII <CR> and/or <LF> characters, are inserted by the message formatting engine in the receiver between the GREIS standard messages to make the resulting message stream more human-readable when it is sent to a terminal or generic text viewer or editor application.

Besides GREIS standard messages and non-standard text messages, JAVAD GNSS receivers typically support plenty of other formats (e.g., RTCM, BINEX, CMR). However, those formats are incompatible with the format of standard GREIS message stream. Should a stream contain messages of those formats, it can't be called GREIS standard message stream anymore, and can't be parsed by the same rules as the standard stream.[1]

3.3 General Format of Messages

3.3.1 Standard Messages

The format of every standard message is as follows:

```
struct StdMessage {var} {
  a1 id[2];          // Identifier
  a1 length[3];      // Hexadecimal body length, [000…FFF]
  u1 body[length];   // Body
};
```

Each standard message begins with the unique message identifier comprising two ASCII characters. Any characters from the subset "0" through "~" (i.e., decimal ASCII codes in the range of [48…126]) are allowed in identifier.

1. In fact, the format of GREIS standard messages is so flexible that it can incorporate any data stream into the standard GREIS data stream, but then the original incompatible stream should be wrapped into a sequence of special GREIS messages. The predefined message with identifier ">>" serves this purpose.

Message identifier is followed by the length of message body field. This field, which comprises three upper-case hexadecimal digits, specifies the length of the message body in bytes. Thus the maximum message body length is 4095 (0xFFF) bytes.

Message body follows immediately after the length field and contains exactly the number of bytes specified by the length field. There are no restrictions on the contents of the message body implied by the general format. The format of the message body in a message is implicitly defined by the message identifier. Formats of message bodies of all the predefined messages

3.3.2 Non-standard Text Messages

The format of non-standard text messages is as follows:

```
struct NonStdTextMessage {var} {
  a1 id;          // Identifier, [!…/]
  a1 body[];      // Body of arbitrary length, [0…)
  a1 eom;         // End of message (<CR> or <LF>)
};
```

Message identifier is any character in the range [!… /] (decimal ASCII codes in the range [33…47]). Message identifier is optional. If absent, the message body should have length zero (i.e., should be absent as well).

Message body is a sequence of ASCII characters except <CR> (decimal code 13) and <LF> (decimal code 10) characters. No limitation on the body length is imposed by the format.

The end of message marker is either <CR> or <LF> character.

Note that the format allows for non-standard messages comprising only CR or LF characters. This feature allows to make standard GREIS message streams look more human-readable when outputting data to a general-purpose terminal or viewing with generic text viewer or editor.

One of the non-standard text message identifiers, the character “$”, is already reserved as the identifier for standard NMEA messages. No other non-standard text messages should use the “$” as identifier.

3.3.3 Parsing Message Stream

In this section, you will find some hints and tips on how to write code intended to parse a GREIS receiver's message streams. Although we are not going to discuss this subject in detail in this reference manual, we'd like to emphasize here that the standard message

format will allow you to effectively process/parse nearly any GREIS message stream you may encounter in practice.

Synchronization

When parsing a message stream, you first need to find nearest message boundary. This is what is usually called "synchronization". Message synchronization is carried out when parsing is started or when synchronization is lost due to an error in the data stream. In fact, to simplify the algorithm, you may consider that you are already synchronized when you start to parse the data stream. If it happens that it's not indeed the case, the parsing error should occur. You then skip one character from the input stream and pretend you are synchronized again. Such approach effectively eliminates synchronization task as a separate part of the parsing algorithm.

Note: Due to the fact that the errors rate in a reasonably useful data stream should be rather low, the synchronization shouldn't be a frequent task. In addition, the GREIS data stream typically consists of rather short messages, so the distance to the nearest message boundary is typically small. Taking into account these considerations, there is no requirement for synchronization algorithm to be very fast.

Skipping to the Next Message

Having the length in the general format of the standard GREIS messages allows you to easily ignore messages without knowing the format of their body. We indeed strongly recommend writing parsers so that they do skip unknown messages.

To go from the current message to the next one, take the following steps:

1. Assume the current message starts at position "N". Determine the current message length (decode characters ## N+2, N+3, N+4). Assume the message length is equal to L. Skip the first L+5 characters starting from position "N".
2. Skip all of <CR> and <LF> characters (if any).

Strictly speaking, we do not recommend that you use in your parsing code any apriori information about the sizes and the contents of the message bodies. If you respect this recommendation, you will not have trouble with the parsing program should some of the messages be changed.

Note: The rules and hints on parsing of message bodies of the standard predefined GREIS messages are discussed later in "Parsing Message Bodies" on page 67.

3.4 Standard Predefined Messages

In this section we will familiarize the reader with the predefined set of standard GREIS messages. When referring to a message with the identifier XX, we use the notation [XX]. While most messages are called by their message identifier in GREIS, some of them, specifically those that have non-alphanumeric identifiers, have names that are different. For such messages the notation [XX](NN) is used, where XX is message identifier, and NN is message name to be used in the GREIS commands. For example the message [~~](RT) has header "~~" and is called `/msg/jps/RT` in GREIS commands.

This section defines the formats of the bodies for all the standard predefined messages. Bear in mind that in a data stream every message has a standard header defined by the general format as well.

3.4.1 Parsing Message Bodies

Allowed Format Extensions

Formats of binary messages having fixed message size allow to add more data fields in the future. New fields are allowed to be inserted only at the end of message body just before the checksum field (if any). Such modifications to the message bodies are considered to be format extensions, not incompatible changes.

Though standard GREIS text messages aren't messages with fixed message size, new fields may still appear in these messages in the future. New fields can be either inserted at the end of an existing text message just before the checksum field, or immediately before any right-hand brace (`}`). For example, a message that is currently read as:

```
…1,{21,22},3,@CS
```

can be later extended to

```
…1,{2.1,2.2,2.3},3,4,@CS
```

where two additional fields, "2.3" and "4", were added.

Implement your parsing algorithms taking into account the following rules to make them work even with future format extensions:

1. Don't assume that the size of message body of the received message should exactly match specific size defined in this document. Only if the message is too short does it mean you can't use its contents. If the message is longer than expected, just ignore the excess data.
2. Address the checksum field relative to the end of the message body.

3. Address other data fields relative to the beginning of the message body.
4. Take into consideration the above rule for extending of text messages when writing data extractors for text messages.

Checksums

After a message has been extracted from the data stream using techniques described in the “Parsing Message Stream” on page 65, and the message identifier appears to be one of those the application is interested in, the message body should be parsed to extract the data. Before extracting the contents, the message checksum should be calculated and compared against the checksum contained in the message.

Most of predefined messages contain checksum. Checksum is computed using both the message header (i.e., “message identifier” plus “the length of message body”) and the body itself. See “Computing Checksums” on page 577 for more information on checksum computation.

The checksum is always put at the very end of the message body. If a message's structure is modified by adding a new data field(s), the new data fields will be added before the checksum field. This explains why it is recommended to address the checksum field relative to the end of the message body.

3.4.2 General Notes

Time Scales

There are several time scales your receiver may handle:

`Tr` – receiver time

`Tg` – GPS system time

`Tn` – GLONASS system time

`Te` – Galileo system time

`Tw` – SBAS system time

`Tq` – QZSS system time

`Tb` – BeiDou system time

`Ti` – IRNSS system time

`Tu` – UTC(USNO). Universal Coordinated Time supported by the U.S. Naval Observatory.

`Ts` – UTC(SU). Universal Coordinated Time supported by the State Time and Frequency Service, Russia.

"Receiver time" is the only time grid that is always available in your receiver (i.e., the other time grids from the above list may or may not be currently available).

Receiver always synchronizes its receiver time with the one specified by the `/par/raw/time/ref` parameter. The time grid thus selected is referred to as "receiver reference time" (Trr) hereafter in this section.

Different time systems may have different time notations (formats) associated with them (e.g., for GPS time, we use such terms as "week number", "time of week", etc.). Note, however, that the "receiver time" representation will not depend on the selected receiver reference time and is always represented as receiver date and time of day.

Most of the predefined messages don't contain reference time information inside. When outputting receiver information available for the current epoch, you usually get various messages. Instead of supplying each of them with an individual time tag data field, we use a special message that carries receiver time information common for these messages. This message is called "Receiver Time" and has the identifier [~~].

There could be, however, modes of operation, such as RTK delayed mode, when at a given epoch receiver may produce solution referenced to some other epoch in the past. To provide time tag for such solution, special "Solution Time-Tag" [ST] message is provided. In fact this message contains correct time tag for a solution in all modes of operations, and in most modes it has exactly the same time as [~~].

There are some other messages having a time tag data field. Those are messages that contain information that appears independently on the receiver epoch grid. An example of such a message is "Event" [==].

Delimiters

In fact, "Receiver Time" message is supposed to precede all of the other messages generated at the current epoch thus delimiting messages corresponding to different epochs. From a formal point of view, it is up to the user to define the order of messages in the output stream. However, care should be taken to ensure that the order in which messages are written into the output stream does not break the "epoch synchronization", which is very essential for post-processing the logged data with JAVAD GNSS software packages. For more details on the default set of messages see "Message Sets" on page 560.

For real-time applications it's essential to determine the end of epoch as soon as possible. For such applications just delimiting epochs by a "start of epoch" marker is not convenient. We suggest to use the "Epoch Time" [::](ET) message as the "end of epoch" marker. This message contains the same time of day field that is found in the "Receiver Time" message that allows for better integrity checking. The idea is to compare time tag

from [::] message against the time tag from corresponding [~~] message. Mismatched tags are an indication of broken epoch.

You will notice that most of the messages have identifiers comprising only digits and/or English letters. In fact, “Receiver Time” [~~] is the only message whose identifier uses the character “~”. It makes sense as the [~~] message plays a very important part serving as an epoch delimiter. Thus there are special precautions in order to minimize the probability of losing this key message. Similarly, the identifier of the “Event” ([==]) message, too, must be as distinctive as possible since application software may use free-form events just as delimiters.

The idea of using “highly distinctive” identifiers for the messages that serve as delimiters is very clear. Should a message's checksum be wrong, just check its identifier. If neither of the identifier's characters coincides with “~”, then it is very unlikely that this is a corrupted [~~] message. Therefore, you needn't skip to the next [~~] message in this case.

On the other hand, if a message has the correct checksum yet one of the identifier characters is “~”, then it would be safer to treat this message as a corrupted [~~] message. In this case – skip to the next [~~] message.

Solution Types

The field “solType” used in many of the predefined messages designates the type of corresponding solution and may have the following values:

Table 3-3. Solution Types

Value	Meaning
0	no solution
1	stand-alone solution
2	code-differential (DGPS) solution
3	phase-differential (RTK) solution with float ambiguities
4	phase-differential (RTK) solution with fixed ambiguities
5	fixed. I.e., the value was entered, not calculated.
6	PPP code. Not converged Precise Point Position.
7	PPP float. Converged Float Precise Point Position with float ambiguities
8	PPP fixed. Converged Precise Point Position with fixed ambiguities

Satellite Navigation Status

Fields containing navigation status are used in a few of the predefined messages. Such fields designate the status of particular satellite with respect to position computation.

Codes [0…3], [40…62], and [64…255] indicate that given satellite is used in position computation and show which measurements are used. The rest of codes indicate that satellite is not used in position computation and indicate why this satellite is excluded from position computation. The table below describes assigned values and their meanings.

Table 3-4. Satellite Navigation Status

Value	Meaning
00	CA/L1 data used for position computation
01	P/L1 data used for position computation
02	P/L2 data used for position computation
03	Ionosphere-free combination used for position computation
04	Measurements are not available
05	Ephemeris is not available, similar to 29 below
06	Unhealthy SV (as follows from operational (=ephemeris) SV health)
07	Time-Frequency parameters from the ephemeris data set may be wrong[1]
08	Initial conditions (position and velocity vectors) from the ephemeris data set may be wrong[1]
09	Almanac SV health indicator is not available for this satellite[1]
10	Unhealthy SV (as follows from the almanac SV health indicator)[1]
11	"Alert" flag (from the word "HOW") is set[2]
12	Navigation accuracy is worse than those specified by `/par/pos/SYS/ura/mask` parameter
13	This SV is excluded from position computation by the user
14	SV with this frequency channel number is excluded from position computation by the user[1]
15	This SV is excluded from solution since its system number is unknown[1]
16	This SV has elevation lower than the specified mask angle
17	Reserved
18	Ephemeris data is too old
19	This SV does not belong to the constellation the user has selected
20	No data from reference station are available for given satellite (DGPS mode only)
21	Ghost
22	Wrong measurements have been detected by RAIM
23	SNR below specified minimum level
24	Ionospheric and/or tropospheric corrections are not available
25	Reserved
26	CLL is not settled
27	Ionospheric corrections are not received from base
28	Coarse code outlier has been detected
29	Ephemeris is not available, similar to 05 above
30,31	No data from reference station are available for given satellite (RTK mode only)
32…41	Reserved
42	L2C data used for position computation
43,44	Reserved
45	L5 data used for position computation
46	L1C data used for position computation
47…50	Reserved
51	CA/L1 slot is used in RTK processing
52	P/L1 slot is used in RTK processing
53	P/L2 slot is used in RTK processing
54	P/L1 and P/L2 measurements are used in RTK processing
55	CA/L1 and P/L2 measurements are used in RTK processing
56	L2C slot is used in RTK processing
57	P/L1 and L2C measurements are used in RTK processing
58	CA/L1 and L2C measurements are used in RTK processing

Table 3-4. Satellite Navigation Status

Value	Meaning
59…62	Reserved
63	Satellite navigation status is undefined
64…70	Reserved
71	L5 slot is used in RTK processing
72	CA/L1 and L5 slots are used in RTK processing
73	P/L1 and L5 slots are used in RTK processing
74	P/L2 and L5 slot are used in RTK processing
75	P/L1, P/L2, and L5 slots are used in RTK processing
76	CA/L1, P/L2, and L5 slots are used in RTK processing
77	L2C and L5 slots are in RTK processing
78	P/L1, L2C, and L5 slots are used in RTK processing
79	C/A L1, L2C, and L5 slots are used in RTK processing
80	L1C and P1 slots are used in RTK processing
81	L1C slot is used in RTK processing
82	L1C and L5 slots are used in RTK processing
83,84	Reserved
85	L1C and P2 slots are used in RTK processing
86,87	Reserved
88	L1C and L2C slots are used in RTK processing
89…255	Reserved

1. GLONASS only
2. GPS only

3.4.3 General Purpose Messages

[JP] File Identifier

```
struct FileId {85} {
  a1 id[5];            // File type identifier
  a1 description[80];  // Human-readable stream description
};
```

This message, that is intended to be put at the beginning of the file, serves two purposes. First, it enables the processing program to easily identify the file type. Second, this message usually contains some additional information about the origin of the corresponding file (e.g., what particular hardware was used to collect data this file contains).

Both the “id” and the “description” fields are padded to the required size with spaces if necessary.

For JAVAD GNSS receivers, the [JP] message always contains the following information: “id” = “RLOGF”, and “description” = “JPS NAME Receiver log-file” (blanks are omitted here), where the sub-string “NAME” stands for the specific receiver name.

Note: The size of this message is not subject to change. Therefore, the first 5 bytes of this message are always “JP055”, and specifically for this message generated by receiver, the first 10 bytes are always “JP055RLOGF”.

[MF] Messages Format

```
struct MsgFmt {9} {
  a1 id[2]=”JP”;     // JP identifier
  a1 majorVer[2];    // Format major version as decimal (e.g., '01')
  a1 minorVer[2];    // Format minor version as decimal
  a1 order;          // Bytes order
                     // '0' – LSB first;
                     // '1' – MSB first
  a1 cs[2];          // Checksum formatted as hexadecimal
};
```

Note: The size of this message is not subject to change. Therefore, the first 7 bytes of this message are always “MF009JP”.

The data field `order` describes how multi-byte binary types are stored inside the message bodies.

Note: For message format version 1.0, `order` is always set to “0”. Receiver always generates data in the least significant bytes first order.

The message format's major version is updated if and only if some backward incompatible changes to the existing message format are made. Any other changes to the existing messages result in updating only the minor version.

3.4.4 Time Messages

[~~](RT) Receiver Time[1]

This message contains the “time of day” part of the full receiver time representation (Tr).

```
struct RcvTime {5} {
  u4 tod;  // Tr modulo 1 day (86400000 ms) [ms]
  u1 cs;   // Checksum
};
```

This message is intended to be used as a “start of epoch” marker.

1. Use message name /msg/jps/RT to enable/disable the message.

[::](ET) Epoch Time[1]

```
struct EpochTime {5} {
  u4 tod;  // Tr modulo 1 day (86400000 ms) [ms]
  u1 cs;   // Checksum
};
```

This message is intended to be used as an “end of epoch” marker. Provided the [~~](RT) message is used as “start of epoch” marker and [::](ET) is used as “end of epoch” marker, one can check that time tags from the messages from given epoch match to increase integrity checking capability of the stream decoding algorithm.

[RD] Receiver Date

```
struct RcvDate {6} {
  u2 year;   // Current year [1…65534][]
  u1 month;  // Current month [1…12] []
  u1 day;    // Current day [1…31] []
  u1 base;   // Receiver reference time [enumerated]
             //       0 — GPS
             //       1 — UTC_USNO
             //       2 — GLONASS
             //       3 — UTC_SU
             //   4…254 — Reserved
  u1 cs;     // Checksum
};
```

This message contains the “date” part of the full receiver time representation (Tr).

[TO] Reference Time to Receiver Time Offset

```
struct RcvTimeOffset {17} {
  f8 val;   // Tr - Trr [s]
  f8 sval;  // Smoothed (Tr - Trr) [s]
  u1 cs;    // Checksum
};
```

[DO] Derivative of Receiver Time Offset

```
struct RcvTimeOffsetDot {9} {
  f4 val;   // Derivative of (Tr - Trr) [s/s]
  f4 sval;  // Smoothed derivative of (Tr - Trr) [s/s]
  u1 cs;    // Checksum
};
```

1. Use message name /msg/jps/ET to enable/disable the message.

[BP] Rough Accuracy of Time Approximation

```
struct RcvTimeAccuracy {5} {
  f4 acc;  // Accuracy [s]
  u1 cs;   // Checksum
};
```

If the value of accuracy is greater than 10^{-3}[s], it means that receiver clock may not be properly synchronized to receiver reference time (Trr).

[GT] GPS Time

```
struct GPSTime {8} {
  u4 tow;   // Time of week [ms]
  u2 wn;    // GPS week number (modulo 1024) []
  u1 cycle; // number of 1024-weeks cycle
  u1 cs;    // Checksum
};
```

[GO] GPS to Receiver Time Offset

```
struct RcvGPSTimeOffset {17} {
  f8 val;   // (Tr - Tg) [s]
  f8 sval;  // Smoothed (Tr - Tg) [s]
  u1 cs;    // Checksum
};
```

[NT] GLONASS Time

```
struct GLOTime {8} {
  u4 tod;   // time of day [ms]
  u2 dn;    // GLONASS day number (modulo 4 years
            // starting from 1996) []
  u1 cycle; // number of 4-years cycle
  u1 cs;    // Checksum
};
```

[NO] GLONASS to Receiver Time Offset

```
struct RcvGLOTimeOffset {17} {
  f8 val;   // (Tr - Tn) [s]
  f8 sval;  // Smoothed (Tr - Tn) [s]
  u1 cs;    // Checksum
};
```

[EO] Galileo to Receiver Time Offset

```
struct RcvGALTimeOffset {17} {
  f8 val;   // (Tr - Te) [s]
  f8 sval;  // Smoothed (Tr - Te) [s]
  u1 cs;    // Checksum
};
```

[WO] SBAS to Receiver Time Offset

```
struct RcvSBASTimeOffset {17} {
  f8 val;   // (Tr - Tw) [s]
  f8 sval;  // Smoothed (Tr - Tw) [s]
  u1 cs;    // Checksum
};
```

[QO] QZSS to Receiver Time Offset

```
struct RcvQZSSTimeOffset {17} {
  f8 val;   // (Tr - Tq) [s]
  f8 sval;  // Smoothed (Tr - Tq) [s]
  u1 cs;    // Checksum
};
```

[CO] BeiDou to Receiver Time Offset

```
struct RcvBeiDouTimeOffset {17} {
  f8 val;   // (Tr - Tb) [s]
  f8 sval;  // Smoothed (Tr - Tb) [s]
  u1 cs;    // Checksum
};
```

[Io] IRNSS to Receiver Time Offset

```
struct RcvIrnssTimeOffset {17} {
  f8 val;   // (Tr - Ti) [s]
  f8 sval;  // Smoothed (Tr - Ti) [s]
  u1 cs;    // Checksum
};
```

[UO] GPS UTC Time Parameters

```
struct GpsUtcParam {24} {
  UtcOffs utc; // GPS UTC time offset parameters
  u1 cs;       // Checksum
};

struct UtcOffs {23} {
  f8 a0;     // Constant term of polynomial [s]
  f4 a1;     // First order term of polynomial [s/s]
  u4 tot;    // Reference time of week [s]
  u2 wnt;    // Reference week number []
  i1 dtls;   // Delta time due to leap seconds [s]
  u1 dn;     // 'Future' reference day number [1…7] []
  u2 wnlsf;  // 'Future' reference week number []
  i1 dtlsf;  // 'Future' delta time due to leap seconds [s]
};
```

This message describes the relationship between UTC(USNO) and GPS time as specified by GPS subframe 4, page 18.

For how to convert GPS time into UTC(USNO), see ICD-GPS-200C, Revision IRN-200C-004 April 12, 2000.

[WU] SBAS UTC Time Parameters

```
struct SbasUtcParam {32} {
  UtcOffs utc; // SBAS to UTC time offset parameters
  i1 utcsi;    // UTC Standard Identifier[]
  u4 tow;      // Reference time of week [s]
  u2 wn;       // Reference week number []
  u1 flags;    // Flags, reserved (always 0)
  u1 cs;       // Checksum
};
```

This message has much in common with the [UO] message. The `utcsi` field may have one of the following values:

Table 3-5. UTC Standard Identifier

Value	Meaning
0	UTC as operated by the Communications Research Laboratory (CRL), Tokyo, Japan
1	UTC as operated by the National Institute of Standards and Technology (NIST)
2	UTC as operated by the U. S. Naval Observatory (USNO)
3	UTC as operated by the International Bureau of Weights and Measures (BIPM)
[4…7]	Reserved

[EU] Galileo UTC and GPS Time Parameters

```
struct GalUtcGpsParam {40} {
  UtcOffs utc; // Galileo to UTC time offset parameters
  // Galileo to GPS time offset parameters
  f4 a0g;      // Constant term of time offset [s]
  f4 a1g;      // Rate of time offset [s/s]
  u4 t0g;      // Reference time of week
  u2 wn0g;     // Reference week number
  u2 flags;    // Flags of data availability [bitfield]
               //     0 - GGTO availability
               //  1…15 - reserved
  u1 cs;       // Checksum
};
```

[QU] QZSS UTC Time Parameters

```
struct QzssUtcParam {24} {
  UtcOffs utc; // QZSS UTC time offset parameters
  u1 cs;       // Checksum
};
```

[CU] BeiDou UTC Time Parameters

```
struct BeiDouUtcParam {24} {
  UtcOffs utc; // BeiDou UTC time offset parameters
  u1 cs;       // Checksum
};
```

[IU] IRNSS UTC Time Parameters

```
struct IrnssUtcParam {24} {
  UtcOffs utc; // IRNSS UTC time offset parameters
  u1 cs;       // Checksum
};
```

[NU] GLONASS UTC and GPS Time Parameters

```
struct GloUtcGpsParam {27} {
  f8 tauSys;  // Time correction to GLONASS time scale (vs. UTC(SU))
              //   tauSys = Tutc(su) - Tglo [s]
  f4 tauGps;  // tauGps = Tgps - Tglo [s]
  f4 B1;      // Coefficient for calculation of UT1
  f4 B2;      // Coefficient for calculation of UT1
  u1 KP;      // Leap second information
  u1 N4;      // Number of 4-year cycle [1…31]
  i2 Dn;      // Day number within 4-year period []
  i2 Nt;      // Current day number at the decoding time
  u1 cs;      // Checksum
};
```

This message contains GLONASS UTC and GPS time parameters. Please refer to GLONASS ICD for details.

3.4.5 Position/Velocity Messages

[ST] Solution Time-Tag

```
struct SolutionTime {6} {
  u4 time;     // Solution time. Tr modulo 1 day (86400000 ms)[ms]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

Specifies the receiver time of the current position solution. Note that this time-tag may differ from the current receiver time if the receiver runs in RTK delay mode. In this case the time tag from this message is typically in the past with respect to the time tag of the current epoch.

[PO] Cartesian Position

```
struct Pos {30} {
  f8 x, y, z;  // Cartesian coordinates [m]
  f4 pSigma;   // Position 3D RMS [m]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

[Po] (PoWgs,PoLoc) Cartesian Position in Specific System

```
struct SpecificCrtPos {38} {
  f8 x, y, z;     // Cartesian coordinates [m]
  f4 pSigma;      // Position 3D RMS [m]
  u1 solType;     // Solution type
  u1 system;      // Source of position
                  // 0 - WGS
                  // 1 - Local
  a1 crsCode[5]; // Name of the coordinate reference system
  u2 chIssue;     // Counter incrementing on every potential change
                  // of user grid system
  u1 cs;          // Checksum
};
```

When enabled as `/msg/jps/PoWgs`, the resulting message will contain WGS coordinates and corresponding value `0` in its `system` field.

When enabled as `/msg/jps/PoLoc`, the resulting message will contain coordinates in local coordinate system and corresponding value `1` in its `system` field.

[VE] Cartesian Velocity

```
struct Vel {18} {
  f4 x, y, z;  // Cartesian velocity vector [m/s]
  f4 vSigma;   // Velocity 3D RMS [m/s]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

[PV] Cartesian Position and Velocity

```
struct PosVel {46} {
! f8 x, y, z;     // Cartesian coordinates [m]
! f4 pSigma;      // Position 3D RMS [m]
! f4 vx, vy, vz;  // Cartesian velocities [m/s]
! f4 vSigma;      // Velocity 3D RMS [m/s]
  u1 solType;     // Solution type
  u1 cs;          // Checksum
};
```

[PG] Geodetic Position

```
struct GeoPos {30} {
  f8 lat;       // Latitude [rad]
  f8 lon;       // Longitude [rad]
  f8 alt;       // Ellipsoidal height [m]
  f4 pSigma;    // Position 3D RMS [m]
  u1 solType;   // Solution type
  u1 cs;        // Checksum
};
```

[Pg] (PgWgs,PgLoc) Geodetic Position in Specific System

```
struct SpecificGeoPos {38} {
  f8 lat;         // Latitude [rad]
  f8 lon;         // Longitude [rad]
  f8 alt;         // Ellipsoidal height [m]
  f4 pSigma;      // Position 3D RMS [m]
  u1 solType;     // Solution type
  u1 system;      // Coordinate system
                  // 0 - WGS
                  // 1 - Local
  a1 crsCode[5]; // Name of the coordinate reference system
  u2 chIssue;     // Counter incrementing on every potential change
                  // of user grid system
  u1 cs;          // Checksum
};
```

When enabled as `/msg/jps/PgWgs`, the resulting message will contain WGS coordinates and corresponding value `0` in its `system` field.

When enabled as `/msg/jps/PgLoc`, the resulting message will contain coordinates in local coordinate system and corresponding value `1` in its `system` field.

[VG] Geodetic Velocity

```
struct GeoVel {18} {
  f4 lat;       // Northing velocity [m/s]
  f4 lon;       // Easting velocity [m/s]
  f4 alt;       // Height velocity [m/s]
  f4 vSigma;    // Velocity 3D RMS [m/s]
  u1 solType;   // Solution type
  u1 cs;        // Checksum
};
```

[SG] Position and Velocity RMS Errors

```
struct Rms {18} {
! f4 hpos;      // Horizontal position RMS error [m]
! f4 vpos;      // Vertical position RMS error [m]
! f4 hvel;      // Horizontal velocity RMS error [m/s]
```

```
! f4 vvel;     // Vertical velocity RMS error [m/s]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

[mp] Position in Local Plane

```
struct LocalPlanePos {45} {
  f8 n;         // Northern coordinate [m]
  f8 e;         // Eastern coordinate [m]
  f8 u;         // Altitude above local ellipsoid [m]
  f8 sep;       // Geoid separation relatively to local ellipsoid [m]
  f4 pSigma;    // Position 3D RMS [m]
  u1 solType;   // Solution type
  u1 grid;      // Grid source
                //   0 - none
                //   1 - predefined grid
                //   2 - user defined grid
                //   3 - result of localization
                //   4 - grid got from external source
  u1 geoid;     // Geoid source
                //   0 - none
                //   1 - predefined geoid
                //   2 - user defined geoid
                //   4 - geoid got from external source
  u2 prj;       // EPSG code of used projection
  u1 gridZone; // Grid zone for global systems UTM and UPS, 0 otherwise
  u2 chIssue;  // Counter incrementing on every potential change
                // of user grid system
  u1 cs;        // Checksum
};
```

[bp] Reference Station Position in Local Plane

```
struct RSLocalPlanePos {42} {
  f8 n;         // Northern coordinate [m]
  f8 e;         // Eastern coordinate [m]
  f8 u;         // Altitude above local ellipsoid [m]
  f8 sep;       // Geoid separation relatively to local ellipsoid [m]
  f4 pSigma;    // Position 3D RMS [m]
  u1 solType;   // Solution type
  u1 grid;      // Grid source
                //   0 - none
                //   1 - predefined grid
                //   2 - user defined grid
                //   3 - result of localization
                //   4 - grid got from external source
  u1 geoid;     // Geoid source
                //   0 - none
                //   1 - predefined geoid
                //   2 - user defined geoid
                //   4 - geoid got from external source
  u2 prj;       // EPSG code of used projection
  u1 cs;        // Checksum
};
```

[DP] Dilution of Precision (DOP)

```
struct Dops {18} {
  f4 hdop;      // Horizontal dilution of precision (HDOP)[]
  f4 vdop;      // Vertical dilution of precision (VDOP) []
  f4 tdop;      // Time dilution of precision (TDOP) []
  u1 solType;   // Solution type
  f4 edop;      // East dilution of precision (eDOP) []
  u1 cs;        // Checksum
};
```

[SP] Position Covariance Matrix

```
struct PosCov {42} {
  f4 xx;        // [m^2]
  f4 yy;        // [m^2]
  f4 zz;        // [m^2]
  f4 tt;        // [m^2]
  f4 xy;        // [m^2]
  f4 xz;        // [m^2]
  f4 xt;        // [m^2]
  f4 yz;        // [m^2]
  f4 yt;        // [m^2]
  f4 zt;        // [m^2]
  u1 solType;   // Solution type
  u1 cs;        // Checksum
};
```

[SV] Velocity Covariance Matrix

```
struct VelCov {42} {
  f4 xx;        // [(m/s)^2]
  f4 yy;        // [(m/s)^2]
  f4 zz;        // [(m/s)^2]
  f4 tt;        // [(m/s)^2]
  f4 xy;        // [(m/s)^2]
  f4 xz;        // [(m/s)^2]
  f4 xt;        // [(m/s)^2]
  f4 yz;        // [(m/s)^2]
  f4 yt;        // [(m/s)^2]
  f4 zt;        // [(m/s)^2]
  u1 solType;   // Solution type
  u1 cs;        // Checksum
};
```

[BL] Baseline

For dual-antenna receivers: baseline vector from master to slave antenna.

For all other receivers: baseline vector from external reference station.

```
struct Baseline {34} {
  f8 x, y, z;  // Calculated baseline vector coordinates [m]
  f4 sigma;    // Baseline 3D RMS [m]
  u1 solType;  // Solution type
  i4 time;     // receiver time of the baseline estimate [s]
  u1 cs;       // Checksum
};
```

[SH] Heading Baseline Covariance Matrix

For dual-antenna receivers: covariance matrix of baseline vector from master to slave antenna.

```
struct HeadCov {26} {
  f4 xx;       // [m^2]
  f4 yy;       // [m^2]
  f4 zz;       // [m^2]
  f4 xy;       // [m^2]
  f4 xz;       // [m^2]
  f4 yz;       // [m^2]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

[EB] External Baseline

This message contains baseline vector from external reference station even for dual-antenna receivers where `[BL]` reports baseline between two antennas.

```
struct Baseline {34} {
  f8 x, y, z;  // Calculated baseline vector coordinates [m]
  f4 sigma;    // Baseline 3D RMS [m]
  u1 solType;  // Solution type
  i4 time;     // receiver time of the baseline estimate [s]
  u1 cs;       // Checksum
};
```

[bL] Attitude Baselines

```
struct Baselines {52} {
  f4 bl0[3];     // baseline vector M-S0 [m]
  f4 bl1[3];     // baseline vector M-S1 [m]
  f4 bl2[3];     // baseline vector M-S2 [m]
  f4 rms[3];     // estimated accuracies for baseline vectors [m]
  u1 solType[3]; // solution types for baseline vectors
  u1 cs; // Checksum
};
```

[mR] Attitude Full Rotation Matrix

```
struct FullRotationMatrix {37} {
  f4 q00, q01, q02; // components of the rotation matrix Q []
  f4 q10, q11, q12; // components of the rotation matrix Q []
  f4 q20, q21, q22; // components of the rotation matrix Q []
  u1 cs;            // Checksum
};
```

[PS] Position Statistics

```
struct PosStat {9 + 3 × N_SYS} {
  u1 solType;    // Solution type
  u1 gpsLocked;  // Number of GPS SVs locked
  u1 gloLocked;  // Number of GLONASS SVs locked
  u1 gpsAvail;   // Number of GPS SVs available for positioning
  u1 gloAvail;   // Number of GLONASS SVs available for positioning
  u1 gpsUsed;    // Number of GPS SVs used in positioning
  u1 gloUsed;    // Number of GLONASS SVs used in positioning
  u1 fixProg;    // Ambiguity fixing progress indicator
                 // controllable by RTK engine [%]
  SysPosStat stat[N_SYS]; // Statistics for the rest of the systems
  u1 cs;         // Checksum
};

struct SysPosStat {
  u1 locked;  // Number of SVs locked
  u1 avail;   // Number of SVs available for positioning
  u1 used;    // Number of SVs used in positioning
}
```

The systems in the `stat[N_SYS]` array are the rest of the systems supported by given receiver, in the following order: Galileo, SBAS, QZSS, BeiDou, IRNSS.

The `fixProg` field may vary from 0% to 100%, though in practice if raw measurements are good enough, the `fixProg` field rarely takes values other than zero. Just occasionally you can see `fixProg` to be 100%. This means that the engine has just finished fixing all available ambiguities. The `fixProg` will be dropped to zero immediately after it has reached 100%.

If the `fixProg` field keeps varying between 0% and 100% exclusive, this means that not all of the ambiguities have been fixed. Here are possible reasons for such behavior:

- You have just launched the RTK engine and it is trying to get a first fixed solution.
- There is one or more problem satellites whose measurements prevent the engine from fixing all available ambiguities “in batch”.
- The receiver has just started tracking one or more “new” satellites. It will take the RTK engine some time to fix these new ambiguities.

Also note that if the solution type is “RTK fixed”, the number of SVs with float ambiguities is:

gpsAvail + gloAvail - (gpsUsed + gloUsed)

[PT] Time of Continuous Position Computation

```
struct PosCompTime {5} {
  u4 pt;  // Continuous position computation time [s]
  u1 cs;  // Checksum
};
```

Specifies the time interval over which continuous position computation has been possible. If the receiver is unable to compute any position at the current epoch, the Time of Continuous Position Computation counter is zeroed.

3.4.6 Satellite Measurements

Generic Messages Description

In this section we will focus on messages containing “satellite specific information”. These kinds of messages include satellite measurements (code and carrier phase measurements, elevations, azimuths, etc.).

Different applications may utilize different sets of measurements. It is almost impossible to select a fixed set of combinations of satellite measurements that would be enough universal yet compact. Instead receiver provides dedicated message for each particular measurement type. Every individual measurement message contains some specific (“homogeneous”) data for the satellites tracked.

For any given epoch, data for particular satellite is put at the same position (index) in all the messages. *Satellite Index,* that from now on is called `SvsIdx` in this manual, will establish correspondence between given satellite and the place of its observables in the messages. Message [SX] “Extended Satellite Index” is used to represent the `SvsIdx` in the data stream and should be used to establish and update the `SvsIdx` as the data stream is being processed. The number of satellites in `SvsIdx`, `nSats`, should also be obtained from the [SX] message.

For multi-antenna receivers, given satellite will typically appear multiple times in `SvsIdx`, indexing observables obtained from different antennas. The [AN] message could then be used to determine which antenna the observables are obtained from.

Most of the measurements messages may contain special values of corresponding types to indicate lack of data for particular satellite(s). Refer to Table 3-2, “Special Values for Fields,” on page 63 for details.

Extended Satellite Identifier (ESI)

Note: ESI obsoletes old enumeration system based on USI which range has been exhausted. Refer to "Backward Compatibility Considerations" on page 90 for further discussion

To handle data corresponding to satellites of different systems in a universal manner, we assign each satellite its Extended Satellite Identifier (ESI):

```
struct ESI {2} {
! u1 ssid;  // Satellite system identifier (SSID)
! u1 svid;  // Satellite identifier (SVID) inside system
};
```

In particular, ESI is used in the [SX] message to represent the `SvsIdx`.

The following table describes ESI:

Table 3-6. Extended Satellite Identifier (ESI)

SSID Value	GNSS System	SVID Meaning and Range
0	Unused. Ignore satellites with this SSID	0
1	GPS (NAVSTAR)	GPS PRNs [1…254]
2	GLONASS	FCN [-7…24][1],25[2],127[3]
3	SBAS	PRN [120…254]
4	Galileo	PRN [1…254]
5	QZSS	PRN [193…254]
6	BeiDou	PRN [1…254]
7	IRNSS	PRN [1…254]
8	L_BAND	PRN [1…254]
9	GLCDMA	PRN [1…254]
[10…254]	Reserved	0
255	Unused. Ignore satellites with this ESI	0

1. Represented in two's complement (i.e., as 'i1' rather than 'u1' field type)
2. Represents WCDMA satellite with unknown FCN.
3. Represents satellite with unknown FCN. Could be useful when converting third-party GLONASS measurement file into GREIS format.

Universal Satellite Identifier (USI)

Warning: *USI is obsolete. ESI should be preferred. Refer to "Backward Compatibility Considerations" on page 90 for further discussion*

Each satellite is assigned its Universal Satellite Identifier (USI) as well as ESI. In particular, USI is used in now obsolete [SI] message — the historical way of representing the `SvsIdx`.

The following table describes USI allocation:

Table 3-7. Universal Satellite Identifiers (USI) Allocation

USI Range	Assigned Satellites
0	Unsupported SVs: ignore
[1…37]	GPS PRNs [1…37]
[38…69]	GLONASS FCNs [-7…24]
70	GLONASS satellite with unknown FCN[1]
[71…119]	Galileo PRNs [1…49]
[120…192]	SBAS PRNs [120…192]
[193…210]	QZSS PRNs [193…210]
[211…254]	BeiDou (COMPASS) PRNs [1…44]
255	Unsupported SVs: ignore

1. Could be useful when converting third-party GLONASS measurement file into GREIS format.

Satellite Signals Allocation and Frequencies

Even though every satellite system has its own set of signals and their names, we conventionally call all the signals CA/L1, P/L1, P/L2, CA/L2, L5, and L1C in GREIS. We also define numeric `SlotId` identifier for each signal.

Standard signal names of all supported satellite systems and their carrier frequencies, along with corresponding GREIS names and `SlotId`, are shown in the following table:

Table 3-8. Satellite Signals Allocation

GREIS SlotId	CA/L1 0	P/L1 1	P/L2 2	CA/L2 3	L5 4	L1C 5
GPS MHz	C/A 1575.42	P1 1575.42	P2 1227.60	L2C(L+M) 1227.60	L5(I+Q) 1176.45	L1C(I+Q) 1575.42
QZSS MHz	C/A 1575.42	L1S 1575.42	L6 1278.75	L2C(L+M) 1227.60	L5(I+Q) 1176.45	L1C(I+Q) 1575.42
SBAS MHz	L1 1575.42				L5(I+Q) 1176.45	
Galileo MHz	E1(B+C) 1575.42	E5 altboc 1191.795	E5B(I+Q) 1207.14	E6(B+C) 1278.75	E5A(I+Q) 1176.45	
GLONASS MHz	CA/L1 $L1_{frq}$[1]	P1 $L1_{frq}$	P2 $L2_{frq}$[2]	CA/L2 $L2_{frq}$	L3(I+Q) 1202.025	
BeiDou MHz	B1 1561.098	altboc 1191.795	B2B(I+Q)[3] 1207.14	B3 1268.52	B2A(I+Q) 1176.45	B1C(I+Q) 1575.42
IRNSS MHz	S 2492.028				L5 1176.45	L1 1575.42

Table 3-8. Satellite Signals Allocation

GREIS SlotId	CA/L1 0	P/L1 1	P/L2 2	CA/L2 3	L5 4	L1C 5
L_BAND MHz	L L_{frq}[4]					
GLCDMA MHz	L1(D+P) 1600.995			L2(D+P) 1248.06	L3(I+Q) 1202.025	

1. $L1_{frq} = 1602 + FCN \times 0.5625$
2. $L2_{frq} = 1246 + FCN \times 0.4375$
3. B2 for BeiDou phase 2 satellites
4. L_{frq} is in range 1545-1559 MHz. Actual value could be obtained from `/par/jppp/beam/cfnom` parameter.

Alignment of Phase Measurements

Receiver provides phase measurements “as-is”, applying no corrections whatsoever (no 1/4 cycle, no 1/2 cycle, no any other), therefore phase relations (e.g., between GPS P2 and L2C signals) are exactly as described in corresponding system ICD. For complex signals (composed of pilot and data signals) receiver phase always refers to the pilot (dataless) signal.

Special Cases for BeiDou Phase Alignment

BeiDou B1/B2/B3 signals have two different variants - without secondary code for GEO satellites (numbers [1…5] and [59…63]), and with secondary code for other non-GEO satellites (numbers [5…58]. Due to this fact it is possible to process these two variants differently. Depending on details of hardware and software implementation it may happen that phase double differences between GEO and non-GEO satellites between receivers of different manufacturers may have 0.5 cycles shift. To avoid this, all manufacturers have to come to common representation.

It so happened that the variant which appears to be the de-facto standard as of today (phases representation in RTCM MSM and RINEX data), and the variant which JAVAD GNSS uses for the raw measurements in JPS file, differ[1]. As it’s inconvenient to change data format of the raw measurements, additional step is needed to convert JPS files to RINEX and RTCM MSM messages.

BeiDou B2 frequency band is even more complex. BeiDou phase 3 satellites transmit B2B signal instead of B2, on the same frequency. Some phase-differential techniques may process signals from this frequency band together, so phase alignment between B2

1. The representation being used in JPS file is quite explainable: multiplication on "+1" element of secondary code changes nothing, while multiplication on "-1" of secondary code inverts phase.

and B2B also may be helpful. Still there are too few manufacturers which provide B2B measurements, and it is unknown what variant of phase alignment will be de-facto standard for RINEX and RTCM MSM.

Overall, to convert BeiDou phase measurement from JAVAD GNSS to RINEX/MSM it is required to add 0.5 cycles to B1/B2/B3 signal phases of the following non-Geo satellites:

- for B1 and B3 signals of satellites [6…58]
- for B2 signals of satellites [6…18].

JAVAD GNSS receivers perform this conversion internally, outputting RTCM MSM messages, and jps2rin utility also does it when converting to RINEX. This function may be changed when de-facto variant of representation of B2B phases (compared B2 phases) appears.

Backward Compatibility Considerations

There are a few considerations that should be taken into account when designing an application that needs to support decoding of satellite measurements generated by old versions of the receiver firmware:

1. Obsoleting of USI /[SI] by ESI/[SX] for enumeration of satellites.
2. Introducing of [RX], [CR], [rx], [cr], [DX], [0d] messages to be able to represent satellites with no CA/L1 signal being track in delta-messages.
3. Changes of K_{sys} and A_{sys} coefficients of “Integer Pseudo-ranges” messages for some of GNSS systems while their support matured.
4. Changes of satellite signals allocation for BeiDou
5. Changes of USI allocation for BeiDou and IRNSS.

Let’s discuss them in turn.

ESI/[SX] Replacing USI/[SI]

Once USI lack of space for new satellites and systems became an issue, the ESI has been introduced to replace USI, and, along with this, the [SX] message containing ESIs was implemented. For smooth transition, the default set of messages was set to include both [SI] and [SX] messages for a few years, and now [SI] has been removed from the default set.

New satellites that have no USI mapping have USI set to 0 in the [SI] message. 0 was always reserved, and thus satellites with this USI should be ignored by all the conforming software. This lets carefully written old software continue to operate correctly if

receiver outputs [SI] (either along with [SX] or not), as [SX] will be ignored as unknown message.

New software caring for compatibility with old data should decode both [SX] and [SI] and use them as sources for SvsIdx. Simplest implementation seems to just update SvsIdx from both messages, taking care to never replace known satellite in the SvsIdx with an unknown one (0 or 255 USI or ESI.SSID).

Decoding of Delta-messages

Delta-messages formats rely on common (for given satellite at given epoch) reference being subtracted from all the values to minimize data size. These references pr_ref, PR_REF, and DP_REF, are used for different kinds of delta-messages.

Historically, these references used to be taken from corresponding CA/L1 messages, [rc], [RC], and [DC], as receiver firmware was not able to track other signals without CA/L1 signal. At some point, requirement for CA/L1 tracking was relaxed, and thus the need to make delta-messages independent of CA/L1 tracking arose. At this point separate messages to hold reference pseudo-ranges and doppler were introduced. This messages are [rx], [RX], and [DX]. In addition, new delta-messages to contain CA/L1 delta-pseudo-range and delta-doppler were implemented to make the entire system symmetric and CA/L1-independent. Notice that in the old representation these latter messages were not needed, as CA/L1 delta-pseudo-range and delta-doppler were always exactly zero due to definition of corresponding references.

To preserve backward compatibility, the references are still defined so that they hold CA/L1 signal data, provided it is available, but newer software should not rely on this feature. Instead, it should prefer data from [rx], [RX], and [DX] over data from [rc], [RC], and [DC] for the purpose of defining of references pr_ref, PR_REF, and DP_REF, respectively.

Overall, an algorithm for decoding of phases and pseudo-ranges from delta-messages could look like this (for dopplers the logic is similar to pseudo-ranges):

```
step 0: clear 'ignore_rc' and 'ignore_RC' flags.
step 1: loop: look for [rx], [rc], [RX], and [RC] messages:

    NOTE: do not populate values in CA/L1 pseudo-range arrays from
      either [rc] or [RC] here - better do this in loop 2, as part of
      common phases/pseudo-ranges decoding.

    - if [rx]: populate values in pr_ref array from corresponding valid
```

```
      values from [rx], unconditionally. (This will override value got
      from [rc], if any.) Set 'ignore_rc' flag.
    - if [rc]: if 'ignore_rc' is set, entirely ignore [rc] in this loop.
      Otherwise populate values in pr_ref array from corresponding valid
      values from [rc].
    - if [RX]: populate values in PR_REF array from corresponding valid
      values from [RX], unconditionally. (This will override value got
      from [RC], if any.) Set 'ignore_RC' flag.
    - if [RC]: if 'ignore_RC' is set, entirely ignore [RC] in this loop.
      Otherwise populate values in PR_REF array from corresponding valid
      values from [RC].

step 3: loop: for all pseudo-range and phase messages excluding [rx]/[RX]:

    NOTE: [rc] and [RC] should be decoded/processed here the same way
    as other pseudo-range messages.

    - To decode relative carrier phases [CP], [1P], …, [lP], use PR_REF.
    - To decode relative carrier phases [cp], [1p], …, [lp], use pr_ref.
    - To decode all the relative pseudo-range messages, use PR_REF if
      available, otherwise use pr_ref if available.
```

A_{sys} and K_{sys} Changes

Unfortunately, over time a few changes to A_{sys} and K_{sys} coefficients (see Table 3-9 on page 95) have been made for some GNSS systems. Here is the history of these changes:

Firmware Version	Issue Date	What	From	To
3.2.7b0	2011-04-12	Galileo A_{sys}	0.075	0.090
3.5.6	2014-03-06	Galileo A_{sys} SBAS A_{sys}	0.090 0.115	0.085 0.125
3.7.0	2017-03-03	Galileo K_{sys}[1]	1×10^{-11}	2×10^{-11}

1. If there is either of [RX] or [CR] in the stream, then it's safe to assume new Ksys values as well.

USI Allocation for BeiDou and IRNSS

Starting from firmware version 3.7.0, IRNSS has been removed from USI, and BeiDou range has been extended into the cleared space. Firmware version could usually be taken from standard GREIS file from [PM] message being output at the beginning of each file.

Satellite Signals Allocation for BeiDou

Starting from firmware version 3.7.0, BeiDou `B2` signal has been moved from `L5` to `P2` slot. Firmware version could usually be taken from standard GREIS file from [PM] message being output at the beginning of each file.

[SX] Extended Satellite Indices

Note: This message obsoletes [SI] message. Refer to "Backward Compatibility Considerations" on page 90 for further discussion

```
struct ExtSatIndex {2×nSats+1} {
  ESI esi[nSats];  // ESI array []
  u1 cs;           // Checksum
};
```

The [SX] message contains ESI for each satellite in `SvsIdx`, thus establishing mutual correspondence between satellite identifier and array index allocated to this satellite. Decoding of [SX] is required to build and update the `SvsIdx` from the GREIS message stream.

The number of satellites in the `SvsIdx`, `nSats`, should be calculated from the length of [SX] message body (taken from the message header):

$$nSats = (length - 1) / 2$$

From practical point of view, [SX] is the way to build and update the SvsIdx from the GREIS message stream.

[AN] Antenna Names

```
struct AntName{nSats+1} {
  a1 name[nSats];  // Antenna names[a…z]
  u1 cs;           // Checksum
};
```

This message contains antenna name (ASCII character in the range [a…z]) for every satellite in `SvsIdx`.

This message is only available for multi-antenna receivers.

[NN] GLONASS Satellite System Numbers

```
struct SatNumbers {nGloSats+1} {
! u1 osn[nGloSats];  // GLONASS SV orbit slot number []
  u1 cs;             // Checksum
};
```

The [NN] message contains the orbit slot number for every GLONASS satellite in SvsIdx. Here nGloSats designates the number of GLONASS satellites in SvsIdx.

[EL] Satellite Elevations

```
struct SatElevation {nSats+1} {
! i1 elev[nSats];  // Elevation angle [degrees] [-90…90)
  u1 cs;           // Checksum
};
```

This message contains elevations for all the satellites in SvsIdx.

[AZ] Satellite Azimuths

```
struct SatAzimuth {nSats+1} {
! u1 azim[nSats];  // Azimuth angle [degrees×2] [0…180)
  u1 cs;           // Checksum
};
```

This message contains azimuths for all the satellites in SvsIdx. The notation [degrees×2] means that the values from the message must be multiplied by 2 to restore actual azimuths in degrees.

[RX], [RC], [R1], [R2], [R3], [R5], [RI]: Pseudo-ranges

```
struct PR {8×nSats+1} {
! f8 pr[nSats];  // Pseudo-range, [s]
  u1 cs;         // Checksum
};
```

These messages contain corresponding pseudo-ranges for all the satellites in SvsIdx. The [RX] message contains *virtual references* PR_REF. The rest of the messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C pseudo-ranges, respectively.

The PR_REF from the [RX] message is used for definition of relative pseudo-range and relative carrier phase messages. For backward compatibility, virtual reference pseudo-range PR_REF is defined so that its value is equal to CA/L1 pseudo-range obtained from [RC] message whenever CA/L1 pseudo-range is available. This way old software that uses values from [RC] message to decode dependent messages will still obtain correct results.

[rx], [rc], [r1], [r2], [r3], [r5], [rl]: Integer Pseudo-ranges

```
struct SPR {4×nSats+1} {
! i4 spr[nSats];  // (PR[s] - A_sys) / K_sys
  u1 cs;          // Checksum
};
```

These messages contain corresponding short pseudo-ranges for all the satellites in `SvsIdx`. The `[rx]` message contains short *virtual references* `pr_ref`. The rest of the messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C pseudo-ranges, respectively.

Use the following formula to restore true pseudo-ranges in seconds:

$$pr = spr \times K_{sys} + A_{sys}$$

where:

Table 3-9. System-specific Coefficients K_{sys} and A_{sys}

GNSS	K_{sys}	A_{sys}
GPS	1×10^{-11}	0.075
GLONASS	1×10^{-11}	0.075
Galileo	2×10^{-11}	0.085
SBAS	1×10^{-11}	0.125
QZSS	2×10^{-11}	0.125
BeiDou	2×10^{-11}	0.105
IRNSS	2×10^{-11}	0.105
GLCDMA	1×10^{-11}	0,075

The `pr_ref` from the `[rx]` message is used for definition of relative pseudo-range and integer relative carrier phase messages. For backward compatibility, short virtual reference pseudo-range `pr_ref` is defined so that its value is equal to CA/L1 pseudo-range obtained from `[rc]` message whenever CA/L1 pseudo-range is available. This way old software that uses values from `[rc]` message to decode dependent messages will still obtain correct results.

Note that `pr_ref` could also be calculated from `PR_REF`, but not vice versa. This could help to decode exotic combinations of messages where `[rx]` is not available, but there are some other messages that need `pr_ref` for their decoding, and `[RX]` happens to be there. To calculate `pr_ref` from `PR_REF`, use:

$$pr\_ref = trunc((PR\_REF - A_{sys}) / K_{sys}) \times K_{sys} + A_{sys}$$

where `trunc(x)` function rounds `x` toward zero.

[prr]: CA/L1 Relative Pseudo-range Combo

This virtual message enables output of [rc], [rx], and [cr] messages, implementing some interdependency rules to save space in the default set of messages and to promote smooth transition from [rc] to [rx] - based decoding.

The interdependency rules are:

1. [rc] is output as usual, if there is it least one CA/L1 pseudo-range measurement for it.
2. [rx] is output only if there is at least one pseudo-range or phase measurement other than CA/L1 for an SVs with no CA/L1 measurements.
3. [cr] is output only if both [rc] and [rx] are output.

[CR], [1R], [2R], [3R], [5R], [IR]: Relative Pseudo-ranges

```
struct RPR {4×nSats+1} {
! f4 rpr[nSats];  // PR - REF, [s]
  u1 cs;          // Checksum
};
```

These messages contain relative CA/L1, P/L1, P/L2, CA/L2, L5, and L1C pseudo-ranges, respectively, for all the satellites in `SvsIdx`.

Use the following formula to restore true pseudo-range in seconds:

$$pr = rpr + REF$$

where `REF` is either of corresponding virtual reference pseudo-ranges `pr_ref`, or `PR_REF`, whatever is available.

[cr], [1r], [2r], [3r], [5r], [Ir]: Integer Relative Pseudo-ranges

```
struct SRPR {2×nSats+1} {
! i2 srpr[nSats];  // (PR[s] - REF[s] - 2×10^-7) × 10^11
  u1 cs;           // Checksum
};
```

These messages contain short relative CA/L1, P/L1, P/L2, CA/L2, L5, and L1C pseudo-ranges, respectively, for all the satellites in `SvsIdx`.

Use the following formula to restore true pseudo-range in seconds:

$$pr = srpr \times 10^{-11} + 2\times10^{-7} + REF$$

where `REF` is either of corresponding virtual reference pseudo-ranges `pr_ref`, or `PR_REF`, whatever is available.

[cm], [1m], [2m], [3m], [5m], [lm]: Pseudo-range Corrections

```
struct PrCorr {2×nSats+2} {
! i2 prc[nSats]; // Correction [Seconds × 1e11]
  u1 mode;       // Mode
  u1 cs;         // Checksum
};
```

Where `mode` corresponds to the value of `/par/raw/corr/ca/code` parameter as follows:

- `0` – normal
- `1` – data
- `2` – mpear
- `3` – mpnew
- `4` – mp2ne
- `5` – mpxne

These messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C pseudo-range corrections, respectively, for all the satellites in `SvIdx`. They allow to compute alternative pseudo-ranges to those being output in pseudo-range messages (i.e., to compute corrected pseudo-ranges when `mode` is 0, and raw pseudo-ranges otherwise.)

When `mode` is `1` (data), corrections are "pilot" pseudo-range minus "data" pseudo-range. For for all the other values of `mode` corrections are multipath corrections.

Use the following formula to compute corrections in seconds:

$$prcs = prc \times 10^{-11}$$

Use the following formula to compute alternative pseudo-ranges:

$$aprs = prs + prcs$$

where `prs` is pseudo-range taken from corresponding pseudo-range message (e.g., from [RC]).

[CC],[C1],[C2],[C3],[C5],[CI]: Smoothing Corrections

```
struct SC {6×nSats+1} {
! Smooth smooth[nSats];  // PR smoothing
  u1 cs;                 // Checksum
};
```

where "Smooth" format is defined as follows:

```
struct Smooth {6} {
! f4 value;     // Smoothing correction [s]
! u2 interval;  // Smoothing interval [s]
};
```

These messages contain corresponding pseudo-range smoothing corrections and corresponding smoothing intervals for all the satellites in SvsIdx. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C smoothing corrections, respectively.

Use the following formula to compute smoothed pseudo-ranges in seconds:

$$pr\_sm = pr + smooth.value$$

[cc],[c1],[c2],[c3],[c5],[cl]: Smoothing Corrections

```
struct SS {2×nSats+1} {
! i2 smooth[nSats];  // Smoothing correction [s×10^-11]
  u1 cs;             // Checksum
};
```

These messages contain corresponding short pseudo-range smoothing corrections for all the satellites in SvsIdx. The [cc], [c1], [c2], [c3], [c5], and [cl] messages contain short CA/L1, P/L1, P/L2, CA/L2, L5, and L1C smoothing corrections, respectively.

Use the following formula to compute smoothed pseudo-ranges in seconds:

$$pr\_sm = pr + smooth \times 10^{-11}$$

[PC], [P1], [P2], [P3], [P5], [Pl]: Carrier Phases

```
struct CP {8×nSats+1} {
! f8 cp[nSats];  // CP, [cycles]
  u1 cs;         // Checksum
};
```

These messages contain corresponding carrier phases for all the satellites in SvsIdx. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C carrier phases, respectively.

[pc], [p1], [p2], [p3], [p5], [pl]: Integer Carrier Phases

```
struct SCP {4×nSats+1} {
! u4 scp[nSats];  // CP, [cycles/1024]
  u1 cs;          // Checksum
};
```

Note: The “scp” field will have discontinuities due to rollovers. Refer to “Compensating for Phase Rollovers” on page 581 for details.

These messages contain corresponding short carrier phases for all the satellites in SvsIdx. The messages contain short CA/L1, P/L1, P/L2, CA/L2, L5, and L1C carrier phases, respectively.

Use the following formula to compute full carrier phases in cycles:

$$cp = scp / 1024.0$$

[CP],[1P],[2P],[3P],[5P],[IP]: Relative Carrier Phases

```
struct RCP_RC {4×nSats+1} {
! f4 rcp[nSats];  // cp / F_Ln - PR_REF [s]
  u1 cs;          // Checksum
};
```

These messages contain differences between the full corresponding carrier phases and the matching [RC] pseudo-ranges for all the satellites in `SvsIdx`. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C carrier phases, respectively.

Use the following formula to compute true carrier phases in cycles:

$$cp = (rcp + PR\_REF) \times F_{Ln}$$

where:

`PR_REF` – s the value taken from corresponding [RX] message, and converted to seconds as specified in the description of the [RX] message

F_{Ln} – is nominal Ln carrier frequency for corresponding satellite, e.g., nominal L2 frequency for [2P] and [3P] messages, and nominal L1 frequency for [CP] and [1P] messages.

[cp],[1p],[2p],[3p],[5p],[Ip]: Integer Relative Carrier Phases

```
struct RCP_rc {4×nSats+1} {
! i4 rcp[nSats];  // cp / F_Ln - pr_ref, [s×2^-40]
  u1 cs;          // Checksum
};
```

These messages contain the differences between the full corresponding carrier phases and the matching [rx] pseudo-ranges for all the satellites in `SvsIdx`. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C carrier phases, respectively.

Use the following formula to retrieve true carrier phases in cycles:

$$cp = (rcp \times 2^{-40} + pr\_ref) \times F_{Ln}$$

where:

`pr_ref` – is the value taken from corresponding [rx] message, and converted to seconds as specified in the description of the [rx] message.

F_{Ln} – is nominal Ln carrier frequency for corresponding satellite, e.g., nominal L2 frequency for [2p] and [3p] messages, and nominal L1 frequency for [cp] and [1p] messages.

[cf], [1f], [2f], [3f], [5f], [lf]: Phase Corrections

```
struct PhCorr {2×nSats+2} {
! i2 phc[nSats]; // Correction [cycles × 1024]
  u1 mode;       // Mode
  u1 cs;         // Checksum
};
```

Where `mode` corresponds to the value of `/par/raw/corr/ca/carrier` parameter as follows:

`0` - normal

`3` - mpnew

These messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C phase corrections, respectively, for all the satellites in `SvIdx`. They allow to compute alternative phases to those being output in phase messages (i.e., to compute corrected phases when `mode` is 0, and raw phases otherwise.)

When `/par/raw/corr/ca/code` is set to `data` and `mode` is `0`, corrections are "data" phase minus "pilot" phase, otherwise the corrections are multipath corrections.

Use the following formula to compute corrections in cycles:

$$phcc = phc / 1024$$

Use the following formula to compute alternative phases:

$$aphcc = phc + phcc$$

where `phc` is phase taken from corresponding phase message (e.g., from [PC]).

[DX], [DC], [D1], [D2], [D3], [D5], [Dl]: Doppler

```
struct DP {4×nSats+1} {
! i4 dp[nSats];  // DP [Hz×10^-4]
  u1 cs;         // Checksum
};
```

These messages contain corresponding doppler estimates for all the satellites in `SvsIdx`. The messages contain *virtual reference* `DP_REF`, CA/L1, P/L1, P/L2, CA/L2, L5, and L1C doppler, respectively.

Use the following formula to compute true doppler:

$$doppler = dp \times 10^{-4}$$

The `DP_REF` from the `[DX]` message is used for definition of relative doppler messages. For backward compatibility, virtual reference doppler `DP_REF` is defined so that its value is equal to CA/L1 doppler obtained from `[DC]` message whenever CA/L1 doppler is available. This way old software that uses values from `[DC]` message to decode dependent messages will still obtain correct results.

[drr]: CA/L1 Relative Doppler Combo

This virtual message enables output of [DC], [DX], and [0d] messages, implementing some interdependency rules to save space in the default set of messages and to promote smooth transition from [DC] to [DX] - based decoding.

The interdependency rules are:

1. [DC] is output as usual, if there is it least one CA/L1 doppler measurement for it.
2. [DX] is output only if there is at least one doppler measurement other than CA/L1 for an SVs with no CA/L1 doppler.
3. [0d] is output only if both [DC] and [DX] are output.

[0d],[1d], [2d], [3d], [5d], [Id]: Relative Doppler

```
struct SRDP {2×nSats+1} {
! i2 srdp[nSats];  // (dp × F_L1/F_Ln - dpCA1)[Hz×10^-4]
  u1 cs;           // Checksum
};
```

These messages contain corresponding short doppler relative to virtual reference doppler for all the satellites in `SvsIdx`. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C short relative doppler, respectively.

Use the following formula to compute true doppler:

$$\text{doppler} = (\text{srdp} + \text{DP\_REF}) \times F_{Ln} / F_{L1} \times 10^{-4}$$

where:

`DP_REF` – is the value `dp` taken from the [DX] message for given SV

F_{L1} – is the nominal CA/L1 frequency of the corresponding satellite (refer to Table 3-8 on page 88).

F_{Ln} – is the nominal Ln frequency of the corresponding satellite.

[EC], [E1], [E2], [E3], [E5], [EI]: SNR

```
struct CNR {nSats+1} {
! u1 cnr[nSats];  // C/N0 [dB×Hz]
  u1 cs;          // Checksum
};
```

These messages contain corresponding carrier to noise ratios for all the satellites in `SvsIdx`. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C carrier to noise ratio, respectively.

[CE], [1E], [2E], [3E], [5E], [IE]: SNR x 4

```
struct CNR_4 {nSats+1} {
! u1 cnrX4[nSats];  // C/N0 [0.25×dB×Hz]
  u1 cs;            // Checksum
};
```

These messages contain corresponding carrier to noise ratios for all the satellites in SvsIdx. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C carrier to noise ratio multiplied by 4, respectively.

Use the following formula to compute true carrier to noise ratio in dB×Hz:

$$cnr = cnrX4 \times 0.25$$

[s0], [s1], [s2], [s3], [s5], [sI]: SNR x 256

```
struct CNR_256 {2×nSats+1} {
! u2 cnrX256[nSats];  // C/N0 [(1/256)×dB×Hz]
  u1 cs;              // Checksum
};
```

These messages contain corresponding carrier to noise ratios for all the satellites in SvsIdx. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C carrier to noise ratio multiplied by 256, respectively.

Use the following formula to compute true carrier to noise ratio in dB×Hz:

$$cnr = cnrX256 / 256$$

[j0], [j1], [j2], [j3], [j5], [jI]: Data SNR x 256

```
struct CNR_256 {2×nSats+1} {
! u2 cnrX256[nSats];  // C/N0 [(1/256)×dB×Hz]
  u1 cs;              // Checksum
};
```

These messages contain corresponding "data" (as opposed to “pilot”) sub-signal carrier to noise ratios for all the satellites in SvsIdx. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C carrier to noise ratio multiplied by 256, respectively.

Use the following formula to compute true carrier to noise ratio in dB×Hz:

$$cnr = cnrX256 / 256$$

[FC],[F1],[F2],[F3],[F5],[FI]: Signal Lock Loop Flags

```
struct Flags {2×nSats+1} {
  u2 flags[nSats];  // Lock Loop Flags [bitfield]
  u1 cs;            // Checksum
};
```

These messages contain an array of corresponding signal lock loop flags for all the satellites in `SvsIdx`. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C signal lock loop flags, respectively.

The following flags are defined:

bit#	Hex Mask	Description
0	0x0001	PLL is in phase lock
1	0x0002	Satellite signal strength is sufficient
2…3	0x000C	Unused
4	0x0010	CLL is in steady state phase lock
5	0x0020	Loss-of-lock occurred in PLL between the previous and the current epochs
6	0x0040	Integral data quality indicator
7	0x0080	Not Used
8	0x0100	Preamble detected
9	0x0200	“Data” signal is used in PLL/DLL (for “data+pilot” signals only)
10	0x0400	“Pilot” signal is used in PLL/DLL (for “data+pilot” signals only)
11	0x0800	Pseudo-range is not full pseudo-range and thus should not be used in position computation. This flag may appear when parameter `/par/raw/meas/mode` is set to `modulo`
12	0x1000	Spoofing detected
13	0x2000	Jamming detected
14,15	0xC000	Reserved for internal purposes

Note: bit#5 is not suitable for loss-of-lock detection in applications, – use [TC] message for this purpose instead.

Note: bits #9 and #10: receiver tries to use both sub-signals, but in case of low C/N0 it automatically switches to pilot-only tracking.

The simplest approach to data validation is keeping track of only bit #6. As long as this bit remains set for a particular satellite, all of this satellite's measurements for corresponding signal type are considered good. Note that the receiver normally utilizes very narrow CLL bandwidths, thus, quite a long settling-down time (tens of seconds). In fact, it is not worth waiting until the pseudo-range noise error reaches its steady-state level.

Note that bit#6 is set as soon as the measured pseudo-ranges become “accurate enough” (i.e., irrespective of whether the formal settling-down period is over or not). On the other hand, for code differential applications, pseudo-range accuracy is of critical importance. If bit#4 is set, this indicates that the corresponding pseudo-ranges are generated after the loop having reached the “steady state” and therefore are considered the least noisy.

In fact, it is not infrequent that raw data are used even if bit #6 is not set. In such cases, however, all responsibility for providing valid results rests with the user.

Bits ##0…3 are used for internal purposes.

[ec], [e1], [e2], [e3], [e5]: Raw Inphases (I)

```
struct IAmp {2×nSats+1} {
! i2 amp[nSats];  // (I) amplitudes
  u1 cs;          // Checksum
};
```

These messages contain signal inphase (I) amplitudes for all the satellites in `SvsIdx`. The messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 amplitudes, respectively. The amplitudes are smoothed over the interval specified by the `/par/raw/iqsmi` parameter.

[qc], [q1], [q2], [q3], [q5]: Raw Quadratures (Q)

```
struct QAmp {2×nSats+1} {
! i2 amp[nSats];  // (Q) amplitudes
  u1 cs;          // Checksum
};
```

These messages contain signal quadrature (Q) amplitudes for all the satellites in `SvsIdx`. The messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 amplitudes, respectively. The amplitudes are smoothed over the interval specified by the `/par/raw/iqsmi` parameter.

[x0], [x1], [x2], [x3], [x5], [x4]: S4

```
struct S4 {nSats+1} {
  f4 s4[nSats]; // S4
  u1 cs;        // Checksum
};
```

These messages contain corresponding s4 value for all the satellites specified in `SvsIdx`. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C S4, respectively.

[y0], [y1], [y2], [y3], [y5], [y4]: SigmaPhi

```
struct SigmaPhi {nSats+1} {
  f4 sigmaPhi[nSats]; // SigmaPhi [radians]
  u1 cs;              // Checksum
};
```

These messages contain corresponding `sigmaPhi` value for all the satellites in `SvsIdx`. The messages contain CA/L1, P/L1, P/L2, CA/L2, L5, and L1C `sigmaPhi`, respectively.

[IQ] 1-millisecond I and Q Samples

```
struct IQSamples {var} {
  ESI esi;    // ESI
  u1  slotId; // SlotId
  i1  isQ;    // 1 if Q output is enabled, 0 otherwise
  IQ  iq[n];  // I or IQ samples; n is variable number.
  u1  cs;     // Checksum
};
```

These messages contain 1000Hz I and Q samples. The format of the message changes depending on `/par/raw/out/q parameter`.

`IQ` definition when `isQ` is `0` (`/par/raw/out/q` is `off`):

```
struct IQ {
  i2 i; // I sample
};
```

`IQ` definition when `isQ` is `1` (`/par/raw/out/q` is `on`):

```
struct IQ {
  i2 i; // I sample
  i2 q; // Q sample
};
```

[TC] CA/L1 Continuous Tracking Time

```
struct TrackingTimeCA {2×nSats+1} {
  u2 tt[nSats];  // tracking time [s]
  u1 cs;         // Checksum
};
```

This message contains time elapsed since the last loss-of-lock on the CA/L1 signal for every satellite in `SvsIdx`.

[TC] time is measured in seconds. Each satellite is allocated its own TC-time counter. Count-up begins with zero and stops when the counter reaches the maximum value the "u2" data type allows. Please note that the TC-time counters are not subject to rollovers.

Given a satellite, TC-time count starts as soon as the C/A signal is locked on. Should a loss of lock occur when tracking the C/A signal, the TC-time counter is reset to zero.

[SS] Satellite Navigation Status

```
struct NavStatus {nSats+2} {
  u1 ns[nSats];  // Navigation Status
  u1 solType;    // Solution type
  u1 cs;         // Checksum
};
```

This message contains navigation status for all the satellites in SvsIdx. In addition, this message indicates which receiver positioning mode the status belongs to. For detailed information on the navigation status, see "Satellite Navigation Status" on page 70.

[ID] Ionospheric Delays

```
struct IonoDelay {4×nSats+1} {
! f4 delay[nSats];  // Ionospheric delay [s]
  u1 cs;            // Checksum
};
```

This message contains estimated ionospheric delays as computed by using the L1 minus L2 frequency combination for all the satellites in SvsIdx.

[rr] Satellite Range Residuals

```
struct RangeResidual {4×nSats+1} {
! f4 res[nSats];  // Range residual [m]
  u1 cs;          // Checksum
};
```

[vr] Satellite Velocity Residuals

```
struct VelocityResidual {4×nSats+1} {
! f4 res[nSats];  // Radial velocity residual [m]
  u1 cs;          // Checksum
};
```

3.4.7 Almanacs and Ephemeris

[GA] GPS Almanac

```
struct GPSAlm {47} {
  u1 sv;       // SV PRN number within the range [1…37]
  i2 wna;      // Almanac reference week []
  i4 toa;      // Almanac reference time of week [s]
  u1 healthA;  // Health summary (from almanac), [bitfield]
               //   0…4 - code for health of SV signal components
               //   5…7 - navigation data health indicators
  u1 healthS;  // Satellite health (page 25 of subframe 5) []
  u1 config;   // Satellite configuration (page 25 of subframe 4)
               //   [bitfield]:
               //   0…2 - satellite configuration
               //     3 - anti-spoofing flag
               //   4…7 - reserved
  //======= Clock data =======
  f4 af1;      // Polynomial coefficient [s/s]
  f4 af0;      // Polynomial coefficient [s]
  //===== Ephemeris data =====
  //--- Keplerian orbital parameters ---
  f4 rootA;    // Square root of the semi-major axis [m^0.5]
```

```
  f4 ecc;      // Eccentricity []
  f4 m0;       // Mean Anomaly at reference time [semi-circles]
  f4 omega0;   // Longitude of ascending node of orbit plane
               //   at the start of week 'wna' [semi-circles]
  f4 argPer;   // Argument of perigee [semi-circles]
  //--- Corrections to orbital parameters ---
  f4 deli;     // Correction to inclination angle [semi-circles]
  f4 omegaDot; // Rate of right ascension [semi-circle/s]
  u1 cs;       // Checksum
};
```

[EA] Galileo Almanac

```
struct GALAlm {49} {
  // GPS-alike data
  GPSAlm gps; // Without 'cs' field, gps.sv within the range [1…37]
  // Galileo-Specific data
  i2 iod;     // Issue of almanac data []
  u1 cs;      // Checksum
};
```

[QA] QZSS Almanac

```
struct QZSSAlm {47} {
  // GPS-alike data
  GPSAlm gps; // 'gps.sv' within the range [193…199]
};
```

[CA] BeiDou Almanac

```
struct BeiDouAlm {47} {
  // GPS-alike data
  GPSAlm gps; // 'gps.sv' within the range [1…63]
};
```

[IA] IRNSS Almanac

```
struct IrnssAlm {47} {
  // GPS-alike data
  GPSAlm gps; // 'gps.sv' within the range [1…30]
};
```

[NA] GLONASS Almanac

```
struct GLOAlmanac {47 | 52} {
  u1 sv;       // Satellite orbit slot number within [1…32] []
  i1 frqNum;   // Satellite frequency channel number [-7…24] []
  i2 dna;      // Day number within 4-year period starting
               //   with the leap year []
  f4 tlam;     // Time of the first ascending node passage
               //   on day 'dna' [s]
  u1 flags;    // Satellite flags [bitfield]:
               //     0 - health: 1 - healthy SV, as specified
               //         by 'Cn', 0 - unhealthy
               //     1 - SVs type: 0 - GLONASS, 1 - GLONASS-M
               //   2…7 - reserved
  //======= Clock data =======
  f4 tauN;     // Coarse time correction to SV clock
               //   with respect to GLONASS system time [s]
  f8 tauSys;   // Correction to GLONASS system time with respect
               //   to UTC(SU) [s]
  //===== Ephemeris data =====
  f4 ecc;      // Eccentricity at reference time 'tlam' []
  f4 lambda;   // Longitude of ascending node
               //   at reference time 'tlam' [semi-circles]
  f4 argPer;   // Argument of perigee
               //   at reference time 'tlam' [semi-circles]
  f4 delT;     // Correction to mean Draconic period
               //   at reference time 'tlam' [s/period]
  f4 delTdt;   // Rate of change of Draconic period [s/period^2]
  f4 deli;     // Correction to inclination
               //   at reference time 'tlam'[semi-circles]
  u1 n4;       // Number of 4-year period []
  // --- Optional data block ---
  u1 reserved; // <reserved>
  f4 gammaN;   // Rate of coarse satellite clock correction to
               // GLONASS time scale [s/s]
  // --- End of optional data block ---
  u1 cs;       // Checksum
};
```

[WA] SBAS Almanac

```
struct SBASAlmanac {51} {
  u1 waasPrn;          // SBAS SV PRN number within [120…142]
  u1 gpsPrn;           // GPS SV PRN associated with SBAS SV
  u1 id;               // Data ID
  u1 healthS;          // Satellite health [bitfield]:
                       //     0 - 0-Ranging on, 1-off
                       //     1 - 0-Corrections on, 1-off
                       //     2 - 0-Broadcast Integrity on, 1-off
                       //     3 - reserved
                       //   4…7 - are set to zero
  u4 tod;              // Time of the day [s]
  f8 xg, yg, zg;       // ECEF coordinates [m]
  f4 vxg, vyg, vzg;    // ECEF velocity [m/s]
  u4 tow;              // time of GPS week almanac was received at
  u2 wn;               // GPS week this almanac was received at
  u1 cs;               // Checksum
};
```

[GE] GPS Ephemeris

```
struct GpsEphemeris {123 | 160 | 168} {
  GpsEphReqData req;  // Required data
  GpsEphOptData opt;  // Optional data. Present when length > 123.
  u1 cs;              // Checksum
};

struct GpsEphReqData {122} {
  u1 sv;      // SV PRN number within the range [1…37]
  u4 tow;     // Time of week [s]
  u1 flags;   // Flags (see GPS ICD for details)[bitfield]:
              //   0 - curve fit interval
              //   1 - data flag for L2 P-code
              // 2,3 - code on L2 channel
              //   4 - anti-spoof (A-S) flag (from HOW)
              //   5 - 'Alert' flag (from HOW)
               //   6 - ephemeris was retrieved from non-volatile memory
              //   7 - reserved
  //===== Clock data (Subframe 1) =====
  i2 iodc;       // Issue of data, clock []
  i4 toc;        // Clock data reference time [s]
  i1 ura;        // User range accuracy []
  u1 healthS;    // Satellite health []
  i2 wn;         // Week number []
  f4 tgd;        // Estimated group delay differential [s]
  f4 af2;        // Polynomial coefficient [s/(s^2)]
  f4 af1;        // Polynomial coefficient [s/s]
  f4 af0;        // Polynomial coefficient [s]
  //===== Ephemeris data (Subframes 2 and 3) =====
  i4 toe;        // Ephemeris reference time [s]
  i2 iode;       // Issue of data, ephemeris []
  //--- Keplerian orbital parameters ---
  f8 rootA;      // Square root of the semi-major axis [m^0.5]
  f8 ecc;        // Eccentricity []
  f8 m0;         // Mean Anomaly at reference time (wn,toe)
                 //   [semi-circles]
  f8 omega0;     // Longitude of ascending node of orbit plane at the
                 //   start of week 'wn' [semi-circles]
  f8 inc0;       // Inclination angle at reference time [semi-circles]
  f8 argPer;     // Argument of perigee [semi-circles]
  //--- Corrections to orbital parameters ---
  f4 deln;       // Mean motion difference from computed value
                 //   [semi-circle/s]
  f4 omegaDot;   // Rate of right ascension [semi-circle/s]
  f4 incDot;     // Rate of inclination angle [semi-circle/s]
  f4 crc;        // Amplitude of the cosine harmonic correction term
                 //   to the orbit radius [m]
  f4 crs;        // Amplitude of the sine harmonic correction term
                 //   to the orbit radius [m]
  f4 cuc;        // Amplitude of the cosine harmonic correction term
                 //   to the argument of latitude [rad]
  f4 cus;        // Amplitude of the sine harmonic correction term
                 //   to the argument of latitude [rad]
  f4 cic;        // Amplitude of the cosine harmonic correction term
                 //   to the angle of inclination [rad]
  f4 cis;        // Amplitude of the sine harmonic correction term
                 //   to the angle of inclination [rad]
};
```

```
struct GpsEphOptData {37 | 45} {
  u1 navType;   // Signal type [bitfield]
                //   #0: L1 NAV
                //   #1: L2C CNAV
                //   #2: L5 CNAV
                //   #3: L1C CNAV2 or NavIC L1
  i4 lTope;     // Time of prediction for ephemeris data
  i4 lTopc;     // Time of prediction for clock data
  f8 dADot;     // Change rate in semi-major axis
  f4 fDelnDot;  // Rate of mean motion
  i1 cURAoe;    // SV ephemeris URA index
  i1 cURAoc;    // SV clock URA index (RSF for NavIC)
  i1 cURAoc1;   // SV clock URA change index
  i1 cURAoc2;   // SV clock URA change rate index
  union {       // One of the variants below, depending on 'navType'
    GpsEphCnavIsc;  // when 'navType' bit #3 is 0
    GpsEphCnav2Isc; // when 'navType' bit #3 is 1
  } isc;
  f4 DAf0;      // correction to 'af0'. Exact term = af0 + DAF0
};

struct GpsEphCnavIsc {16} {
  f4 fIscL1CA; // Inter-signal correction between L1P(Y) and L1 C/A
  f4 fIscL2C;  // Inter-signal correction between L1P(Y) and L2C
  f4 fIscL5I5; // Inter-signal correction between L1P(Y) and L5I5
  f4 fIscL5Q5; // Inter-signal correction between L1P(Y) and L5Q5
};

struct GpsEphCnav2Isc {8} {
  f4 fIscL1CP; // Inter-signal correction between L1P(Y) and L1CP
  f4 fIscL1CD; // Inter-signal correction between L1P(Y) and L1CD
};
```

[EN] Galileo Ephemeris

```
struct GALEphemeris {149} {
  GpsEphReqData req; // GPS required data, 'req.sv' within the range [1…37]
  // --- Galileo-specific data block ---
  f4 bgdE1E5a; // broadcast group delay E1 - E5A [s]
  f4 bgdE1E5b; // broadcast group delay E1 - E5B [s]
  f4 ai0;      // Effective ionisation level 1-st order parameter []
  f4 ai1;      // Effective ionisation level 2-nd order parameter []
  f4 ai2;      // Effective ionisation level 3-rd order parameter []
  u1 sfi;      // Ionospheric disturbance flags [bitfield]
  u1 navType;  // Signal type [bitfield]:
               //   0 - Galileo E1B(INAV)
               //   1 - Galileo E5A(FNAV)
               //   2 - Galileo E5B(INAV)
               //   3 - GIOVE E1B (historical)
               //   4 - GIOVE E5A (historical)
               //   5 - <reserved>
               //   6 - Galileo E6
  f4 DAf0;     // correction to 'af0'. Exact term = af0 + DAF0
  // --- End of Galileo-specific data block ---
  u1 cs;       // Checksum
};
```

[QE] QZSS Ephemeris

```
struct QZSSEphemeris {123 | 160} {
  // GPS-alike data
  GpsEphemeris gps;  // Without 'cs', 'gps.sv' within the range[193…199]
  u1 cs;             // Checksum
};
```

[CN] BeiDou Ephemeris

```
struct BeiDouEphemerisS3 {134|152|160} {
  GpsEphReqData req; // GPS required data, 'req.sv' within the range [1…63]
  // --- BeiDou-specific data block ---
  f4 tgd2;      // tgd between B2A and B3 signals
                // Note that tgd in GpsEphReqData block contains
                // tgd between B1C and B3 signals
  u1 navType;   // Signal type[bitfield]
                //   0 - B1
                //   1 - B2
                //   2 - B3
                //   3 - B1C
                //   5 - B2A
                //   6 - B2B
  f4 DAf0;      // correction to 'af0'. Exact term = af0 + DAF0
  union {       // One of the variants below, depending on 'navType'
    EphBds2;    // when any of 'navType' bits 0, 1, or 2 is set
    EphBds3;    // when any of 'navType' bits 3, 5, or 6 is set
  } isc;
  // --- End of BeiDou-specific data block ---
  u1 cs;        // Checksum
};

struct EphBds2 {2} {
  u1 AODC;
  u1 AODE;
};

struct EphBds3 {20|28} {
  f8 ADot;      // derivative of A
  f4 DelNDot;   // derivative of DelN
  f4 isc_B1C;   // ISC of B1C signal
  f4 isc_B2A;   // ISC of B1A signal
  f4 tgd_B2Bdata;  // Optional, only if receiver supports B2B
  f4 tgd_B2Bpilot; // Optional, only if receiver supports B2B
};
```

[NE] GLONASS Ephemeris

```
struct GLOEphemeris {88 | 103} {
! u1 sv;       // Satellite orbit slot number [1…32] []
  i1 frqNum;   // Satellite frequency channel number [-7…24] []
! i2 dne;      // Day number within 4-year period []
  i4 tk;       // Frame start time within current day [s]
  i4 tb;       // Ephemeris reference time (for day 'dne') [s]
  u1 health;   // Satellite health [bitfield]
               //     0 - MSB taken from Bn word which indicates
               //         satellite health:
               //           1 - satellite is unhealthy
               //           0 - satellite is healthy
               //     1 - If set, this flag indicates that params
               //         'tau' and 'gamma' may be wrong
               //         (Note that receiver performs several
               //         'internal' data consistency checks allowing
               //         detection of problem broadcast parameters)
               //     2 - If set, this flag indicates that initial
               //         conditions 'r[3]' and 'v[3]' may be wrong
               //     3 - SV health (Cn word) status from almanac:
               //         0 - satellite is unhealthy
               //         1 - satellite is healthy
               //     4 - If set, this flag indicates that SV health
               //         status from almanac is available
               //   5…7 - reserved
  //===== Ephemeris data ======
  u1 age;      // Age of operational information (En) [days]
  u1 flags;    // Flags (for details, see GLONASS ICD) [bitfield]:
               //   0…1 - p1 word
               //     2 - p2 word
               //     3 - p3 word
               //   4…5 - 2 LSB taken from Bn word
               //     6 - ephemeris was retrieved from NV-memory
               //     7 - SV is GLONASS-M or newer
  f8 r[3];     // Satellite PE-90 coordinates [km]
  f4 v[3];     // Satellite PE-90 velocities [km/s]
  f4 w[3];     // Satellite PE-90 accelerations due to Luni-Solar
               //   gravitational perturbations [km/s^2]
//===== Clock data ======
  f8 tauSys;   // Time correction to GLONASS time scale (vs. UTC(SU))
               //   tauSys = TUTC(SU) - TGLN [s]
  f4 tau;      // Correction to satellite clock (vs. GLONASS time)
               //   tau = TGLN - TSV [s]
  f4 gamma;    // Rate of satellite clock offset [s/s]
//===== GLONASS-M data ======
  f4 fDelTauN; // Delta Tau N - delay between L1 and L2 [s]
  u1 nFt;       // Ft (User Range Accuracy), see GLONASS ICD for values
  u1 nN4;       // Number of 4-year cycle [1…31]
  u2 flags2;   // Flags (for details, see GLONASS ICD) [bitfield]:
               //     0 - GLONASS-M ln (third string)
               //   1…2 - GLONASS-M P
               //     3 - GLONASS-M P4
               //   4…5 - GLONASS-M M
               //     6 - GLONASS-M ln (fifth string)
               //  7…15 - reserved
```

```
  // --- Optional data block ---
  u1 navType; // Signal type [bitfield]
              //   0 - L1
              //   1 - L3
              //   2 - L2C
              //   3 - P1
              //   4 - P2
  f4 beta;    // Derivation of rate of satellite clock offset [s/s/s]
  f4 tauSysDot; // Derivation of 'tauSys' [s/s]
  u1 ec;        // Age of clock information (days)
  u1 ee;        // Age of ephemeris information (days)
  i1 fc;        // Clock accuracy index
  i1 fe;        // Ephemeris accuracy index
  u2 reserv;
  // --- End of optional data block ---
  u1 cs;       // Checksum
};
```

[WE] SBAS Ephemeris

```
struct SBASEhemeris {73} {
  u1 waasPrn;           // SBAS SV PRN number within [120…142]
  u1 gpsPrn;            // GPS SV PRN associated with SBAS SV
  u1 iod;               // Issue of data
  u1 acc;               // SBAS SV accuracy[1]
  u4 tod;               // Reference time (seconds of the day)[s]
  f8 xg, yg, zg;        // ECEF coordinates [m]
  f4 vxg, vyg, vzg;     // ECEF velocity [m/s]
  f4 vvxg, vvyg, vvzg;  // ECEF acceleration [m/s^2]
  f4 agf0;              // SBAS SV clock offset factor 'ao' [s]
  f4 agf1;              // SBAS SV clock offset factor 'a1' [s/s]
  u4 tow;               // Time of GPS week this ephemeris was
                        //   received at
  u2 wn;                // GPS week this ephemeris was received at
  u2 flags;             // Flags [bitfield]:
                        //   0…5 - reserved
                        //     6 - ephemeris was retrieved from NV-memory
                        //  7…15 - reserved
  u1 cs;                // Checksum
};
```

[IE] IRNSS Ephemeris

```
struct IrnssEphemeris {124 or 160} {
  GPSEphemeris gps; // GPS-alike data; without 'cs' field,
  union { // One of the variants below, depending on message length
    GpsEphOptData opt; // When message length is greater than 124
    u1 navType; // Signal type [bitfield], when message length is 124
                // #0: Navic L5
                // #1: Navic S
                // #2: Reserved
```

1. For details, see ICD-GPS-200C, Revision IRN-200C-004 April 12, 2000.

```
                // #3: Navic L1
  }
  u1 cs;      // Checksum
}
```

3.4.8 Raw Navigation Data

Most raw navigation data messages have the `errCorr` field, which meaning is as follows:

Values	Meaning
-128	Error(s) have been detected and left uncorrected due to user settings. See `/par/raw/data/mode` parameter
[-127…0]	Data check algorithm detected this number of errors, but error recovery is not available or had failed
0	No errors detected
[1…127]	This many errors have been detected and corrected

[gd] GPS Raw Navigation Data

```
struct GpsRawNavData {len×4+9} {
  u1 prn;        // Pseudo-Range Number (PRN)
  u4 time;       // Time of receiving of message [s]
  u1 type;       // Type of data:
                 //   0 - L1 NAV
                 //   1 - L2C CNAV
                 //   2 - L5 CNAV
                 //   3 - L1C CNAV2
  u1 len;        // Length of the navigation data block 'data'
  u4 data[len]; // Navigation data block
  i1 errCorr;    // Error corrections
  u1 cs; // Checksum
};
```

In the `data` field of the message, when `type` field is set to `L1`, every element contains 30 LSBits of navigation data; when `type` field is set to anything else, all 32 bits are used. The most significant bit corresponds to the first broadcast symbol.

For L1C, the `data` field of the message contains no sync bits, and has the following layout:

Table 3-10. L1C Raw Data Layout

#	Field Name	Field Length [bits]
1	toi	9
2	data2	600
3	data3	274

[qd] QZSS Raw Navigation Data

```
struct QzssRawNavData {len×4+9} {
  GpsRawNavData data;
};
```

[lD] GLONASS Raw Navigation Data

```
struct GloRawNavData {len×4+10} {
! u1 num;        // SV number
! i1 fcn;        // SV frequency code number
  u4 time;       // GLONASS time of receiving of message [s]
  u1 type;       // Type of data:
                 //   0 - L1
                 //   1 - unused (historical L3)
                 //   2 - L2C
                 //   3 - P1
                 //   4 - P2
  u1 len;        // Length of the navigation data block 'data'
  u4 data[len]; // Navigation data block
  i1 errCorr;    // Error corrections
  u1 cs;         // Checksum
};
```

In the `data` field of the message, when `type` field is set to `L1`, every element contains 25 LSBits of the string of GLONASS sub-frame; when `type` field is set to `L3`, all 32 bits are used. The most significant bit corresponds to the first broadcast symbol.

[ud] GLCDMA Raw Navigation Data

```
struct GLCDMARawNavData {len×4+9} {
  u1 prn;        // SV PRN number
  u4 time;       // GLONASS time of receiving of message [s]
  u1 type;       // Type of data:
                 //   0 - L1
                 //   1 - L2
                 //   2 - L3
  u1 len;        // Length of the navigation data block 'data'
  u4 data[len]; // Navigation data block
  i1 errCorr;    // Error corrections
  u1 cs;         // Checksum
};
```

[WD] SBAS Raw Navigation Data

```
struct SbasRawNavData {len+9} {
  u1 prn;        // SV PRN number within the range [120…147,183…202]
  u4 time;       // Time of receiving of message [s]
  u1 type;       // Type of data:
                 //   0 - L1 SBAS, L1S QZSS, or L1Sb QZSS
                 //   1 - L5 SBAS, or L5S QZSS
  u1 len;        // Length of the navigation data block 'data'
```

```
  u1 data[len]; // Navigation data block
  i1 errCorr;   // Error corrections
  u1 cs;        // Checksum
};
```

This message can carry raw navigation data from any SBAS-like signals, including SBAS L1 or L5, as well as QZSS L1S, L5S or L1Sb.

In the `data` field of the message the most significant bit of the first byte corresponds to the first broadcast 4-ms data symbol. The field contains data starting from SBAS preamble up to and including SBAS checksum.

Note: This message contains native PRN numbers for given system (e.g., PRN number of L5S of QZS-2 satellite is 196 and PRN number of L1S is 184).

[ED] Galileo Raw Navigation Data

```
struct GalRawNavData {len+9} {
  u1 prn;       // SV PRN number within the range [1…30]
  u4 time;      // Time of receiving of message [s]
  u1 type;      // Type of data:
                //   0 - Galileo E1B(INAV)
                //   1 - Galileo E5A(FNAV)
                //   2 - Galileo E5B(INAV)
                //   3…5 <reserved>
                //   6 - Galileo E6
  u1 len;       // Length of the navigation data block 'data'
  u1 data[len]; // Navigation data block
  i1 errCorr;   // Error corrections
  u1 cs; // Checksum
};
```

In the `data` field of the message the most significant bit of the first byte corresponds to the first broadcast symbol.

For INAV, the `data` field of the message contains no sync bits, and has the following layout:

Table 3-11. INAV Raw Data Layout

#	Field Name	Field Length [bits]
1	even/odd	1
2	page type	1
3	Data (1/2)	112
4	Tail	6
5	even/odd	1
6	page type	1
7	Data (2/2)	16
8	Field1	64

Table 3-11. INAV Raw Data Layout

#	Field Name	Field Length [bits]
9	CRC	24
10	Field2	8
11	Tail	6

For E1-B: Field1 = Reserved1 (40) + SAR (22) + Spare (2). Field2 = SSP

For E5b-I: Field1 = Reserved1. Field2 = Reserved2

For FNAV, the `data` field of the message contains no sync bits, and has the following layout:

Table 3-12. FNAV Raw Data Layout

#	Field Name	Field Length [bits]
1	page type	6
2	nav data	208
3	crc	24
4	tail	6

For Galileo E6, the `data` field of the message contains no sync bits.

[hd] Galileo HAS Message Type1 Data

```
struct HasType1Data {len+9} {
  u1 prn;        // Reserved; always 0
  u4 time;       // Time of decoding of message [s]
  u1 type;       // Type of Data:
                 //  0 - Galileo E6 Type 1
  u1 len;        // Length of the data block 'data'
  u1 data[len]; // Data
  i1 errCorr;    // Reserved; always 0
  u1 cs;         // Checksum
};
```

This message contains Galileo HAS Message Type1 after Reed-Solomon decoding of Galileo E6 raw data.

Note: Depending on particular receiver and firmware version, one might need to set `/par/pos/pp/type` parameter to `"has"` to make this message available.

Table 3-13. HAS Message Type1 Data Layout

#	Field Name	Field Length [bits]
1	MT1 Header	32
2	MT1 Body	variable

[cd] BeiDou Raw Navigation Data

```
struct CompRawNavData {len×4+9} {
  u1 prn;        // SV PRN number
  u4 time;       // BeiDou Time of receiving of message [s]
  u1 type;       // Type of data [bitfield]:
                 // bits 5…0: Signal type:
                 //   0 - B1
                 //   1 - B2
                 //   2 - B3
                 //   3 - B1C
                 //   4 - <reserved>
                 //   5 - B2a
                 //   6 - B2b
                 // bit 6: 1 - B2b data was decoded from B2bq component
                 // bit 7: 1 - D2 structure from GEO BeiDou satellite
  u1 len;        // Length of the navigation data block 'data'
  u4 data[len]; // Navigation data block
  i1 errCorr;    // Error corrections
  u1 cs; // Checksum
};
```

Example: `type` from B1 of BeiDou GEO (D2 data): 0x80 (D2 data) | 0x00 (B1) = 0x80
`type` from B2a: 0x05 (B2a) = 0x05
`type` from B2b pilot (B2bq): 0x40 (pilot) | 0x06 (B2b) = 0x46

For B1C, the `data` field of the message contains no sync bits, and has the following layout:

Table 3-14. B1C Raw Data Layout

#	Field Name	Field Length [bits]
1	data2	600
2	errCorr2[1]	8
3	data3	264
4	soh[2]	8

1. This field has the meaning of errCorr field, but for subframe 2, while errCorr field is for subframe 3
2. from the first subframe

For B2a, the data field of the message has the following layout:

Table 3-15. B2a Raw Data Layout

#	Field Name	Field Length [bits]
1	PRN	6
2	message type	6
3	SOW	18
4	data	234
5	CRC	24

For B2b, the data field of the message has the following layout:

Table 3-16. B2b Raw Data Layout

#	Field Name	Field Length [bits]
1	message type	6
2	SOW	20
3	data	436
4	CRC	24
5	Rev	6

[id] IRNSS Raw Navigation Data

```
struct IrnssRawNavData {len×4+9} {
  u1 prn;        // SV PRN number
  u4 time;       // IRNSS time of receiving of message [s]
  u1 type;       // Type of data:
                 //   0 - L5
                 //   1 - S
                 //   2 - reserved (former L1)
                 //   3 - L1
  u1 len;        // Length of navigation data block 'data'
  u4 data[len]; // Navigation data block
  i1 errCorr;    // Error corrections
  u1 cs; // Checksum
};
```

For `L1` signal, the layout of the `data` field is:

`data[0]` – subframe 1 (toi)

`data[1…19]` – subframe 2

`data[20…28]` – subframe 3

Note: NavIC L1 shares the same overall message structure as GPS/QZSS L1C, but uses 29 long-words (vs. 28 in [gd]/[qd]) in a different way (each aligned to a new longword), and SF1 being right aligned.

[xd] QZSS L6 Raw Navigation Data

```
struct QzssL6RawNavData {len+9} {
  u1 svid;       // SVID from ESI
  u4 time;       // QZSS time of receiving of message [s]
  u1 type;       // Type of data: // 0 - L6D;  1 - L6E
  u1 len;        // Length of navigation data block 'data'
  u1 data[len]; // Navigation data block
  i1 errCorr;    // Error corrections
  u1 cs;         // Checksum
};
```

[ad] Raw Navigation Data With Minimal Latency.

```
struct ADData {var} {
  u1 count;                 // Number of FastData records
  FastData fastData[count]; // FastData records
  u1 cs;                    // Checksum
};

struct FastData {var} {
  u1 sysSig        // (SSID[1] << 4) | SlotId[2]
  u1 sat;          // SVID[1]
! u1 dtime;        // Time offset of the last bit from the epoch time
                   // dtime = (epochTime - satelliteTime) mod 256
                   // dtime = 255, if satellite time is unknown
  u1 nBits;        // Number of raw data bits in the 'data' field
  u1 data[nBytes]; // Raw data array; nBytes = (nBits + 7) / 8
           // oldest bit is MSB (left), most recent - LSB (right)
           // In the last byte unused LSB bits are zero.
};
```

Description of SlotId -GREIS table 3-8; SSID, SVID - GREIS table 3-6

3.4.9 Spectrum Messages

In the spectrum messages, 'n' denotes the number of spectra. It depends on receiver type and is equal to the number of RF bands implemented in the receiver. The order of particular set of spectra always matches those of the following list:

```
GPS L1, GPS L2, GPS L5, GLONASS L1, GLONASS L2, Galileo E5B
```

For example, single-frequency GPS/GLONASS receiver will have 2 spectra, GPS L1, and GLONASS L1, in this particular order.

Refer to “Spectrum Parameters” on page 547 for more information

[sp] Spectrum

```
struct Spectrum {n×m×2+7} {
  i2 currFrq;         // Current frequency [Hz×10^4]
  i2 finalFrq;        // Frequency of the last message [Hz×10^4]
  u1 n;               // Number of spectra in this
  u1 m;               // Number of spectrum blocks in this message
  SpecData s[m];      // Spectrum data
  u1 cs;              // Checksum
};
```

1. See Table 3-6 on page 87
2. See Table 3-8 on page 88

```
struct SpecData {2×n} {
  i2 spec[n];  // Spectrum values for n spectra [dB×0.1]
};
```

[sP] Extended Spectrum

```
struct Spectrum {n×m×4+7} {
  i2 currFrq;          // Current frequency [Hz×10^4]
  i2 finalFrq;         // Frequency of the last message [Hz×10^4]
  u1 n;                // Number of spectra in this message
  u1 m;                // Number of spectrum blocks in this message
  ExtSpecData s[m];    // Extended spectrum data
  u1 cs;               // Checksum
};

struct ExtSpecData {4×n} {
  i2 spec[n];    // Spectrum values for n spectra [dB×0.1]
  u1 agcmin[n];  // Min AGC values []
  u1 agcmax[n];  // Max AGC values []
};
```

[Sp] Single Spectrum

This message is an improved format used for getting spectrum data in newer receivers instead of `[sp]` and `[sP]` messages.

Unlike `[sp]` and `[sP]`, every `[Sp]` message contains only one spectrum. First and last messages of entire spectrum contain additional `extData` field that allows to plot spectrum without knowledge of any spectrum parameters.

```
struct Spectrum {m×2+9+(sizeof(extData) for first and last nessage only)} {
  i4 pointNumber;      // Number of fists spectrum value in this message
  u1 nameId;           // Spectrum name ID (see /par/specrt/name parameter)
  u1 m;                // Type of spectrum data (currently always 1)
  u2 n;                // Number of spectrum data in this message
  SpecData s[n];       // Spectrum data
  extData ext;         // extended data (only first and last message)
  u1 cs;               // Checksum
};

struct SpecData {2×m} {
  i2 spec[m];          // Spectrum values [dB×0.1]
};

struct extData {33} {
  f4 freq0;   // Frequency of spectrum first point [Hz]
  f4 step;    // Spectrum frequency step [Hz]
  i4 points;  // Total number of spectrum points
  a1 name[8]; // Spectrum name. String with '\0' at the end
  i4 timeTag; // Spectrum timeTag - weekTime [ms]
  a1 logPoint[8]; // Spectrum log point
  a1 antenna; // Spectrum antenna
};
```

[ms] Modem Spectrum

```
struct MDM_Spectrum {9} {
  i4 frq; // Current frequency [Hz]
  i4 pwr; // Current signal (or noise) power [dBm]
  u1 cs;  // Checksum
};
```

3.4.10 Hardware Calibrator Messages

Messages described in this section contain measurements obtained by the hardware calibrator (refer to “Hardware Calibrator” on page 232).

[gC], [g1], [g2], [g3]: GLONASS Delays

```
struct SvDelays {9} {
  i1 fcn;    // GLONASS FCN[-7…7]
  f4 phase;  // Phase delay [cycles]
  f4 range;  // Range delay [s]
};

struct GloDelays {9×nFcn+1} {
  SvDelay del[nFcn]; // Delays
  u1 cs;             // Checksum
};
```

The [gC], [g1], [g2], and [g3] messages contain CA/L1, P/L1, P/L2, and CA/L2 phase and code delays for GLONASS FCNs.

[gR]: Code Delays of Receiver RF Bands

```
struct BandDelay {6} {
  i1 band;    // Band, see table below
  i1 code_frq;// 0 - 1.023 MHz (C/A-like), 1 - 10.23 MHz (P-like)
  f4 delay;   // Code delay [s]
};

struct CalBandsDelay {6×n+1} {
  BandDelay d[n]; // 'n' may vary depending on receiver model
                  // and calibrator data readiness
  u1 cs;          // Checksum
};
```

band	name	frequency [Hz]
0	gps1	1575420000
1	gps2	1227600000

band	name	frequency [Hz]
2	gps5	1176450000
3	glo1	1602000000
4	glo2	1246000000
5	glo3	1202025000
6	gal5b	1207140000
7	gal5	1191795000
8	bei1	1561098000
9	gal6	1278750000
10	bei3	1268520000
11	bei12	1589742000
12	ind1	2492028000

3.4.11 Device Data Messages

[dv] Device Data

```
Dv {var} {
  u1 type;    // Device type
  DvDev data; // Device data (type-specific)
  u1  cs;     // Checksum
};
```

Provides data obtained from a device, in device-specific format. The following device types, along with the name of the message to be requested, are currently defined:

0	reserved
1	magnetometer (dv_mag)
2	IMU accelerometer (dv_accl)
3	IMU gyroscope (dv_gyro)

Example: enable data output from magnetometer to the current port:

⇒ `em,/cur/term,/msg/jps/dv_mag`

[dv](dv_mag) Magnetometer Raw Measurements

```
DvMag {18} {
  u1 type;      // Set to 1 to indicate Magnetometer
  DvDev {
    u4 time;    // Measurements time. Tr modulo 1 day [ms]
    i4 x;       // X-axis magnitude
    i4 y;       // Y-axis magnitude
    i4 z;       // Z-axis magnitude
  }
  u1 cs;        // Checksum
};
```

XYZ axes correspond to ENU receiver body frame.

[dv](dv_accl) IMU Accelerometer Raw Measurements

```
DvMag {18} {
  u1 type;      // Set to 2 to indicate IMU Accelerometer
  DvDev {
    u4 time;    // Measurements time. Tr modulo 1 day [ms]
    f4 x;       // X-axis accelerometer [m/sec^2]
    f4 y;       // Y-axis accelerometer [m/sec^2]
    f4 z;       // Z-axis accelerometer [m/sec^2]
  }
  u1 cs;        // Checksum
};
```

XYZ axes correspond to ENU receiver body frame.

[dv](dv_gyro) IMU Gyroscope Raw Measurements

```
DvMag {18} {
  u1 type;      // Set to 3 to indicate IMU Gyroscope
  DvDev {
    u4 time;    // Measurements time. Tr modulo 1 day [ms]
    f4 x;       // X-axis gyroscope [deg/sec]
    f4 y;       // Y-axis gyroscope [deg/sec]
    f4 z;       // Z-axis gyroscope [deg/sec]
  }
  u1 cs;        // Checksum
};
```

XYZ axes correspond to ENU receiver body frame.

3.4.12 ADU Messages

[MR] Rotation Matrix

```
struct RotationMatrix {37} {
  u4 time;      // receiver time [ms]
  f4 q00, q01, q02, q12;  // components of the rotation matrix Q []
  f4 rms[3];    // estimated accuracy for three baseline vectors [m]
```

```
  u1 solType[3];  // solution type[1] for three baseline vectors
  u1 flag;      // 0 - components of matrix Q are invalid, 1 - valid
  u1 cs;        // Checksum
};
```

[mr] Rotation Matrix and Vectors

```
struct RotationMatrixAndVectors {73} {
  u4 time;    // receiver time [ms]
  f4 q00, q01, q02, q12;  // components of the rotation matrix Q []
  f4 rms[3];  // estimated accuracy for three baseline vectors [m]
  u1 solType[3];  // solution type[1] for three baseline vectors
  u1 flag;    // 0 - components of matrix Q are invalid, 1 - valid
  f4 bl0[3];  // baseline vector M-S0 in the current epoch [m]
  f4 bl1[3];  // baseline vector M-S1 in the current epoch [m]
  f4 bl2[3];  // baseline vector M-S2 in the current epoch [m]
  u1 cs;      // Checksum
};
```

[AR] Rotation Angles

```
struct RotationAngles {33 or 35} {
  u4 time;         // Receiver time [ms]
  f4 p,r,h;        // Pitch ,roll , heading angles [deg]
  f4 sp,sr,sh;     // Pitch, roll, heading angles RMS [deg]
  u1 solType[3];   // Solution type for 3 base lines
  u1 flags;        // flags [bitfield]:
                   // #0: data validity
                   //     0 - no data available ('insErrors' is still valid)
                   //     1 - data are valid
                   // #1: solution source
                   //     0 - GNSS solution
                   //     1 - INS solution
                   // #2…#7: reserved
  // The 'insErrors' field only exists for "INS solution"
  u2 insErrors;    // INS-specific error flags [bitfield]
  u1 cs;           // Checksum
};
```

The `insError` field is logical OR of the following bits that help diagnosing possible problems:

Value	Description
0x01	No data from IMU
0x02	No GNSS measurements: tracking issue
0x04	No GNSS position: no RTK fix
0x08	Failure to compute GNSS velocity

1. See Table 3-3, “Solution Types,” on page 70

Value	Description
0x10	Not enough motion or no baseline fix for proper initialization
0x20	Solution not yet converged
0x40	Bad gyroscope measurements
0x80	Bad accelerometer measurements
0x100	No baseline fix

[AV] Angular Velocities

```
struct AngularVelocity {22 or 24} {
  u4 time;    // receiver time [ms]
  f4 x;       // X component of angular velocity [rad/s]
  f4 y;       // Y component of angular velocity [rad/s]
  f4 z;       // Z component of angular velocity [rad/s]
  f4 rms;     // Angular velocity RMS [rad/s]
  u1 flags;   // flags [bitfield]:
              // #0: data validity
              //     0 - no data available ('insErrors' is still valid)
              //     1 - data are valid
              // #1: solution source
              //     0 - GNSS solution
              //     1 - INS solution
              // #2…#7: reserved
  // The 'insErrors' field only exists for "INS solution"
  u2 insErrors; // INS-specific error flags [bitfield]
  u1 cs;        // Checksum
};
```

This message contains angular velocities in WGS-84.

3.4.13 Tilt-compensated Solution Messages

Refer to “Tilt-Compensated Position Parameters” on page 556 for description of the tilt-compensation feature.

[pg] Pole Tip Geodetic Position

```
struct Pg {46} {
! f8 lat;     // Pole tip latitude [rad]
! f8 lon;     // Pole tip longitude [rad]
! f8 alt;     // Ellipsoidal height [m]
! f4 pSigma;  // 3D RMS of pole tip position [m]
  u1 solType; // Solution type
  u4 solTime; // Solution time. Tr modulo 1 day [ms]
  u2 error;   // Error flags.
! f4 theta;   // Pole tilt angle from local vertical Up [deg]
```

```
  ! f4 phi;      // Pole tilt azimuthal angle relative to East direction [deg]
    u2 st;       // Time span of receiver being stationary (0 if moving) [ms]
    u1 cs;       // Checksum
  };
```

To convert `phi` angle to navigation azimuth (taken from North to East, in the range `[0…360]`), one may use this code in C:

```
azimuth = ((phi < 90) ? 90 : 450) - phi;
```

The `error` field is logical OR of the following bits that help diagnosing possible problems:

Value	Description
0x01	No data from IMU
0x02	No GNSS measurements: tracking issue
0x04	No GNSS position: no RTK fix
0x08	Failure to compute GNSS velocity
0x10	Not enough motion or no baseline fix for proper initialization
0x20	Solution not yet converged
0x40	Bad gyroscope measurements
0x80	Bad accelerometer measurements
0x100	No baseline fix

3.4.14 Event Marker and PPS Messages

The event marker and PPS have their own reference time settings governed by corresponding parameters[1]. As a consequence, some of the event marker and PPS messages described below utilize the “time scale” field of the following format:

Table 3-17. Event Marker and PPS time scale

Value	Description
0	GPS system time
1	UTC(USNO). Universal Coordinated Time supported by the U.S. Naval Observatory
2	GLONASS system time

1. `/par/dev/event/[a|b]/time`, and `/par/dev/pps/[a|b]/time`

Table 3-17. Event Marker and PPS time scale

Value	Description
3	UTC(SU). Universal Coordinated Time supported by the State Time and Frequency Service, Russia
4	SBAS system time
5	UTC SBAS
6	Galileo system time
7	UTC Galileo
8	BeiDou system time
9	UTC BeiDou
10	QZSS system time
11	UTC QZSS
12	IRNSS system time
13	UTC IRNSS
14…255	Reserved

[XA], [XB] External Event

```
struct ExtEvent {10} {
  i4 ms;          // ms part of event time tag
  i4 ns;          // ns part of event time tag
  u1 timeScale;   // time scale
  u1 cs;          // Checksum
};
```

The event time tag is the time in corresponding time scale modulo one day.

To make your receiver generate these messages, you additionally need to turn on external event processing on corresponding external event input (using `/par/dev/event/[a|b]/in` parameters).

[ZA], [ZB] PPS Offset

```
struct PPSOffset {5} {
  f4 offs;  // PPS offset in nanoseconds
  u1 cs;    // Checksum
};
```

Due to a hardware limitation, PPS signals are discrete with resolution that depends on particular receiver model. JAVAD GNSS receiver allows you to compensate for this discreteness error by means of utilizing this message. It contains the offset between the scheduled PPS time and the actual pulse edge's arrival time. When the pulse edge is earlier than the scheduled time, the offset is positive. When the pulse edge is delayed relative to the scheduled time, the offset is negative.

[YA], [YB] Time Offset at PPS Generation Time

```
struct RcvTimeOffsAtPPS {10} {
  f8 offs;       // [Tpps-Tr] offset [s]
  u1 timeScale;  // time scale
  u1 cs;         // Checksum
};
```

This message contains PPS reference time to receiver time offset at the moment of PPS generation.

PPS is usually output before solution for given epoch is ready, and therefore clock offset is extrapolated from those computed at previous epoch. It's this clock offset that is output in this message and that is why it could be slightly different than those output in messages such as [TO].

3.4.15 Heading and Pitch Messages

[ha] Heading and Pitch

```
struct HeadAndPitch {10} {
  f4 heading;  // Heading of the baseline between the base and the
               // rover receiver [degrees]
  f4 pitch;    // Pitch of the baseline between the base and the
               // rover receiver [degrees]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

This message contains heading and pitch calculated by the RTK engine.

[RO] Lever Arm Cartesian Position

This message contains the position of the master antenna corrected by the rotated lever arm vector. It has exactly the same format as the [PO] message described on page 80.

[RG] Lever Arm Geodetic Position

This message contains the position of the master antenna corrected by the rotated lever arm vector. It has exactly the same format as the [PG] message described on page 81.

3.4.16 Interactive Messages

Commands sent to the receiver may generate reply messages from the receiver. These human-readable text messages are output immediately as a response to corresponding commands. Interactive applications are the target for this class of messages.

[RE] Reply

```
struct RE {var} {
  a1 reply[];  // Reply
};
```

The contents of a reply message depends on what particular command has invoked this reply message (see Chapter 2 for more information about GREIS receiver commands and possible replies).

[ER] Error

```
struct ER {var} {
  a1 error[];  // Error description
};
```

If receiver gets a command that, for some reason, can't be executed, or produce an error during execution, then an error message is generated. The contents of the error message specifies what is wrong with the issued command.

3.4.17 Miscellaneous Messages

[IO] GPS Ionospheric Parameters

```
struct IonoParams {39} {
  u4 tot;     // Time of week [s]
  u2 wn;      // Week number (taken from the first subframe)
  // The coefficients of a cubic equation representing
  // the amplitude of the vertical delay
  f4 alpha0;  // [s]
  f4 alpha1;  // [s/semicircles]
  f4 alpha2;  // [s/semicircles^2]
  f4 alpha3;  // [s/semicircles^3]
  // The coefficients of a cubic equation representing
  // the period of the model
  f4 beta0;   // [s]
  f4 beta1;   // [s/semicircles]
  f4 beta2;   // [s/semicircles^2]
  f4 beta3;   // [s/semicircles^3]
  u1 cs;      // Checksum
};
```

This message contains ionospheric correction parameters from GPS subframe 4, page 18. These parameters relate to an ionospheric model mainly used by single frequency GPS receivers.

For more information about this ionosphere model, please see ICD-GPS-200C, Revision IRN-200C-004 April 12, 2000.

[QI] QZSS Ionospheric Parameters

```
struct QzssIonoParams {39} {
  IonoParams par;
};
```

This message contains ionospheric correction parameters from QZSS sub-frame 4, page 18. These parameters belong to the ionospheric model generally being used by single frequency GPS/QZSS receivers and are optimized for Japan area.

[CI] BeiDou Ionospheric Parameters

```
struct BeiDouIonoParams {39} {
  IonoParams par;
};
```

This message contains ionospheric correction parameters from BeiDou sub-frame 1.

These parameters relate to an ionospheric model mainly used by single frequency BeiDou receivers.

[II] IRNSS Ionospheric Parameters

```
struct IrnssIonoParams {39} {
  IonoParams par;
};
```

This message contains ionospheric correction parameters from IRNSS data. These parameters relate to an ionospheric model mainly used by single frequency IRNSS receivers.

[sj] Spoofing/Jamming Information

```
struct SpoofingJamming {var} {
  u1 ssid;           // SSID from ESI
  u1 slotId;         // SlotId
  u2 num;            // Number of satellites in this record
  u2 numSpoofed;     // Number of spoofed (with 2 correlation peaks)
                     // satellites
  u2 meanNoise;      // Noise mean value (percents)
  u2 devNoise;       // Noise deviation value (percents)
  SpoofData sj[num]; // Spoofing information for each satellite
  u1 cs;             // checksum
}

struct SpoofData {10} {
  u2 svid;   // SVID from ESI, but 2 bytes long
  u2 peak1;  // Main peak value [percents]
  u2 peak2;  // Second peak value [percents]
  i4 dRange; // Delta range between peaks [meters]
}
```

[==](EV) Event

```
struct Event {var} {
  u4 time;    // Receiver time of event occurrence modulo day, [ms]
  u1 type;    // Event type (see below), []
  u1 data[];  // Event contents
  u1 cs;      // Checksum
};
```

This message is generated (if enabled) every time some event occurs in the receiver. Currently the following event types are defined:

- `0` – free-form event. Is generated by the "event" command (see "event" on page 50).
- `1` – firmware warning. The "data" field describes the warning in human-readable form.

[LT] Message Output Latency

```
struct Latency {2} {
  u1 lt;  // output latency [ms]
  u1 cs;  // Checksum
};
```

This message contains the difference between the actual output time of the first of the messages sent to the output stream at the given epoch, and this epoch's time-tag. Note that latency for an output stream may depend on the amount of messages requested to a different stream. For example, the more messages are output to port A, the bigger the latency of port B; this is because the receiver begins generation of messages for port B only after it has finished generating messages for port A.

[>>] Wrapper

```
struct Wrapper {var} {
  u1 id;          // Source identifier
  u1 data[size];  // Data from the source
  a1 cs[2];       // Checksum formatted as hexadecimal
};
```

This message is intended to wrap up arbitrary data. The size of the wrapped data (in bytes) is equal to the message length from the header less 3 (size=L-3).

This message is used for two different purposes:

1. To wrap data from an input stream that has been set to the "wrapped echo mode" (see `/par/[port]/ewrap` and `/par/[port]/echo` parameters). In this case it

is generated whenever some data come to the stream. The "id" field then contains input stream identifier:

id	Source Stream
'a'…'d'	serial ports A…D, /dev/ser/a…/dev/ser/d
'A'…'E'	TCP ports A…E, /dev/tcp/a…/dev/tcp/d
'U'	USB port A, /dev/usb/a
'H'	Bluetooth port A, /dev/blt/a
'N'	CAN port A, /dev/can/a
'P'	TCP client port, /dev/tcpcl/a
'V'	UDP port

2. To wrap arbitrary message(s) during periodic messages output, as specified by the em command for corresponding messages. In this case the id field is set to be numerically equal to (-1 - count), where count is the field from the message scheduling parameters. See "Periodic Output" on page 22 and "em & out" on page 37 for details.

Note: This message is not subject to enabling/disabling using the em and dm commands. It is generated and output using its own rules.

[PM] Parameters

```
struct Params {var} {
  a1 params[];          // Parameters description
  a1 delim[2] = ",@";   // Checksum delimiter
  a1 cs[2];             // Checksum formatted as hexadecimal
};
```

This message contains information on (most of) receiver parameters. When enabled, it will also be output every time one of the receiver parameters is changed.

Due to large number of parameters only part of a whole receiver parameter tree is output at every epoch, and multiple [PM] messages are typically output per epoch. In addition, a few starting [PM] messages containing values for specific parameters are output at the first receiver epoch after enabling the message.

The starting messages and messages generated at the time of updating of receiver parameters have the following format:

NAME=VALUE

where NAME denotes the parameter name, and VALUE denotes the parameter value.

The other messages have a slightly different format, specifically:

{[ITEM[,ITEM…]]}

where ITEM denotes either the value of a parameter, or a comma-separated list of ITEMs surrounded by braces.

[LH] Logging History

```
struct LoggingHistory {var} {
  u1 svsCount;         // Number of SVs
  u1 targetStream;     // Stream ID
  u2 issue;            // Issue of the history
  u2 bitsCount;        // Number of bits
  u4 lastBitTime;      // Time since the last history shift [ms]
  u1 uids[svsCount];   // SVs UIDs
  u1 pad[padCount];    // Padding
  u4 hist[elemsCount][svsCount];  // History bits
};
```

This message contains history of logging of satellites data into particular stream. For a description of how logging history works and parameters governing logging history, see “Logging History” on page 380.

Fields description:

`svsCount` – Number of SVs in this history.

`targetStream` – The stream ID the history is gathered for (see description of the [>>] message for details).

`issue` – The issue of the history. It is incremented every time the history is changed.

`bitsCount` – Number of bits in this history. This history contains this number of bits for every SV specified in the `uids` field.

`lastBitTime` – Time in milliseconds since the last history shift.

`uids[svsCount]` – Array of SVs UIDs

`pad[padCount]` – Padding (with zeroes) to align the next field on 4 bytes boundary,

$$\text{fillCount} = (4 - (\text{svsCount} \% 4)) \% 4$$

`hist[elemsCount][svsCount]` – History bits. For every SV, the bits are packed into array of `u4` values. Most significant bit of the first element of the array represents most recent bit of the history. Least significant bit of the last element of the array represents the oldest bit of the history when there are enough history bits to fill the last `u4` element. The number of `u4` elements in the array is just enough to hold `bitsCount` bits:

$$\text{elemsCount} = (\text{bitsCount} + 31) / 32$$

Exactly `svsCount` bit arrays are put into the message in the order specified by the `uids` field.

[AI] Antenna Information

```
struct AntInfo {var} {
  u1   apcNum;      // APC length [0|3]. 0 if APC is not set
  f8   apc[apcNum]; // APC(GPS L1) WGS84(x,y,z) [meters]
  u1   arpNum;      // ARP length [0|3]. 0 if ARP is not set
  f8   arp[arpNum]; // ARP WGS84(x,y,z) [meters]
  u1   aidLen;      // Antenna ID Length
  a1   aid[aidLen]; // Antenna ID
  u1   cs;          // Checksum
};
```

[BI] Base Station Information

```
struct BaseInfo {28} {
  f8 x, y, z;  // ECEF coordinates [m]
  u2 id;       // Reference station ID
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

[SE] Security

```
struct Security {6} {
  u1 data[5];  // Opaque data
  u1 cs;       // Checksum
};
```

This message is for JAVAD GNSS internal use.

[SM] Security for [rM]

```
struct Security {8} {
  u1 data[6];  // Opaque data
  u2 crc16;    // 16-bit CRC
};
```

This message is for JAVAD GNSS internal use.

[TT] CA/L1 Overall Continuous Tracking Time

```
struct TrackingTime {5} {
  u4 tt;  // tracking time [s]
  u1 cs;  // Checksum
};
```

This message contains time elapsed since the last loss-of-lock of all CA/L1 signals. Time count starts as soon as the first CA/L1 signal is locked on. Should a loss of lock of the last CA/L1 signal occur, the time counter is reset to zero.

[OO] Oscillator Offset

```
struct RcvOscOffs {5} {
  f4 val;  // Oscillator offset [s/s]
  u1 cs;   // Checksum
};
```

The offset is the difference between oscillator frequency when oscillator control voltage has nominal value, and the nominal frequency of the oscillator, normalized to the nominal frequency of the oscillator. $(F_{Un} - F_n)/F_n$.

Note: The contents of this message are not suitable as a parameter for calculations based on receiver measurements. Use [DO] message instead.

[||](EE) Epoch End

```
struct EpochEnd {1} {
  u1 cs;   // Checksum
};
```

This message carries no information. It is intended as an "end of epoch" marker useful for real-time applications. This message is not recommended. We suggest to use the [::](ET) message instead, see "[::](ET) Epoch Time" on page 75.

3.4.18 Text Messages

GREIS Format for Text Messages

All the text messages have the following format:

```
struct Text {var} {
  a1 text[];
};
```

General format of the "text" field of GREIS predefined text messages as well as format notation used to specify particular text messages is described in this section.

The format of the "text" field is as follows:

```
,TITLE[,ITEM[,ITEM,…]],@CS
```

where square brackets designate optional parts,

`TITLE` – the title of particular message;

`ITEM` – either a field of particular type described below, or an item list surrounded by braces:

```
{[ITEM[,ITEM,…]]}
```

`CS` – checksum formatted as two hexadecimal uppercase digits.

A GREIS text message's format specifies its structure, field types, and the number of significant digits for each field.

Field format notation always starts with symbol "%" (hex 25).

The following data type characters (aka "data type specifiers") are used to distinguish between different data types:

- `D` – decimal integer;
- `X` – hexadecimal integer;
- `C` – character;
- `S` – string type (note that strings may have arbitrary lengths);
- `F` – floating point;
- `E` – exponential format for floating point.

Given a numeric field, digits preceding the data type specifier designate the number of digits in the format of this field[1]. There may be two, one or no digits specified before the data type specifier. For floating point fields, the first digit defines the length of the integer part of the field representation ("integer part descriptor") whereas the second digit defines the length of the fractional part ("fractional part descriptor"). If there are two digits used in the field format notation, these are separated by a dot "." (hex 2E). A field's fractional part descriptor can be variable, i.e., its length is programmable with an appropriate receiver command. In this case the fractional part descriptor shows the parameter range, which is put inside square brackets (e.g., %0.[1-4]F). Integer part descriptor may be omitted, which means that the integer part of the output value is allowed to be as long as necessary. Note that the delimiting dot before the corresponding fractional part descriptor will still exist.

If an integer field format has no integer part descriptor before its data type specifier (either "X" or "D"), the receiver will output all the significant digits. Leading zeroes will be added if the actual length of the integer part is shorter than specified by the descriptor.

If a format notation includes the plus symbol "+", receiver will output signed values (as usual, characters "+" (hex 2B) and "-"(hex 2D) are used to denote positive and negative values, respectively).

If a field format is surrounded by round brackets, it means that this format applies to a batch of "homogeneous" fields (note that the number of fields in such a batch may vary).

A back slash followed by two hexadecimal digits designates that the text character with corresponding ASCII code will be put into the message in this specific place. In addition, the reader will notice that various non-reserved symbols (lower and upper case

1. This applies to integer, float and double fields only.

English letters, arithmetic operation signs, braces, etc.,) are used together with the above described field formats. Here are some examples illustrating the format notation as defined above:

Format	Output data representation
%5.2F	00027.89
%5.2F × %+4.2F	67283.67 × +5678.22
Value: %+5.2F	Value: +00000.35
%+2.0F	+07
%0.4F	%0.4F 0.1234
%+7.5F	-0000039.67432
5-2=%D	5-2=3
%D	2467
%+3D	-052
%+3D	+964
%2X	0A
{%2D,%+.4F}	{04,-45.8027}
%C%C%C	XYZ
%S	This is a string
\4A	J
\50	P
(%2.1F)	45.8,98.0,04.7,88.3
%.3E	1.234E-04
%.3E	-5.789E+02

[DL] Data Link Status

This message displays the status of the data links associated with the corresponding serial ports/modem.

#	Format	Description
1	DLINK	Message title
2	%1D	Total number of data links (0…5). The rest of the message is available only if this number is non-zero. Otherwise the total number of data links value is immediately followed by the checksum.

#	Format	Description
3	({%C,%C,%S,%3D,%4D,%4D,%.2F[,%.2F][,%.2F] })	Group of fields associated with the given data link (note that the total number of such groups is determined by the previous field). These fields are - Data link identifier (refer to Table 3-18 below). - Decoder identifier (“R” – RTCM decoder, “T” – RTCM 3.0 decoder, “C” – CMR decoder, “J” – JPS decoder, “W” – SISNeT decoder). - Reference station identifier. - Time [in seconds] elapsed since receiving last message (maximum value = 999). Estimated with an accuracy of ±1 second. - Number of received messages (between 0001 and 9999). If no message has been received, this data field contains zero. - Number of corrupt messages (between 0001 and 9999). If no corrupt messages have been detected, this data field is set to zero. - Data link quality in percent (0-100); - Data link latency in seconds(0-3600), empty if unavailable - Output period of corrections being received from reference station, in seconds (0-3600), empty if unavailable
4	@%2X	Checksum

Table 3-18. Data Link Identifiers

Id	Corresponding Stream
‘A’…’D’	serial ports A…D, `/dev/ser/a…/dev/ser/d`
‘E’…’I’	TCP ports A…E, `/dev/tcp/a…/dev/tcp/d`
‘P’,‘Q’	TCP client ports, `/dev/tcpcl/a,/dev/tcpcl/b`
‘U’	USB port A, `/dev/usb/a`
‘L’,‘K’	Bluetooth ports, `/dev/blt/a,/dev/blt/b`
‘g’	CAN port A, `/dev/can/a`

[GS] GPS SVs Status

This message describes the status of GPS satellites.

#	Format	Description
1	GPSVST	Message title
2	%2D	Total number of GPS SVs being track

#	Format	Description
3	({%2D,%2D,%3D, {%2D[,%2D…]},%2D})	This 5-field section comprises: 1. GPS PRN number 2. Elevation in degrees 3. Azimuth in degrees, 4. Signal-to-noise ratios in [dB×Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. C/A 2. P1 3. P2 4. L2C(L+M) 5. L5(I+Q) 6. L1C(I+Q) 5. Satellite navigation status (see “Satellite Navigation Status” on page 70) The total number of such 5-field sections will match the number of SVs being track.
4	@%2X	Checksum

[ES] Galileo SVs Status

This message describes the status of Galileo satellites.

#	Format	Description
1	ESSVST	Message title
2	%2D	Total number of Galileo SVs being track
3	({%2D,%2D,%3D, {%2D[,%2D…]},%2D})	This 5-field section comprises: 1. Galileo PRN number 2. Elevation in degrees 3. Azimuth in degrees, 4. Signal-to-noise ratios in [dB×Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. E1(B+C) 2. E5 altboc 3. E5B(I+Q) 4. E6(B+C) 5. E5A(I+Q) 5. Satellite navigation status (see “Satellite Navigation Status” on page 70) The total number of such 5-field sections will match the number of SVs being track.
4	@%2X	Checksum

[WS] SBAS SVs Status

This message describes the status of SBAS satellites.

#	Format	Description
1	WSSVST	Message title
2	%2D	Total number of SBAS SVs being track
3	({%2D,%2D,%3D, {%2D[,%2D…]},%2D})	This 5-field section comprises: 1. SBAS or L1S(b) PRN number 2. Elevation in degrees 3. Azimuth in degrees, 4. Signal-to-noise ratios in [dB×Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. L1 2. Reserved (always empty) 3. Reserved (always empty) 4. Reserved (always empty) 5. L5(I+Q) 5. Satellite navigation status (see “Satellite Navigation Status” on page 70) The total number of such 5-field sections will match the number of SVs being track.
4	@%2X	Checksum

[NS] GLONASS SVs Status

This message describes the status of GLONASS satellites.

#	Format	Description
1	GLSVST	Message title
2	%2D	Total number of GLONASS SVs being track

#	Format	Description
3	({%2D,%2D,%2D,%3D,{%2D[,%2D…]},%2D})	This 6-element list comprises: 1. GLONASS SV Orbit Slot Number[1] 2. GLONASS SV Frequency Channel Number 3. Elevation in degrees 4. Azimuth in degrees 5. Signal-to-noise ratios in [dB×Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. CA/L1 2. P1 3. P2 4. CA/L2 5. L3(I+Q) 6. Satellite navigation status (see “Satellite Navigation Status” on page 70) The total number of such 6-element lists will match the number of SVs being track.
4	@%2X	Checksum

1.If orbit slot number is reported as zero, the slot number hasn't yet been determined.

[US] GLCDMA SVs Status

This message describes the status of GLONASS satellites.

#	Format	Description
1	USSVST	Message title
2	%2D	Total number of GLCDMA SVs being track
3	({%2D,%2D,%3D,{%2D[,%2D…]},%2D})	This 5-element list comprises: 1. GLCDMA PRN number 2. Elevation in degrees 3. Azimuth in degrees 4. Signal-to-noise ratios in [dB×Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. L1 2. L2 3. L3 5. Satellite navigation status (see “Satellite Navigation Status” on page 70) The total number of such 6-element lists will match the number of SVs being track.
4	@%2X	Checksum

[QS] QZSS SVs Status

This message describes the status of QZSS satellites.

#	Format	Description
1	QSSVST	Message title
2	%2D	Total number of GPS SVs being track
3	({%2D,%2D,%3D, {%2D[,%2D…]},%2D})	This 5-field section comprises: 1. QZSS PRN number [193…199] 2. Elevation in degrees 3. Azimuth in degrees, 4. Signal-to-noise ratios in [dB×Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. C/A 2. L1S 3. L6 4. L2C(L+M) 5. L5(I+Q) 6. L1C(I+Q) 5. Satellite navigation status (see “Satellite Navigation Status” on page 70) The total number of such 6-element lists will match the number of SVs being track.
4	@%2X	Checksum

[CS] BeiDou SVs Status

This message describes the status of BeiDou satellites.

#	Format	Description
1	CSSVST	Message title
2	%2D	Total number of BeiDou SVs being track

#	Format	Description
3	({%2D,%2D,%3D, {%2D[,%2D…]},%2D})	This 5-field section comprises: 1. BeiDou PRN number 2. Elevation in degrees 3. Azimuth in degrees, 4. Signal-to-noise ratios in [dB×Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. B1 2. altboc 3. B2B(I+Q)[1] 4. B3 5. B2A(I+Q) 6. B1C(I+Q) 5. Satellite navigation status (see “Satellite Navigation Status” on page 70) The total number of such 6-element lists will match the number of SVs being track.
4	@%2X	Checksum

1. B2 for BeiDou phase 2 satellites

[Is] IRNSS SVs Status

This message describes the status of IRNSS satellites.

#	Format	Description
1	ISSVST	Message title
2	%2D	Total number of IRNSS SVs being track
3	({%2D,%2D,%3D, {%2D[,%2D…]},%2D})	This 5-field section comprises: 1. IRNSS PRN number 2. Elevation in degrees 3. Azimuth in degrees, 4. Signal-to-noise ratios in [dB×Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. Reserved (always empty) 2. Reserved (always empty) 3. Reserved (always empty) 4. Reserved (always empty) 5. L5 5. Satellite navigation status (see “Satellite Navigation Status” on page 70) The total number of such 6-element lists will match the number of SVs being track.
4	@%2X	Checksum

[LS] L-band SVs Status

This message describes the status of L-band satellites.

#	Format	Description
1	LSSVST	Message title
2	%2D	Total number of L-band SVs being track
3	({%2D,%2D,%2D,%3D,{%2D},%2D)	This 6-element list comprises: 1. L-band SV Signal Frequency Identifier (see below) 2. L-band SV Frequency Channel Number 3. Elevation in degrees 4. Azimuth in degrees 5. Signal-to-noise ratio in [dB×Hz]. This is a list containing single element: CA/L1 6. Satellite navigation status (see “Satellite Navigation Status” on page 70) The total number of such 6-element lists will match the number of SVs being track.
4	@%2X	Checksum

Signal Frequency Identifier is computed by the formula:

$$ID = (frq - 1545825000) / 500$$

where *frq* is L-band signal frequency in Hz.

[RS] Reference Station Status

This message contains parameters related to the reference station status.

#	Format	Description
1	REFST	Message title
2	%C	UTC time indicator: “V” means that UTC time is valid “N” means that UTC time is not valid
	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest digits designate seconds)

#	Format	Description
4	{%C,%C,%C, %C}	Checking reference station location: - the first field relates to the reference station APC coordinates used for referencing GPS data (see /par/ref/pos/gps/… parameters), - the second field relates to the reference station APC coordinates used for referencing GLONASS data (see /par/ref/pos/glo/… parameters), - the third field relates to the reference station ARP coordinates used for referencing GPS data (see /par/ref/arp/gps/… parameters), - the fourth field relates to the reference station ARP coordinates used for referencing GLONASS data (see /par/ref/arp/glo/… parameters).[1] "V" – means that the difference between the current receiver coordinates and the user-defined reference coordinates does not exceed the specified limit; "N" – means that the difference between the current receiver coordinates and the user-defined reference coordinates is greater than the specified limit; (see "Maximum Allowed Error in Reference Position" on page 335)
5	@%2X	Checksum

1. Parameters mentioned above are described in the section "Reference Station Coordinates" on page 332.

[TX] RTCM/CMR Text Message

This message allows the user to view text information derived on the rover end from messages RTCM 2.x Type 16, 36, 23, 24, RTCM 3.0 and CMR Type 2.

#	Format	Description
1	TEXT	Message title
2	%1D	Total number of the messages (0…2). The rest of the message is available only if this number is non-zero. Otherwise the total number of the messages value is immediately followed by the checksum.
3	({%02D/%02D ,%02D:%02D: %02D,"%S",% C,%C,%S})	Group of fields associated with the given text message (note that the total number of such groups is determined by the previous field). These fields are - Date of receiving of the message (MM/DD - month/day). -UTC time of receiving of the message (HH:MM:SS) -String value of up to 90 characters. -Decoder identifier (see below) -Data link identifier ("A"…"D" – serial ports, "M" – modem). -Reference station identifier.
4	@%2X	Checksum

Decoder identifier:

- A – shows that information has been decoded from RTCM messages 23 and 24;
- R – shows that information has been decoded from RTCM messages 16 and 36;
- T – shows that information has been decoded from RTCM 3.0 messages 1007 and 1008;
- N – shows that information has been decoded from RTCM 3.0 message 4091 (Topcon proprietary message, `/msg/rtcm3/4091t`);
- C – CMR decoder
- J – GREIS messages decoder.

[RM] Results of RAIM Processing

This message contains RAIM output data.

#	Format	Description
1	RAIM	Message title
2	%C	UTC time indicator: “V” means that UTC time is valid “N” means that UTC time is not valid
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%1D	RAIM indication: 0 - no anomalous measurements have been detected (position is valid); 1 - RAIM is not able to detect anomalous measurements (for example, due to poor geometry or limited number of visible satellites) – position may be badly affected by anomalous measurements; 2 - RAIM has detected and excluded anomalous measurements (position is valid). Note: Generally speaking, if more than one bad measurement is detected, there is no guarantee that the RAIM has excluded all bad measurements with the specified probability. However, in most cases, RAIM runs properly even if more than one bad measurement has been detected. If the RAIM indicator is other than “2”, this data field is followed by the checksum). 3 – RAIM is turned off (position may be badly affected by poor measurements);
5	%D	Total number of excluded bad measurements
6	(%C%2D)	IDs of the satellites with bad measurements. Note that the total number of such satellites is determined by the previous field. - Navigation system identifier: “G” – designates GPS satellites, “R” or “F” – both designate GLONASS satellites. “F” is used for “R” until the receiver has determined the satellite's slot number. - Satellite system number (or, for GLONASS, satellite frequency channel number): GPS SV PRN (follows after the “G” flag), GLONASS SV slot number (follows after the “R” flag) or GLONASS SV frequency channel number (follows after the “F” flag);

#	Format	Description
7	@%2X	Checksum

[NP] Navigation Position

This message includes the receiver's navigational and positioning parameters.

#	Format	Description
1	NAVPOS	Message title
2	%C	UTC time indicator: “V” means that UTC time is valid “N” means that UTC time is not valid
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%1D	Position computation indicator. If it equals “0”, this message contains more meaningful information (see the following data fields). If this indicator is non-zero, this data field is followed by the checksum). Position is valid only if the position computation indicator is equal to “0”. For how to interpret other values, see Table 3-19 on page 149.
5	%C%C	Position (first symbol) and velocity (second symbol) computation mode. See Table 3-20 on page 150.
6	{%2D,…,%2D}	Number of satellites used in position computation; from each system. Places of systems are always the same: {gps,glo,gal,sbas,qzss,bei,irnss}. Empty fields (two consecutive commas) are output where number of used satellites is zero. Any number of consecutive commas at the end are not output
7	%S	Reference geodetic datum identifier
8	%C%2Do%2D’%2.6F"	Latitude: hemisphere (“N” – northern, “S” – southern), degrees, minutes and seconds
9	%C%3Do%2D’%2.6F"	Longitude: hemisphere (“E” – eastern, “W” – western), degrees, minutes and seconds
10	%+5.4F	Altitude above ellipsoid of the reference datum [meters]
11	%C	Geoidal separation indicator: “V” means geoidal separation is valid; “N” means geoidal separation is unavailable.
12	%+.4F	Geoidal separation: the distance between ellipsoid of the reference datum and geoid (mean-sea-level) [meters]
13	%.2F	Horizontal dilution of precision (HDOP) []

#	Format	Description
14	%.2F	Vertical dilution of precision (VDOP) []
15	%.3F	Horizontal position RMS error [meters]
16	%.3F	Vertical position RMS error [meters]
17	%.4F	Horizontal velocity [kilometers/hour]
18	%+.4F	Vertical velocity [kilometers/hour]
19	%3.3F	True heading [degrees]
20	%C	Magnetic heading indicator: “V” means magnetic heading is valid; “N” means magnetic heading is not valid.
21	%3.3F	Magnetic heading [degrees]
22	%.3F	Horizontal velocity RMS error [meters/second]
23	%.3F	Vertical velocity RMS error [meters/second]
24	%.2F	Data link quality in percent (0…100)
25	%3D	Time [in seconds] elapsed since last RTCM, CMR or JPS message was received (maximum value = 999). Estimated with an accuracy of ±1 second.
26	@%2X	Checksum

Table 3-19. Position Computation Indicator

0	Position is valid
1	Too many iterations have been made (position is not valid)
2	Singular matrix (position is not valid)
3	Not enough data for position computation (position is not valid)
4	Either or both altitude and speed exceed specified threshold values (position is not valid)
5	PDOP exceeds specified threshold value (position is not valid) See the parameter `/par/pos/pdop` in “Positioning Parameters” on page 238.
6	Wrong position. Calculated position is outside of sensible margins.
7	Position was computed, but the output of given type of solution is disabled by the user (e.g., stand-alone position is not output due to `/par/pos/mode/sp` parameter is set to `off`)
8	Position output is disabled by option

Table 3-20. Position/Velocity Computation Mode

A	Autonomous mode
D	Code differential mode
C	RTK positioning with codes
F	RTK positioning with float integers
R	RTK positioning with fixed integers
P	Fixed position, i.e., entered by user (not computed)

[MP] Position in Map Projection

The message describes receiver position in the specified map projection or local coordinate system.

#	Format	Description
1	MAPRJ	Message title
2	%C	UTC time indicator: "V" means that UTC time is valid "N" means that UTC time is not valid
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest digits designate seconds)
4	%1D	Position computation indicator. If it equals "0", this message contains meaningful information (see the following data fields). If this indicator is non-zero, this data field is followed by the checksum. Position is valid only if the position computation indicator is equal to "0". For how to interpret other values, see Table 3-19 on page 149.
5	%C	Grid (or local) coordinates indicator: "V" means that Grid (or local) coordinates are valid "N" means that Grid (or local) coordinates are not valid
6	%S	Grid system ID (see Table below)
7	%2D	Zone of the grid system ("00", if not available)
8	%+.4F	Northern component of grid coordinates or "X" component of local coordinates [meters]
9	%+.4F	Eastern component of grid coordinates or "Y" component of local coordinates [meters]
10	%+5.4F	Altitude above ellipsoid or local horizon (for local coordinates) [meters]

#	Format	Description
11	@%2X	Checksum

Grid system IDs

UTM	Universal Transversal Mercator
TM	User-defined Transversal Mercator projection
STER	User-defined Stereographic projection
LOC	Local coordinates

[NR] Lever Arm Position

This message contains the position of the master antenna corrected by the rotated lever arm vector.

#	Format	Description
1	ARMPOS	Message title
2	%C	UTC time indicator: “V” means that UTC time is valid “N” means that UTC time is not valid
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%1D	Position computation indicator. If it equals “0”, this message contains more meaningful information (see the following data fields). If this indicator is non-zero, this data field is followed by the checksum). Position is valid only if the position computation indicator is equal to “0”. For how to interpret other values, see Table 3-19 on page 149.
5	%C%C	Position (first symbol) and velocity (second symbol) computation mode (below). See Table 3-20 on page 150.
6	{%2D,…,%2D}	Number of satellites used in position computation; from each system. Refer to [NP] message for description.
7	%S	Reference geodetic datum identifier
8	%C%2Do%2D’%2.6F"	Latitude: hemisphere (“N” – northern, “S” – southern), degrees, minutes and seconds
9	%C%3Do%2D’%2.6F"	Longitude: hemisphere (“E” – eastern, “W” – western), degrees, minutes and seconds
10	%+5.4F	Altitude above ellipsoid [meters]

#	Format	Description
11	%.3F	Horizontal position RMS error [meters]
12	%.3F	Vertical position RMS error [meters]
13	@%2X	Checksum

[SY] Geographic Position Near Poles

The message contains position and velocity suitable near poles including quasi-course and quasi-heading.

#	Format	Description
1	QDIR	Message title
2	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds). “000000.00” if UTC time is unknown
3	%C	Indicator. Always equals to ‘A’
4	%2D%2D%.7F	Latitude: degrees, minutes and fractions of minutes
5	%C	‘N’- North, ‘S’ - South hemisphere
6	%3D%2D%.7F	Longitude: degrees, minutes and fractions of minutes
7	%C	‘E’ - Eastern, ‘W’ - Western hemisphere
8	%.4F	Ground speed [knots]
9	%.3F	True course [degrees]
10	%2D	Gregorian day
11	%2D	Gregorian month
12	%2D	Gregorian year modulo 100
13	%.3F	Heading [degrees]
14	%.3F	Quasi-course [degrees]
15	%.3F	Quasi-heading [degrees]
16	%%C	Attitude solution type. ‘N’ – not available, ‘A’ – autonomous, ‘D’ – code differential, ‘F’ – float, ‘R’ – fixed
17	@%2X	Checksum

[TR] Time Residuals

This message is intended for various time transfer applications. It contains information allowing the user to “match” an external clock to a specific GPS/GLONASS satellite's time scale.

#	Format	Description
1	TIMERES	Message title
2	%C	UTC time indicator: “V” means that UTC time is valid “N” means that UTC time is not valid
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest digits designate seconds)
4	%2D	Total number of satellites used in position computation
5	({%C%2D,%.2F,%D})	Group of fields associated with the given satellite (note that the total number of such groups is determined by the previous field). These fields are - Navigation system identifier: “G” – designates GPS satellites, “R” or “F” – both designate GLONASS satellites. “F” is used for “R” until the receiver has determined the satellite's slot number. - Satellite system number (or, for GLONASS, satellite frequency channel number): GPS SV PRN (follows after the “G” flag), GLONASS SV slot number (follows after the “R” flag) or GLONASS SV frequency channel number (follows after the “F” flag); - “Time residual” in nanoseconds for the given satellite. This satellite-specific time correction is defined as V1 - V2, where - V1 is the system time[1] to the receiver time offset as estimated using this particular satellite (i.e., the difference between the satellite's own clock[2] and the receiver clock). - V2 is the system time to the receiver time offset as estimated using all of the satellites belonging to this navigation system. Note that it is V2 that governs PPS signals generated in the receiver. - Issue of data for satellite ephemeris: For GPS satellites this field includes IODE (Issue Of Data, Ephemeris); For GLONASS satellites this field includes Ephemeris reference time t_b (7 LSB) and the indicator (MSB), which is set to “1” if the ephemeris has been updated without a change in tb. Note: The interpretation of this field for GLONASS satellites is subject to change in the future.
6	@%2X	Checksum

1. “System time” means GPS and GLONASS system time for GPS and GLONASS satellites, respectively.
2. Assuming that the satellite clock has already been corrected to the corresponding system time, either GPS or GLONASS time, by applying the broadcast frequency-time corrections. The reader will notice that these “satellite-specific time residuals” first of all describe such effects as SA, multipath and atmospheric delay, which are all satellite-specific.

[TM] Clock Offsets and Time Derivatives

The message contains clock offsets (the difference between receiver time scale and GPS/GLONASS system time) and their derivatives.

#	Format	Description
1	TIMING	Message title
2	%C	UTC time indicator: “V” means that UTC time is valid “N” means that UTC time is not valid
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%1D	Position computation indicator. If it equals “0”, this message contains more meaningful information (see the following data fields). If this indicator is non-zero, this data field is followed by the checksum. Position is valid only if the position computation indicator is equal to “0”. For how to interpret other values, see Table 3-19 on page 149.
5	%C	Position computation mode. See Table 3-20 on page 150.
6	{%2D,…,%2D}	Number of satellites used in position computation; from each system. Refer to [NP] message for description.
7	{{%C	GPS time fields indicator: “V” means that GPS time fields are valid “N” means that GPS time fields are not valid
8	%+.4F}	Tr - Tg, [meters]
9	%+.4F}	Derivative of (Tr - Tg), [meters/second]
10	{%C	GLONASS time fields indicator: “V” means that GLONASS time fields are valid “N” means that GLONASS time fields are not valid
11	%+.4F	Tr - Tn, [meters]
12	%+.4F}}	Derivative of (Tr - Tn), [meters/second]
13	%.2F	Time dilution of the precision (TDOP)
14	%.3F	Clock offsets RMS error [meters]
15	%.3F	RMS of the derivatives of clock offsets [meters/second]
16	%1D	The Improved Timing mode indicator. If it equals “0”, this means the Improved Timing mode is turned off. If this field is “1” the mode is turned on.

#	Format	Description
17	@%2X	Checksum

[RP] Reference Station Parameters

This message contains the reference station parameters such as the station's coordinates, antenna offsets, station ID, etc. These parameters are used in RTK on the rover side. These parameters are available via RTCM Messages Types 3, 22, 23, 24 or 31, RTCM 3.0, or CMR Message Type 1.

#	Format	Description
1	REFPAR	Message title
2	%1D	Total number of groups containing the reference station fields. In the current version of the message, this field can be set to 0 or 1. "0" means that the reference station fields are not valid. "1" means that this message contains valid information.
3	({%S	Reference geodetic datum identifier
4	%C%2Dd%2D m%2.6Fs	Latitude: hemisphere ("N" – northern, "S" – southern), degrees, minutes and seconds
5	%C%3Dd%2D m%2.6Fs	Longitude: hemisphere ("E" – eastern, "W" – western), degrees, minutes and seconds
6	%.4F	Altitude above ellipsoid [meters]
7	%C	Antenna height indicator: "V" means antenna height is valid; "N" means antenna height is not valid.
8	%.4F	Antenna height [meters]
9	%C	Antenna East/North offsets indicator: "V" means East/North offsets are valid; "N" means East/North offsets are not valid.
10	%.4F	Antenna North offset [meters]
11	%.4F	Antenna East offset [meters]
12	%C	Decoder identifier, R, T, C, J, where A – shows that information has been decoded from RTCM messages 23 and 24; R – shows that information has been decoded from RTCM messages 3, 22 and 31; T – shows that information has been decoded from RTCM 3.0; C – CMR decoder J – GREIS messages decoder.

#	Format	Description
13	%C	Data link identifier (“A”…“D” – serial ports, “M” – modem, “U” – by user input).
14	%S	Reference station identifier.
15	%S})	Antenna ID. Two types of antenna IDs are supported: 1.IDs that are approved and standardized by NGS. 2.IDs that are used in the Trimble’s CMR format. CMR antenna IDs will always contain three digits.
16	@%2X	Checksum

Note: By default this message is output only after its contents have changed.

[RK] RTK Solution Parameters

This message contains some parameters of an RTK solution.

#	Format	Description
1	RTKPAR	Message title
2	%C	UTC time indicator: “V” means that UTC time is valid “N” means that UTC time is not valid
3	%6.2F	UTC time of the position fix (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%C	RTK data availability indicator: “V” means that RTK data are available “N” means that RTK data are unavailable. If this indicator is “N”, the indicator is followed by the checksum.
5	%D	Fixing ambiguity progress, in percentage []
6	%.1F	Estimated probability of fixing ambiguities correctly []
7	%.2F	χ^2 for the position fix []
8	%+.4F	Time shift between the rover and base receiver times [multiplied by the speed of light and presented in meters]
9	%+.4F	Derivative of time offset between the rover and the base [meters/second]
10	%.2F	Root-mean-squared single differenced ionosphere error as estimated by the RTK engine [meters]

#	Format	Description
11	%.2F	Corrections age, [seconds]. When RTK works in `extrap` mode, this field contains extrapolation time of data from reference station. When RTK works in `delay` mode, this field is zero.
12	@%2X	Checksum

[AP] Position Covariance Matrix

This message is a text version of the message [SP].

#	Format	Description
1	POSCOV	Message title
2	%C	UTC time indicator: “V” means UTC time is valid “N” means UTC time is not valid
3	%6.2F	UTC time of the position fix (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%C	Position covariance matrix indicator: “V” means that Position covariance matrix is available “N” means that Position covariance matrix is not available. If this indicator is “N”, the indicator is followed by the checksum.
5	%C	Position computation mode. See Table 3-20 on page 150.
6	%.3E	Cov(1,1)
7	%.3E	Cov(2,2)
8	%.3E	Cov(3,3)
9	%.3E	Cov(4,4)
10	%.3E	Cov(1,2)
11	%.3E	Cov(1,3)
12	%.3E	Cov(1,4)
13	%.3E	Cov(2,3)
14	%.3E	Cov(2,4)
15	%.3E	Cov(3,4)
16	@%2X	Checksum

[AB] Baseline

This message contains coordinates of baseline, defined as vector from rover to reference station.

#	Format	Description
1	BASLIN	Message title
2	%C	UTC time indicator: "V" means UTC time is valid "N" means UTC time is not valid
3	%6.2F	UTC time of the position fix (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%C	Availability indicator: "V" means that baseline is available "N" means that baseline is not available If this indicator is "N", the indicator is followed by the checksum.
5	%C	Position computation mode. See Table 3-20 on page 150.
6	%.4F	X_{ref} - X_{rov} [meters]
7	%.4F	Y_{ref} - Y_{rov} [meters]
8	%.4F	Z_{ref} - Z_{rov} [meters]
9	%.4F	Position 3D RMS [meters]
10	@%2X	Checksum

[TD] Text Data

Text Data is a generic text message containing different kinds of data.

#	Format	Description
1	%S	Message Type Designator
2	Data Type Dependent	Message Data
4	@%2X	Checksum

The following data designators are currently defined:

#	Designator	Name	Description
1	CGGTTS	/msg/jps/cgg	CGGTTS Auxiliary Data
2	TEMP	/msg/jps/temp	Temperature measurements

Designator Formats:

1. CGGTTS

Comma-separated list of fields in the form `NAME=VALUE`, surrounded by curly braces:

```
{NAME=VALUE,…,NAME=VALUE}
```

The data for this message is provided by `/par/timing/cggtts/hdrs` parameters.

Example:
```
⇐ TD036,CGGTTS,{FY=2005,REF=UTC(USNO),CH=816 (UNIVERSAL)},@7C
```

2. TEMP

Comma-separated list of fields in the form `NAME=VALUE`, surrounded by curly braces:

```
{NAME=VALUE,…,NAME=VALUE}
```

where `NAME` is the point of measurements, and `VALUE` is temperature in Celsius.

Example:
```
⇐ TD014,TEMP,{brd=46.2},@8D
```

3.5 Predefined Foreign Messages

3.5.1 Approved NMEA sentences

The NMEA-0183 (National Marine Electronic Association) standard v4.30 [1] is a specification intended to facilitate interconnection and interchangeability of equipment produced by different manufacturers. The standard defines data transmission specifications, message types and a data exchange protocol between Talker and Listener. It is widely used not only in marine applications but in many other applications too.

1. NMEA 0183 “Standard for Interfacing Marine Electronic Devices” Version 4.30

The NMEA-0183 standard provides, together with other information, the description of the "approved" sentences. "Approved sentences" are those having predefined formats. Every approved sentence has "talker identifier" and "sentence identifier" and is uniquely characterized by the corresponding (predefined) set of fields. There is a whole variety of devices that may serve as "talkers" in NMEA applications (e.g., marine sounders and weather instruments).

A specific "talker", however, may handle only a particular set of approved messages. For example, a combined GPS/GLONASS receiver utilizes only a limited number of the existing approved sentences (specifically, sentences containing GNSS–related navigational/positioning information).

By default, WGS84 position is output in NMEA sentences, see parameter "NMEA Datum" on page 382 for details.

General Format of Approved NMEA Sentences

Each approved NMEA sentence has the following format:

$AACCC,c---c×hh<CR><LF>

where

"$" (HEX 24) – Start of sentence.

AACCC – Address field. The first two characters identify "Talker". The last three characters identify the sentence type.

"," (HEX 2C) – Field delimiter.

c---c – Data sentence block.

"×" (HEX 2A) – Checksum delimiter.

hh – Checksum field. This value is computed by exclusive-OR'ing the eight data bits of each character in the sentence, between, but excluding, "$" and "×". The hexadecimal value of the most significant and least significant four bits of the result are converted into two ASCII characters (0…9,A…F) for transmission. The most significant character is transmitted first.

<CR><LF> (HEX 0D,0A) – sentence terminators. Approved NMEA sentences are allowed to contain the so-called "null fields". Null fields are used when one or more values in the message are unreliable or unavailable. A null field may be delimited by two commas (",,") or by a comma and a multiplication sign "×" (",×") depending on its position in the sentence. JAVAD GNSS receivers support the following "talker identifiers":

"GP" — Global Positioning Systems (GPS)

"GL" — GLONASS

"GN" — Global Navigation Satellite System (GNSS)

Generally speaking, "talker identifier" is supposed to inform Listener whether the positioning information contained in the message is "GPS only", "GLONASS only" or "combined GPS plus GLONASS". In reality, this is not always true: there are sentences whose "talker identifiers" will not indicate the true constellation used in position computation (please see notes to specific sentences listed in the section below).

NMEA-Specific Format Limitations

The NMEA standard forbids the use of character "+" inside approved NMEA sentences. Note that this limitation "overrides" the general format conventions described in "GREIS Format for Text Messages" on page 136.

In other words, in approved NMEA sentences, "+" is omitted before non-negative numbers even if the corresponding field formats contain the plus sign (e.g., %+7.5F).

It should be noted that the NMEA standard, as a rule, does not specify exact mantissa lengths for the sentence fields. The user is free to allocate to every field as many digits as necessary to ensure required accuracy. For example, since JAVAD GNSS receivers provide millimeter-level positioning accuracy in differential modes, receiver geodesic coordinates (latitude - longitude - ellipsoidal height) should have mantissas long enough to enable coordinate presentation with sub-millimeter accuracy. For the format conventions for the following sentences, please see section , "GREIS Format for Text Messages" on page 136.

GGA – Global Positioning System Fix Data

This message comprises time, position and other fix related data for JAVAD GNSS receiver.

#	Format	Description
1	%6.[0-2]F	UTC time of position fix (first two digits designate hours, the next two designate minutes and the rest digits designate seconds)
2	%4.[1-7]F	Latitude in selected datum (first two digits designate degrees and the rest designates minutes of arc)
3	%C	Latitude hemisphere: N – northern, S – southern
4	%5.[1-7]F	Longitude in selected datum (first three digits designate degrees and the rest digits designate minutes of arc)
5	%C	Longitude hemisphere: E – eastern, W – western

#	Format	Description
6	%1D	GPS quality indicator (see below for details)
7	%2D	Number of satellites used for position computation
8	%.2F	Horizontal dilution of precision (HDOP) [-]
9	%+5.[1-4]F	Altitude above geoid (mean-sea-level) [meters]
10	%C	Symbol “M” (denote that altitude is in meters)
11	%+.[1-4]F	Geoidal separation: the distance between ellipsoid of the reference datum and geoid (mean-sea-level) [meters]
12	%C	Symbol “M” (denotes that geoidal separation is in meters)
13	%.1F	Age of differential GPS data [seconds]
14	%4D	Differential reference station ID (an integer between 0000 and 1023)
15	×%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 160)

Note: The [GGA] message talker identifier uses the following JAVAD GNSS convention: whatever constellation is used for position computation (GPS only, GLONASS only, or GPS plus GLONASS), the talker identifier is always set to “GP”.

If your receiver uses combined GPS+GLONASS data for position computation in RTK or DGPS, “age of differential GPS data” and “differential reference station ID” from [GGA] message will relate to GPS data. On the other hand, if the receiver uses pure GLONASS data when computing the position in RTK or DGPS, the fields “age of differential GPS data” and “differential reference station ID” will relate to GLONASS data.

Generally speaking, it is not recommended to use [GGA] message when operating a full-functionality GPS/GLONASS receiver. Note that [GGA] is mainly intended for pure GPS receivers. For combined receivers, we recommend using [GNS] for [GGA].

GPS quality indicator:

0	Fix not available or invalid
1	GPS SPS Mode (single point mode), fix valid
2	Differential GPS SPS Mode, fix valid
3	GPS PPS Mode (single point mode), fix valid
4	RTK fixed
5	RTK float
6	Estimated (dead reckoning) mode

7	Manual input mode
8	Simulator mode

GLL – Geographic Position – Latitude/Longitude

Current latitude/longitude, time and status of position fix.

#	Format	Description
1	%4.[1-7]F	Latitude in selected datum (first two digits designate degrees and the rest designates minutes of arc)
2	%C	Latitude hemisphere: N – northern, S – southern
3	%5.[1-7]F	Longitude in selected datum (first three digits designate degrees and the rest designates minutes of arc)
4	%C	Longitude hemisphere: E – eastern, W – western
5	%6.[0-2]F	UTC time of position (first two digits designate hours, the next two digits designate minutes and the rest designates seconds)
6	%C	Status field (shall be set “V” = Invalid for all values of positioning system mode indicator (see next field) except for “A”= Autonomous, “D”= Differential, “P” = Precise, “R” = RTK with fixed integers and “F” = RTK with floating integers
7	%C	Positioning system mode indicator (see below).
8	×%2X\0D\0A	Checksum (see“General Format of Approved NMEA Sentences” on page 160)

Mode indicator:

A	Autonomous. Satellite system used in non-differential mode in position fix
D	Differential. Satellite system used in differential mode in position fix
E	Estimated (dead reckoning) mode
M	Manual input mode
S	Simulator mode
N	Data not valid

GNS – GNSS Fix Data

This message intended for combined navigation systems (GNSS). It comprises time/position/status fix data.

#	Format	Description
1	%6.[0-2]F	UTC time of position fix (first two digits designate hours, the next two digits designate minutes and the rest designate seconds)
2	%4.[1-7]F	Latitude in selected datum (first two digits designate degrees and the rest designates minutes of arc)
3	%C	Latitude hemisphere: N – northern, S – southern
4	%5.[1-7]F	Longitude in selected datum (first three digits designate degrees and the rest designates minutes of arc)
5	%C	Longitude hemisphere: E – eastern, W – western
6	%C%C	Mode indicator (see below): variable length valid character field type with the first two characters currently defined. The first character indicates the use of GPS satellites, the second character indicates the use of GLONASS satellites.
7	%2D	Total number of satellites used for position computation
8	%.2F	Horizontal dilution of precision (HDOP) []
	%+5.[1-4]F	Altitude above geoid (mean-sea-level) [meters]
	%+.[1-4]F	Geoidal separation: the distance between ellipsoid of the reference datum and geoid (mean-sea-level) [meters]
11	%.1F	Age of differential data [seconds] (see the note below)
12	%4D	Differential reference station ID (this is an integer between 0000 and 1023) (see the note below)
13	×%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 160)

Note: If your JAVAD GNSS receiver runs in “pure GPS” or “pure GLONASS” RTK or DGPS, it outputs one GNS message per position fix. If running in “dual system” RTK or DGPS (i.e., both GPS and GLONASS differential correction data are used simultaneously), the receiver outputs, in accordance with the NMEA standard, a “GNS triplet” for every position fix.

The first message in a GNS triplet plays the most important part carrying the lion's share of information. The other two messages provide some GPS-specific and GLONASS-specific information, specifically: “total number of satellites”, “age of differential data” and “differential reference station ID”[1].

The following is the example of a typical GNS triplet:

```
$GNGNS,122310.20,3722.425671,N,12258.856215,W,DD,14,0.9,1005.543,6.5,,×74<CR><LF>
$GPGNS,122310.20,,,,,,7,,,,5.2,23×4D<CR><LF>
$GLGNS,122310.20,,,,,,7,,,,3.0,23×55<CR><LF>
```

Positioning system mode indicator for [GNS] message

N	No fix.
A	Autonomous. Satellite system used in non-differential mode in position fix
D	Differential. Satellite system used in differential mode in position fix
P	Precise. Satellite system used in precision mode. Precision mode is defined as: no deliberate degradation (such as Selective Availability) and higher resolution code (P-code) is used to compute position fix
R	Real time kinematic. Satellite system used in RTK mode with fixed integers
F	Float RTK. Satellite system used in real time kinematic mode with floating integers
E	Estimated (dead reckoning) mode
M	Manual input mode
S	Simulator mode

GRS – GNSS Range Residuals

This message contains “range residuals”. These kinds of data are used to support Receiver Autonomous Integrity Monitoring (RAIM).

#	Format	Description
1	%6.[0-2]F	UTC time (the first two digits designate hours, the next two digits designate minutes and the rest designates seconds)
2	%1D	Mode: 0 = residuals were used to calculate the position given in the matching GGA or GNS sentence; 1 = residuals were recomputed after the GGA or GNS position was computed. Currently, the receiver uses only the first mode (Mode = 0).
3	(%+.3F) or (%+.0F)	A sequence of range residuals (in meters). Sequence length depends on the number of satellites used in the position solution. Order must match order of satellite ID numbers in GSA. When GRS is used GSA and GSV are generally required. If the range residual exceeds ±99.9 meters, then the decimal part is discarded, resulting in an integer (-103.7 becomes -103). The maximum value for this field is ±999.
4	×%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 160)

Note: The NMEA standard states the following:

1. Not to mention “UTC time of position fix”, which is the same for all three messages in the triple.

- If either GPS or GLONASS is used for position computation, the talker ID is set to “GP” or “GL”, respectively.

- If GPS and GLONASS are used together, the receiver will generate two GRS messages at one time. The first of these messages will describe the GPS range residuals whereas the other will describe the GLONASS range residuals. Either message will have the same talker ID, “GN”, which indicates that the range residuals actually relate to GNSS.

GSA – GNSS DOP and Active Satellites

This message describes GNSS receiver operating mode, satellites used in the position solution reported by the GGA or GNS sentence, and DOP values.

#	Format	Description
1	%C	Mode: M = Manual, forced to operate in 2D or 3D mode, A = Automatic, allowed to automatically switch 2D/3D
2	%D	Mode: 1 = Fix not available, 2 = 2D, 3 = 3D
3	%.2D	A sequence of satellite ID numbers. Sequence length is variable (depends on the amount of satellites used in solution). For more details on satellite ID numbers, see below.
4	%.2F	Position dilution of precision (PDOP)
5	%.2F	Horizontal dilution of precision (HDOP)
6	%.2F	Vertical dilution of precision (VDOP)
7	×%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 160)

The NMEA standard states the following:

- If either GPS or GLONASS is used for position computation, the talker ID is set to “GP” or “GL”, respectively.
- If GPS and GLONASS are used together, two GSA messages are generated at one time. The first and second messages relate to the GPS and GLONASS satellites, respectively. Both the messages, however, will have the same talker ID, “GN”, and the same DOP values (the latter are actually computed for the combined constellation). The talker ID “GN” indicates that this pair of messages relate to the one and the same GNSS solution.

Table 3-21. Satellite ID Numbers

NMEA Satellite ID Numbers	System Numbers
1…32	GPS PRN numbers 1…32

Table 3-21. Satellite ID Numbers

NMEA Satellite ID Numbers	System Numbers
65…88	GLONASS slot numbers [1…24]

GST – GNSS Pseudo-range Error Statistics

This message is used to support Receiver Autonomous Integrity Monitoring (RAIM).

#	Format	Meaning
1	%6.[0-2]F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest designate seconds)
2	%.3F	Estimated standard deviation of the range input's error. “SV Range input”, which is used in the navigation process, includes this satellite’s pseudo-range and the corresponding DGNSS correction [meters]
3	%.3F	Semi-major axis of error ellipse [meters]
4	%.3F	Semi-minor axis of error ellipse [meters]
5	%.3F	Orientation of semi-major axis of error ellipse [degrees from true north]
6	%.3F	RMS latitude error [meters]
7	%.3F	RMS longitude error [meters]
8	%.3F	RMS altitude error [meters]
9	×%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 160)

Note: In case the solution computed by the receiver is not RTK fixed or RTK float, fields 3, 4 and 5 are null fields.

GSV – GNSS Satellites in View

Number of satellites in view, satellite ID numbers, elevation, azimuth and SNR value.

#	Format	Meaning
1	%1D	Total number of messages, 1 to 3
2	%1D	Message number, 1 to 3
3	%2D	Total number of satellites in view

#	Format	Meaning
4	(%2D,%2D, %3D,%2D)	Satellite ID number (see GSA for ID numbers), elevation in degrees, azimuth in degrees and C/A signal-to-noise ratio (SNR) in dB×Hz
5	%1X	Signal ID (1 - GPS, 2 - GLO, 3 - Galileo)
6	×%2X\0D\ 0A	Checksum (see "General Format of Approved NMEA Sentences" on page 160)

Note: A variable number of "Satellite ID-Elevation-Azimuth-SNR" sets are allowed (up to a maximum of four sets per message).

In case the number of visible SVs exceeds 4, multiple messages are transmitted. The first field specifies the total number of messages (minimum value 1) whereas the second field identifies the order of this message (i.e., message number), minimum value 1.

Messages for GPS and GLONASS are generated separately: GPS messages will have Talker ID "GP" and GLONASS messages - Talker ID "GL".

RMC – Recommended Minimum Specific

GNSS Data Time, date, position, course and speed data provided by a GNSS navigation receiver.

#	Format	Description
1	%6.[0-2]F	UTC time of position fix (first two digits designate be hours, the next two designate minutes and the rest digits designate seconds)
2	%C	Status: A – Data valid, V – Navigation receiver warning
3	%4.[1-7]F	Latitude in selected datum (first two digits designate degrees and the rest designates minutes of arc)
4	%C	Latitude hemisphere: N – northern, S – southern
5	%5.[1-7]F	Longitude in selected datum (first three digits designate degrees and the rest digits designate minutes of arc)
6	%C	Longitude hemisphere: E – eastern, W – western
7	%.[1-4]F	Speed over ground (horizontal speed) [knots]
8	%3.[1-3]F	Course over ground (true course) [degrees]
9	%S	Date: DDMMYY
10	%3.[1-3]F	Magnetic variation [degrees]

#	Format	Description
11	%C	Magnetic variation direction: E – eastern, W – western
12	%C	Mode indicator (see Positioning system mode indicator in GLL message above)
13	×%2X\0D\0A	Checksum (see "General Format of Approved NMEA Sentences" on page 160)

HDT – Heading, True

#	Format	Description
1	%.3F	True Heading in degrees
2	%C	Symbol "T" indicates true heading
3	×%2X\0D\0A	Checksum (see "General Format of Approved NMEA Sentences" on page 160)

VTG – Course Over Ground and Ground Speed

The actual course and speed relative to the ground.

#	Format	Description
1	%3.[1-3]F	True course [degrees]
2	%C	Symbol "T" indicates True course
3	%3.[1-3]F	Magnetic course [degrees]
4	%C	Symbol "M" indicates Magnetic course
5	%.[1-4]F	Horizontal speed [knots]
6	%C	Symbol "N" indicates that horizontal speed is given in knots
7	%.[1-4]F	Horizontal speed [km/h]
8	%C	Symbol "K" indicates that horizontal speed is given in km/h
9	%C	Mode indicator (see "GGA – Global Positioning System Fix Data" on page 161)
10	×%2X\0D\0A	Checksum (see "General Format of Approved NMEA Sentences" on page 160)

ROT – Rate of Turn

#	Format	Description
1	%.3F	Rate of turn, degrees/minute. If negative, bow turns to port else bow turns to starboard 2
2	%C	A = data valid, V = data invalid
3	×%2X\0D\0A	Checksum (see "General Format of Approved NMEA Sentences" on page 160)

Note: This message is available if heading mode is turned on (i.e., `/par/pos/pd/hd/mode` is set to `on`).

ZDA – UTC Time and Date

#	Format	Description
1		%6.[0-2]F UTC time (first two digits designate hours, next two digits designate minutes and the rest designates seconds)
2	%2D	Day (varies between 01…31)
3	%2D	Month (varies between 01…12) 4 6
4	%4D	Year
5	%2D	Local zone hours (varies from -13 to +13)
6	%2D	Local zone minutes (varies from 00 to 59)
7	×%2X\0D\0A	Checksum (see "General Format of Approved NMEA Sentences" on page 160)

Note: Local time zone is the magnitude of hours plus the magnitude of minutes added, with the sign of local zone hours, to local time to obtain UTC.

To specify values of local zone hours and local zone minutes, use the command `set,/par/pos/ltz,{H,M}` (see"Positioning Parameters" on page 238).

GMP - GNSS Map Projection Fix Data

This message contains fix data for single or combined navigation systems (GNSS) in grid (or local) coordinates expressed in the given map projection.

#	Format	Description
1	%6.[0-2]F	UTC time of position fix (first two digits designate hours, the next two digits designate minutes and the rest digits designate seconds).
2	%S	Map projection identification.
3	%S	Map zone.
4	%.[1-4]F	Y (Northern) component of grid (or local) coordinates in meters.
5	%.[1-4]F	X (Eastern) component of grid (or local) coordinates in meters.
6	%C%C	Mode indicator (see GNS message): variable length valid character field type with the first two characters currently defined. The first character indicates the use of GPS satellites, the second character indicates the use of GLONASS satellites.
7	%2D	Total number of satellites used for position computation.
8	%.2F	Horizontal dilution of precision (HDOP) [-]
9	%+5.[1-4]F	Altitude above geoid (mean-sea-level) [meters]
10	%+.[1-4]F	Geoidal separation: the distance between ellipsoid of the reference datum and geoid (mean-sea-level) [meters]
11	%.1F	Age of differential data [seconds]. See the note below.
12	%4D	Differential reference station ID (this is an integer between 0000 and 1023). See the note below.
13	×%2X\0D\0A	Checksum

Note: All the notes applicable to the NMEA GNS sentence are valid for GMP message as well.

3.5.2 JNS Proprietary NMEA Sentences

All JNS proprietary NMEA sentences should have the following format:

$PTPSR,MsgID,c---c×hh<CR><LF>

ATT – Attitude

#	Format	Description
1	%C	UTC time indicator: "V" means that UTC time is valid "N" means that UTC time is not valid
2	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
3	%.3F	True Heading in degrees
4	%.3F	Pitch in degrees
5	%C	Base position type ('N' – not available, 'A' – autonomous, 'D' – code differential, 'F' – float, 'R' – fixed). This field can be useful provided JPS messages are used for broadcasting RTK data
6	%.3F	RMS of the True Heading in degrees
7	×%2X\0D\0A	Checksum (see , "General Format of Approved NMEA Sentences" on page 160)

Note: To enable [ATT] message, use the message name `/msg/nmea/P_ATT`

3.5.3 Meteo NMEA Sentences

XDR - Raw Meteo Data

$WIXDR,…<CR><LF>

This message contains data from connected meteo-sensor as is. The input mode of the port to which the sensor is connected should be set to `jps` for this message to be actually output.

3.5.4 RTCM 2.x Messages

Introduction to RTCM 2.x Messages

RTCM (Radio Technical Commission For Maritime Services) SC-104 (Special Committee 104) has developed a standard for differential GNSS (Global Navigation Satellite Systems) service[1]. The standard formulates recommendations in the following areas:

1. See for details: RTCM recommended standards for differential GNSS (Global Navigation Satellite System) service, version 2.3, August 20, 2001. (RTCM PAPER 136-2001/SC104-STD)

1. Data message and format – The message elements that make up the corrections, the status messages, the station parameters and ancillary data are defined in some details.
2. User interface – A standard interface is defined which enables a receiver to be used in concert with a variety of different data links.

The formats both GPS/GLONASS RTK data and GPS/GLONASS differential corrections are defined as well as the formats of the messages, which includes, e.g., reference station parameters, special message etc.

This standard is used, for example, for broadcasting the differential corrections by the radio-beacon based differential services located near coastal waters all over the world.

Every RTCM message consists of a variable number of 30-bit words (the reader will notice some resemblance to the structure of the GPS Navigation Message). The first two words in any RTCM message serve as the message header. The header comprises the following data fields: preamble, message type, reference station ID, modified Z-count, sequence number, length of frame and station health. The contents of the other words will depend on the type of the message (see 9 for more details).

Supported RTCM 2.x Messages

Table below contains a list of RTCM messages currently supported by JAVAD GNSS receivers. The column “type” indicates the message type number as specified in the RTCM standard.

The second column shows the message's GREIS-specific identifier (or simply “ID”). When the user enables a particular RTCM message with an appropriate command, he/she specifies the message “ID”, not the message “type”.

The third column, “title”, describes the message contents. 203The fourth column, “period”, specifies the default periods of the corresponding RTCM messages.

The last column explains how to calculate an RTCM message's length in 30-bit words.

Note that according to the RTCM standard it takes the receiver five bytes52 to transmit a 30-bit word. This is because the two MSB in each of these five bytes are “reserved” (more precisely, set to some predefined values). Thus the following formula to compute the actual length of an RTCM message [in bits]:

$$[\text{Length in bits}] = [\text{Length in 30-bit words}] \times 5 \times 8$$

Note: N designates the total number of satellites, and S designates the length (in characters) of the user-specified text

Type	ID	Title	Period [s]	Length in 30-bit words
1	1	Differential GPS corrections	1	2+N×2-N/3
3	3	GPS reference station parameters	10	2+4
6	6	GPS null frame	30	2
9	9	GPS partial correction set	1	2+N×2-N/3
15	15	Ionospheric delay message	60	2+N×2-N/2
16	16	GPS special message	30	2+1+(S-1)/3
18	18	RTK uncorrected carrier phases	1	3+N×2 for C/A or P L1 data 2×(3+N×2) for L1+L2 data
19	19	RTK uncorrected pseudo-ranges	1	3+N×2 for C/A or P L1 data 2×(3+N×2) for L1+L2 data
20	20	RTK carrier phase corrections	1	3+N×2 for C/A or P L1 data 2×(3+N×2) for L1+L2 data
21	21	RTK/High accuracy pseudo-range corrections	1	3+N×2 for C/A or P L1 data 2×(3+N×2) for L1+L2 data
22	22	Extended reference station parameters	10	(2+1) to (2+3)
23	23	Antenna Type Definition Record	10	2+1+[Nant+Nser] Nant - the number of characters used for antenna descriptor; Nser – the number of characters used for serial number.
24	24	Reference Station Antenna Reference Point Parameter, which provides the exact location of the reference station and the antenna height as the distance to the Antenna Reference Point (ARP).	10	5 (if the antenna height is not included in the message) or 6 (if the antenna height is included in the message).
31	31	Differential GLONASS corrections	1	2+N×2-N/3
32	32	GLONASS reference station parameters	10	2+4
34	34	GLONASS partial correction set	1	2+N×2-N/3
34	65	GLONASS null frame (34th message with N=0)	30	2
36	36	GLONASS special message	30	2+1+(S-1)/3

Type	ID	Title	Period [s]	Length in 30-bit words
41	41	Differential GNSS corrections	1	3+(2+9×N)
42	42	GNSS partial correction set	1	3+(2+9×N)
44	44	GNSS ionospheric delay message	60	3+N

Note: Unless you want to provide backward compatibility with earlier versions of the RTCM format, JAVAD GNSS recommends that you use messages 23 and 24 instead of messages 3 and 22. To retain backward compatibility with previous versions of RTCM format, it is recommended to transmit both pairs of the messages in the same RTK data stream.

This table indicates how many 30-bit words each specific RTCM message takes. For information on the total amount of data (in bytes) transmitted by the reference station in RTK or DGPS, see *"GREIS User's Manual"* available from *http://www.javad.com*.

3.5.5 RTCM 3.2 Messages

RTCM (Radio Technical Commission For Maritime Services) SC-104 (Special Committee 104) has developed a standard for differential GNSS (Global Navigation Satellite Systems) service[1].

The following RTCM 3.2 messages are supported:

- 1001 – L1-Only GPS RTK Observables
- 1002 – Extended L1-Only GPS RTK Observables
- 1003 – L1&L2 GPS RTK Observables
- 1004 – Extended L1&L2 GPS RTK Observables
- 1005 – Stationary RTK Reference Station ARP
- 1006 – Stationary RTK Reference Station ARP with Antenna Height
- 1007 – Antenna Descriptor
- 1008 – Antenna Descriptor & Serial Number
- 1009 – L1-Only GLONASS RTK Observables
- 1010 – Extended L1-Only GLONASS RTK Observables
- 1011 – L1&L2 GLONASS RTK Observables
- 1012 – Extended L1&L2 GLONASS RTK Observables
- 1019 – GPS ephemeris

1. See for details: RTCM STANDARD 10403.3, October 7, 2016

- 1020 – GLONASS ephemeris
- 1033 – Receiver and Antenna Descriptor
- 1044 – QZSS ephemeris
- 1045 – Galileo ephemeris
- 1071-1077 – GPS MSM RTK Observables
- 1081-1087 – GLONASS MSM RTK Observables
- 1091- 1097 – Galileo MSM RTK Observables
- 1101- 1107 – SBAS MSM RTK Observables
- 1111-1117 – QZSS MSM RTK Observables
- 1121-1127 – BeiDou MSM RTK Observables
- 4090t – Proprietary text message. This message provides possibility to transmit some text from the base to the rover receiver. Receivers of other brands may not understand this message.

Example: To enable output of messages 1004, 1012 with period 1 second, and 1006 with period of 30 seconds, on serial port C, issue the following command:

```
⇒ em,/dev/ser/c,/msg/rtcm3/{1004,1012,1006:30}:1
```

3.5.6 CMR Messages

Introduction to CMR Messages

The Compact Measurement Record (CMR) format was developed by Trimble Navigation Limited and now is approved for public use. The format is suitable for communication links that have a minimum of 2400 baud throughput (assuming that only GPS data is used). It provides significant advantages over RTCM messages (the latter are at least twice as long as compared against their CMR counterparts). It should be noted however that the original version of the CMR format does not allow for GLONASS. Therefore the original format definition needs to be expanded to include GLONASS.

For a detailed description of the CMR format see ftp://ftp.trimble.com/pub/survey/cmr.

Supported CMR Messages

Table below lists CMR messages currently supported by JAVAD GNSS receivers.

The column “type” indicates the message type as specified in the CMR standard except the message type var(3). GREIS defined this type to overcome some limitations existing in the CMR protocol.

The second column shows what GREIS specific identifiers (IDs) are assigned to different CMR messages. When enabling a CMR message with an appropriate GREIS command, the user specifies its “ID”, not “type”.

The third column, “title”, describes what kind of data each CMR message contains.

The fourth column, “period”, defines default periods for CMR messages.

The last column explains how to calculate the length of a CMR message (in bytes).

The following notations are used:

`N` – designates the total number of satellites.

`S` – is the length (in characters) of Long Station ID.

`FREQ` – takes 1 and 2 for single- and dual-frequency measurements, respectively.

`Var` – indicates that the type is subject to change

Type	ID	Title	Period (seconds)	Length in 30-bit words
0	0	GPS measurements	1	6 + N × (8 + (FREQ-1) × 7)
1	1	Reference station coordinates	10	31
2	2	Reference station description	10	31+S
var(3)	10	GLONASS measurements (GREIS proprietary message)	1	6 + N × (8 + (FREQ-1) × 7)
+	9[1]	Scrolling Station Information	1	16

1. This message has been developed for reducing the peak throughput of the CMR format. It is used for CMR message types 1 and 2. Data from this message type is transmitted by “frames”. Each frame is 16 bytes in size (7-byte message body plus 9 auxiliary bytes), and is transmitted together with CMR Message Types 0 and 10, which allows considerable reduction of the data link peak load. For more details about this message, see ftp://ftp.trimble.com/pub/survey/bin/uconf97.exe

3.5.7 BINEX Messages

BINEX, for “BINary EXchange”, is an operational binary format standard for GPS/GLONASS/SBAS research purposes. It has been designed to allow encapsulation of all (or most) of the information currently allowed for in RINEX OBS, GPS RINEX

NAV, GLONASS RINEX NAV, RINEX MET, IONEX, SP3, SINEX, and so on, plus other GNSS-related data and meta-data as encountered, including next-generation GNSS.

For a detailed information on BINEX specifications, refer to UNAVCO's Web site at *http://www.binex.unavco.org*.

This group comprises the following BINEX messages:

/msg/binex/00_00 - BINEX record 0x00-00
/msg/binex/01_01 - BINEX record 0x01-01
/msg/binex/01_02 - BINEX record 0x01-02
/msg/binex/01_03 - BINEX record 0x01-03
/msg/binex/01_04 - BINEX record 0x01-04
/msg/binex/01_06 - BINEX record 0x01-06
/msg/binex/7D_00 - BINEX record 0x7D-00
/msg/binex/7E_00 - BINEX record 0x7E-00
/msg/binex/7F_02 - BINEX record 0x7F-02
/msg/binex/7F_03 - BINEX record 0x7F-03
/msg/binex/7F_04 - BINEX record 0x7F-04
/msg/binex/7F_14 - BINEX record 0x7F-14
/msg/binex/7F_05 - BINEX record 0x7F-05

For the BINEX record 0x00-00, the following fields are supported:

```
0x04 0x0f 0x17 0x19 0x1a 0x1b 0x1d 0x1f
```

It's possible to turn on/off the output of each of the above fields using the /par/binex/00_00 parameters.

The values for fields 0x04 and 0x0f could be specified using parameters /par/binex/site and /par/binex/data_id, respectively.

Meteorological data for BINEX record 0x7E-00 could be obtained by connecting MET3-compatible sensor to a receiver port, setting the imode of the port to jps and enabling output of /msg/misc/MET3 to this port.

Example: Program receiver to get data from MET3 sensor working at 9600 baud and connected to the serial port B. The MET3 data are requested every 60 seconds with offset -2 seconds (phase =86400-2) so that the data are ready by the time BINEX record 0x7E-00 is out-

put. Then request output of BINEX record `0x7E-00` every 60 seconds to the serial port A:

```
⇒ set,/par/dev/ser/b/rate,9600
⇒ set,/par/dev/ser/b/imode,jps
⇒ em,/dev/ser/b,/msg/misc/MET3:{60,86398}
⇒ em,/dev/ser/a,/msg/binex/7E_00:60
```

RECEIVER OBJECTS

In this chapter we will describe all the receiver objects in details.

4.1 Overview

Recall that every object has an unique identifier, or *name*, that is used to address the object in GREIS commands, and that all the objects are organized into single tree-like structure that not only groups related objects together, but also allows to apply a command to a group of objects. The object tree starts at the single root list, and ends at the tree leafs. As all non-leaf objects have the same type *list* and behave similarly with respect to GREIS commands[1], we mostly describe leaf objects in this chapter.

Most of the (leaf) objects could be used both in the `print` (or `list`) and `set` commands. We call such objects *read-write* objects. Those objects that can't be used in the `set` command, are called *read-only* objects, whereas objects that can't be used in the `print` command, are called *write-only* objects. Description of each object contains the field *access* that specifies if the object is read-write or read-only. If an object could be used as an argument in some other GREIS commands, this ability will be explicitly mentioned in the description of the object.

Each object has a *type* associated with it. Object type defines the formats that are accepted by the `set` command for this object, and the format that the `print` command will use when it reports the state of the object. Note that the `set` command may accept multiple formats for given type, whereas the `print` command will always use one fixed format from those supported by the `set` command. For example, the `set` command for an *integer* type will accept values in decimal, hexadecimal, or octal format, while the `print` command will always use one of these formats for given object. The format that is typically used by the `print` command for a given type is called the *default format* for this type. Should the `print` command use non-default format for an object, the format is either explicitly specified in the description of this object, or matches those that is used to specify the allowed values and the default value of the object.

1. Due to limitations of the current implementation of the `set` command, it doesn't support most of non-leaf objects. Those non-leaf objects that nevertheless are supported by the `set` command are explicitly described in this chapter.

When appropriate, an object description contains a range (or a list) of allowed values, as well as the default value of the object. The allowed values and the default value are always specified in the format that the `print` command will use for this object.

4.2 Conventions

4.2.1 Object Specification

Every object specification found in the section "Objects Reference" on page 190 has the following representation:

```
Name:     name
Access:   access
Type:     type
Values:   allowed_values
Default:  default_value
Options:  options_spec
```

<DESCRIPTION>

where:

`name` - is the full name of the object (object identifier).

`access` - access type. `rw` – for read-write object, `r` – for read-only object, or `w` – for write-only object.

`type` - the type of the object and the measurement units of the object, the latter being taken into square brackets.

`allowed_values` - specification of the range of values allowed for the object. For integer or float values, the range is specified in the form `[A…B]`, where `A` and `B` are the lower and upper bounds of the range, inclusive. If a bound is excluded, then round bracket is used instead of square one, e.g., `[A…B)` means the range where `A` is included and `B` is excluded. For a list of allowed values, the values listed are delimited either by comma or by the vertical bar (`|`) character.

`default_value` - the default value of the object in the format that the `print` command will use, or the text `(empty string)` that for objects of type string denotes the string comprising zero characters.

`options_spec` - the specification of options the `set` command may take for given object in the format `{op1: type1,…,opN: typeN}`, where `opX` is the description of option, and `typeX` is corresponding option type.

`<DESCRIPTION>` - textual description of the object and the meaning of its values.

4.2.2 Input and Output Ports Notations

Receiver may support many input/output ports. To denote receiver ports in the object specifications, the notations described in this section are used.

[port] – input/output port

The `[port]` denotes any of ports suitable both for input and for output. It may take one of the following values:

`dev/ser/X, X=[a…d]` - RS232 (serial) ports

`dev/tcp/X, X=[a…e]` - TCP ports

`dev/usb/a` - USB port

`dev/tcpcl/X, X=[a,b]` - TCP client ports

`dev/can/X, X=[a,b]` - CAN ports

`dev/blt/X, X=[a,b]` - Bluetooth ports

`dev/prl/a` - parallel port (almost obsolete, – no newer receivers support this)

[oport] – output port

The `[oport]` denotes any of ports suitable for output only. It may take one of the following values:

`[port]` - input/output ports

`cur/file/X, X=[a,b]` - current log-files

`dev/udp/X, X=[a…e]` - UDP ports

`dev/tcpo/X, X=[a…e]` - TCP output ports

`dev/ntrip/X, X=[a…e]` - NTRIP Caster output ports

cur/term – current terminal

You can use the string `cur/term` to denote the port the command is issued by wherever `[port]` or `[oport]` is allowed. The `cur/term` will be substituted by the actual port name before the command is executed. Therefore, for example, if you set some parameter to `/cur/term` when sending command through `/dev/ser/a`, the value of the parameter, once the command is executed, will become `/dev/ser/a`.

4.3 Primary Object Types

In this section, for object types that are frequently used, we describe formats that are accepted by the `set` command and could be used by the `print` commands. Formats for the object types that are used for single object are described along with corresponding objects.

4.3.1 list

The `list` format is a comma-separated sequence of fields surrounded by braces (`{` and `}`). When an object of this type is assigned a value using `set` command, some of the fields could be omitted, in which case corresponding fields will retain their previous values.

4.3.2 array

The type `array` is a kind of type `list` where all the fields have the same type and have names assigned after the decimal representations of their indexes.

The notation

```
array [N…M] of <type>
```

is used in descriptions of array objects, where:

`N` - is the index of the first element of the array

`M` - is the index of the last element of the array

`<type>` - is the name of the type of elements of the array

The format for `array` type is the same as for `list` type.

For arrays of `boolean`, in addition to the format of the `list` type, the value for the `set` command could be given as an integer number, where the bits of the number correspond to elements of the array. Least significant bit (bit #0) corresponds to the first element of the array, and bit #`K`, where `K=M-N`, corresponds to the last element of the array. If bit is set to 1, corresponding element will have *true* value; if bit is set to 0, corresponding element will have `false` value.

4.3.3 integer

Integer values could be specified in one of the following formats:

`decimal` - optional plus or minus sign, then one or more digits in the range `[0…9]`, where the first digit is not `0`. For example, `493`.

`octal` - optional plus or minus sign, then the digit `0` followed by one or more digits in the range `[0…7]`. For example, `0371`.

`hexadecimal` - optional plus or minus sign, then the string `0x` followed by one or more characters in the range `[0…9,a…f,A…F]`. For example, `0x03f0`, or `-0xCAF`.

Decimal representation is the default one for `print` command.

4.3.4 float

Float values could be specified in the following format:

- An optional plus or minus sign (`+` or `-`).
- A nonempty sequence of digits optionally containing a decimal-point character (`.`).
- An optional exponent part, consisting of a character `e` or `E`, an optional sign, and a sequence of digits.

For example, `+3.24e-10`, or `-0.001`.

4.3.5 enumerated

An object of type `enumerated` may posses one of the values taken from predefined set of value. The set of possible values is defined for each object of this type individually.

4.3.6 boolean

An object of type boolean may posses the following values:

`y, yes, on` - stand for *true*

`n, no, off` - stand for *false*

The `print` command will use variant specified in the parameter description.

4.3.7 string

The type `string` denotes possibly empty sequence of characters. When upper- and lower-number of characters in the string is defined, the notation `string [N…M]` is used,

where N is the minimum, and M is the maximum allowed number of characters in the string.

4.3.8 sched_params

The type sched_params denotes message scheduling parameters. The supported format is as follows:

```
{period,phase,count,flags}
```

where:

- period - field of type float denoting message output period in seconds within the range [0…86400).
- phase - field of type float denoting message output phase or forced output period in seconds within the range [0…86400).
- count - message output count of type integer in the range [-256…32767).
- flags - message scheduling flags of type integer formatted as hexadecimal.

For detailed description of the message scheduling parameters, refer to "Periodic Output" on page 22.

4.3.9 timespec

The timespec type is used to define time specifications. The canonical form of timespec is as follows:

```
DdHHhMMmSSs
```

where D, HH, MM, and SS fields are either integer numbers in corresponding range or special value "__" serving as wild-card.

- D - number of day inside a week [1…7]. 1-Sunday… 7-Saturday.
- HH - number of hour inside a day [0…23].
- MM - number of minute inside an hour [0…59].
- SS - number of second inside a minute [0…59].

Here are two examples of valid timespec: 4d17h40m18s __d__h00m__s

Receivers print command always outputs timespec in its canonical form. Receivers set command, however, accepts timespec not only in canonical form, but also in simplified forms. The rules for set command are as follows:

- If some field is omitted, it is assumed to be "__".

- Single underscore is the same as double underscore.
- Any number of integer digits is accepted.

Thus, for example, empty string `timespec` is taken as `__d__h__m__s`, and `8h2s` is taken as `__d08h__m02s`.

4.3.10 ip_address

The `ip_address` type format is standard Internet IPv4 address in numbers-and-dots notation.

4.3.11 datum_id

The `datum_id` type is a string of up to 5 upper-case characters designating datum identifier. Refer to "Datums" on page 247 for details.

4.3.12 pos_xyz

The `pos_xyz` type is used to denote Cartesian coordinates. It has the following format:

```
{datum_id,x,y,z}
```

where:

`datum_id` - field of the type `datum_id` specifying the datum to which coordinates are referenced.

`x,y,z` - Cartesian coordinates in meters. Allowed range is `[-10000000…10000000]`.

4.3.13 pos_geo

The `pos_geo` type is used to denote Geodetic coordinates. It has the following format:

```
{datum_id,lat,lon,alt}
```

where:

`lat` - latitude (see below for accepted formats). Allowed range is `[-90…90]` degrees. Negative latitude corresponds to the Southern hemisphere.

`lon` - longitude (see below for accepted formats). Allowed range is `[-180…180]` degrees. Negative longitude corresponds to the Western hemisphere

`alt` - altitude in meters. Allowed range is `[-20000…20000]`.

Output Format for Angles

For latitude and longitude, the `print` command uses the following formats:

`[N|S]DDdMMmSS.SSSSSSs` – for latitude

`[E|W]DDDdMMmSS.SSSSSSs` – for longitude

where

`N, S, E, and W` – designate Northern, Southern, Eastern, and Western hemisphere, respectively,

`DD, DDD` – integer degrees,

`MM` – integer minutes,

`SS.SSSSSS` – integer and fractional seconds,

`d, m, s` – delimiters.

For example,

`N83d42m47.556000s` – means 83 degrees 42 minutes and 47.556 seconds Northern latitude.

`E083d42m47.556000s` – means 83 degrees 42 minutes and 47.556 seconds Eastern longitude.

The `set` command supports two different formats for latitude and longitude. These are the *general format* and the *almost fixed format*.

General Input Format for Angles

General format for entering latitudes and longitudes is an extended version of the format that receiver uses when it outputs these angles.

As the name of this format implies, this is a very flexible format enabling you to specify latitude and longitude in a number of different ways. You can use various angular units (specifically, degrees, minutes, seconds, and radians) and their combinations.

Angle representation may comprise one or more floating point numbers. Every float number in the angle representation except the right-most one must have a delimiter after it. Allowed delimiters are "`d`", "`m`", "`s`", or "`r`", which denote degrees, minutes, seconds, and radians, respectively. Using a delimiter after the right-most float number in the angle representation is optional. If you omit the delimiter after the right-most float number, the

receiver will first identify the preceding delimiter and then retrieve the omitted one by using the following decision rule:

Preceding Delimiter	Assumed Omitted Delimiter
none	d
d	m
m	s
s	s
r	r

An angle representation may or may not have a direction sign. Direction signs are "E" and "W" to denote Eastern and Western longitude, "N" and "S" to denote Northern and Southern latitude, respectively. A direction sign may be placed either at the very beginning or at the very end of the angle representation (see the examples below).

Each of the separate floats describing "degrees", "minutes", "seconds" and "radians" is multiplied by an appropriate factor and the resulting products are accumulated. Also, if there is either "W" or "S" in the angle representation, the resulting sum's sign is inverted. If there is no direction sign in the angle representation, such notation is still valid.

Example: `37.87` and `37.87d` are equivalent. Either means 37.87 degrees Eastern longitude or 37.87 degrees Northern latitude.

Example: `37.87W, W37.87, 37.87dW,` and `W37.87d` are all equivalent meaning 37.87 degrees Western longitude.

Example: `27d37m20.45sE` and `27d37m20.45E` are equivalent representations meaning 27 degrees 37 minutes and 20.45 seconds Eastern longitude.

Example: `-0.85r` means either 0.85 radians Western longitude or 0.85 radians Southern latitude.

Example: `27.13d-12.6s34dW` means 27.13 degrees minus 12.6 seconds plus 34 degrees, Western longitude. It is equivalent to 61.13 degrees minus 12.6 seconds, Western longitude; or simply 61.1265 degrees Western longitude.

Almost Fixed Input Format for Angles

This format is modeled after the format used to represent latitudes and longitudes in the NMEA sentences.

This format comprises the following data fields (from left to right):

- letter "x" (in lower case),
- one or more decimal digits before the decimal point,

- optional decimal point ".",
- zero or more decimal digits after the decimal point,
- direction sign ("E" or "W" for longitude, "N" or "S" for latitude).

The following rules are used to extract integer degrees, integer minutes, and fractional minutes from this format:

- Digits after decimal point (if any) denote fractional part of minutes.
- Up to two decimal digits immediately before the decimal point (or immediately before the "E", "W", "N" or "S" signs provided there is no decimal point), denote integer number of minutes.
- The remaining decimal digits (the leftmost ones) denote integer number of degrees.

Example: x12023.234E means 120 degrees and 23.234 minutes Eastern longitude.

Example: x1202E means 12 degrees and 2 minutes Eastern longitude.

4.4 Objects Reference

This section contains the complete list of objects used to control the operation and behavior of the receiver.

4.4.1 Power Management

Reset receiver

Name: /par/reset
Access: w
Type: boolean
Values: yes,no
Default: no

yes - setting this parameters to yes will reset (reboot) the receiver. From a functional point of view, the reset is equivalent to turning the power off and then back on. The value of this parameter will be automatically returned back to no after the reset.

no - setting to no is ignored.

Power Off

Name: /par/power
Access: rw
Type: boolean
Values: on,off
Default: on

on - setting this parameter to on is silently ignored.

off - Setting this parameter to off turns the receiver off. There is no way to turn the receiver on after that using GREIS commands, and after receiver is turned on by other means (e.g., using a power button), this parameter returns back to on.

Sleep Mode

Name: /par/sleep
Access: rw
Type: boolean
Values: on,off
Default: off
Options: {wakeup_time: timespec}

on - put receiver to sleep mode.

off - ignored.

When in sleep mode, receiver could be woken up by one of the methods supported by particular receiver model. For example, some or all receiver serial ports may be able to wake up receiver whenever some character is received. Pushing the power button will wake up receiver from the sleep mode as well.

If wakeup_time option is specified, receiver will be woken up on specified date and time, unless it's woken up earlier by other means.

Example: Put receiver into sleep mode so that it will wake up on Monday (2d) at 23h3m55s GPS time.

⇒ set,/par/sleep,on:2d23h3m55s

Low Power Mode

Name: /par/lpm
Access: rw
Type: boolean
Values: on,off
Default: on

on - enable processor to enter low power mode when idling.

off - disable processor to enter low power mode when idling.

Auto Power Off Threshold

Name: /par/pwr/mvoff
Access: rw
Type: integer [millivolt]
Values: [0…100000]
Default: 0

Turn power off if external voltage (/par/pwr/ext) drops below the threshold specified by this parameter. At default value of 0 never turn power off. Receiver will ensure it saw some external power applied for a few seconds and then it saw power below the threshold for a few seconds before it turns power off.

Power Off Level

Name: /par/pwr/level/off
Access: rw
Type: float [volts]
Values: [0…100]
Default: 0

This parameter specifies voltage level of external power at which receiver will be turned off. 0 - inactive.

When receiver is turned off due to this parameter, it's subject to be switched on due to /par/pwr/level/on parameter.

Power On Level

Name: /par/pwr/level/on
Access: rw
Type: float [volts]
Values: [0…100]
Default: 0

This parameter specifies voltage level of external power at which receiver will be turned on. 0 - inactive.

Even when non-zero, the parameter will be active only when receiver was turned off due to power drop below /par/pwr/level/off parameter.

SIGMA Power Management

SIGMA Power Mode

Name: /par/pwr/mode
Access: rw
Type: enumerated
Values: auto,ext
Default: auto

auto - external power or batteries

ext - external power

SIGMA Current Power Mode

Name: /par/pwr/curmode
Access: r
Type: enumerated
Values: auto,ext

This parameter reflects current mode of operation of SIGMA power management. It will switch to ext when /par/pwr/mode is set to ext and both power-on and power-off levels are set to suitable values.

SIGMA Low Power Mode

Name: /par/pwr/lpm
Access: rw
Type: boolean
Values: on,off
Default: off

on - enable SIGMA reduced power consumption mode

off - disable SIGMA reduce power consumption mode

Note: This mode is supported only for revision 3 and higher of SIGMA power board.

Note: Operation of Bluetooth module is not supported yet (as of April, 2011 and power board rev.4) when this mode is turned on.

SIGMA/DELTA Enable External Power Output

Name: /par/pwr/ic/ext
Access: rw
Type: boolean
Values: on,off
Default: off

on - enable output of power to the external connector

`off` - disable output of power to the external connector

4.4.2 Receiver Information

Receiver Serial Number

Name: `/par/rcv/sn`
Access: `r`
Type: `string[0…31]`

This parameter contains serial number assigned to the receiver on the factory.

Receiver Electronic ID

Name: `/par/rcv/id`
Access: `r`
Type: `string[11]`

This parameter contains a piece of text uniquely identifying your receiver.

Receiver Model

Name: `/par/rcv/model`
Access: `r`
Type: `string`

The model of the receiver, e.g., `DELTA`.

Receiver Vendor

Name: `/par/rcv/vendor`
Access: `r`
Type: `enumerated`
Values: `JAVAD,UNKNOWN`

Receiver IGS Name

Name: `/par/rcv/igs`
Access: `r`
Type: `string[0…20]`

Receiver Up-time

Name: `/par/rcv/uptime`
Access: `r`
Type: `timespec`

Time elapsed since last receiver reboot.

Receiver RAM Size

Name: /par/rcv/mem
Access: r
Type: integer [kilobytes]

Receiver Configuration Word

Name: /par/rcv/cfgw
Access: r
Type: integer

The receiver configuration word formatted as hexadecimal, or empty if not available.

4.4.3 Version Information

Hardware Version

Name: /par/rcv/ver/hw
Access: r
Type: integer

Boot-loader Version

Name: /par/rcv/ver/boot
Access: r
Type: integer

Firmware Version

Name: /par/rcv/ver/main
Access: r
Type: string

Board Version

Name: /par/rcv/ver/board
Access: r
Type: integer

Power Board Hardware Version

Name: /par/rcv/ver/pow/hw
Access: r
Type: integer

Internal Modem Board Version

Name: /par/rcv/ver/modem
Access: r
Type: string

If internal modem is not supported, the value of this parameter will be none.

Modem Board Hardware Version

Name: /par/rcv/ver/mdm/hw
Access: r
Type: integer

Modem Board Type

Name: /par/rcv/ver/mdm/type
Access: r
Type: integer

4.4.4 Locking on Satellite Signals

Notation

A lot of parameters in this section are similar for each GNSS system. We use the following notation in the descriptions below:

SYS – stands for: gps, glo, sbas, glcdma, gal, bei, qzss, irnss

[a…b] – stands for the following SVs numbers ranges, depending on particular SYS:

SYS	Maximum [a…b] range
gps	[1…32]
glo	[-7…7] GLONASS FCN
sbas	[120…147] SBAS; [183…189] L1S(b)
glcdma	[1…30]
gal	[1…36]
bei	[1…37]
qzss	[193…199]
irnss	[1…14]

N – the number of particular satellite in the range [a…b]

SIG – stands for the following signal names, depending on particular SYS:

SYS	SIG
gps	ca, p1, p2, l2c, l5, l1c
glo	ca, p1, p2, l2c
sbas	l1, l5
glcdma	l1, l2, l3
gal	e1, aboc, e5b, e6, e5a
bei	b1, aboc, b2, b3, b2a, b1c
qzss	ca, l6, l2c, l5, l1c
irnss	s, l5, l1

Overview

Parameters described in this section allow to tune locking on and subsequent tracking of GNSS signals in multiple interesting ways.

Probably most important is enabling/disabling locking on signals, at different levels. These levels, progressively more specific, along with the names of corresponding parameters, are:

#	Level	Parameter(s)
1	global	/par/lock/mode
2	system	/par/lock/sys/SYS
3	satellite	/par/lock/sat/SYS/N
	signal	/par/lock/sig/SYS/SIG/N

For particular GNSS signal to be tracked, its locking must be enabled at all these levels. Disabling of locking to a signal at any of these levels disables tracking of the signal. If signal is being tracked at the time of disabling, tracking of the signal will be dropped.

For example, setting /par/lock/sys/gal to 'n' will disable tracking of all signals of all Galileo satellites no matter what the rest of parameters say.

Even when signal locking is enabled at all the levels, receiver may decide not to track particular signal due to information about lack of given signal on given satellite that receiver gets from the constellation. When and if this information appears to be wrong, it's possible to force receiver to ignore it using corresponding /par/lock/adv/force/SYS/SIG/N parameter, to make receiver try to lock to such signal.

For the cases where GNSS signal has complex structure, the parameters /par/lock/adv/data/SYS/SIG/N and /par/lock/adv/pilot/SYS/SIG/N govern the kind of the sub-signal that receiver must track. When both of them are 'n', the signal is to be tracked as a whole complex signal. When only one of them is set to 'y' for given signal, those sub-signal will be subject to track. When both parameters are set to 'y' for given signal, the "data" sub-signal wins.

Guided vs independent tracking could be configured using parameters /par/lock/adv/guide/SYS/SIG/N. For signals that could be tracked only in guided or only in independent mode receiver will ignore these settings.

There are a more parameters in this group that specify other, usually particular GNSS specific, aspects of tracking of satellite signals, as well as some tracking limits.

Here are a few examples that demonstrate using of the locking parameters:

Example: turn off tracking of GPS SVs #14

⇒ set,/par/lock/sat/gps/14,n

Example: turn off tracking of all GPS SVs but SVs #8

⇒ set,/par/lock/sat/gps,n
⇒ set,/par/lock/sat/gps/8,y

Example: turn off Galileo E6 signal tracking for all SVs

⇒ set,/par/lock/sig/gal/e6,n

Generic Locking Parameters

Enable Tracking of Satellite Signals

Name: /par/lock/mode
Access: rw
Type: enumerated
Values: on,off
Default: on

on - turn on SVs tracking

off - turn off SVs tracking

Enable Tracking by Satellite System

Name: /par/lock/sys
Access: rw
Type: list {SYS} of boolean
Values: {y|n,…,y|n}
Default: <receiver-dependent>

This parameter allows to select satellite constellation(s) enabled for tracking. The fields correspond to each of the SYS, respectively.

Enable Tracking of Given Satellite System

Name: /par/lock/sys/SYS
Access: rw
Type: boolean
Values: y,n
Default: <SYS-dependent>

y - enable tracking of SYS

n - disable tracking of SYS

Enable GNSS Satellites Tracking by Their Numbers

Name: /par/lock/sat/SYS
Access: rw
Type: array [a…b] of boolean
Values: {y|n,…,y|n}
Default: <SYS-dependent> usually {y,…,y}

Enable Tracking of GNSS Satellite Number N

Name: /par/lock/sat/SYS/N (N=[a…b])
Access: rw
Type: boolean
Values: y,n
Default: <SYS-dependent> usually y

y - enable tracking of GNSS satellite number N

n - disable tracking of GNSS satellite number N

Enable Tracking of Particular Signal

Name: /par/lock/sig/SYS/SIG
Access: rw
Type: array [a…b] of boolean
Values: {y|n,…,y|n}
Default: <SYS-dependent> usually {y,…,y}

Enable SIG tracking for each GNSS Satellite of SYS

Enable Tracking of Particular Signal for Satellite Number N

Name: /par/lock/sig/SYS/SIG/N
Access: rw
Type: boolean
Values: y,n
Default: <SYS-dependent> usually y

Enable SIG tracking for GNSS Satellite N of SYS

Advanced Tracking Parameters

Guide Tracking Of Particular Signal

Name: /par/lock/adv/guide/SYS/SIG
Access: rw
Type: array [a…b] of boolean
Values: {y|n,…,y|n}
Default: <SYS-dependent> usually {n,…,n}

Guide SIG tracking for each GNSS Satellite of SYS

Guide Tracking Of Particular Signal for Satellite Number N

Name: /par/lock/adv/guide/SYS/SIG/N
Access: rw
Type: boolean
Values: y,n
Default: <SYS-dependent> usually y

y – guide SIG tracking for GNSS satellite N of SYS, i.e., turn on guided tracking mode.

n – don't guide SIG tracking for GNSS satellite N of SYS.

Force Tracking of Particular Signal

Name: /par/lock/adv/force/SYS/SIG
Access: rw
Type: array [a…b] of boolean
Values: {n|y,…,n|y}
Default: {n,…,n}

Force SIG tracking for each GNSS satellite of SYS.

Force Tracking of Particular Signal for Satellite Number N

Name: /par/lock/adv/force/SYS/SIG/N
Access: rw
Type: boolean
Values: n,y
Default: n

y - force SIG tracking for GNSS Satellite N of SYS, i.e. ignore information from SYS that signal does not exist.

n - don't force SIG tracking for GNSS Satellite N of SYS, – use SYS-provided information to decide to track the signal

Track Only "data" Sub-signal of Given Signal

Name: /par/lock/adv/data/SYS/SIG
Access: rw
Type: array [a…b] of boolean
Values: {n|y,…,n|y}
Default: SYS/SIG dependent, usually {n,…,n}

Track only "data" sub-signal of SIG for each GNSS Satellite of SYS

Track Only "data" Sub-signal of Given Signal for Satellite Number N

Name: /par/lock/adv/data/SYS/SIG/N
Access: rw
Type: boolean
Values: n,y
Default: SYS/SIG dependent, usually n

y - track only "data" sub-signal of SIG for GNSS Satellite N of SYS

n - don't track "data" sub-signal of SIG for GNSS Satellite N of SYS

Track Only "pilot" Sub-signal of Given Signal

Name: /par/lock/adv/pilot/SYS/SIG
Access: rw
Type: array [a…b] of boolean
Values: {n|y,…,n|y}
Default: {n,…,n}

Track only "pilot" sub-signal of SIG for each GNSS Satellite of SYS

Track Only "pilot" Sub-signal of Given Signal for Satellite Number N

Name: /par/lock/adv/pilot/SYS/SIG/N
Access: rw
Type: boolean
Values: n,y
Default: n

y - track only "pilot" sub-signal of SIG for GNSS Satellite N of SYS, unless /par/lock/adv/data/SYS/SIG/N is set to 'y', in which case track the "data" sub-signal.

n - don't track "pilot" sub-signal of SIG for GNSS Satellite N of SYS

Upper Limit for Maximum Number of Simultaneously Tracked Signals

Name: /par/lock/adv/max_sig
Access: rw
Type: integer
Values: [1…MAX], where MAX depends on receiver type
Default: MAX (the value of /par/lock/adv/max_sig&max)

Limit maximum number of simultaneously tracked signals by this value. Lower numbers cause lower CPU load.

Upper Limit Maximum Value

Name: /par/lock/adv/max_sig&max
Access: r
Type: integer

The maximum value for /par/lock/adv/max_sig parameter, i.e., the value of the MAX in the description of /par/lock/adv/max_sig.

Enable Dual Tracking

Warning: *this parameter is for internal use and for testing, – never set it in production!*

Name: /par/lock/adv/dual
Access: rw
Type: boolean
Values: n,y
Default: n

n - guided' bit being set for a signal disables independent tracking of the signal

y - guided' bit being set for a signal does not disable independent tracking of the same signal

Current Locking Parameters

These read-only parameters show the current state of firmware variables that actually guide tracking and that are calculated from the user parameters already described.

Warning: *these parameters are for internal and troubleshooting purposes only and are subject to change without notice.*

Current Tracking Mode

Name: `/par/lock/cur/mode`
Access: `r`
Type: `enumerated`
Values: `on,off`

The current tracking mode. Should always match `/par/lock/mode`.

Current Maximum Number of Simultaneously Tracked Signals

Name: `/par/lock/cur/max_sig`
Access: `r`
Type: `integer`
Values: `[1…MAX], where MAX is the value of /par/lock/adv/max_sig`

The current maximum number of channels receiver will use for SVs tracking.

Current Independent Tracking of Particular Signal for Satellite Number N

Name: `/par/lock/cur/indep/SYS/SIG/N`
Access: `r`
Type: `boolean`
Values: `y,n`

`y` – independent tracking for given signal is enabled

`n` – independent tracking for given signal is disabled

Current Guided Tracking of Particular Signal for Satellite Number N

Name: `/par/lock/cur/guide/SYS/SIG/N`
Access: `r`
Type: `boolean`
Values: `y,n`

`y` – guided tracking for given signal is enabled

`n` – guided tracking for given signal is disabled

Current Force Tracking of Particular Signal for Satellite Number N

Name: /par/lock/cur/force/SYS/SIG/N
Access: r
Type: boolean
Values: n,y

Always matches /par/lock/adv/force/SYS/SIG/N

Current Track "data" Sub-signal

Name: /par/lock/cur/data/SYS/SIG/N
Access: r
Type: boolean
Values: n,y

Current Track "pilot" Sub-signal

Name: /par/lock/cur/pilot/SYS/SIG/N
Access: r
Type: boolean
Values: n,y

GPS Specific Parameters

Enable Tracking of GPS L1C TMBOC Sub-signal by PRN

Name: /par/lock/gps/tmboc
Access: rw
Type: array [1…32] of boolean
Values: {y|n,…,y|n}
Default: <firmware dependent>

Enables receiver to track L1C TMBOC sub-signal of GPS satellites by their PRN.

Enable Tracking of GPS L1C TMBOC Sub-signal for PRN #N

Name: /par/lock/gps/tmboc/N (N=[1…32])
Access: rw
Type: boolean
Values: y,n
Default: <firmware dependent>

y - enable L1C TMBOC sub-signal tracking of GPS satellite #N

n - disable L1C TMBOC sub-signal tracking of GPS satellite #N

Galileo Specific Parameters

Enable Tracking Of Galileo CBOC by PRN

Name: /par/lock/gal/cboc
Access: rw
Type: array [1…30] of boolean
Values: {y|n,…,y|n}
Default: firmware dependent

Enables/disables CBOC tracking of Galileo satellites by their PRN.

Enable Tracking Of Galileo CBOC for PRN N

Name: /par/lock/gal/cboc/N (N=[1…30])
Access: rw
Type: boolean
Values: y,n
Default: firmware dependent

y - enable CBOC tracking of Galileo satellite number N

n - disable CBOC tracking of Galileo satellite number N

QZSS Specific Parameters

Enable Tracking of QZSS L1C TMBOC Sub-signal by PRN

Name: /par/lock/qzss/tmboc
Access: rw
Type: array [193…199] of boolean
Values: {y|n,…,y|n}
Default: <firmware dependent>

Enables receiver to track L1C TMBOC sub-signal of QZSS satellites by their PRN.

Enable Tracking of QZSS L1C TMBOC Sub-signal for PRN #N

Name: /par/lock/qzss/tmboc/N (N=[193…199])
Access: rw
Type: boolean
Values: y,n
Default: <firmware dependent>

y - enable L1C TMBOC sub-signal tracking of QZSS satellite #N

n - disable L1C TMBOC sub-signal tracking of QZSS satellite #N

BeiDou Specific Parameters

Enable Tracking of BeiDou B1C TMBOC Sub-signals by PRN

Name: /par/lock/bei/tmboc
Access: rw
Type: array [1…37] of boolean
Values: {y|n,…,y|n}
Default: <firmware dependent>

Enables receiver to track B1C TMBOC sub-signals of BeiDou satellites by their PRN.

Enable Tracking of BeiDou B1C TMBOC Sub-signal for PRN #N

Name: /par/lock/bei/tmboc/N (N=[1…37])
Access: rw
Type: boolean
Values: y,n
Default: firmware dependent

y – enable B1C TMBOC sub-signals tracking of BeiDou satellite #N

n – disable B1C TMBOC sub-signals tracking of BeiDou satellite #N

BeiDou Phase3 Satellite Generation by PRN

Name: /par/lock/bei/phase3
Access: rw
Type: array [1…37] of boolean
Values: {y|n,…,y|n}
Default: <firmware dependent>

Specifies which BeiDou satellites are phase3 satellites by their PRN.

BeiDou Phase3 Satellite Generationfor PRN #N

Name: /par/lock/bei/phase3/N (N=[1…37])
Access: rw
Type: boolean
Values: y,n
Default: n for N=[1…18], y for the rest

y – handle BeiDou satellite #N as phase3 generation satellite

n – handle BeiDou satellite #N as phase2 generation satellite

Locking Limits

Elevation Mask for SVs Locking

Name: /par/lock/elm
Access: rw
Type: integer [degrees]
Values: [-90…90]
Default: -1 or 5 (receiver dependent)

Receiver will not lock to satellites below this elevation mask.

Sector Mask for SVs Locking

Name: /par/lock/elazm/N N=[1,2,3]
Access: rw
Type: list {el1,el2,az1,az2} of float [degrees]
Values: {[-90…90],[-90…90],[0…360],[0…360]}
Default: {0,0,0,0}

Receiver will not lock to a satellite if its elevation is between el1 and el2 and at the same time its azimuth is between az1 and az2, taking into account zero crossing.

Maximum Velocity

Name: /par/lock/vmax
Access: rw
Type: integer [m/s]
Values: [1000…10000]
Default: 1000

This parameter specifies maximum possible velocity of the receiver antenna for the purpose of computation of the required SVs search zone.

Maximum Acceleration

Name: /par/lock/amax
Access: rw
Type: integer [m/s/s]
Values: [10…100]
Default: 20

This parameter specifies maximum possible acceleration of the receiver antenna for the purpose of computation of the required SVs search zone.

Antennae Tracking Mask

Name: /par/lock/ant
Access: rw
Type: array [a,b,c,d] of boolean
Values: {y|n,y|n,y|n,y|n}
Default: {y,y,y,y}

Each element of the array enables tracking of SVs on corresponding antenna.

Note: This parameter is only available for multi-antenna receivers.

Antenna N Tracking

Name: /par/lock/ant/N (N=[a,b,c,d])
Access: rw
Type: boolean
Values: y,n
Default: y

This parameter enables tracking of SVs on antenna N.

Note: This parameter is only available for multi-antenna receivers.

Misc Locking Parameters

Desired L1/E1/B1 Center Frequency.

Name: /par/l1center/frq
Access: rw
Type: integer [MHz]
Values: [1561…1575]
Default: 1575

Use this parameter to let receiver track BeiDou B1 signal. The value of this parameter will be made current center frequency after next receiver reboot and will be reflected then in the /par/l1center/curfrq parameter.

Warning: *This parameter is only necessary on JAVAD GNSS receivers that were designed before BeiDou system was launched.*

Note: You will also need to turn on BeiDou SVs tracking, see /par/lock/comp/sat parameter.

Meaningful values:

1575 – This is default. BeiDou B1 reception is impossible. Best GPS/Galileo/SBAS/QZSS L1/E1 performance. Best multipath reduction in mpnew mode.

1573 – BeiDou B1 reception is possible, but it's data quality is poor. Good GPS/Galileo/SBAS/QZSS L1/E1 performance. Good GPS/Galileo/SBAS/QZSS multipath

reduction in `mpnew` mode. Recommended for low-precision BeiDou B1 applications and/or for collecting BeiDou navigation data.

`1563` – BeiDou B1 reception is good, it's data quality is good. Good BeiDou multipath reduction using "mpnew" setting. Poor GPS/Galileo/SBAS/QZSS L1/E1 performance. Recommended for high-precision BeiDou B1 applications.

Warning: Setting this parameter to `1561` is not recommended, because receiver won't be able to track GPSs.

Current L1/E1/B1 Center Frequency.

Name: `/par/l1center/curfrq`
Access: `r`
Type: `integer [MHz]`
Values: `[1561… 1575]`
Default: `1575`

Current L1/E1/B1 center frequency. To change this parameter, set `/par/l1center/frq` parameter to desired value and reboot receiver.

Digital Filters Bandwidths.

Name: `/par/filt/band`
Access: `rw`
Type: `list {[BAND]}`

The list of all the digital filters bands supported by receiver. The `[BAND]` is receiver dependent and is a subset of:

```
glo1, gps1, beiB1, galE6, beiB3, glo2, gps2, galE5B,
glo3, galE5, galE5, gps5
```

Particular Digital Filter Bandwidth.

Warning: *WARNING: do not change these parameter unless you fully understand the consequences*

Name: `/par/filt/band/BAND`
Access: `rw`
Type: `integer [MHz]`
Values: `[1…MAX] (receiver dependent)`
Default: `(receiver dependent)`

4.4.5 Measurements Parameters

Generic Measurements Parameters

Measurements Update Rate

Name: /par/raw/msint
Access: rw
Type: integer [milliseconds]
Values: [MIN…1000], multiple of MIN
Default: 100

MIN – either 5 or 10, depending on receiver type

This parameter specifies the required period of the internal receiver time grid. Receiver will calculate effective period of the time grid using the value of this parameter and values of relevant receiver options (see /par/raw/curmsint below). In turn, this time grid defines the rate of receiver generating pseudo-ranges, carrier phases and other measurements, as well as defines the base time grid for periodic messages output.

Effective Measurements Update Rate

Name: /par/raw/curmsint
Access: r
Type: integer [milliseconds]

Although the user can formally set /par/raw/msint to arbitrary allowed value, the receiver may need to adjust this user setting in order to make it consistent with the receiver options. The adjusted setting is stored to this read-only parameter and defines effective internal time grid.

The formula used to calculate curmsint is as follows:

$$\mathtt{curmsint} = max\,(\mathtt{msint}, \mathtt{1000} / max\,(\mathtt{1, \_RAW, \_POS, PDIF}))$$

where _RAW, _POS, and PDIF are current values of corresponding receiver options, and msint is the value of /par/raw/msint parameter.

The actual period at which receiver will allow user to get measurements depends on the value of /par/raw/curmsint parameter and the current value of receiver _RAW option. Actual measurements update period is calculated as follows:

$$\mathtt{meas\_period} = max\,(1, \lfloor \mathtt{1000} / \mathtt{\_RAW} / \mathtt{curmsint} \rfloor) \times \mathtt{curmsint}$$

where _RAW is the current value of corresponding receiver option, and $\lfloor x \rfloor$ denotes integer part of x.

In addition, this parameter along with the PDIF receiver option specify the update rate of carrier phase-differential (RTK) position that is computed as follows:

$$\text{cpd\_period} = max\,(1,\ \lvert 1000\ /\text{PDIF}\ /\ \text{curmsint}\ \rvert) \times \text{curmsint}$$

where PDIF is the current value of corresponding receiver option.

The value of this parameter also indirectly affects update rate of stand-alone and code-differential position, refer to description of /par/pos/msint parameter for details.

Note: While the formulas seem rather complex, what they basically mean in practice, is that if you set /par/raw/msint to a value that is multiple of 1000/_RAW, then both the /par/raw/curmsint and actual allowed measurements output period will be equal to the specified value.

Pseudo-range Smoothing Interval

Name: /par/raw/smi
Access: rw
Type: integer [seconds]
Values: [0…900]
Default: 100

0 – receiver will not use carrier phases to smooth pseudo-ranges. Therefore, in this case the pseudo-range noise error will depend only on the corresponding CLL bandwidth (see the parameter /par/raw/cll/band on page 223).

[1…900] – receiver will smooth pseudo-ranges based on a Kalman filter whose time constant is set to the value of this parameter.

Doppler Smoothing Bandwidth

Name: /par/raw/dopp/smi
Access: rw
Type: float [Hz]
Values: [0.1…50]
Default: 3

For example, when the parameter is 10Hz, i.e. smoothing interval is 0.1 seconds, the velocity will be rather noisy. On the other hand, when the parameter is 1Hz (smoothing interval is 1 second), velocity will be very smooth, but problems with latency may appear.

Doppler Smart Smoothing Mode

Name: /par/raw/dopp/smsm
Access: rw
Type: boolean
Values: on,off
Default: off

on – Smart algorithm of doppler smoothing is turned on. Doppler is less noisy, but sometimes loop settling can be seen. When in this mode, it is highly recommended to have CA/L1 PLL order set to 3 (the default) and increase doppler smoothing bandwidth.

off – Regular doppler smoothing algorithm is turned on.

Ionosphere Corrections Smoothing Interval

Name: /par/raw/iono/smi
Access: rw
Type: integer [seconds]
Values: [0…900]
Default: 60

This parameter specifies the nominal interval (T_{nom}) over which raw ionospheric corrections are smoothed (assuming the receiver has been working for some time and has already obtained enough raw ionospheric corrections to perform such smoothing).

Note that the current ionosphere smoothing interval will vary in time. After you switch receiver on, the current smoothing interval will be growing from zero to the nominal value as new raw ionospheric corrections are computed. Once the nominal value is reached, this smoothing interval will be fixed. Smoothing filter is a simple n-point running average.

Minimum Ionosphere Corrections Smoothing Interval

Name: /par/raw/iono/minsmi
Access: rw
Type: integer [seconds]
Values: [0…900]
Default: 30

The parameter specifies the minimum smoothing interval (T_{min}) for the receiver to filter raw ionospheric corrections before they can be used in position computation.

Raw Inphases/Quadratures Smoothing Interval

Name: /par/raw/iqsmi
Access: rw
Type: integer[5ms]
Values: [1…200]
Default: 1

Specifies raw Inphases/Quadratures smoothing interval in 5 millisecond units. This smoothing affects nothing else but values output in corresponding messages. When smoothing interval defined by this parameter is greater than time interval defined by /par/raw/curmsint parameter, the I and Q are smoothed over the latter time interval.

Enable Q Output in [IQ] Message

Name: /par/raw/out/q
Access: rw
Type: boolean
Values: off,on
Default: off

Satellite Signals for [IQ] Message

Name: /par/raw/out/SYS/SIG/N
Access: rw
Type: boolean
Values: y,n
Default: y

Refer to /par/lock/sig description for possible values of SYS, SIG, and N.

Warning: *1000Hz data output causes extremely high data traffic! Do not output more data than can be transmitted through selected connection.*

Satellite Signals for [ad] Message

Name: /par/raw/fastdata/SYS/SIG
Access: rw
Type: boolean
Values: y,n
Default: n

Enable corresponding signal in [ad] message. Refer to /par/lock/sig description for possible values of SYS, SIG.

Example: Output GPS C/A, GPS L5 and Galileo E5B raw navigation data along with time-tag to the current port (you may wish to skip "RT" message if no time-tags are needed):

```
⇒ set,/par/raw/fastdata/gps/ca,y
⇒ set,/par/raw/fastdata/gps/l5,y
⇒ set,/par/raw/fastdata/gal/e5b,y
⇒ em,/cur/term,/msg/jps/{RT,ad}:0
```

This will output raw navigation data in [ad] message with /par/raw/curmsint period (each 100 milliseconds by default), that you can change to a lower value, say 10 milliseconds, to minimize latency further (provided 100 Hz raw data update option is purchased):

```
⇒ set,/par/raw/msint,10
```

Please be warned that reducing latency increases messages overhead.

Alternatively, if it's fine to get data less often, specify longer period of [ad] message output, say, 0.5 seconds:

```
⇒ em,/cur/term,/msg/jps/{RT,ad}:0.5
```

Use Signal Propagation Delay

Name: /par/raw/rfdel
Access: rw
Type: boolean
Values: on,off
Default: on

on - to compute time moment for taking of measurements, receiver will take into account signal propagation delay due to receiver RF part schematic. Propagation delay is hard-coded number (for given receiver) which approximates actual delay with about +-10ns accuracy

off - receiver will assume zero propagation delay

Measurements Mode

Name: /par/raw/meas/mode
Access: rw
Type: enumerated
Values: normal,modulo
Default: normal

Warning: *set this parameter to* modulo *for investigation purposes only.*

`modulo` - receiver will start to output measurements (phase, pseudo-range, doppler, C/No) even if preamble is not found yet and correct full pseudorange is not available.

`normal` - regular behavior

Millisecond Mode for Independent Loops

Name: `/par/raw/msecmode`
Access: `rw`
Type: `enumerated`
Values: `off,iq,loops`
Default: `off`

`off` - default loops closing and IQ collection period (4 or 5 milliseconds).

`iq` - collect I and Q samples with one millisecond period. Close loops with default period.

`loops` - collect I and Q samples and close loops with default period.

Note: setting this parameter to `loops` may be useful for high dynamic applications because PLL bandwidth can be meaningfully raised up to 200Hz with this setting.

Warning: *firmware restart required to activate change of this parameter.*

Warning: *receiver may be unable to track all satellites and all signals with "loops" setting due to high CPU demands. Turn off tracking of unneeded signals for best results.*

Millisecond Mode for Guided Loops

Name: `/par/raw/msecmode_g`
Access: `rw`
Type: `enumerated`
Values: `off,iq,loops`
Default: `off`

`off` - default loops closing and IQ collection period (4 or 5 milliseconds).

`iq` - collect I and Q samples with one millisecond period. Close loops with default period.

`loops` - collect I and Q samples and close loops with default period.

Warning: *firmware restart required to activate change of this parameter.*

Warning: *receiver may be unable to track all satellites and all signals with "loops" setting due to high CPU demands. Turn off tracking of unneeded signals for best results.*

Measurements to RINEX Conversion Parameters

Name: /par/raw/rinex/SYS/SIG
Access: r
Type: {rinex_id,phase_offset}
Values:

where:

SYS - one of (gps,glo,gal,sbas,qzss,irnss)

SIG - one of (ca,p1,p2,l2c,l5,l1c)[1]

rinex_id - RINEX identifier of signal. Type: string[2]

phase_offset - phase offset which should be added to phase measurements provided by receiver to get RINEX phase measurements. Type: float, [cycles].

QZSS Signal Tracking

Parameters described in this section allow to tune QZSS-specific signals tracking. Receiver restart is required for changes to these parameters to take effect properly.

Example: Suppose QZSS satellite number with ESI SVID 196 transmits L1 CB signal using PRN #203, and the signal has no navigation data. To track such signal, the following settings are to be used:

⇒ set,/par/qzss/cabprn/196,203
⇒ set,/par/qzss/cb/196,y
⇒ set,/par/raw/meas/mode,modulo
⇒ set,reset,y

Here the /par/raw/meas/mode parameter being set to modulo allows to track signals without correct navigation data structure.

Note: It appears that if satellite transmits C/A signal, but receiver is set to receive C/B signal, receiver sometimes is still able to receive it somehow (with lower SNR), because during 1/2 of time prn bits sequence of C/B signal is the same, as for C/A signal.

QZSS L1 CA/CB PRN

Name: /par/qzss/cabprn
Access: rw
Type: array [193…199] of integers [CA/CB PRN]
Values: {[193…210],…}
Default: {193,194,195,196,197,198,199}

1. ca, p1, p2, l2c,l 5, l1c are historical signal names that came from GPS.

Name: /par/qzss/cabprn/SVID (SVID=[193…199])
Access: rw
Type: integer
Values: [193…210]
Default: <SVID>

This correspondence allows to select L1 CA/CB PRN number which differs from satellite's ESI SVID number.

Enable QZSS C/B signal

Name: /par/qzss/cb
Access: rw
Type: array [193…199] of boolean
Values: {y|n,…,y|n}
Default: {n,n,n,n,n,n,n}

Name: /par/qzss/cb/N (N=[193…199])
Access: rw
Type: boolean
Values: y,n
Default: n

n - trakc L1 C/A signal

y - track L1 C/B signal instead of C/A

Correspondence Between PRNs of QZSS L5S and L1S Signals

Name: /par/qzss/l5sprn
Access: rw
Type: array [183…189] of integers
Values: {X,X,…,X}, where X is L5S PRN; X=[183…210]
Default: {183,196,200,186,187,188,197}

Name: /par/qzss/l5sprn/L1S_PRN, (L1S_PRN=[183…189])
Access: rw
Type: integer
Values: [183…210]
Default: <satellite dependent> (see above)

This correspondence allows to represent L1S and L5S signals of QZSS satellite by the same ESI.

Spoofing Detection Parameters

Receiver detects possible spoofing by analyzing the number of correlation peaks. In normal conditions only one peak is present. In case of spoofing, there often are 2 peaks - the original one and another one from the spoofer. If receiver detects 2 peaks it marks signal as "spoofed" and optionally does not use it in positioning (in case `/par/raw/spoof/pos` is set to `off`.)

If number of spoofed satellites appeared to be more than one at given signal, that signal is flagged as "spoofed" for the rest of satellites as well.

Jamming detection is based on analysis of incoming noise. In case of jamming the "noise floor" gets higher than usual. Also, deviation of the noise may increase. Receiver measures these parameters and once at least one of them ends up being above the threshold (it's value is higher than 100 in the `[sj]` message), that signal for all satellites is flagged as "jammed".

Note: for spoofing/jamming detection, signal should be tracked in independent (not guided) mode, so for checking of all possible signals it is required to switch tracking to independent mode, e.g., using `set,/par/lock/adv/guide,n` command.

Spoofing Detection Mode

Name: `/par/raw/spoof/mode`
Access: `rw`
Type: `boolean`
Values: `off,on`
Default: `off`

`on` - turn on spoofing/jamming detection

`off` - turn off spoofing/jamming detection

Use Spoofed Signals in Position

Name: `/par/raw/spoof/pos`
Access: `rw`
Type: `boolean`
Values: `off,on`
Default: `off`

`off` - do not use measurements from satellite signals detected as spoofed or jammed in position computation

`on` – ignore spoofed and jammed status of signals in position computation, so that such signals could still be used

S4 and sigmaPhi Parameters

Historically S4 and sigmaPhi have been computed based on GPS C/A code only. Now they could be computed from any/all sources.

Example: In this example all signals are used. PLL loop bandwidth is 25 Hz (it is default). Receiver computes S4 and sigmaPhi only for signals tracked independently, so tracking is set to independent mode:

```
⇒ %%set,/par/lock/adv/guide,n
⇒ %%set,/par/raw/msecmode,iq
⇒ %%set,/par/raw/sigmaphi/per,30
⇒ %%set,/par/raw/s4/per,60
⇒ %%em,,jps/{x0,x1,x2,x3,x5,x4,y0,y1,y2,y3,y3,y5,y4}
⇒ %%set,reset,y
```

Period of S4 Computation

Name: `/par/raw/s4/per`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…100]`
Default: `0`

`0` – turn off S4 computation

Period of sigmaPhi Computation

Name: `/par/raw/sigmaphi/per`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…100]`
Default: `0`

`0` – turn off sigmaPhi computation

Receiver Time Parameters

Receiver time is the only time grid that is always available in your receiver, and most of receiver operations, such as periodic messages output and computation of measurements, are based on this time scale. In turn, receiver tries its best to synchronize the

receiver time to one of the GNSS system times. In this section the parameters governing the receiver time are described.

Receiver Reference Time

Name: /par/raw/time/ref
Access: rw
Type: enumerated
Values: gps,glo
Default: gps

This parameter specifies particular system time scale to which receiver time will be synchronized.

- gps - receiver time will be synchronized to the GPS system time scale. When GPS time scale is not available, its best approximation based on other available time scales will be used.
- glo - receiver time will be synchronized to the GLONASS system time scale. When GLONASS time scale is not available, its best approximation based on other available time scales will be used.

Note: For GLONASS-only operation it is recommended to set this parameter to glo.

Receiver Time Synchronization Mode

Name: /par/raw/time/sync
Access: rw
Type: enumerated
Values: ms,ms0,steady
Default: ms

This parameter specifies particular algorithm of synchronization of receiver time to receiver reference time.

- ms - when time difference exceeds 0.5 milliseconds, correct receiver time by 1 millisecond.
- ms0 - correct time once after startup to be as close as possible to receiver reference time. Then, when time difference exceeds 0.5 milliseconds, correct receiver time by this time difference.
- steady - instantly synchronize receiver time to receiver reference time. This mode is also called *time steering*.

Correlator Parameters

Code Correlator Shape

Name: /par/raw/corr/ca/code
Access: rw
Type: enumerated
Values: normal,mpear,mpnew,mp2ne,mpxne,data
Default: mpnew

normal - code multipath reduction is off.

mpear - - medium multipath reduction. Symmetric multipath strobe is used.

mpnew - recommended for multipath reduction; best trade-off between multipath reduction and possible side effects, like noise increase.

mp2ne - even stronger than "mppnew" multipath reduction. Noisier measurements. Some shift in range measurements is possible.

mpxne - even stronger than "mp2ne" multipath reduction. Yet noisier measurements. Some shift in range measurements is possible

data - special setting for separate tracking of “pilot” and “data” sub-signals of complex signals. In this mode, receiver will store “data” range measurements in conventional messages, deltas between “pilot” and “data” measurements in “Code Corrections”. For phase corrections, receiver stores “pilot” phase measurements in conventional messages, deltas between “data” and “pilot” measurements in “Phase Corrections” messages.

Warning: *"mp2ne" and "mpxne" are recommended for research/investigation purposes only. Do not use them in production setups.*

Note: if this parameter is set to data, parameter /par/raw/corr/ca/carrier should be set to normal to have correct phases in "Phase Corrections" messages ([cf], etc.) messages.

Code Correlator Width

Warning: *we advice to leave this parameter at its default value if you use any correlator but normal, as set by the /par/raw/corr/ca/code parameter.*

Name: /par/raw/corr/ca/width
Access: rw
Type: float [fraction of C/A code chip width]
Values: [0.02…1]
Default: 0.02

Width (spacing) of conventional correlators. Despite its name, this parameter changes spacing for all correlators, not only for C/A ones. Actual correlator width for a signal is computed according to the formula:

$$\text{width} = \min(\text{width\_ca} \times \text{chip\_rate} / \text{ca\_chip\_rate}, 1)$$

where:

`width_ca` - the value of this parameter

`chip_rate` - chip rate of corresponding signal

`chip_rate_ca` - chip rate of GPS C/A code that is equal to 1023 [1/ms]

For example, if this parameter is set to `0.05`, GPS C/A code correlator width will be set to `0.05`, and GPS P-code one will be set to `0.5` (i.e. `0.5` of P-code chip width), as GPS P-code chip rate is 10230 [1/ms]:

$$0.05 \times 10230/1023 = 0.5$$

Carrier Correlator Shape

Name: `/par/raw/corr/ca/carrier`
Access: `rw`
Type: `enumerated`
Values: `normal,mpnew`
Default: `mpnew`

`normal` - carrier phase multipath reduction is off.

`mpnew` - carrier phase multipath reduction is on.

Tracking Loops Parameters

Warning: *It is not recommended to change tracking loop parameters from their default values. Only a few special applications would require different settings.*

There are three kinds of tracking loops in JAVAD GNSS receivers:

1. Tracking loops for CA/L1.
2. Tracking loops for strong signals, or simply *strong loops*. These include GPS L2C and L5, GLONASS P/L1 and P/L2, SBAS L5, and Galileo L5 signals.
3. Tracking loops for weak signals, or simply *weak loops*. These include GPS P/L1 and P/L2 signals.

Each kind of loops has its own set of parameters.

In this section, the PLL and CLL abbreviations stand for Phase Lock Loop and Code Lock Loop, respectively. Note that frequently used abbreviation DLL (Delay Lock Loop) is synonym for CLL.

CA/L1 PLL Bandwidth

Name: /par/raw/pll/band
Access: rw
Type: float [Hz]
Values: [0.1…200]
Default: 25

Warning: *If the value of this parameter is set outside of [10…50] interval, satellite tracking may become unstable.*

CA/L1 PLL Order

Name: /par/raw/pll/order
Access: rw
Type: integer
Values: 2,3
Default: 3

Care should be taken when changing this parameter from the default value to "2". We don't recommend you to use a 2nd order PLL unless you are certain that the receiver's trajectory is far from being a uniformly accelerated motion, and therefore using a lower-order PLL might make a difference.

CA/L1 CLL Bandwidth

Name: /par/raw/cll/band
Access: rw
Type: float [Hz]
Values: [0.1…200]
Default: 3

Care should be taken that the setting used doesn't result in receiver malfunction. Values in the range [0.2…5] Hz are usually safe.

Note: The ionosphere fluctuations and quick multipath are main limitations in setting lower values for this parameter.

CA/L1 CLL Order

Name: /par/raw/cll/order
Access: rw
Type: integer
Values: 1,2,3
Default: 1

CA/L1 CLL by CA/L1 PLL Guide Factor

Name: /par/raw/cll/guid
Access: rw
Type: integer [percents]
Values: [0…100]
Default: 100

The more guiding is set, the less CA/L1 CLL bandwidth could be set and weaker signals could be track. Only when guide factor is set to 100, the CA/L1 CLL order could be set to 1, otherwise it should be set to either 2 or 3.

Strong Loops PLL Bandwidth

Name: /par/raw/plls/band
Access: rw
Type: float [Hz]
Values: [0.1…200]
Default: 3

Strong Loops PLL Order

Name: /par/raw/plls/order
Access: rw
Type: integer
Values: 1,2,3
Default: 1

Strong Loops PLL by CA/L1 PLL Guide Factor

Name: /par/raw/plls/guid
Access: rw
Type: integer [percents]
Values: [0…100]
Default: 100

Strong Loops CLL Bandwidth

Name: /par/raw/clls/band
Access: rw
Type: float [Hz]
Values: [0.1…200]
Default: 3

Strong Loops CLL Order

Name: /par/raw/clls/order
Access: rw
Type: integer
Values: 1,2,3
Default: 1

Strong Loops CLL by CA/L1 PLL Guide Factor

Name: /par/raw/clls/guid
Access: rw
Type: integer [percents]
Values: [0…100]
Default: 100

Strong Loops CLL by Its Own PLL Guide Factor

Name: /par/raw/clls/pllguid
Access: rw
Type: integer [percents]
Values: [0…100]
Default: 0

Weak Loops PLL Bandwidth

Name: /par/raw/gdplls/band
Access: rw
Type: float [Hz]
Values: [0.1…200]
Default: 3

Weak Loops PLL Order

Name: /par/raw/gdplls/order
Access: rw
Type: integer
Values: 1,2,3
Default: 1

Weak Loops PLL by CA/L1 PLL Guide Factor

Name: /par/raw/gdplls/guid
Access: rw
Type: integer [percents]
Values: [0…100]
Default: 100

Weak Loops CLL Bandwidth

Name: /par/raw/gdclls/band
Access: rw
Type: float [Hz]
Values: [0.1…200]
Default: 3

Weak Loops CLL Order

Name: /par/raw/gdclls/order
Access: rw
Type: integer
Values: 1,2,3
Default: 1

Weak Loops CLL by CA/L1 PLL Guide Factor

Name: /par/raw/gdclls/guid
Access: rw
Type: integer [percents]
Values: [0…100]
Default: 100

Weak Loops CLL by Its Own PLL Guide Factor

Name: /par/raw/gdclls/pllguid
Access: rw
Type: integer [percents]
Values: [0…100]
Default: 0

RF Configuration Parameters

These parameters are hard for the user to fiddle with. Please ask JAVAD GNSS support for advice should you feel a need to tune these parameters for your hardware.

RF Chip Identifier

Name: /par/rf/id
Access: r
Type: enumerated
Values: NT1066

RF Chip Revision

Name: /par/rf/rev
Access: r
Type: integer

RF Low-Noise Amplifier Gain

Name: /par/rf/lna/gain
Access: rw
Type: enumerated
Values: low,high
Default: <receiver dependent>

For /par/rf/id equal to NT1066:

low - use LNA2

high - use LNA1

RF High Dynamic Mode

Name: /par/rf/hdm/mode
Access: rw
Type: boolean
Values: on,off
Default: <receiver dependent>

RF Pulse-Width Modulation

Name: /par/rf/pwm/wbN, N=[1,2,3]
Access: rw
Type: integer
Values: [0…127]
Default: <receiver dependent>

RF Calibrator Attenuator Mode

Name: /par/rf/cal/att
Access: rw
Type: boolean
Values: on,off
Default: <receiver dependent>

RF Chip Power Managment

Name: /par/rf/pwr/mode
Access: rw
Type: enumerated
Values: auto,on,off
Default: auto

on - power is always on

off - power is always off

auto - power is controlled automatically according to satellite tracking

RF Chip Low-Level Access

Name: /par/rf/rw/[a|b|c|d]/A, A=[0…16383]
Access: rw
Type: enumerated
Values: [rd,0…255]
Default: 0

rd - to send request to read

[0…255] - write specific value

Anti-jamming Parameters

Anti-jamming Mode

Name: /par/ajm/mode
Access: rw
Type: enumerated
Values: on,off
Default: <receiver dependent>

on - anti-jamming is turned on in the mode specified by the rest of parameters in this section.

off - anti-jamming is turned off.

Enable Anti-jamming Bands

Name: /par/ajm/band
Access: rw
Type: list {<receiver dependent>} of boolean
Values: {on|off,…,on|off}
Default: {on,…,on}

on - enable anti-jamming on corresponding band.

off - disable anti-jamming on corresponding band.

Enable Anti-jamming on Band B.

Name: /par/ajm/band/B (B=[<receiver dependent>])
Access: rw
Type: boolean
Values: on,off
Default: on

on - enable anti-jamming on band B.

off - disable anti-jamming on band B.

Antenna Input Parameters

Antenna Input

Name: /par/ant/inp
Access: rw
Type: enumerated
Values: int,ext,auto (receiver-dependent)
Default: (receiver-dependent)

Note that allowed parameter values and the default value are receiver-dependent.

Antenna Current Input

Name: /par/ant/curinp
Access: r
Type: enumerated
Values: int,ext

This parameter always reflects the actual antenna input that is currently in use. This is of primary interest when /par/ant/inp is set to auto.

Status of External Antenna

Name: /par/ant/dc
Access: r
Type: enumerated
Values: off,normal,overload

off - external antenna does not draw any DC

normal - external antenna draws normal DC

overload - external antenna draws too high DC

Enable External Antenna Power

Name: /par/pwr/extanten
Access: rw
Type: boolean
Values: on,off
Default: on

on - enable power output to external antenna

off - disable power output to external antenna

Frequency Input and Output Parameters

Frequency Mode

Name: /par/frq/mode
Access: rw
Type: enumerated
Values: off,use,gen
Default: off

off - use internal oscillator as frequency source and do not generate output frequency.

use - use external frequency input as frequency source and do not generate output frequency.

gen - use internal oscillator as frequency source and generate output frequency. Should you set this value, you will probably need to set /par/osc/mode parameter as well.

Input Frequency Value

Name: /par/frq/in/frq
Access: rw
Type: integer [MHz]
Values: [5…40]
Default: 10

The frequency of the external oscillator. The receiver will not lock on satellites if this parameter is set to a wrong value while /par/frq/mode is set to use.

Input Frequency Status

Name: /par/frq/in/status
Access: r
Type: enumerated
Values: off,wait,locked

`off` - the receiver is using the internal oscillator.

`wait` - the receiver is waiting for the external frequency lock. The receiver will return this value if, after the user has set `/par/frq/mode` to `use`, the external frequency oscillator is disconnected, its amplitude is too low, or the actual external source frequency is different from that specified via `/par/frq/in/frq` parameter.

`locked` - external frequency source is being used.

Input Frequency Amplitude

Name: `/par/frq/in/amp`
Access: `r`
Type: `enumerated`
Values: `off,low,ok`

`off` - the internal oscillator is used, – the amplitude can't be measured.

`low` - external frequency signal's amplitude is lower than required.

`ok` - external frequency signal's amplitude meets the required specifications.

Output Frequency Value

Name: `/par/frq/out/frq`
Access: `r | rw (hardware dependent)`
Type: `integer [MHz]`
Values: `(hardware dependent, 10,20 - typical)`
Default: `(hardware dependent, 20 - typical)`

The output frequency. This frequency will appear on the frequency output when `/par/frq/mode` is set to `gen`.

Oscillator Frequency Offset Reduction Mode

Name: `/par/osc/mode`
Access: `rw`
Values: `off,locked,tied`
Default: `off`

`off` - receiver will not adjust internal oscillator's frequency.

`locked` - receiver will adjust internal oscillator's frequency so that measured frequency offset is reduced to about zero. Receiver clock offset in this mode becomes almost constant, though being subject to slow random drifts.

`tied` - receiver will adjust the internal oscillator's frequency so that both measured frequency offset and receiver clock offset are reduced to about zero.

After switching from `off` to `locked`, it may take the receiver up to a minute to adjust the internal oscillator frequency to the nominal value. This time could be even longer for switching to `tied` mode.

After switching from `locked` or `tied` to `off`, the internal oscillator frequency will be reset to its mean value instantly, which may result in temporary loss of lock to satellites.

Setting this parameter to `locked` or `tied` only guarantees that the receiver's frequency output will have high long-term stability, not necessarily high short-term stability (the latter depends on the internal oscillator type).

Hardware Calibrator

Parameters described in this section govern the behavior of the hardware calibrator.

Hardware calibrator instantly measures phase and code delays introduced by the receiver RF part for different GLONASS FCNs (let's call them GLONASS inter-channel corrections), as well as (provided corresponding hardware support is present) phase and code delays of other RF bands. It could be programmed to either apply resulting GLONASS corrections to the code and/or phase measurements, or just to store corrections inside receiver and output them by user request.

Despite the fact that calibrator measurements are relative to the delays introduced by calibrator hardware, resulting delays in different RF bands may be interpreted as absolute and can be used for different purposes like precise timing, absolute ionosphere delay computations and so on, because self calibrator delays should be extremely small and constant. Calibrator tracks any change of RF delay(s), independent on the cause, for example, resulted from temperature changes.

Warning: *Calibration process should be performed only when GNSS antenna is connected to the receiver, otherwise wrong corrections could be computed.*

Calibrator Mode

Name: `/par/calib/hard/mode`
Access: `rw`
Type: `boolean`
Values: `on,off,alt`
Default: `alt (if E5b supported), off (otherwise)`

`on` - calibrator is active.

`off` - calibrator is turned off.

`alt` - calibrator is active, but only delays between L5(E5a) and E5b bands are calculated. This helps receiver to track E5altBoc signal as best as possible. Please do not turn calibrator off if you track Galileo E5altBoc signal.

Attenuator Support

Name: /par/calib/hard/atten
Access: r
Type: boolean
Values: y,n
Default: (hardware dependent)

If the value of this parameter is n, we do not recommend to use calibrator code measurements for precise applications as they will be rather noisy. Only some early designed boards do not have attenuator.

Supported Bands

Name: /par/calib/hard/bands
Access: r
Type: enumerated
Values: glo,all
Default: (hardware dependent)

glo - only measurements on GLONASS L1 and L2 (if present) bands are supported.

all - measurements on all receiver-supported bands are supported.

Calibrator Measurements Availability

Name: /par/calib/hard/valid
Access: r
Type: boolean
Values: y,n
Default: n

y - calibrator measurements are available (calibrator performed at least one full round of all required measurements.)

n - no measurements are available.

Apply Calibrator Corrections

Name: /par/calib/hard/use
Access: rw
Type: boolean
Values: on,off
Default: off

on - apply GLONASS inter-frequency corrections, provided they are available. Particular corrections to be applied are determined by the /par/calib/hard/apply/x parameters.

`off` – do not apply measured corrections. Corrections are still kept in memory and are available through hardware calibrator GREIS messages (refer to "Hardware Calibrator Messages" on page 122).

Enable Specific Code Corrections

Name: `/par/calib/hard/apply/code`
Access: `rw`
Type: `list {ca,p1,l2c,p2} of boolean`
Values: `on,off`
Default: `{on,on,on,on}`

`on` – apply corresponding GLONASS inter-channel correction to code measurements (provided parameter `/par/calib/hard/use` is set to `on`.)

`off` – never apply corresponding correction.

Enable Specific Phase Corrections

Name: `/par/calib/hard/apply/phase`
Access: `rw`
Type: `list {ca,p1,l2c,p2} of boolean`
Values: `on,off`
Default: `{on,on,on,on}`

`on` – apply corresponding GLONASS inter-channel correction to phase measurements (provided parameter `/par/calib/hard/use` is set to `on`.)

`off` – never apply corresponding correction.

Smooth GLONASS Code Corrections

Name: `/par/calib/hard/smooth/code`
Access: `rw`
Type: `integer`
Values: `0,1,2`
Default: `0`

`0` – do not smooth code corrections

`1` – smooth code corrections using linear polynomial approximation

`2` – smooth code corrections using quadrature polynomial approximation

We do not recommend to change this parameter from its default value.

Smooth GLONASS Phase Corrections

Name: /par/calib/hard/smooth/phase
Access: rw
Type: integer
Values: 1,2,3
Default: 1

1 - do not smooth phase corrections

2 - smooth phase corrections using linear polynomial approximation

3 - smooth phase corrections using quadrature polynomial approximation

We do not recommend to change this parameter from its default value.

Input Point of Calibrator Signal

Name: /par/calib/hard/inp
Access: rw
Type: enumerated
Values: rf, antint, antext
Default: rf

rf - mix calibrator signal at the entry of RF board

antint - mix in internal antenna

antext - mix in external antenna

Usually calibrator signal is mixed into the receiver signal at the point where satellite signal comes to receiver RF board from antenna. For special purposes and with special hardware it is possible to provide calibrator signal to internal or external antenna via second antenna cable coming from special receiver socket to special antenna socket. This allows to calibrate antenna circuity in addition to receiver RF. This is useful for antenna parameters investigation, as well as for precise timing applications, as calibrator will calculate temperature-dependent signal delay for all receiver/antenna path.

Do not change this parameter unless you have corresponding special hardware.

Measured Code Delays

Name: /par/calib/hard/delay/B/C
(B=[gps1,gps2,gps5,glo1,glo2,glo3]) (C=[ca,p])
Access: r
Type: float [meters]
Values: [0…500]
Default: 0

These parameters contain measured code delays for corresponding bands and code types. The value is 0 if delay is not yet measured or if calibrator hardware does not support this band and code type.

Measured Code GLONASS Corrections

Name: /par/calib/hard/delay/X (X=[c1delay,p1delay,c2delay,p2delay])
Access: r
Type: integer [millimeters]
Values: [0…10000]
Default: 0

Value is 0 if not measured yet.

Measured Phase GLONASS Corrections

Name: /par/calib/hard/delay/X (X=[c1phase,p1phase,c2phase,p2phase])
Access: r
Type: integer [10^{-4} cycles]
Values: [0…10000]
Default: 0

Value is 0 if not measured yet.

4.4.6 Ephemeris Handling

Enable Ephemeris Double-check

Name: /par/pos/eph/dblchk
Access: rw
Type: boolean
Values: y|n
Default: n

When enabled, ensure two exactly the same ephemeris are received before updating ephemeris data.

Note: currently affects only GPS and GLONASS ephemeris.

4.4.7 Almanac Status

Almanac Status

Name: `/par/alm`
Access: `r`
Type: `list {gps,glo}`

Almanac Status for GPS Satellites

Name: `/par/alm/gps`
Access: `r`
Type: `array [1…32] of boolean`
Values: `{y|n,…,y|n}`

Almanac Status for GPS Satellite Number N

Name: `/par/alm/gps/N (N=[1…32])`
Access: `r`
Type: `boolean`
Values: `y,n`

`y` – the almanac data are available for the satellite

`n` – the almanac data are unavailable for the satellite

Almanac Status for GLONASS Satellites

Name: `/par/alm/glo`
Access: `r`
Type: `array [1…24] of boolean`
Values: `{y|n,…,y|n}`

Almanac Status for GLONASS Satellite Number N

Name: `/par/alm/glo/N (N=[1…24])`
Access: `r`
Type: `boolean`
Values: `y,n`

`y` – the almanac data are available for the satellite

`n` – the almanac data are unavailable for the satellite

4.4.8 Positioning Parameters

Generic Positioning Parameters

Position Update Rate

Name: /par/pos/msint
Access: rw
Type: integer [milliseconds]
Values: [MIN…5000], multiple of MIN
Default: 100

MIN – either 5 or 10, depending on receiver type

This parameter specifies the required period of stand-alone and code-differential positions update rate. Carrier Phase-differential (RTK) position is not affected. Receiver will calculate effective period of the position updates using the value of this parameter and the value of the parameter /par/raw/curmsint (see /par/pos/curmsint below).

Effective Position Update Rate

Name: /par/pos/curmsint
Access: r
Type: integer [milliseconds]

Although the user can formally set /par/pos/msint to arbitrary value allowed by the specification, receiver may need to adjust this user setting in order to make it consistent with the value of /par/raw/curmsint parameter. The adjusted setting is stored to this read-only parameter and defines internal effective position update rate.

The formula used to calculate curmsint is as follows:

$$\mathtt{curmsint} = max\,(1, \lfloor \mathtt{msint} \,/\, \mathtt{raw} \rfloor) \times \mathtt{raw}$$

where msint is the value of /par/pos/msint, raw is the value of /par/raw/curmsint, and $\lfloor x \rfloor$ denotes integer part of x.

The actual period at which receiver will allow user to get position depends on the value of /par/pos/curmsint parameter and the current value of the _POS receiver option. Actual position update period is calculated as follows:

$$\mathtt{pos\_period} = max\,(1, \lfloor 1000 \,/\, \mathtt{\_POS} \,/\, \mathtt{curmsint} \rfloor) \times \mathtt{curmsint}$$

where _POS is the current value of corresponding receiver option, and $\lfloor x \rfloor$ denotes integer part of x.

Note: While the formulas seem rather complex, what they basically mean in practice, is that if you set `/par/pos/msint` to a value that is multiple of both `/par/raw/curmsint` and `1000/opt`, then both the `/par/pos/curmsint` and actual allowed position output period will be equal to the specified value.

Elevation Computation Mode

Name: `/par/pos/elmode`
Access: `rw`
Type: `enumerated`
Values: `astrohor,truehor`
Default: `astrohor`

`astrohor` - satellites elevations will be computed with respect to the astronomical horizon.

`truehor` - satellites elevations will be computed with respect to the true visible horizon.

Both methods give the same result at zero height over Earth ellipsoid and begin to differ significantly only when antenna gets rather high.

Elevation Mask for Position Computation

Name: `/par/pos/elm`
Access: `rw`
Type: `integer [degrees]`
Values: `[-90…90]`
Default: `5 or 10 (receiver dependent)`

Satellites with elevations lower than this mask will be excluded from position computation.

Sector Mask for Position Computation

Name: `/par/pos/elazm/N N=[1,2,3]`
Access: `rw`
Type: `list {el1,el2,az1,az2} of float [degrees]`
Values: `{[-90…90],[-90…90],[0…360],[0…360]}`
Default: `{0,0,0,0}`

Receiver will exclude a satellite from position computation if its elevation is between `el1` and `el2` and at the same time its azimuth is between `az1` and `az2`, taking into account zero crossing.

SNR Mask for Position Computation

Name: /par/pos/minsnr
Access: rw
Type: integer [dB×Hz]
Values: [0…50]
Default: 30

Satellites whose signal-to-noise ratios are lower than this value will be excluded from position computation.

Position Computation Mode

Name: /par/pos/mode/cur
Access: rw
Type: enumerated
Values: pd,pf,cd,wd,gb,sp
Default: sp

pd – carrier phase differential (RTK) with fixed ambiguities

pf – carrier phase differential (RTK) with float ambiguities

cd – code differential (DGPS) mode

wd – wide area code differential mode (WDGPS) using SBAS corrections

gb – GBAS mode

sp – single point positioning mode[1]

These computation modes are arranged in the order of increasing of their typical accuracy, sp being the lowest, and pd being the highest accuracy mode. Receiver will try to compute position according to the mode specified by this parameter. If for whatever reason it fails to compute corresponding position, it will try to use the next mode below the current one, provided the mode is enabled by corresponding parameter (see below). This process continues down the list of modes until either some position is enabled and could be computed, or all the modes are exhausted.

Note: In fact receiver will still compute single point position for its internal purposes, though it will not make it available for output if disabled.

Note: Current implementation of code differential mode has a limitation. There are cases when receiver will not try to compute single point position when it failed to compute code differential position even though there are enough data.

Note: GBAS mode is never used implicitly.

1. Also known as “absolute positioning”, “stand-alone positioning” or simply “point positioning”

Enable Single Point Position

Name: /par/pos/mode/sp
Access: rw
Type: boolean
Values: on,off
Default: on

on - when receiver is running in pd, pf, or cd mode and is unable to output pd, pf, or cd solution, it will output sp solution, if available.

off - receiver will not output sp solution unless it runs in sp mode.

This parameter doesn't affect behavior of receiver running in sp modes.

Enable Code Differential Position

Name: /par/pos/mode/cd
Access: rw
Type: boolean
Values: on,off
Default: off

on - when receiver is running in pd or pf mode and is unable to output pd or pf solution, it will output cd solution, if available.

off - receiver will not output cd solution unless it runs in cd mode.

This parameter doesn't affect behavior of receiver running in cd or sp modes.

Note: Code differential mode requires broadcasting the corresponding DGPS (not RTK) messages from the reference receiver and accepting them on the rover receiver. If any of these requirements are not met, then enabling this parameter will not have any effect.

Enable SBAS Code Differential Position

Name: /par/pos/mode/wd
Access: rw
Type: boolean
Values: on,off
Default: off

on - when receiver is running in pd or pf mode and is unable to output pd, pf, or cd solution, it will output wd solution, if available.

off - receiver will not output wd solution unless it runs in wd mode.

This parameter doesn't affect behavior of receiver running in cd, wd, or sp modes.

Note: SBAS code differential mode requires tracking of at least one SBAS satellite. If thise requirement is not met, then enabling this parameter will not have any effect.

Enable RTK Solution with Float Ambiguities

Name: /par/pos/mode/pf
Access: rw
Type: boolean
Values: on,off
Default: on

on - when receiver is running in pd mode and is unable to output pd solution, it will output pf solution, if available.

off - receiver will not output pf solution unless it is running in pf mode.

This parameter doesn't affect behavior of receiver running in pf, cd, or sp modes.

Navigation Data Source

Name: /par/pos/navdata/source
Access: rw
Type: enumeration
Values: sat,ext,any,off
Default: sat

sat - get navigation data from satellites only.

ext - get navigation data from external sources only.

any - enable all sources of navigation data.

off - disable all sources of navigation data.

Almanac Data Source

Name: /par/pos/almdata/source
Access: rw
Type: enumeration
Values: sat,ext,any,off
Default: sat

sat - get almanac data from satellites only.

ext - get almanac data from external sources only.

any - enable all sources of almanac data.

off - disable all sources of almanac data.

Clear Working Ephemeris Data

Name: /par/pos/navdata/reset
Access: rw
Type: boolean
Values: n,y (reads always 'n')
Default: n

`y` - -clear active working ephemeris data. No non-volatile data are cleared, nor receiver reboot is performed.

`n` - no effect

Clear Working Almanac Data

Name: `/par/pos/almdata/reset`
Access: `rw`
Type: `boolean`
Values: `n,y (reads always 'n')`
Default: `n`

`y` - -clear active working almanacs data. No non-volatile data are cleared, nor receiver reboot is performed.

`n` - no effect

Enable Satellite System

Name: `/par/pos/sys`
Access: `rw`
Type: `list {gps,glo,gal,sbas,qzss,comp,irnss} of boolean`
Values: `{y|n,y|n,y|n,y|n,y|n,y|n,y|n}`
Default: `<receiver-dependent>`

This parameter allows you to select satellite constellation(s) used for position computation. The fields correspond to GPS, GLONASS, Galileo, SBAS, QZSS, BeiDou (COMPASS), and IRNSS systems, respectively.

Per- GNSS System Parameters

Parameters in this section are similar for each GNSS system. In the descriptions below, `SYS` stands for one of `(gps,glo,gal,sbas,qzss,comp,irnss)`, and `[a…b]` stands for the following SVs numbers ranges:

`gps` - [1…32]

`glo` - [1…30]

`gal` - [1…36]

`sbas` - [120…147]

`qzss` - [193…199]

`comp` - [1…37]

`irnss` - [1…14]

Enable GNSS Satellites by Their Numbers

Name: /par/pos/SYS/sat
Access: rw
Type: array [a…b] of boolean
Values: {y|n,…,y|n}
Default: {y,…,y}

Enable GNSS Satellite Number N

Name: /par/pos/SYS/sat/N (N=[a…b])
Access: rw
Type: boolean
Values: y,n
Default: y

y – enable using of GNSS satellite number N for position computation

n – disable using of GNSS satellite number N for position computation

Note: If given GLONASS satellite is enabled by this parameter, has frequency code number M, and is disabled by parameter /par/pos/glo/fcn/M, the satellite will be still disabled.

Enable Applicability of Measurements by Signal

Name: /par/pos/SYS/meas/SIG
Access: rw
Type: boolean
Values: n,y
Default: y

y – enable applicability of given SIG of SYS

n – disable applicability of given SIG of SYS

Particular mode of using of one or more applicable measurements is defined by /par/pos/sp/meas parameter.

The values of SIG for every SYS are as follows:

gps – ca,p1,p2,l2c,l5,l1c

glo – c1,p1,p2,c2,l3

gal – e1,aboc,e5b,e5a,e6

sbas – ca,l5

qzss – ca,l2c,l5,l1c,l6

comp – b1,aboc,b2,b3,b2a,b1c

irnss – l5

Enable GLONASS Satellites by FCN

Name: /par/pos/glo/fcn
Access: rw
Type: array [-7…7] of boolean
Values: {y|n,…,y|n}
Default: {y,…,y}

Enable GLONASS Satellite with FCN #N

Name: /par/pos/glo/fcn/N (N=[-7…7])
Access: rw
Type: boolean
Values: y,n
Default: y

y – enable using of GLONASS satellite with frequency code number N for position computation

n – disable using of GLONASS satellite with frequency code number N for position computation

Note: If given satellite is enabled by this parameter, has orbit number M, and is disabled by parameter /par/pos/glo/sat/M, the satellite will be still disabled.

Enable Check of Health for GNSS Satellites

Name: /par/pos/SYS/health/check
Access: rw
Type: array [a…b] of boolean
Values: {y|n,…,y|n}
Default: {y,…,y}

Enable Check of Health for GNSS Satellite Number N

Name: /par/pos/SYS/health/check/N (N=[a…b])
Access: rw
Type: boolean
Values: y,n
Default: y

y – exclude satellite number N from position computation when its health value from navigation data indicates that the satellite is unhealthy.

n – do not exclude satellite number N from position computation no matter what the value of health is.

Enable Check of URA for GNSS Satellites

Name: /par/pos/SYS/ura/check
Access: rw
Type: array [a…b] of boolean
Values: {y|n,…,y|n}
Default: {y,…,y}

Enable Check of URA for GNSS Satellite Number N

Name: /par/pos/SYS/ura/check/N (N=[a…b])
Access: rw
Type: boolean
Values: y,n
Default: y

y – exclude satellite number N from position computation when URA value from navigation data for this satellite exceeds the limit specified by the /par/pos/SYS/ura/mask parameter.

n – do not exclude satellite number N from position computation no matter what the value of URA is.

URA Mask for GNSS Satellites

Name: /par/pos/SYS/ura/mask
Access: rw
Type: float [meters]
Values: [0.01…10000]
Default: 10.0

This parameter specifies the limit for checking of URA values for satellites. Refer to description of the parameter /par/pos/SYS/ura/check/N for details.

System Datum

Name: /par/pos/SYS/datum (SYS=gps,glo,gal,sbas,qzss)
Access: rw
Type: datum_id
Default: P90 for SYS=glo, W84 for others

These parameters specify datum that will be used for computation of satellites position and velocity for GNSS system SYS.

System Time

Name: /par/pos/SYS/systime (SYS=gps,glo,gal,sbas,qzss)
Access: rw
Type: enumerated
Values: gps,SYS
Default: glo for glo, irnss for irnss, gps for others

This parameter specifies time scale that will be used for computation of satellites position and velocity for GNSS system SYS.

System Navigation Data

Name: /par/pos/SYS/navdata (SYS=gps,glo,gal,sbas,qzss,irnss)
Access: rw
Type: enumerated
Values: any,nav,cnav,cnav2 for 'gps'
any,nav,l3nav for 'glo'
any,inav,fnav for 'gal'
any,nav for 'sbas'
any,nav,cnav,cnav2 for 'qzss'
any,l5nav for 'irnss'
Default: any

This parameter specifies the type of navigation data that will be used for computation of satellites position and velocity for GNSS system SYS.

Datums

Notations

Each datum supported by the receiver has unique datum identifier assigned. Datum identifier is a string of up to 5 upper-case characters. We will use the type datum_id to refer datum identifiers. Though the set of supported datums may vary from receiver to receiver and from one firmware version to another, the following datums are always supported:

W84 - WGS-84 datum; GPS system datum

P90 - PE-90 datum; GLONASS system datum

W72 - WGS-72 datum

USER - user-defined datum

GOT - datum got from external source (e.g., from RTCM3 data)

In addition, receiver may support a subset of datums described in the *"Reference Ellipsoids and Local Datums supported by JAVAD GNSS Receivers"* guide.

Note: You can get the list of supported datums along with their parameters from receiver itself using `print,/par/pos/datum:on` command.

Every datum has corresponding ellipsoid parameters as well as a set of parameters for standard 7-parameters transformation. For most datums these parameters are read-only. User may change only parameters of `USER` and `P90` datum.

Ellipsoid parameters have type called `ell_params` of the following format:

```
{ell_id,axis,inv_flat}
```

where

`ell_id` - ellipsoid identifier. String comprising two characters.

`axis` - ellipsoid's major semi-axis in the range `[6300000…6500000]` meters.

`inv_flat` - ellipsoid's inverse flattening in the range `[280…300]`, dimensionless.

Set of parameters for 7-parameters transformation have type called `datum_params` of the following format:

```
{datum_id,ref,dx,dy,dz,rx,ry,rz,scale}
```

where

`datum_id` - datum identifier (see above).

`ref` - flag indicating whether the datum's transformation parameters are specified with respect to WGS-84 (`ref=0`) or PE-90 (`ref=1`)

`dx, dy, dz` - translations in X-, Y- and Z-direction, respectively. Each component is in the range `[-10000…10000]` meters.

`rx, ry, rz` - rotations around X-, Y- and Z-axis, respectively. Each component is in the range `[-60…60]` seconds of arc.

`scale` - scale in ppm (true scale = `scale` x 10^{-6}) ranging within `[-100…100]`.

The above parameters specify a coordinate transformation from given datum to WGS-84 (or PE-90) according to the following equations:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{W84|P90} = \begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{bmatrix} + (1 + s \cdot 10^{-6}) \cdot \begin{bmatrix} 1 & R_z & -R_y \\ -R_z & 1 & R_x \\ -R_y & -R_x & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{local}$$

Finally, in the descriptions below, [D] denotes either one of the valid datum identifiers, or the string INUSE that designates the datum that is set as current for position computation.

Current Datum for Position Computation

Name: /par/pos/datum/cur
Access: rw
Type: datum_id
Default: W84

This parameter specifies the identifier of the datum that will be used for position computation. Note that some of the receiver messages always contain position referenced to WGS-84 datum. This parameter has no effect on such messages.

When this parameter is set to the value GOT, the datum taken from the /par/pos/datum/GOT parameter will be used.

Use Datum Full Precision Transformation

Name: /par/pos/datum/ft
Access: rw
Type: boolean
Values: on,off
Default: off

on - receiver will use full precise transformation formulas for datum transformations. This mode is rather time-consuming, so we don't recommend it unless datum rotation angles are large enough (tens seconds of arc or more).

off - receiver will use approximate datum transformations valid for small rotation angles. This preserves CPU

Parameters of Datum [D]

Name: /par/pos/datum/[D]
Access: r
Type: list {ell,datum}

ell - reference ellipsoid parameters for this datum

par - 7-parameters transformation for this datum

Reference Ellipsoid for Datum [D]

Name: /par/pos/datum/[D]/ell
Access: r
Type: ell_params

7-parameters Transformation for Datum [D]

Name: /par/pos/datum/[D]/par
Access: r
Type: datum_params

User Defined Datum

Name: /par/pos/datum/USER
Access: rw
Type: list {ell,par}
Default:
```
{{U-,6378137.0000,298.257223563},
 {USER,0,+0.0000,+0.0000,+0.0000,
  +0.00000,+0.00000,+0.00000,+0.00000}}
```

ell - user-defined ellipsoid parameters for this datum. This should conform to the format of the type ell_params. The ell_id field of the format should be set to the string U- (the letter "U" in the upper case and the "minus" sign), or, alternatively, could be omitted (note that the delimiting comma should still exist).

par - user-defined set of parameters for 7-parameters coordinates transformation. This should conform to the format of type datum_params. The datum_id field of the format should be set to the string USER, or, alternatively, could be omitted (note that the delimiting comma should still exist).

Example: Set parameters of user-defined datum, then print them:

```
⇒ set,/par/pos/datum/USER,{{U-,6378136,298},{USER,0,0,0,1,0,0,-0.2,0}}
⇒ print,/par/pos/datum/USER
⇐ RE021{{U-,6378136.0000,298.000000000},
  RE046 {USER,0,+0.0000,+0.0000,+1.0000,+0.00000,+0.00000,
    -0.20000,+0.00000}}
```

Datum Got From External Source

Name: /par/pos/datum/GOT
Access: r
Type: list {ell,par}
Default:
```
{{U-,6378137.0000,298.257223563},
 {GOT,0,+0.0000,+0.0000,+0.0000,
  +0.00000,+0.00000,+0.00000,+0.00000}}
```

ell - ellipsoid parameters for this datum got from external source. This should conform to the format of the type ell_params. The ell_id field of the format should be set to the string G- (the letter "G" in the upper case and the "minus" sign), or, alternatively, could be omitted (note that the delimiting comma should still exist).

par - set of parameters for 7-parameters coordinates transformation got from external source. This should conform to the format of type datum_params. The

datum_id field of the format should be set to the string GOT, or, alternatively, could be omitted (note that the delimiting comma should still exist).

7-parameters Transformation for PE-90

Name: /par/pos/datum/P90/par
Access: rw
Type: datum_params
Default: {P90,0,+0.0000,+0.0000,+0.0000,
+0.00000,+0.00000,-0.00000,+0.00000}

The datum_id field of the format should be set to the string P90, or, alternatively, could be omitted (note that the delimiting comma should still exist).

The ref field of the format should be set to the value 0, or, alternatively, could be omitted (note that the delimiting comma should still exist).

There are no universally accepted Helmert transformation parameters for the PE-90 datum so far. This explains why the user is allowed to define his/her own version of the transformation using this parameter.

Note: Under use of combined GPS/GLONASS receivers in DGPS modes, be sure that the same transformation parameters for PE-90 datum is used at both the base station and the rover. You may relax this requirement provided referencing corrections to local datum is turned on (see /par/rtcm/base/locdtm parameter on page 361).

Grid Systems

Current Grid System

Name: /par/pos/grid/cur
Access: rw
Type: enumerated
Values: NONE,UTM,USER,LOC,GOT
Default: UTM

NONE – receiver will not compute grid coordinates (this mode allows some reduction of the processor's computation load).

UTM – Universal Transverse Mercator (automatic selection of the right zone). 6-degree zones with the scale = 0.9996.

USER – user-defined grid system.

LOC – local coordinates (as a result of the localization procedure).

GOT – grid system got from external source (e.g., from RTCM3 data).

Note that computation of grid and local coordinates depends on the parameter /par/pos/datum/cur. Ensure that this parameter is specified properly.

User-defined Grid System

Name: /par/pos/grid/USER
Access: rw
Type: grid_spec
Default: {TM,N00d00m00.000000s,E000d00m00.000000s,
1.000000000,0.0000,0.0000}

grid_spec has the following format:

{proj,par1,…,parN}

where proj is map projection identifier, and the list of parameters depends on particular projection. Map projection identifier is one of:

TM	Transverse Mercator
TMS	Transverse Mercator South oriented
STER	Stereographic
LCC1SP	Lambert Conic Conformal 1SP
LCC2SP	Lambert Conic Conformal 2SP
LCCW	Lambert Conic Conformal 1SP west orientated
CS	Cassini-Soldner
MC	Mercator
OM	Oblique Mercator (Hotine B)

The entire format is:

- for LCC2SP:

{LCC2SP,latFO,lonFO,latSP1,latSP2,NFO,EFO}

- for OM:

{OM,latPC,lonPC,SIL,AzIL,ARSG,NPC,EPC}

- for all the other values of proj:

{proj,lat0,lon0,scale,falseN,falseE}

Where:

lat0 - latitude of the origin of the grid projection, in latitude format.
lon0 - longitude of the central meridian of the projection, in longitude format.
scale - scale factor; [0.1…10].
falseN - false Northing; $[-10^9 \ldots 10^9]$ meters.
falseE - false Easting; $[-10^9 \ldots 10^9]$ meters.

`latFO` – latitude of the false origin, in latitude format.

`lonFO` – longitude of the false origin, in longitude format.

`latSP1` – latitude of 1-st standard parallel, in latitude format.

`latSP2` – latitude of 2-nd standard parallel, in latitude format.

`NFO` – northing of false origin; [-10e7…10e7] meters.

`EFO` – easting of false origin; [-10e7…10e7] meters.

`latPC` – latitude of projection center, in latitude format.

`lonPC` – longitude of projection center, in longitude format.

`SIL` – scale factor on initial line; [0.1…10].

`AzIL` – azimuth of initial line; [0…360] degrees.

`ARSG` – angle from rectified to skew grid; [0…360] degrees.

`NPC` – northing of projection center; [-10e7…10e7] meters.

`EPC` – easting of projection center; [-10e7…10e7] meters.

Grid System Got From External Source

Name: `/par/pos/grid/GOT`
Access: `r`
Type: `grid_spec`
Default: `{TM,N00d00m00.000000s,E000d00m00.000000s, 1.000000000,0.0000,0.0000}`

For description of `grid_spec`, see `/par/pos/grid/USER`.

Specific Map Projection

Name: `/par/pos/grid/spc`
Access: `rw`
Type: `enumerated`
Values: `NONE,S34J,S34S,S34B`
Default: `NONE`

There are map projections that require additional computations, for example, with the use of polynomials. This parameter allows the user to select such a specific map projection. Currently the receiver supports three such projections which are used on Denmark maps: System 34 Jutland (`S34J`), System 34 Seeland (`S34S`), and System 34 Bornholm (`S34B`).

Note: For correct use of a projection from this list, you should set up the parameters of the desired Transverse Mercator projection using the `/par/pos/grid/USER` parameter and specify the correct datum transformation parameters using the `/par/pos/datum/USER` parameter.

Local Coordinates

Parameters of Transformation to Local Coordinates

Name: /par/pos/local/par
Access: rw
Type: {lat0,lon0,scalePrj,falseN,falseE,delN,delE, scaleLoc,rotation,H_n,H_e,H_0}
Default: {N00d00m00.000000s,E000d00m00.000000s,1,0,0,0,0, 1,000d00m00.000000s,0,0,0}

Stereographic projection parameters:

`lat0` - latitude of the origin of the projection, in latitude format.

`lon0` - longitude of the origin of the projection, in longitude format.

`scalePrj` - scale factor of the projection; `[0.1…10]`.

`falseN` - false Northing; $[-10^7 \ldots 10^7]$ meters.

`falseE` - false Easting; $[-10^7 \ldots 10^7]$ meters.

Parameters obtained from the localization procedure:

`delN` - offset in North direction; $[-10^7 \ldots 10^7]$ meters.

`delE` - offset in East direction; $[-10^7 \ldots 10^7]$ meters.

`scaleLoc` - scale factor; `[0.1…10]`.

`rotation` - rotation angle α; `[0…360)` degrees.

H_n, H_e, H_0 - parameters that define height transformation; H_n and H_e are dimensionless, [-1…1]; H_0 is in range $[-10^4 \ldots 10^4]$ meters.

The equations that define transformation from stereographic coordinates to local coordinates are as follows:

$$n = \text{scale} \cdot (n_{\text{stereo}} \cdot \sin\alpha + e_{\text{stereo}} \cdot \cos\alpha) + \Delta N$$

$$e = \text{scale} \cdot (n_{\text{stereo}} \cdot \cos\alpha - e_{\text{stereo}} \cdot \sin\alpha) + \Delta E$$

$$h = h_{\text{ell}} + H_{\text{n}} \cdot n_{\text{stereo}} + H_{\text{e}} \cdot e_{\text{stereo}} + H_0$$

Generic Single Point Parameters

Parameters described in this section affect single point positioning. Please note that due to specific of code differential (DGPS) positioning, most of these parameters will affect DGPS positioning as well.

Measurements Type to Use

Name: /par/pos/sp/meas
Access: rw
Type: enumerated
Values: any,ionofree,all
Default: any

This parameter specifies which measurements receiver will use for single point position computation.

any - use any one of the available single-frequency measurements

single - synonym for any

ionofree - use ionosphere-free combination of code measurements

dual - synonym for ionofree

all - use all the available signals. In this case optimal combination of signals is used for every SV

Note: Old ca, p1, p2, l2c, l5, and l1c values are still supported but are now obsolete and deprecated.

Consider SNR in Weighting of Measurements

Name: /par/pos/weight/use/snr
Access: rw
Type: boolean
Values: on,off
Default: on

on - enable using of SNR in the weighting of measurements

off - disable using of SNR in the weighting of measurements

Enable Ionospheric Corrections

Name: /par/pos/sp/iono
Access: rw
Type: boolean
Values: on,off
Default: on

on - receiver will correct the measured pseudo-ranges for ionospheric delay errors before computing the point position. For the ionospheric model used, please refer to ICD-GPS-200C, Revision IRN-200C-004 April 12, 2000.

off - receiver will not use ionospheric corrections.

Note: Ionospheric corrections are used in the receiver exclusively for computing single point position. Receiver messages will contain raw pseudo-ranges.

Ionospheric Corrections Type

Name: /par/pos/sp/ionom/type
Access: rw
Type: enumerated
Values: model,grid
Default: model

model - receiver will compute ionospheric delay in stand-alone mode using one of implemented ionosphere models

grid - receiver will compute ionospheric delay in stand-alone mode using ionospheric delays received from one of augmentation systems if it broadcasts corresponding grid, otherwise one of the models will be used

Source of Grid of Ionospheric Delays

Name: /par/pos/sp/ionom/grid
Access: rw
Type: enumerated
Values: sbas,comp
Default: sbas

sbas - use SBAS for the source of grid of ionospheric delays

comp - use BeiDou for the source of grid of ionospheric delays

Ionospheric Model

Name: /par/pos/sp/ionom/model
Access: rw
Type: enumeration
Values: klob,nequick,bdgim
Default: klob

This parameter specifies the type of ionospheric model to be used. The data source for each model is either selected by corresponding parameters described below, or user-defined data for the model could be selected by setting /par/pos/sp/ionom/usermode to the value on.

klob - use Klobuchar model using data from the source defined by either /par/pos/sp/ionom/klob/source or /par/pos/sp/ionom/klob/userdata parameter depending on the value of /par/pos/sp/ionom/usermode.

nequick - use NeQuick-G ionosphere model using data broadcast by Galileo satellites

bdgim - use BeiDou global ionosphere model using data broadcast by new BeiDou satellites

Enable User Data for Ionospheric Model

Name: /par/pos/sp/ionom/usermode
Access: rw
Type: boolean
Values: on,off
Default: off

on - use of user-defined data for ionospheric models regardless of availability of received ionospheric models data.

off - use received ionospheric data.

Source of Klobuchar Model

Name: /par/pos/sp/ionom/klob/source
Access: rw
Type: enumerated
Values: gps,qzss,comp,irnss
Default: gps

This parameter defines data source for received Klobuchar ionospheric model data.

gps - use data broadcast in GPS navigation frames

qzss - use data broadcast in QZSS navigation frames

comp - use data broadcast in BeiDou navigation frames

irnss - use data broadcast in IRNSS navigation frames

User Parameters for Klobuchar Model

Name: /par/pos/sp/ionom/klob/userdata
Access: rw
Type: array [0…7] of float
Values: {[-1/8388608…1/8388608],[-1/1048576…1/1048576],
[-1/131072…1/131072],[-1/131072…1/131072],
[-262144…262144],[-2097152…2097152],
[-8388608…8388608],[-8388608…8388608]}
Default: {1.0244548e-08,1.4901161e-08,
-5.9604645e-08,-1.1920929e-07,
88064.0,32768.0,
-196608.0,-196608.0}

This parameter specifies user defined ionospheric Klobuchar model data, to be used when /par/pos/sp/ionom/usermode parameter is set to on.

Enable Tropospheric Corrections

Name: /par/pos/sp/tropo
Access: rw
Type: boolean
Values: on,off
Default: on

on - receiver will correct the measured pseudo-ranges for tropospheric delay errors when computing the point position. For the tropospheric model applied, please refer to "Technical characteristics of the NAVSTAR GPS" (June 1991).

off - receiver will not use tropospheric corrections.

Note: Tropospheric corrections are used in the receiver exclusively for computing single point position. Receiver messages will contain raw pseudo-ranges.

Tropospheric Model

Name: /par/pos/sp/tropom/model
Access: rw
Type: enumeration
Values: gpsdef,waasdef,gcat,niell,unb3,unb4,goadgood
Default: waasdef

This parameter selects tropospheric model to be used.

gpsdef - use GPS tropospheric model (Central radio propagation Laboratory Reference Atmosphere 1958 zenith delay model plus Chao mapping function)

waasdef - use WAAS tropospheric model defined in WAAS MOPS (Saastamoinen zenith delay model plus Black and Eisner 1984 mapping function plus estimation of meteo data)

gcat - use GCAT tropospheric model (simplified zenith delay model with averaged meteo data plus Black and Eisner 1984 mapping function)

niell - use Niell 1996 tropospheric model (simplified zenith delay model with averaged meteo data plus Niell 1996 mapping function)

unb3 - use UNB3 tropospheric model, University of New Brunswick (Saastamoinen zenith delay model plus Niell 1996 mapping function plus estimation of meteo data)

unb4 - use UNB3 tropospheric model University of New Brunswick (Saastamoinen zenith delay model plus Niell 1996 mapping function plus another estimation of meteo data)

goadgood - use Goad-Goodman tropospheric model (Goad-Goodman zenith delay model and mapping function plus estimation of meteo data)

Tropospheric Model Meteo Data Source for Stand-alone Mode

Name: /par/pos/sp/tropom/meteo/source
Access: rw
Type: enumerated
Values: model,user,external
Default: model

model - use data from tropospheric model

user - use user-specified data

external - use data from external meteo-station

Tropospheric Model Meteo Data Source for RTK

Name: /par/pos/pd/tropom/meteo/source
Access: rw
Type: enumerated
Values: model,user,external
Default: model

model - use data from tropospheric model

user - use user-specified data

external - use data from external meteo-station

Tropospheric Model Current Hydrostatic ZTD

Name: /par/pos/tropom/delay/cur/hyd
Access: r
Type: float [mbar] or <empty string>
Values: <empty string>

Tropospheric Model Current Wet ZTD

Name: /par/pos/tropom/delay/cur/wet
Access: r
Type: float [mbar] or <empty string>
Values: <empty string>

PDOP Mask

Name: /par/pos/pdop
Access: rw
Type: float
Values: [0…500]
Default: 30

If the current PDOP value exceeds the specified mask, the receiver will not compute the single point or code differential position.

Local Time Zone

Name: /par/pos/ltz
Access: rw
Type: list {integer,integer}
Values: {[-13…+13],[0…+59]}
Default: {0,00}

The first parameter in the list describes the local zone hour offset from the UTC time. The second parameter in the list describes the local zone minute offset from the UTC time. These local zone hour and minute offsets will be used in NMEA ZDA message.

Geoid Models

Receiver can use fixed geoid separation, no geoid separation, built-in geoid model, or custom geoid model. Geoid model use is limited by receiver option _GEO and is governed by the parameters described in this section.

Custom geoid model could be used by receiver instead of the built-in one, provided geoid model file is loaded into the unit and the name of the file is specified by the /par/pos/geoid/file parameter. Multiple geoid files could be loaded into receiver, but only single of them (if any) will be loaded into memory on start-up and this model will be used till next receiver reboot.

Fixed Geoidal Separation

Name: /par/pos/geoidh
Access: rw
Type: float [meters]
Values: [-1000…1000]
Default: 0

This parameter contains user-defined geoidal separation. Provided the parameter /par/pos/fix/geoidh is set to on, the value of this parameter is used as geoidal separation with respect to WGS84 datum for orthometric height computation.

Use Fixed Geoidal Separation

Name: /par/pos/fix/geoidh
Access: rw
Type: boolean
Values: on,off
Default: off

`on` – receiver will use geoidal separation specified by the parameter `/par/pos/geoidh`.

`off` – if receiver option `_GEO` is enabled, receiver will use geoidal separation computed using builtin or custom geoid height model. If `_GEO` option is disabled, geoidal separation won't be available.

Current Geoid Name

Name: `/par/pos/geoid/name`
Access: `r`
Type: `string[0…255]`
Values: `<any string>`
Default: `"Default EGM2008"`

This parameter specifies the name of current geoid. This geoid will be used, provided `/par/pos/fix/geoidh` is set to `off`. When custom geoid is successfully loaded, this parameter is set to the name of geoid stored in the custom geoid file.

Custom Geoid File Name

Name: `/par/pos/geoid/file`
Access: `rw`
Type: `string[0…32]`
Values: `<any string>`
Default: `(empty string)`

This parameter specifies the name of file which contains custom geoid model. Data from the specified file is loaded at receiver startup, so the new file name will take effect only after receiver re-boot.

If there is no such file, or the value of the parameter is empty string, the built-in model will be used.

Note: In the current implementation geoid file resides on internal disk along with receiver log-files. While not strictly required, the name of the file should start with dot '.', otherwise AFRM could remove the file when there is not enough space on the disk for new log-files.

Maximum Custom Geoid Data Size

Name: `/par/pos/geoid/maxsize`
Access: `r`
Type: `integer`

This parameter specifies maximum size of custom geoid data in bytes. If custom geoid data is larger than this value, receiver will ignore custom geoid file and will revert back to built-in geoid.

Last Computed Geoidal Height

Name: /par/pos/geoid/height
Access: r
Type: float [meters]

This parameter contains the result of last computation of geoidal height using current geoid. It is provided to simplify troubleshooting of custom geoid models.

Positioning With Reduced State Vector

The parameters described in this section provide positioning modes with fixed values of some state vector entries. These modes are useful when the values of some states are known from some external source or from previous positioning epoch. (The conventional example of that kind is 2D positioning with fixed altitude.) These modes of positioning sometimes can improve position accuracy, decrease required number of SVs, and/or increase SVs redundancy.

Fixed Altitude Positioning

Entered Altitude

Name: /par/pos/alt
Access: rw
Type: float [meters]
Values: [-1000…10000]
Default: 0

This parameter allows to enter exact ellipsoidal height of the antenna phase center in the currently used datum. When /par/pos/fix/alt is set to on, this value will be used in position computations decreasing the number of parameters to be calculated.

This parameter serves two purposes:

- Using an apriori ellipsoidal height will allow the receiver to get a position fix in critical situations when there are few satellites in sight and when it is impossible to derive the point solution using the current measurements only.
- Using precise apriori ellipsoidal height estimate allows to have more precise position fixes.

Use Fixed Altitude

Name: /par/pos/fix/alt
Access: rw
Type: boolean
Values: on,off
Default: off

on - enable receiver to use in position computation the fixed ellipsoidal height specified by the /par/pos/alt parameter.

off - receiver will calculate altitude.

Fixed Receiver Clock Drift Positioning

Entered Clock Drift

Name: /par/pos/clkdft
Access: rw
Type: float [m/s]
Values: [-10000…10000] or string "last"
Default: 0

This parameter allows to enter exact value of clock drift parameter. When /par/pos/fix/clkdft is set to on, this value will be used in position computations (for extrapolation of last clock offset value) decreasing the number of parameters to be calculated.

The string last entered instead of numerical value will assign the numerical value of the last computed clock drift value to this parameter.

This parameter serves two purposes:

- Using an apriori clock drift will allow the receiver to get a position fix in critical situations when there are few satellites in sight and when it is impossible to derive the point solution using the current measurements only.
- Using precise apriori clock drift estimate allows to have more precise position fixes.

Note that using of this parameters makes sense only for operation modes utilizing stable external reference oscillator.

Use Fixed Clock Drift

Name: /par/pos/fix/clkdft
Access: rw
Type: boolean
Values: off,on
Default: off

on - enable receiver to use in position computation the fixed clock drift (extrapolated clock offset) specified by the /par/pos/clkdft parameter.

off - receiver will calculate clock drift.

Fixed Inter-System Time Offset Positioning

Fixed Inter-System Time Offset

Name: /par/pos/SYS (SYS=gpsglo,gpsgal,gpssbas,gpsqzss)
Access: rw
Type: float [meters]
Values: [-300000…+300000] or string "last"
Default: 0

This parameter determines the apriori known (constant) inter-system time offset. Note that this offset is entered in meters, not in time units (just divide this value by the speed of light to get the offset in seconds).

The string last entered instead of numerical value will assign the numerical value of the last computed time offset value to this parameter.

This parameter serves two purposes:

- Using apriori time offset will allow the receiver to get a position fix in critical situations when there are few satellites in sight and when it is impossible to derive the point solution using the current measurements only. E.g. if there are only three GPS satellites and one GLONSASS satellite in view, the receiver won't be able to get a position fix unless the user enters a GLONASS vs. GPS time offset or some other apriori data, thus reducing the number of unknowns in the corresponding set of equations.
- Using precise apriori time offset will allow you to have more precise position fixes.

Use Fixed Inter-system Time Offset

Name: /par/pos/fix/SYS (SYS=gpsglo,gpsgal,gpssbas,gpsqzss)
Access: rw
Type: boolean
Values: on,off
Default: off

on - receiver will use constant time offset specified by /par/pos/SYS in position computation.

off - receiver will calculate inter-system time offset.

Held Parameters Positioning

The parameters in this section allow receiver to hold last computed values of some elements of the state vector when number of SVs is not enough to perform computations with all the elements considered as unknown.

Enable to Hold Altitude

Name: /par/pos/hold/alt
Access: rw
Type: boolean
Values: on,off
Default: on

on - receiver can hold last computed altitude for position computation in the case when SVs number is not enough for conventional position computation.

off - receiver is not allowed to hold last computed altitude.

Enable to Hold Clock Drift

Name: /par/pos/hold/clkdft
Access: rw
Type: boolean
Values: off,on
Default: on

on - receiver can hold last computed clock drift (extrapolate last computed clock offset) for position computation in the case when SVs number is not enough for conventional position computation.

off - receiver is not allowed to hold last computed clock drift.

Enable to Hold Inter-system Time Offsets

Name: /par/pos/hold/systime
Access: rw
Type: boolean
Values: on,off
Default: on

on - receiver can hold last computed inter-system time scales offsets for position computation in the case when SVs number is not enough for conventional position computation.

off - receiver is not allowed to hold last computed time offsets.

KFK Parameters

KFK is positioning engine based on Kalman filter. It could be used in standalone or code differential positioning modes with or without SBAS. KFK provides sufficient increase of positioning accuracy, availability, continuity, and integrity, especially:

- for mobile dynamic user
- in case of unfavorable environment conditions (dense canopy, city canyon, etc.)

KFK Mode

Name: /par/pos/kfk/mode
Access: rw
Type: boolean
Values: on,off
Default: off

on - use KFK engine for stand-alone and code-differential positioning modes.

off - use conventional least squares (LMS) based engine for stand-alone and code-differential positioning modes.

This parameter has no effect when receiver is in one of RTK positioning modes, i.e., when /par/pos/mode/cur parameter is set to either pf or pd. In this case the LMS engine is used to compute stand-alone or code-differential solution.

KFK Reset

Name: /par/pos/kfk/reset
Access: w
Type: boolean
Values: on

Setting this pseudo-parameter to on will initialize the KFK engine from conventional least squares based positioning engine.

KFK Dynamics Mode

Name: /par/pos/kfk/dynamic/mode
Access: rw
Type: enumerated
Values: static,walk,car,ship,aircraft,unlim,satellite,user,adapt
Default: car

The dynamic mode for KFK is defined by the values of horizontal and vertical components of dynamic noise variance and maximum velocity.

`static,walk,car,ship,aircraft,satellite,unlim` - use one of preset dynamic modes from the table below.

`user` - use the mode specified by the /par/pos/kfk/dynamic/user parameter.

`adapt` - use adaptive mode for dynamic parameters.

Table 4-1. Preset KFK Dynamic Modes

Mode	var(V_h) m^2/s^2	var(V_v) m^2/s^2	max(V_h) m/s	max(V_v) m/s
static	0.0001	0.0001	1	1
walk	1	1	5	5
car	10	10	70	20
ship	50	5	40	5
aircraft	100	100	500	100
satellite	10	10	20000	20000
unlim	10000	10000	20000	20000

User Defined KFK Dynamics

Name: /par/pos/kfk/dynamic/user
Access: rw
Type: list {VarVh:float,VarVv:float,
MaxVh:float,MaxVv:float}
Values: {[0.0001…10000],[0.0001…10000],[0…5000],[0…5000]}
Default: {10.0,10.0,70.0,20.0}

Using this parameter, the user can set KFK dynamics parameters according to his specific requirements. The values from this parameter take effect only when parameter /par/pos/kfk/dynamic/mode is set to the value `user`.

RAIM Parameters

RAIM Mode

Name: /par/pos/raim/mode
Access: rw
Type: boolean
Values: on,off
Default: on

on - RAIM is active

off - RAIM is turned off

Alarm Limit Mode

Name: /par/pos/raim/al/mode
Access: rw
Type: enumerated
Values: manual,npa,ter,enr
Default: manual

In the description of this parameter, *nmi* stands for International Nautical Mile, that is equal to 1852 meters.

npa - non-precision approach. Limit is equal to 0.3nmi.

ter - terminal. Limit is equal to 1.0nmi.

enr - en route. Limit is equal to 2.0nmi.

manual - RAIM will use the contents of the parameter /par/pos/raim/al/manual as the alarm limit.

Alarm Limit for Manual Mode

Name: /par/pos/raim/al/manual
Access: rw
Type: float [meters]
Values: [10…10000]
Default: 555.6 (it corresponds to "npa" mode);

This parameter specifies alarm limit value for the manual alarm limit mode. The default value of this parameter numerically corresponds to the value used in npa mode.

Filtering Position Estimates

Position Filter Mode

Name: /par/pos/filt/mode
Access: rw
Type: boolean
Values: on,off
Default: off

Position Filter Type

Name: /par/pos/filt/type
Access: r
Type: enumerated
Values: stat

stat - simple N-point weighted moving average is used to smooth the current position.

Note that this type of position filter normally applies only to static (or "nearly static") receivers in sp or cd mode. It is especially useful when the number of tracked satellites changes abruptly; in this case the position accuracy may temporarily drop dramatically unless the position filter is on. For moving receivers, using this type of position filter may adversely affect the receiver trajectory's accuracy.

Position Filter Width

Name: /par/pos/filt/num
Access: rw
Type: integer
Values: [1…10000]
Default: 30

This parameter designates the number of the preceding pre-filtered position measurements used by the least-squares estimator to obtain the current smooth position. Provided /par/pos/filt/type is stat (the only one currently supported), this estimator is just an N-point weighted running average.

Maximum Allowed Time Gap

Name: /par/pos/filt/maxdel
Access: rw
Type: integer [seconds]
Values: [1…3600]
Default: 10

This parameter specifies the maximum allowed time gap mask for two successive position estimates. If the current position estimate is obtained in more than maximum allowed time gap seconds after the previous one, the position filter is reset.

Reset the Position Filter

Name: `/par/pos/filt/reset`
Access: `rw`
Type: `boolean`
Values: `on,off`

`on` – the position filter will be reset. Immediately after the reset, the parameter will automatically be set back to `off`.

`off` – ignored.

Position Filter Weighting Mode

Name: `/par/pos/filt/weight`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`off` – position filter will treat all of the position estimates as uniformly weighted measurements.

`on` – each position estimate will have its own weight depending on the corresponding *rms* error.

Improved Timing Mode

Running receiver in the Improved Timing mode serves two main purposes:

- It enables you to synchronize your receiver with the GNSS time scales even when you have only one satellite in sight.
- This mode provides better synchronization precision as compared to the general case when the receiver has to solve for both coordinates and time offsets.

To correctly use the Improved Timing mode, you need to specify reference coordinates of the receiver antenna L1 phase center as precise as possible. Refer to "Reference Parameters" on page 332 for details. Note that 1 meter of error in reference position will result in about 3.3 nanoseconds error in time offset.

Enable Improved Timing Mode

Name: /par/pos/clk/fixpos
Access: rw
Type: boolean
Values: on, off
Default: off

on - receiver will use the apriori known coordinates from the /par/ref/syspos/gps and /par/ref/syspos/glo parameters to solve for the unknown time offsets between corresponding system time scale and receiver time.

off - receiver will calculate the time offsets as part of usual position computation.

Time Scales Threshold for Gradual Recovery

Name: /par/pos/clk/sync/threshold
Access: rw
Type: integer [ns]
Values: [0…1000000]
Default: 0

If, after positioning is lost and then restored, the computed difference between extrapolated time scale and reference time scale exceeds the threshold specified by this parameter, the extrapolated time scale will be immediately corrected by the computed difference, otherwise correction will be limited by the value of /par/pos/clk/sync/rate parameter.

Setting this parameter to non-zero value will cause gradual approach of extrapolated time scale back to the reference time scale when positioning is restored.

The default value 0 effectively disables gradual time recovery, as any value will be above the threshold 0. The extrapolated time scale will be always corrected immediately in this case.

Time Scales Rate of Gradual Recovery

Name: /par/pos/clk/sync/rate
Access: rw
Type: integer [ns/s]
Values: [1…1000000]
Default: 10000

This parameter defines the rate of recovery of extrapolated time scale to the reference time scale after positioning is restored.

If computed time difference does not exceed the value specified by `/par/pos/clk/sync/threshold` parameter, the extrapolated time scale won't be immediately corrected to the reference one. Instead, the value of this parameter will be (repeatedly) used to gradually bring extrapolated time scale to the reference one.

If the computed time difference exceeds the value specified by `/par/pos/clk/sync/threshold` parameter, the entire correction will be applied immediately. This provides protection against large periods of time when approximated time scale significantly differs from the reference one.

Pulse Per Second (PPS) Parameters

Overview

JAVAD GNSS receivers can generate precise Pulse Per Second (PPS) signals with programmable reference time system, period and offset. PPS signals are available via the corresponding output connector pins.

Note: In static applications where the receiver's precise position is known, we recommend that you switch your receiver to the Improved Timing mode to improve precision of PPS output. Refer to "Improved Timing Mode" on page 270 for details.

The PPS time grid is defined by the PPS `period` as follows:

$$\text{reference\_time} \ \textit{mod}\ \texttt{period} = 0$$

In addition PPS *offset* allows you to generate PPS shifted with respect to the PPS time grid. Positive values of *offset* means shift to the future.

There could be up to two PPS outputs in JAVAD GNSS receivers, "a" (PPSA) and "b" (PPSB). It is possible to use both PPS outputs concurrently.

Due to a hardware limitation, PPS signals are discrete with a resolution dependent on particular receiver model[1]. JAVAD GNSS receivers, however, allow compensation for this "discreteness error". You can force the receiver to generate for each PPS pulse a message containing the offset between the scheduled PPS time and the actual pulse edge's arrival time. Refer to "[ZA], [ZB] PPS Offset" on page 128 for details.

Parameters

In this section, the notation `[a|b]` denotes either PPSA or PPSB. The user should substitute either `a` or `b`.

1. Typical values are a few nanoseconds.

Enable PPS Generation

Name: /par/dev/pps/[a|b]/out
Access: rw
Type: boolean
Values: on,off
Default: on

on - corresponding PPS will be generated

off - corresponding PPS is disabled

PPS Reference Time

Name: /par/dev/pps/[a|b]/time
Access: rw
Type: enumerated
Values: utc,gps,utcusno,glo,utcsu,sbas,utcsbas,gal,utcgal, comp,utcbei,qzss,utcqzss,irnss,utcind
Default: utc

utc - select the best from all available UTC time scales. [XA] and [XB] messages will contain identifier of the actual reference time selected for each particular PPS.

gps, utcusno - GPS system time, UTC(USNO)

glo, utcsu - GLONASS system time, UTC(SU)

sbas, utcspbas - SBAS system and UTC times

gal, utcgal - Galileo system and UTC times

comp, utcbei - BeiDou system and UTC times

qzss, utcqzss - BeiDou system and UTC times

irnss, utcind - IRNSS system and UTC times

When this parameter is set to gps or utcusno, there should be at least one GPS satellite being locked for receiver to be able to correctly synchronize to corresponding time scale. Similar rule holds true for the rest of the systems.

Tie PPS to its Reference Time

Name: /par/dev/pps/[a|b]/tied
Access: rw
Type: boolean
Values: on,off
Default: on

There are applications where the user needs to synchronize PPS signals with either the receiver's internal clock or an external frequency, not with the selected reference time. This parameter allows not to tie PPS to its reference time.

`on` – PPS pulses are synchronized with the selected reference time, which is the common practice.

`off` – PPS signals are synchronized either with the receiver's internal clock or, provided the parameter `/par/frq/mode` has been set to `use`, with the external frequency.

PPS Period

Name: `/par/dev/pps/[a|b]/per/ms`
Access: `rw`
Type: `integer [milliseconds]`
Values: `[10…10^9]`
Default: `1000`

Milliseconds of PPS Offset

Name: `/par/dev/pps/[a|b]/offs/ms`
Access: `rw`
Type: `integer [milliseconds]`
Values: `[-10^9/2…10^9/2]`
Default: `0`

Nanoseconds of PPS Offset

Name: `/par/dev/pps/[a|b]/offs/ns`
Access: `rw`
Type: `integer [nanoseconds]`
Values: `[-10^6/2…10^6/2]`
Default: `0`

PPS Reference Edge

Name: `/par/dev/pps/[a|b]/edge`
Access: `rw`
Type: `enumerated`
Values: `rise,fall`
Default: `rise`

`rise` – rising edge of the PPS pulse will be tied to the reference time

`fall` – falling edge of the PPS pulse will be tied to the reference time

PPS Pulse Length

Name: `/par/dev/pps/[a|b]/len`
Access: `rw`
Type: `integer [nanoseconds]`
Values: `[20…MAX] (receiver dependent)`
Default: `(receiver dependent) 10^6 or 2×10^6 typical`

This parameter specifies PPS pulse length. Actual PPS length will be set to the closest possible value according to particular hardware limitations.

Period of Marked PPS Pulses

Name: /par/dev/pps/[a|b]/mper
Access: rw
Type: integer [milliseconds]
Values: 0,[20…10^9]
Default: 0

The JAVAD GNSS receiver can generate either or both normal and marked PPS pulses. This parameter specifies the period of the marked PPS signal.

0 – receiver will generate no marked pulses.

NM=[20…10^9] – provided the parameter governing the period of normal pulses is set to NN, then the receiver will generate both normal and marked pulses, but marked pulses will be output only every N milliseconds, where N is equal to the least common multiple of NN and NM.

Length of Marked PPS Pulses

Name: /par/dev/pps/[a|b]/mlen
Access: rw
Type: integer [nanoseconds]
Values: [20…MAX] (receiver dependent)
Default: (receiver dependent) 2×10^9 or 3×10^9 typical

This parameter specifies the length of marked PPS pulses. Actual PPS length will be set to the closest possible value according to particular hardware limitations.

Use Signal Propagation Delay in PPS

Name: /par/dev/pps/[a|b]/rfdel
Access: rw
Type: boolean
Values: on,off
Default: on

on – to compute time moment for PPS output, receiver will take into account signal propagation delay due to receiver RF part schematic. Propagation delay is hard-coded number (for given receiver) which approximates actual delay with about +-10ns accuracy.

off – receiver will assume zero propagation delay for the purposes of PPS output.

If global /par/raw/rfdel parameter is set to on, signal propagation delay will be effectively taken into account no matter what the value of this PPS-specific parameter is.

External Event Parameters

Overview

JAVAD GNSS receivers have the “event marking” function allowing the user to measure/record event times in the specified reference time system with high accuracy. You may have your JAVAD GNSS receiver measure the time of either the rising edge or falling edge of the input event signal. Most of the JAVAD GNSS receivers may accommodate up to two external event pins, EventA and EventB.

The measured event times are buffered[1] inside the receiver and could be then output by the corresponding receiver message(s), please see “[XA], [XB] External Event” on page 128 for details.

Parameters

In this section, the notation `[a|b]` denotes either EventA or EventB. The user should substitute either `a` or `b`.

Enable Event Acquisition

Name: `/par/dev/event/[a|b]/in`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – corresponding event input is active and events will be acquired and buffered.

`off` – corresponding event input will be inactive.

Event Reference Time

Name: `/par/dev/event/[a|b]/time`
Access: `rw`
Type: `enumerated`
Values: `utc,gps,utcusno,glo,utcsu,sbas,utcsbas,gal,utcgal,comp,utcbei,qzss,utcqzss,irnss,utcind`
Default: `utc`

`utc` – select the best from all available UTC time scales. [ZA] and [ZB] messages will contain identifier of the actual reference time selected for each particular Event.

`gps, utcusno` – GPS system time, UTC(USNO)

`glo, utcsu` – GLONASS system time, UTC(SU)

1. The internal buffer will hold up to 128 most recent events.

`sbas, utcspbas` - SBAS system and UTC times

`gal, utcgal` - Galileo system and UTC times

`comp, utcbei` - BeiDou system and UTC times

`qzss, utcqzss` - BeiDou system and UTC times

`irnss, utcind` - IRNSS system and UTC times

When this parameter is set to `gps` or `utcusno`, there should be at least one GPS satellite being locked for receiver to be able to correctly synchronize to corresponding time scale. Similar rule holds true for the rest of the systems.

Tie Measured Event Time to its Reference Time

Name: `/par/dev/event/[a|b]/tied`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

With this parameter, the receiver is instructed to measure the event reception time in the selected reference time with or without consideration for the computed receiver clock offset.

`off` - the event time is measured in the receiver time scale that will differ from the selected reference time by the computed clock offset.

`on` - the event time is measured in the selected reference time properly. Thus the name of the parameter, tied (figuratively speaking, we "tie up" event signals rigidly with the selected reference time).

Event Reference Edge

Name: `/par/dev/event/[a|b]/edge`
Access: `rw`
Type: `enumerated`
Values: `rise,fall`
Default: `rise`

`rise` - the time of the rising edge of the event signal will be measured

`fall` - the time of the falling edge of the event signal will be measured

Synchronize Receiver Clock with External Event

Name: /par/dev/event/[a|b]/lock
Access: rw
Type: enumerated
Values: off,on,always,calib
Default: off

off - receiver will not synchronize its clock with external event.

on - receiver will synchronize its one-millisecond cycle grid with the corresponding edge of the next event signal arrived after setting this parameter to on. You may need to set this parameter to on repeatedly to ensure that the receiver maintains synchronization with the time scale governing the external event signals.

always - receiver will synchronize its one-millisecond cycle grid with the corresponding edge of each event signal arrived after setting this parameter to always.

calib[1] - provided /par/dev/event/[a|b]/frqdel is 0, automatically determine delay between positive zero-crossing of external frequency and external Event, measured at receiver inputs, and synchronize using this value. Accuracy of this synchronization (calibration) is about 0.3 [ns]. If /par/dev/event/[a|b]/frqdel is non-zero, use its value for synchronization instead.

Status of the Receiver Clock Synchronization

Name: /par/dev/event/[a|b]/locked
Access: r
Type: boolean
Values: on,off

on - receiver time has been synchronized with an external event.

off - receiver time is not synchronized with external event.

Event Input Impedance

Name: /par/dev/event/[a|b]/imp
Access: rw
Type: enumerated
Values: high, low
Default: high

high - high input impedance, about 5 KOhm

low - low input impedance, about 50 Ohm

Note: Few hardware supports this setting. In particular, TRE-3 rev.5 and higher does.

1. Only special timing receivers (TRE-3 for the moment of writing) have this mode implemented

Event to External Frequency Delay

Name: /par/dev/event/[a|b]/frqdel
Access: rw
Type: float [nanoseconds]
Values: [0…1000]
Default: 0

Specifies delay between positive zero-crossing of external frequency and external Event, measured at receiver inputs.

This parameter is only active when /par/dev/event/[a|b]/lock is set to calib.

Suppose external frequency is used, and PPS is connected to "ExternalEvent" input of the receiver. Provided phase shift between these two inputs is known to the user, specifying this shift in this parameter will allow receiver to synchronize its internal time scale with external Event with accuracy equal to accuracy of the entered number.

Event Offset

Name: /par/dev/event/[a|b]/offs/ns
Access: rw
Type: integer [nanoseconds]
Values: [-500000…+500000]
Default: 0

This offset will be added to external Event time.

Useful when external frequency is used, receiver time is synchronized to external PPS, and phase relationship between these two inputs happens to be unfortunate enough for receiver time to jump between two neighbor stable points.

Use Signal Propagation Delay in Event

Name: /par/dev/event/[a|b]/rfdel
Access: rw
Type: boolean
Values: on,off
Default: on

on - to compute time moment of Event reception, receiver will take into account signal propagation delay due to receiver RF part schematic. Propagation delay is hard-coded number (for given receiver) which approximates actual delay with about +-10ns accuracy.

off - receiver will assume zero propagation delay for the purposes of Event acquisition.

If global /par/raw/rfdel parameter is set to on, signal propagation delay will be effectively taken into account no matter what the value of this Event-specific parameter is.

Current Time

Parameters described in this section hold current time that is updated every millisecond. To receive consistent values of multiple parameters, use single print command to retrieve multiple values, for example:

Example: Get consistent UTC time and date:

```
⇒ print,/par/time/utc
⇐ RE019{2006-12-26,14:43:11.269}
```

Example: Get snapshot of all the times along with their names:

```
⇒ print,/par/time:on
⇐ RE018/par/time={rcv=53091337,
  RE02A utc={date=2006-12-26,clock=14:44:37.337},
  RE01D gps={week=383,ms=225891337},
  RE01C glo={day=1091,ms=63877337}}
```

Current Times

Name: /par/time
Access: r
Type: list {rcv,utc,gps,glo}

rcv – receiver time

utc – UTC time

gps – GPS system time

glo – GLONASS system time

Current Receiver Time

Name: /par/time/rcv
Access: r
Type: integer [milliseconds]
Values: [0…86400000)

This parameter reports the current time in local (i.e., receiver) time scale.

Current UTC Time

Name: /par/time/utc
Access: r
Type: list {date,clock}

`date` - UTC date

`clock` - UTC clock (time of day)

Current UTC Date

Name: `/par/time/utc/date`
Access: `r`
Type: `string`

This parameter reports the current UTC date and this date is represented as `YYYY-MM-DD`, where:

`YYYY` - the year in the Gregorian calendar between 0001 and 9999

`MM` - the month of the year between 01 (January) and 12 (December)

`DD` - the day of the month between 01 and 31. If no time information is available, the receiver reports an empty string.

Current UTC Time of Day

Name: `/par/time/utc/clock`
Access: `r`
Type: `string`

This parameter reports the current time of day in UTC as a value in the `HH:MM:SS.SSS` or empty string format, depending on whether the time information is available or not.

`HH` - hours [00-23]

`MM` - minutes [00-59]

`SS` - seconds [00-60][1]

`SSS` - milliseconds [000-999]

Current GPS Time

Name: `/par/time/gps`
Access: `r`
Type: `list {week,ms}`

`week` - GPS week number

`ms` - GPS time inside week

1. It could be equal to 60 only when UTC leap second happens.

Current GPS Week

Name: /par/time/gps/week
Access: r
Type: integer
Values: [-1…1023]

This parameter reports the current GPS week number. If no time information is available, the receiver reports -1.

Current GPS Time of Week

Name: /par/time/gps/ms
Access: r
Type: integer [milliseconds]
Values: [-1…604800000)

This parameter reports the current GPS time of week. If no time information is available, the receiver reports -1.

Current GLONASS Time

Name: /par/time/glo
Access: r
Type: list {day,ms}

day - GLONASS day number

ms - GLONASS time inside day

Current GLONASS Day Number

Name: /par/time/glo/day
Access: r
Type: integer
Values: [-1,1…1461]

This parameter reports the current GLONASS day number within the 4-year period beginning with the leap year. If no time information is available, the receiver reports -1.

Current GLONASS time of day

Name: /par/time/glo/ms
Access: r
Type: integer [milliseconds]
Values: [-1…86400000)

This parameter reports the current time of day in GLONASS system time. If no time information is available, the receiver reports -1.

4.4.9 Code Differential (DGPS) Parameters

Generic DGPS Parameters

Code Differential Corrections Type

Name: /par/pos/cd/type
Access: rw
Type: enumerated
Values: rtcm,slas
Default: rtcm

rtcm - use classic RTCM/RTCM3/CMR corrections

slas - use QZSS Sub-meter Level Augmentation Service (SLAS) corrections

Source Port of DGPS Corrections

Name: /par/pos/cd/port
Access: rw
Type: enumerated
Values: any,/[port]
Default: any

any - receiver will use differential corrections from whichever port.

/[port] - receiver will only use differential corrections from corresponding port.

Source of DGPS Corrections

Name: /par/pos/cd/src/mode
Access: rw
Type: enumerated
Values: user,best,nearest,any
Default: nearest

user - receiver will use corrections with station ID specified by the parameter /par/pos/cd/src/usersrc.

nearest - receiver will use corrections from the nearest reference station.

best - receiver will use reference station with minimal estimated RMS of navigation solution. Navigation satellites which don't have corrections from this source won't be used in position solution.

any - receiver will use reference station with minimal estimated RMS of navigation solution. Navigation satellites which don't have corrections from this source will get corrections from another source with larger estimated RMS, if possible.

Generate DGPS Corrections from RTK

Name: /par/pos/cd/src/rtkdata
Access: rw
Type: boolean
Values: on,off
Default: off

on – DGPS corrections will be generated from RTCM3 and/or CMR RTK data from reference station.

off – DPGPS corrections won't be generated from RTK data.

Fixed Reference Station ID

Name: /par/pos/cd/src/usersrc
Access: rw
Type: integer
Values: [0…1023]
Default: 0

This parameter specifies user defined source of corrections by reference station ID. Receiver will only use corrections from reference station with given ID, provided /par/pos/cd/src/mode is set to user.

Maximum Age of DGPS Corrections

Name: /par/pos/cd/maxage
Access: rw
Type: integer [seconds]
Values: [1…1200]
Default: 30

Receiver will stop using differential corrections for DGPS solution when their age exceeds specified limit.

Ionosphere-free DGPS Mode

Name: /par/pos/cd/ionofree
Access: rw
Type: boolean
Values: on,off
Default: off

off – receiver will use RTCM ionosphere corrections for DGPS solution. I.e., it will not use data from RTCM message type 15.

on – receiver will not use RTCM ionosphere corrections, provided /par/pos/sp/meas parameter is set to ionofree. I.e., it will use data from RTCM message type 15 to get rid of RTCM ionosphere corrections.

Maximum Age of Ionosphere Corrections

Name: /par/pos/cd/iono/maxage
Access: rw
Type: integer [seconds]
Values: [1…1800]
Default: 300

Receiver will stop using ionosphere corrections for DGPS solution when their age exceeds specified limit.

Range Residual Limit

Name: /par/pos/cd/rlim
Access: rw
Type: float [meters]
Values: [1.0…100.0]
Default: 5.0

Satellites whose range residuals are greater than this limit will be discarded from code differential positioning. This parameter is used only if the parameter /par/pos/raim/mode has been set to on.

Corrections to Reference Stations Coordinates

Name: /par/rover/base/pos/par/X (X=[a…e])[1]
Access: rw
Type: {valid,port,basedID,delNorth,delEast,delUp}

valid - boolean [on|off]. If valid is off, the parameter is associated with no reference station.

port - this field takes same values as, for example, the parameter /par/pos/cd/port. It specifies the port from which the coordinates to be corrected are coming.

baseID - integer [1…1023]. It is identification of the reference station for which the offsets are specified. If baseID is -1, the offsets are applicable to all reference stations used in multi-base mode except the stations for which offsets are already specified with the preceding parameters (see the example below).

delNorth, delEast, delUp - float numbers specifying the north-, east- and up-components of the vector correction to the reference station position in meters.

Each specification from a to e in turn is compared to the port and identifier of the reference station. The first specification that has valid field set to on and which port and

1. Currently GREIS provides only five parameters for coordinate offsets since only up to five reference stations can be used in multi-base mode.

baseId match those of the reference station, provides offsets for the coordinates of this reference station. If no specifications match, the offsets are assumed to be zero.

You can use these parameters to compensate for known "coordinate offsets" of up to five reference stations used in multi-base code differential mode. This correction mechanism is especially important when differential corrections from different reference stations are used together to compute rover position in the mixed solution mode (for details, see parameter /par/pos/cd/src/mode on page 283). Should the transmitted reference coordinates of the base stations be significantly different from their "true" coordinates in this mode, the estimated rover position may prove corrupt unless the user enters appropriate "coordinate offsets" for these stations on the rover side.

Example:
```
⇒ set,rover/base/pos/par/a,{on,/dev/ser/a,any,-0.02,0.033,-0.05}
⇒ set,rover/base/pos/par/b,{on,any,101,0.01,-0.034,-0.22}
⇒ set,rover/base/pos/par/c,{on,any,102,0.002,0.023,-0.011}
```

With these commands, the receiver will apply the offset (-0.02,0.03,-0.05) to the transmitted coordinates of the reference station whose differential corrections are received on serial port A (whatever the station ID). It will apply corresponding corrections to reference stations with IDs 101 and 102, unless data for them are coming from serial port A, in which case the first offsets apply.

SLAS Parameters

This parameters govern the use of QZSS Sub-meter Level Augmentation Service (SLAS) corrections. To be used, the data from given ground station should be enabled by its station ID, and at least 1 satellite, enabled by its PRN and transmitting data for this station, should be tracked.

Enable SLAS Signals as Source of Corrections

Name: /par/pos/cd/slas/sat
Access: rw
Type: array [183…189] of boolean
Values: {y|n,…,y|n}
Default: {y,…,y}

Enable SLAS Signals as Source of Corrections for given PRN

Name: /par/pos/cd/slas/sat/N (N=183…189)
Access: rw
Type: boolean
Values: y,n
Default: n

`y` – enable using of SLAS signal as source of corrections (independently from selected SLAS ground station)

`n` – disable using of SLAS signal as source of corrections (independently from selected SLAS ground station)

Enable SLAS Ground Stations as Sources of Corrections

Name: `/par/pos/cd/slas/stid`
Access: `rw`
Type: `array [0…14] of boolean`
Values: `{y|n,…,y|n}`
Default: `{y,…,y}`

Enable SLAS Ground Station ID as Source of Corrections

Name: `/par/pos/cd/slas/stid/N (N=0…14)`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

`y` – enable using of SLAS ground station as source of corrections (independently from selected SLAS signal PRN)

`n` – disable using of SLAS ground station as source of corrections (independently from selected SLAS signal PRN)

SBAS Parameters

SBAS stands for Satellite Based Augmentation System. JAVAD GNSS receivers support multiple implementations of SBAS: Wide Area Augmentation System (WAAS), European Geostationary Navigation Overlay Service (EGNOS), GPS Aided Geo Augmented Navigation (GAGAN), Japanese Multi-functional Satellite Augmentation System (MSAS), and Russian System for Differential Correction and Monitoring (SDCM). Parameters described in this section define when and how SBAS data will be used by the receiver to increase positioning accuracy.

SBAS Mode

Name: `/par/pos/wd/mode`
Access: `rw`
Type: `enumerated`
Values: `manual,npa,ter,enr`
Default: `manual`

This parameter defines the mode of use of SBAS corrections for DGPS solution.

`manual` – receiver will apply SBAS corrections according to other parameters defined in this section.

`npa,ter,enr` – receiver will apply SBAS corrections according to the DO-229C/D specification.

Enable SBAS Mode for Non-safety Applications

Name: `/par/pos/wd/nsa`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – receiver will apply SBAS corrections as for non-safety application. Currently receiver will interpret non-empty message type 0 as message type 2 in accordance with DO-229C/D. Note that this mode is used only when parameter `/par/pos/wd/mode` is set to `manual`.

`off` – receiver will apply SBAS corrections as for safety applications.

SBAS Elevation Mask

Name: `/par/pos/wd/elm`
Access: `rw`
Type: `integer`
Values: `[-90…90]`
Default: `5`

SBAS corrections from satellites with elevations lower than this mask will be excluded from position computation.

SBAS Corrections Type

Name: `/par/pos/wd/type`
Access: `rw`
Type: `enumerated`
Values: `sbas,comp`
Default: `sbas`

`sbas` – use usual sources of SBAS corrections

`comp` – use BeiDou as the source of SBAS corrections

Enable SBAS Corrections by Satellite Numbers

Name: /par/pos/wd/sat
Access: rw
Type: array [120…147] of boolean
Values: {y|n,…,y|n}
Default: {y,…,y}

Enable SBAS Corrections from Satellite Number N

Name: /par/pos/wd/sat/N (N=[120…147])
Access: rw
Type: boolean
Values: y,n
Default: y

y - enable using of SBAS satellite number N as source of corrections

n - disable using of SBAS satellite number N as source of corrections

Enable SBAS Provider

Name: /par/pos/wd/sbas/provider
Access: rw
Type: array [0…15] of boolean
Values: {y|n,…,y|n}
Default: {y,…,y}

Enable SBAS Provider N

Name: /par/pos/wd/sbas/provider/N (N=[0…15])
Access: rw
Type: boolean
Values: y,n
Default: y

y - enable SBAS provider N

n - disable SBAS provider N

Enable SBAS Corrections by Satellite Numbers

Name: /par/pos/wd/sbas/sat
Access: rw
Type: array [120…147] of boolean
Values: {y|n,…,y|n}
Default: {y,…,y}

Enable SBAS Corrections From Satellite Number N

Name: /par/pos/wd/sbas/sat/N (N=[120…147])
Access: rw
Type: boolean
Values: y,n
Default: y

y – enable corrections from SBAS satellite N

n – disable corrections from SBAS satellite N

Enable BeiDou Wide-area Corrections by Satellite Numbers

Name: /par/pos/wd/comp/sat
Access: rw
Type: array [1…30] of boolean
Values: {y|n,…,y|n}
Default: {y,…,y}

Enable BeiDou Wide-area Corrections from Satellite Number N

Name: /par/pos/wd/comp/sat/N (N=[1…30])
Access: rw
Type: boolean
Values: y,n
Default: y

y – enable using of BeiDou signal of BeiDou Satellite Number N as the source of wide-area corrections

n – disable using of BeiDou signal of BeiDou Satellite Number N as the source of wide-area corrections

Enable SISNeT Source

Name: /par/pos/wd/sisnet
Access: rw
Type: boolean
Values: y,n
Default: y

y – enable SISNeT source for SBAS DGPS mode

n – disable SISNeT source for SBAS DGPS mode

Source of SBAS Corrections

Name: /par/pos/wd/src/mode
Access: rw
Type: enumerated
Values: user,best,any
Default: any

user - enable using of SBAS satellite with PRN specified by the parameter /par/pos/wd/src/usersrc.

best - receiver will choose SBAS satellite with minimal estimated RMS of navigation solution as source. Navigation satellites which don't have corrections from this source are not used in position solution.

any - receiver will choose SBAS satellite with minimal estimated RMS of navigation solution as source. Navigation satellites which don't have corrections from this source will get corrections from another source with larger estimated RMS if possible.

Note: This parameter takes effect only when /par/pos/wd/mode is set to manual.

Fixed SBAS Satellite

Name: /par/pos/wd/src/usersrc
Access: rw
Type: integer
Values: [120…147]
Default: 120

Specify SBAS satellite number as user-defined source of corrections. This parameter is active only when /par/pos/wd/src/mode parameter is set to user.

Note: The SVs specified by this parameter should also be enabled by the /par/pos/wd/sat parameter in order to be used as source of corrections.

Enable SBAS Ionosphere Corrections

Name: /par/pos/wd/ionofree
Access: rw
Type: boolean
Values: off,on
Default: off

off - receiver will use ionosphere corrections from SBAS satellites for DGPS solution.

on - receiver will not use SBAS ionosphere corrections, provided /par/pos/sp/meas parameter is set to ionofree.

Maximum Age of SBAS Satellite Corrections

Name: /par/pos/wd/maxage
Access: rw
Type: integer [seconds]
Values: [1…3600]
Default: 360

Receiver will stop using SBAS satellite corrections for DGPS solution when their age exceeds specified limit.

Note: This limit is active for SBAS fast corrections as well as for SBAS long-term corrections.

Note: This parameter takes effect only when /par/pos/wd/mode is set to manual.

Maximum Age of SBAS Ionosphere Corrections

Name: /par/pos/wd/iono/maxage
Access: rw
Type: integer [seconds]
Values: [1…3600]
Default: 1200

Receiver will stop using SBAS ionosphere corrections for DGPS solution when their age exceeds specified limit.

Note: This parameter takes effect only when /par/pos/wd/mode is set to manual.

Smoothing Interval of SBAS Satellite Corrections

Name: /par/pos/wd/smi
Access: rw
Type: integer [seconds]
Values: [0…3600]
Default: 60

Receiver will smooth SBAS satellite corrections (the sum of SBAS fast corrections and SBAS long-term corrections) before their use for DGPS solution. This parameter defines the value of time constant of the smoothing filter. Zero value stops the smoothing.

Note: This parameter takes effect only when /par/pos/wd/mode is set to manual.

Smoothing Interval of SBAS Ionosphere Corrections

Name: /par/pos/wd/iono/smi
Access: rw
Type: integer [seconds]
Values: [0…3600]
Default: 60

Receiver will smooth SBAS ionosphere corrections before their use for DGPS solution. This parameter defines the value of time constant of the smoothing filter. Zero value stops the smoothing.

Note: This parameter takes effect only when `/par/pos/wd/mode` is set to `manual`.

4.4.10 Phase Differential (RTK) Parameters

Generic RTK Parameters

RTK Position Computation Mode

Name: `/par/pos/pd/mode`
Access: `rw`
Type: `enumerated`
Values: `extrap,delay`
Default: `delay`

`extrap` - in this mode the RTK engine will extrapolate the latest carrier phases received from reference station to the current time. The final positioning accuracy may be somewhat lower due to additional extrapolation errors, which may be up to a few millimeters vertical and horizontal for a one-second extrapolation time. Note that this mode could be used only when reference station is static (i.e., not moving).

`delay` - in this mode, the RTK engine does not extrapolate the base station's carrier phases in position computation. Instead, the engine will either compute a delayed position or simply output the current stand-alone position (while waiting for new differential messages from the base station). Note that the delayed position is computed for the time (epoch) to which the last received base station's carrier phase measurements correspond. Accuracies achievable in delay mode are normally on a level with those of post-processing kinematic.

RTK Delay Mode Variant

Name: `/par/pos/pd/delay`
Access: `rw`
Type: `enumerated`
Values: `last,every`
Default: `last`

This parameter is only active for RTK *delay* mode.

`last` - RTK engine will process the last set of carrier phase differential data received from the reference station.

every - RTK engine will attempt to process all sets of carrier phase differential data sequentially received from the reference station.

Multi-RTK Mode

Name: /par/pos/pd/multirtk
Access: rw
Type: boolean
Values: on,off
Default: <receiver dependent>

on - receiver will run multiple RTK engines with different settings in parallel to select the best solution, for single baseline. The multi-base mode is disabled.

off - receiver will run single RTK engine for every baseline (multi-base mode).

RTK Datum Conversion Mode

Name: Name: /par/pos/pd/local
Access: Type: boolean
Type: Access: rw
Values: Values: on,off
Default: Default: on

on - RTK solution will be converted to local datum specified by /par/pos/datum, similar to other solution types.

off - RTK solution is expected to already be in local datum specified by /par/pos/datum, so no conversion will be applied to RTK solution.

RTK Solution Period

Name: /par/pos/pd/per/sol
Access: rw
Type: float [seconds]
Values: [0…100]
Default: 0

This parameter limits update rate of RTK solution by specified period. Default 0 ensures that RTK is updated as fast as possible, according to other parameters.

Period of RTK Reference Station Output

Name: /par/pos/pd/per/ref
Access: rw
Type: float [seconds]
Values: [0.05…100]
Default: 1

This parameter is effective only in the RTK *delay* mode. Its value should be set to the exact rate at which the base station transmits its differential correction data. This parameter will instruct the rover receiver to output the RTK position at the same rate at which differential corrections are updated.

Period of Ambiguities Estimation

Name: `/par/pos/pd/per/af`
Access: `rw`
Type: `float [seconds]`
Values: `[0.05…5]`
Default: `1`

Period of Base Measurements for Extrapolation

Name: `/par/pos/pd/per/ex`
Access: `rw`
Type: `float [seconds]`
Values: `[0…5]`
Default: `0.1`

The RTK engine will extrapolate the received reference station's carrier phase measurements if the time to which these carrier phase measurements correspond is divisible by the value of this parameter.

Extrapolate Missing Carrier Phase Measurements

Name: `/par/pos/pd/se`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

This parameter is only active when `/par/pos/pd/mode` is set to `delay`.

`on` - in delay mode, RTK engine will extrapolate missing carrier phase measurements for the currently processed epoch (provided that the code measurements have been successfully received for this epoch).

`off` - RTK won't extrapolate missing carrier phase measurements in delay mode.

Maximum Time Gap in Reference Data

Name: `/par/pos/pd/timegap`
Access: `rw`
Type: `integer [seconds]`
Values: `[-1…3600]`
Default: `-1`

`[0…3600]` – RTK will be reset as soon as duration of time gap in the data being received from reference station exceeds specified value.

`-1` – RTK is never reset due to time gaps.

Check Reference Position Change

Name: `/par/pos/pd/crpc`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – RTK engine will be reset whenever change of reference station position is detected, provided reference measurements are being extrapolated, either due to `extrap` value of `/par/pos/pd/mode` parameter, or `on` value of `/par/pos/pd/se` parameter.

`off` – RTK engine will never be reset because of change of reference station position. Setting this value makes it possible to use RTK extrapolation mode with quasi-static reference station, to achieve high update rates.

Source Port of Differential Data

Name: `/par/pos/pd/port`
Access: `rw`
Type: `enumerated`
Values: `any,/[port]`
Default: `any`

`any` – RTK engine will use differential data from whichever port.

`/[port]` – RTK engine will only use differential data received on the corresponding port.

Confidence Level for Ambiguity Fixing

Name: `/par/pos/pd/aflevel`
Access: `rw`
Type: `enumerated`
Values: `low,medium,high`
Default: `medium`

`low` – 95% confidence level

`medium` – 99.5% confidence level

`high` – 99.9% confidence level

The higher the confidence level specified, the longer the integer ambiguity search time and the higher the reliability of the ambiguity fixed solution.

Known Point Initialization

Name: /par/pos/pd/fixpos
Access: rw
Type: boolean
Values: on,off
Default: off

on - RTK engine will use the rover's precise position for ambiguities resolution. This allows the engine to fix ambiguities much faster. The precise coordinates of the L1 phase center of the rover antenna must be specified as described in "Reference Parameters" on page 332. Care should be taken that this parameter is set back to off once the RTK initialization is over and the antenna starts moving. Otherwise the rover's position will be computed incorrectly.

off - RTK engine won't use rover precise position.

RTK Penalty Parameter

Name: /par/pos/pd/pen
Access: rw
Type: float
Values: [0…1000]
Default: 1

The penalty parameter is used for the known point initialization function.

Use CA/L1 Measurements for RTK

Name: /par/pos/pd/meas/ca
Access: rw
Type: boolean
Values: on,off
Default: on

Use P/L1 Measurements for RTK

Name: /par/pos/pd/meas/p1
Access: rw
Type: boolean
Values: on,off
Default: on

Use P/L2 Measurements for RTK

Name: /par/pos/pd/meas/p2
Access: rw
Type: boolean
Values: on,off
Default: on

Use CA/L2 Measurements for RTK

Name: /par/pos/pd/meas/c2
Access: rw
Type: boolean
Values: on,off
Default: on

Use L5 and E5a Measurements for RTK

Name: /par/pos/pd/meas/l5
Access: rw
Type: boolean
Values: on,off
Default: on

This parameter allows using of GPS L5 and Galileo E5a measurements in RTK processing.

Use L1 Only for RTK

Name: /par/pos/pd/l1only
Access: rw
Type: boolean
Values: on,off
Default: off

This parameter allows the receiver to stabilize the fixed position in case of poor L2 phase tracking.

on - RTK engine will calculate the final position using only L1 measurements even if the ambiguities have been resolved for L2 as well.

off - RTK will use L2 if possible.

RTK Weighting Scheme

Name: /par/pos/pd/scale
Access: rw
Type: integer
Values: 0,1,2
Default: 1

This parameter specifies the weights that the engine will apply to single-differenced (SD) carrier phase measurements.

0 – the simplest weighting scheme will be used. Specifically, all SD carrier phases will be used with the apriori weight *(1 / sigma2)*, where sigma is 0.05 cycles. This scheme, which is recommended when running RTK in favorable environment conditions, allows a shorter ambiguity fixing time than the other two (see below).

1 – the first adaptive weighting scheme will be used. Specifically, SD carrier phases for the given satellite will be used with the apriori weight:

$$max(0.2, sin(elev)) / sigma\_est^2$$

where elev designates the satellite's elevation angle, and sigma_est designates the RMS single-differenced carrier phase error.

2 – the second adaptive weighting scheme will be used. This is similar to scheme 1, but different estimator to get sigma_est is used. This scheme is strongly recommended in scenarios where strong multipath is expected.

Enable RTK Ionosphere Model

Name: /par/pos/pd/ion
Access: rw
Type: boolean
Values: on,off
Default: on

on – single-differenced ionosphere model is enabled

off – single-differenced ionosphere model is disabled

Threshold for RTK Ionosphere Model

Name: /par/pos/pd/ionr
Access: rw
Type: float [meters]
Values: [0…10^6)
Default: 10000

The ionospheric delay will be modeled by RTK engine only if the estimated baseline length is greater or equal to the specified value.

Memory Factor For the Float Ambiguity Filter

Name: /par/pos/pd/mem
Access: rw
Type: float
Values: [0.5…1.0]
Default: 0.99970

The smaller the filter memory factor, the "less important" are the older ambiguity estimates for the RTK engine estimating the current ambiguities. On shorter vectors (up to 8 km), it is recommended to set the memory factor to 0.998 when running the receiver under tree canopy.

Half-integer Ambiguity Fixing on L1 and L2

Name: /par/pos/pd/si
Access: rw
Type: list {L1, L2} of boolean
Values: {on|off,on|off}
Default: {off,off}

This is a technology (specialist) parameter. It is mostly used for internal JAVAD GNSS purposes when testing/debugging new firmware versions.

Interval of Verification of Fixed Ambiguities

Name: /par/pos/pd/check
Access: rw
Type: integer
Values: [-1…32767]
Default: 7

This parameter specifies the periodicity N of the engine checking fixed ambiguities for errors. The engine will be forced to recompute/check the ambiguity vector every N epochs. Thus estimated "forced ambiguities" will be compared against the current ones. If the forced ambiguity vector differs from the current ambiguity vector, the float solution mode will be enabled at once.

-1 - ambiguity verification is off.

0, 1 - RTK engine will verify ambiguities every epoch.

[2…32767] - RTK engine will verify ambiguities every specified number of epochs.

Rover Dynamics for RTK

Name: /par/pos/pd/dyn
Access: rw
Type: float
Values: [0…1]
Default: 1

[0…1) - RTK engine will run in static mode.

1 - RTK engine will run in kinematic mode.

When in static mode, the engine uses a running average over a few consecutive raw estimates to decrease the resulting position's noise error.

Note that the RTK engine also provides a “static watchdog mechanism”. When in static mode, the receiver will automatically monitor estimated coordinates X, Y, Z. Should the position change over 4 centimeters in one of the coordinates, the receiver will immediately switch to kinematic mode and will run in this mode for the next 30 seconds.

RTK Computational Scheme Version

Name: /par/pos/pd/ver
Access: r
Type: integer
Values: 2

Reset RTK Engine

Name: /par/pos/pd/reset
Access: w
Type: boolean
Values: on,off
Default: off

on - RTK engine will be reset and the value of this parameter will be set back to off.

off - ignored.

Reset Multi-RTK Engines Independently

Name: /par/pos/pd/resetonly
Access: w
Type: array [0…M] of boolean
Values: {y|n,…,y|n}
Default: {n,…,n}

M is the number of multi-RTK engines supported on given receiver.

Reset Multi-RTK Engine N

Name: /par/pos/pd/resetonly/N (N=[0…M])
Access: w
Type: boolean
Values: y,n
Default: n

y - NRTK engine number N will be reset and the value of this parameter will be set back to n.

n - ignored.

Reset Multi-RTK Engine That Fixed

Name: /par/pos/pd/resetfix
Access: rw
Type: boolean
Values: y,n
Default: n

y - When multi-RTK mode is on, reset of the engine that fixed.

n - do not reset engine that fixed.

Enable Use of Kept Reference Coordinates

Name: /par/pos/pd/ref/keep
Access: rw
Type: boolean
Values: on,off
Default: off

on - receiver will use reference coordinates retrieved from NVRAM at receiver startup for RTK.

off - receiver will not begin RTK processing prior to receiving reference coordinates from base station even if the rover has already received all the other necessary data/measurements from the base. Note that reference coordinates are normally transmitted much rarer than measurements and such delays may well be unacceptable for many applications.

Care should be taken when setting this parameter to on. Imagine for a moment that the rover has moved to a different location and started a new RTK session with a different reference station. Should this parameter be set to on, the rover receiver will be misusing the old reference coordinates for some time, which will most likely result in position blunders until the rover receives a first message with the correct reference coordinates.

Factor for Residual Ionosphere Standard Deviation

Name: /par/pos/pd/ionf
Access: rw
Type: float
Values: [0…10^6]
Default: 2

RTK uses the following equation for the residual ionosphere standard deviation (measured in meters):

$$\text{stdev\_iono\_delay} = \texttt{ionf} \times 10^{-6} \times \text{base\_line\_length}$$

where ionf is the value of this parameter.

Environmental Condition Factor

Name: /par/pos/pd/env
Access: rw
Type: enumerated
Values: open,forest
Default: open

open – RTK will use "normal" thresholds when searching for measurement outliers. This setting is used if the environment conditions are considered favorable for RTK (many satellites in sight, few obstructions and low multipath).

forest – RTK will use less rigid thresholds when filtering out measurement outliers. This mode is recommended when working under tree canopy or in other cases of high multipath.

Use Smoothed Pseudo-range in RTK

Name: /par/pos/pd/smr
Access: rw
Type: boolean
Values: on,off
Default: off

on – RTK engine will use smoothed rover pseudo-range measurements.

off – RTK engine will use raw rover pseudo-range measurements.

RTK Maximum Extrapolation Time

Name: /par/pos/pd/textr
Access: rw
Type: float
Values: [0…60]
Default: 30

When RTK works in the extrapolation mode and the time gap between the last received correction and the current rover's time exceeds this value, RTK stops to produce a position.

Maximum Number of Iterations for Float Ambiguity

Name: /par/pos/pd/maxit
Access: rw
Type: integer
Values: [0…10]
Default: 1

This parameter specifies maximum number of iterations RTK will make when estimating float ambiguity.

Maximum Number of Iterations for Fixed Ambiguity

Name: /par/pos/pd/maxitf
Access: rw
Type: integer
Values: [0…10]
Default: 1

This parameter specifies maximum number of iterations RTK will make when estimating residuals of fixed ambiguity.

Use FKP Data From RTCM 2.x Message 59

Name: /par/pos/pd/fkp
Access: rw
Type: boolean
Values: on,off
Default: off

on - instructs the rover receiver to use ionospheric and geometric corrections from RTCM 2.x messages type 59 (FKP) when computing position.

off - RTK engine will ignore FKP data.

Maximum Number of Satellites to Use in RTK

Name: /par/pos/pd/maxsat
Access: rw
Type: integer
Values: [4…44]
Default: 32

With this parameter, the user specifies maximum number of satellites that are used in RTK position computation. If the actual number of satellites in sight exceeds the current

parameter's value, the RTK engine will utilize data only from the satellites with the highest CA/L1 SNRs and the number of satellites used will not be greater than that specified by the parameter.

Drop in Maximum Number of SVs

Name: /par/pos/pd/drop
Access: rw
Type: integer
Values: [0…MAX], where MAX is receiver-dependent
Default: 16

When ambiguities fix is not yet achieved, decrease maximum number of satellites (with respect to /par/pos/pd/maxsat) by specified amount. This allows to keep low the number of supposedly problematic satellites to achieve better fix.

Use Base Doppler in RTK

Name: /par/pos/pd/usebasedop
Access: rw
Type: boolean
Values: on,off
Default: off

on – enable receiver to use the doppler measurements, either received from the base station or obtained by the base measurements extrapolator, in RTK processing.

off – disable the use of doppler measurements.

Enable Measurements Quality Indicators

Name: /par/pos/pd/qcheck
Access: rw
Type: boolean
Values: on,off
Default: on

on – RTK engine will take into account measurements quality indicators.

off – RTK will ignore measurements quality indicators.

RTK VRS Mode

Name: /par/pos/pd/vrs
Access: rw
Type: boolean
Values: on,off
Default: off

`on` – RTK will assume that the reference station in use is Virtual Reference Station (VRS)

`off` – RTK will assume that the reference station is real one.

Correlation Time for Estimating the Residual Ionosphere

Name: `/par/pos/pd/iont`
Access: `rw`
Type: `float [seconds]`
Values: `[0…1800]`
Default: `30`

Minimum CA/L1 SNR

Name: `/par/pos/pd/minpot`
Access: `rw`
Type: `integer [dB×Hz]`
Values: `[1…60]`
Default: `36`

Only those signals will be used in RTK position computation whose signal-to-noise ratios exceed the specified threshold value.

RMS for Pseudo-range Measurements Noise

Name: `/par/pos/pd/rmsc`
Access: `rw`
Type: `float [meters]`
Values: `[0.001…100]`
Default: `5.0`

RMS for Carrier Phase Measurements Noise

Name: `/par/pos/pd/rmsp`
Access: `rw`
Type: `float [cycles]`
Values: `[0.001…1]`
Default: `0.05`

Maximum Distance Between Base and Rover

Name: `/par/pos/pd/range`
Access: `rw`
Type: `float [meters]`
Values: `[0.0…1000000]`
Default: `1000000.0`

With this parameter, the user specifies the maximum allowed distance between the reference and rover stations. If this distance exceeds the specified limit, the rover receiver will stop to provide the RTK position.

Minimum Number of Satellites for Fixing Integer Ambiguities

Name: /par/pos/pd/minsat
Access: rw
Type: integer
Values: [4…20]
Default: 5

RTK Heading Parameters

Heading Mode

Name: /par/pos/pd/hd/mode
Access: rw
Type: boolean
Values: on,off
Default: off

on – this value indicates that the mutual distance between the base and the rover antennas remain fixed during the RTK session. This restriction allows the RTK engine to fix ambiguities faster and more accurately.

Use Fixed Baseline Length

Name: /par/pos/pd/hd/uselen
Access: rw
Type: boolean
Values: on,off
Default: off

on – RTK engine will use in the heading mode the apriori baseline length from the parameter /par/pos/pd/hd/len/0, unless its value is equal to zero.

off – RTK engine will compute its own baseline length estimate, which is obtained by averaging instantaneous baseline estimates over a 30-second interval after the first ambiguity fix. Thus, a derived empirical estimate is then used by the RTK engine to improve ambiguity fixing. Also note that this empirical estimate is constantly refined by the receiver as new measurements arrive.

Fixed Baseline Length

Name: /par/pos/pd/hd/len/N (N=[0…2])
Access: rw
Type: float [meters]
Values: [0…10000]
Default: 0

Currently only parameter with N=0 is used by RTK engine. Unless the value of this parameter is zero, it will be used by RTK engine in the heading mode, provided /par/pos/pd/hd/uselen is set to on.

Penalty Factor for Baseline Length Term

Name: /par/pos/pd/hd/pen
Access: rw
Type: float
Values: [0…10]
Default: 0.05

The larger this weight, the more "critical" for correct ambiguity resolution is the apriori baseline length specified by the parameter /par/pos/pd/hd/len.

The penalty factor must be consistent with the actual accuracy of the specified apriori baseline length. It it is not the case, the RTK engine may not be able to fix ambiguities correctly. In most scenarios, it will be sufficient to use the default value of the penalty factor.

The heuristic formula for the penalty factor is the following:

$$pf = 6 / \texttt{sigma}2$$

where sigma (measured in millimeters) stands for the expected standard deviation of the specified baseline length. If the baseline length is apriori known with an accuracy of 1mm or better, the user is recommended to set the parameter to the maximum value 10.

Memory Factor for Smoothing of Heading Angles

Name: /par/pos/pd/hd/memf
Access: rw
Type: float
Values: [0…1]
Default: 0

The bigger the memory factor value, the more conservative is smoothing.

Geometrical Constraint Tolerance Level

Name: /par/pos/pd/hd/tol
Access: rw
Type: float [meters]
Values: [0…1]
Default: 0.018

Baseline Calibration Mode

Name: /par/pos/pd/hd/tune/mode
Access: rw
Type: boolean
Values: off,on
Default: off

on - enable calibration, the value of this parameter will be set back to off after calibration is complete.

off - disable calibration.

Baseline Length After Calibration

Name: /par/pos/pd/hd/tune/len
Access: r
Type: float [meters] or <empty>
Default: <empty>

Number of Epochs to Use in Baseline Calibration

Name: /par/pos/pd/hd/tune/naver
Access: rw
Type: int
Values: [1…3600]
Default: 300

Baseline Calibration State

Name: /par/pos/pd/hd/tune/state
Access: r
Type: enumerated
Values: off,run,wait,ready
Default: off

off - calibration off

run - calibration on

wait - waiting RTK fixed solution

ready - calibration ready

Baseline Calibration Progress

Name: /par/pos/pd/hd/tune/progress
Access: r
Type: integer [percent]
Values: [0…100] or <empty>
Default: <empty>

Compensation of GLONASS Inter-channel Biases

Parameters described in this section govern compensation of GLONASS inter-channel biases in the measurements received from reference station. Correct compensation may significantly improve RTK performance.

Estimation Mode

Name: /par/calib/soft/mode
Access: rw
Type: enumerated
Values: got,fix
Default: got

got – use brand of reference station received from the reference station to select biases estimations.

fix – use brand of reference station taken from par/calib/soft/brand/fix parameter.

Current Brand

Name: /par/calib/soft/brand/cur
Access: r
Type: enumerated
Values: unknown,javad,ashtech,topcon,trimble,leica,septentrio
Default: unknown

The brand of reference station currently being used by RTK to choose the values of GLONASS inter-channel biases. When unknown, all the biases are set to 0.

This parameter is informational and is changed by receiver depending on other parameters described in this section as well as possibly on the data received from reference station.

Received Brand

Name: /par/calib/soft/brand/got
Access: r
Type: enumerated
Values: unknown,javad,ashtech,topcon,trimble,leica,septentrio
Default: unknown

The brand of reference station receiver got from the reference station.

Fixed Brand

Name: /par/calib/soft/brand/fix
Access: rw
Type: enumerated
Values: unknown,javad,ashtech,topcon,trimble,leica,septentrio
Default: unknown

This parameter allows user to specify fixed brand of reference station. For this parameter to take effect, set /par/calib/mode to the value fix.

The value unknown means all the biases will be 0.

Receiver Name Got From Corrections

Name: /par/calib/soft/brand/rcv
Access: r
Type: string
Default: <empty string>

GLONASS Biases Source

Name: /par/calib/soft/source
Access: r
Type: enumerated
Values: none,preset,data
Default: none

none - no source of GLONASS biases

preset - the source of GLONASS biases is preset table of known brands

data - the source of GLONASS biases is received data values

Current GLONASS Bias on L1 frequency

Name: /par/calib/soft/bias/l1
Access: r
Type: float
Values: 0

Current GLONASS Bias on L2 frequency

Name: /par/calib/soft/bias/l2
Access: r
Type: float
Values: 0

TDMA Multiple Reference Stations

JAVAD GNSS receivers support Multi-base mode in which the rover receiver is able to obtain RTK data from more than one reference station[1]. Running in this mode, the reference stations broadcast their RTK data using TDMA (Time Division Multiple Access) method. This method allows the reference stations to use a single frequency for transmitting their data. It is achieved by setting a transmission delay for each station. This mode is currently supported for CMR Plus messages only.

Timeout for Data From Multiple Reference Stations

Name: /par/pos/pd/mbtimeout
Access: rw
Type: integer
Values: [1…6]
Default: 1

In RTK with multiple reference stations, this parameter specifies how many epochs the RTK engine will wait for a complete data set, for example, from three reference stations before processing, for example, the data from only two of them.

Reference Station ID to Use

Name: /par/pos/pd/inuse
Access: rw
Type: integer
Values: [-1…31]
Default: -1

This parameter allows the user to identify the reference station (by specifying its CMR reference station ID) he/she wants to use in order to compute the RTK position.

-1 – rover receiver will use RTK corrections from whichever station.

[0…31] – receiver will use RTK corrections only from the reference station having the specified ID. Data from all the other reference stations will be ignored. Thereby it guarantees that the rover will work properly if there are two or more reference stations transmitting RTK data on the same frequency.

1. Currently, this mode allows the rover to use up to four reference stations.

Setting the parameter to -1 whereas two or more sources of RTK data are available simultaneously and automatic selection of the nearest reference station is turned off, may result in inability to get the RTK solution since RTK data received from several reference stations may be mixed with each other.

Enable Automatic Selection of the Nearest Reference Station

Name: /par/pos/pd/nrs/mode
Access: rw
Type: boolean
Values: on,off
Default: off

on - rover receiver will use RTK data broadcast by the nearest reference station. The receiver will use this parameter only if /par/pos/pd/inuse is set to -1.

off - rover receiver will use RTK data from any reference station.

Threshold for Switching of Nearest Reference Station

Name: /par/pos/pd/nrs/lim
Access: rw
Type: float [meters]
Values: [1.0….10000.0]
Default: 25.0

Switching to the new reference station will occur only if the difference between the distance to the new reference station and the distance to the current reference station exceeds the specified limit.

Maximum counter of attempts to stay the nearest reference station

Name: /par/pos/pd/nrs/cnt
Access: rw
Type: integer
Values: [1…30000]
Default: 10

The parameter defines the counter that serves for setting up a total number of attempts during which a nearest reference station must remain the nearest one for this station to be selected as the new nearest reference station. If during those attempts, at least one of the other reference stations is detected as the nearest one, the counter will be reset and started again. The rover estimates the distances to the reference stations and makes attempts to select the nearest one each time when it receives CMR message Type 0.

Attitude Parameters

Attitude Mode

Name: /par/att/mode
Access: rw
Type: integer
Values: 0,1
Default: 0

0 – attitude mode is off.

1 – attitude mode is on. In addition, the *Talker ID* field of the NMEA HDT message is set to "HE" instead of the value dictated by the NMEA standard.

Number of Epochs to Use for Self-calibration

Name: /par/att/naver
Access: rw
Type: integer
Values: [1…2000000]
Default: 60

The number of measurement epochs must be set taking into consideration the following:

1. Minimum self-calibration time is 1 hour;
2. Recommended self-calibration time is greater than 1 hour;
3. Maximum self-calibration time is 2 hours.

According to the above, the number of measurement epochs can be determined as:

$$N = T_{calibration} / T_{interval}$$

where

$T_{calibration}$ – self-calibration time,

$T_{interval}$ – differential corrections update period.

For example, if differential correction update period is 0.05 s (1/20 Hz) and $T_{calibration}$ is 1 hour, then

$$N = 3600 / 0.05 = 72000$$

Start Self-calibration

Name: /par/att/tune
Access: rw
Type: boolean
Values: on,off
Default: off

on - the self-calibration process will begin. The value of this parameter is immediately restored to off.

off - ignored.

Baseline Vectors of the Body Frame

Name: /par/att/bl/N (N=[0,1,2])
Access: rw
Type: {n,r,b}
Default: {0,0,0}

n, r, b - Nose-, Right-, Belly- components of the corresponding vector. Float values in the range [-100000…100000] meters.

Pitch, Roll, and Heading Offsets

Name: /par/att/aoff
Access: rw
Type: {p,r,h}
Default: {0,0,0}

p, r - Pitch- and Roll- offsets. Float values in the range [-90…90] degrees.

h - Heading offset. Float values in the range [-180…180] degrees.

Master Input Mode

Name: /par/att/remote/imode
Access: rw
Values: none,rtcm,cmr
Default: none

This parameter contains the input mode to use on the Master side.

Attitude Processing Mode

Name: /par/att/procmode
Access: rw
Type: enumerated
Values: delay,extrap
Default: delay

delay - there will be delays between time tags of the position and the attitude.

`extrap` – the time tags of the position and the attitude are aligned by using the extrapolation filter. Additional smoothing is added as a side effect.

Gain of the Attitude Extrapolation Filter

Name: `/par/att/gain`
Access: `rw`
Type: `float`
Values: `[0.1…10000]`
Default: `1`

Correlation Time of the Attitude Extrapolation Filter

Name: `/par/att/tcor`
Access: `rw`
Type: `float`
Values: `[0.001…10000]`
Default: `1`

Use Baseline Vectors with Fixed Ambiguities Only

Name: `/par/att/fixed`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – the attitude will be calculated with all three baseline vectors being fixed only.

`off` – the attitude will be calculated even with some of baseline vectors having float status.

Lever Arm Computation Mode

Name: `/par/att/armcalc`
Access: `rw`
Type: `enumerated`
Values: `on, off, auto`
Default: `off`

`on` – the "lever arm" position, shifted from the antenna position by the lever arm vector, will be calculated and reported in the messages [RO], [RG], and [NR], provided the attitude is estimated.

`off` – if the attitude is not available, the NaN (Not a Number) values are reported. Also, NaN values are reported in the messages [RO], [RG], [NR].

`auto` – the messages [RO], [RG], [NR] report the position corrected by the rotated lever arm if the attitude is available, otherwise they report the position of the master antenna.

Lever Arm Vector

Name: `/par/att/arm`
Access: `rw`
Type: `{n,r,d}`
Default: `{0,0,0}`

The lever arm vector in the "nose", "right", "down" frame.

`n, r, d` – Nose-, Right-, Down- components of the lever arm vector. Float values in the range `[-10000…10000]` meters.

Heading

Name: `/par/pos/att/head`
Access: `r`
Type: `float [deg]`
Values: `[0…360], or <empty>`
Default: `<empty>`

Pitch

Name: `/par/pos/att/pitch`
Access: `r`
Type: `float [deg]`
Values: `[-90…90], or <empty>`
Default: `<empty>`

Roll

Name: `/par/pos/att/roll`
Access: `r`
Type: `float [deg]`
Values: `[-180…180], or <empty>`
Default: `<empty>`

Ambiguity Fixing Statistics

The parameters described in this section are considered as technology parameters and are subject to change without notice. Note that they are primarily intended for test purposes.

This group of parameters provides the user with statistical (probabilistic) information on ambiguity fixing time. There are two different modes to obtain such information, specifically:

1. Enable *full firmware* reset once the ambiguities are fixed during the test cycle.
2. Enable *RTK only reset* once the ambiguities are fixed during the test cycle. This statistical information is presented as a histogram the i-th point of which specifies the probability of ambiguity fixing time being less than "i" seconds.

Precise Test Baseline

Name: `/par/pos/pd/hist/pr/C (C=[x|y|z])`
Type: `float [meters]`
Values: `[-1000000…1000000]`
Default: `0`

These three related parameters specify apriori precise coordinates of the test baseline in WGS-84 (X-, Y- and Z- coordinates, respectively).

The baseline vector's components estimated in the current iteration are compared against the apriori precise coordinates. Ambiguities are considered fixed correctly if the difference between the apriori and a posteriori coordinates does not exceed R cm in each axis (note that R may be different in different firmware versions; it normally lies between 4 cm and 6 cm).

Ambiguity Fixing Statistics Using Full Firmware Resets

Name: `/par/pos/pd/hist/mode`
Access: `rw`
Values: `on,off`
Default: `off`

`on` – turn the mode on.

`off` – turn the mode off.

In this mode the receiver firmware is fully reset every time the integer ambiguities are re-fixed after the previous iteration. Although the actual time to ambiguity fix in the current iteration may well be less than one minute (and this time is correctly logged in the receiver memory), yet the new firmware reset is executed no sooner than in a minute after the previous one occurs. It is done in order for the receiver to be able to timely update ephemeris used. Should the current ambiguity fixing time exceed one minute, the firmware will fully reset immediately after the ambiguities are fixed.

This mode is available only when both `/par/raw/msint` and `/par/pos/msint` are set to 1000.

Erase the Current Statistics

Name: /par/hist/reset
Access: w
Values: on

Before you start collecting new statistics, you need to delete the previous ones. Setting this parameter to on will do this.

Estimated Probability of Ambiguities Fixing

Name: /par/pos/pd/hist/out
Access: r

Output data will look as follows:

```
⇐ 1<float> probability of fixing ambiguities in = 1 s
  2<float> probability of fixing ambiguities in = 2 s
  .
  .
  120<float> probability of fixing ambiguities in = 120 s
```

Total Number of Fixed Solutions

Name: /par/pos/pd/hist/num
Access: r

Output data will look as follows:

```
⇐ khist=<integer>
```

Percentage of Wrong Fixed Solutions

Name: /par/pos/pd/hist/bad
Access: r

Output data will look as follows:

```
⇐ bad=<float>
```

Note that if you have no apriori baseline coordinates then this estimate is of no avail.

Ambiguity Fixing Statistics Using RTK Engine Resets

Name: /par/rtk/dbi/rest
Access: w
Type: integer
Values: 0,1,2
Default: 0

This parameter is intended to enable the second ambiguity fixing time mode.

`0` – turn the mode off.

`1` – RTK engine will be reset on fixing the current set of ambiguities.

`2` – firmware will immediately output obtained statistics (histogram) to the current terminal and then start collecting data for the next histogram.

This parameter is not stored in NVRAM, therefore switching receiver off then on will turn this mode off.

Example: The following is an example histogram:

```
⇐ khist = 122 kbad = 0
  hist 1 0.000000
  hist 2 0.991803
  hist 3 0.991803
  hist 4 1.000000
  hist 5 1.000000
  …
  hist 120 1.000000
```

where:

`khist` – is the number of trials

`kbad` – is the number of the wrong fixes. Note that if you have no apriori baseline coordinates then this estimate is of no avail.

`hist` – 120 strings in the format "`hist n p`", where `p` is the estimated probability of fixing ambiguity in no more than `n` seconds.

4.4.11 Precise Point Positioning (PPP) Parameters

PPP Update Period

Name: `/par/pos/pp/period`
Access: `rw`
Type: `float [seconds]`
Values: `[0.01…100]`
Default: `1`

Specify PPP solution update rate.

PPP Corrections Type

Name: /par/pos/pp/type
Access: rw
Type: enumerated
Values: jppp,has
Default: jppp

Specify corrections type to use for PPP:

jppp - J-Star LBand

has - Galileo High Accuracy Service (HAS)

JAVAD Precise Point Positioning (JPPP)

These parameters govern JAVAD implementation of Precise Point Positioning service, or JPPP.

JPPP Correction Stream Source

Name: /par/jppp/stream/src
Access: rw
Type: enumerated
Values: auto,beam,ntrip
Default: auto

auto - automatically select either beam or ntrip, ensuring that whenever multiple correction streams are being provided, a specific one is being used. See /par/jppp/stream/auto/prefer for details.

beam - use satellite beam as the source of corrections stream

ntrip - use NTRIP client (to be connected to caster) as a source of corrections stream

JPPP Current Correction Stream Source

Name: /par/jppp/stream/cursrc
Access: r
Type: enumerated
Values: beam,ntrip

JPPP Preferred Correction Stream for Auto mode

Name: /par/jppp/stream/auto/prefer
Access: rw
Type: enumerated
Values: beam,ntrip
Default: beam

beam - prefer satellite beam corrections stream when auto correction stream source is selected by /par/jppp/stream/src parameter.

ntrip - prefer NTRIP corrections stream when auto correction stream source is selected by /par/jppp/stream/src parameter.

JPPP Beam Good Messages Counter

Name: /par/jppp/stream/msg/beam/good
Access: r
Type: integer
Default: 0

JPPP Beam Bad Messages Counter

Name: /par/jppp/stream/msg/beam/bad
Access: r
Type: integer
Default: 0

JPPP NTRIP Good Messages Counter

Name: /par/jppp/stream/msg/ntrip/good
Access: r
Type: integer
Default: 0

JPPP NTRIP Bad Messages Counter

Name: /par/jppp/stream/msg/ntrip/bad
Access: r
Type: integer
Default: 0

JPPP Beam Channel

Name: /par/jppp/beam/chan
Access: rw
Type: enumerated
Values: off,auto,user
Default: off

off - L-Band demodulator is turned off

auto - L-Band beam with the best signal quality is automatically selected

user - use customization parameters to select Beam frequency, data rate and scrambling vector

JPPP Beam Name

Name: /par/jppp/beam/cur/name
Access: r
Type: string
Default: "unknown"

JPPP Beam Coverage

Name: /par/jppp/beam/cur/coverage
Access: r
Type: string
Default: "unknown"

JPPP Beam Carrier Frequency

Name: /par/jppp/beam/cfnom
Access: rw
Type: integer [Hz]
Values: [1525000000…1570000000]
Default: 1542000000

Use specific carrier frequency when /par/jppp/beam/chan is set to user.

JPPP Beam Data Rate

Name: /par/jppp/beam/drnom
Access: rw
Type: enumerated [bps]
Values: 600,1200,2400,4800
Default: 1200

Use specific data rate when /par/jppp/beam/chan is set to user.

JPPP Beam Scrambling Vector

Name: /par/jppp/beam/scrv
Access: rw
Type: string [1…4]
Values: (hexadecimal string)
Default: 5C08

Use specific scrambling vector when /par/jppp/beam/chan is set to user.

JPPP Beam Locking Status

Name: /par/jppp/beam/cur/lock
Access: r
Type: string [0…32]
Default: "unknown"

0 - L-Band beam not locked

1 - L-Band beam is locked

JPPP Beam Preamble Status

Name: /par/jppp/beam/cur/sync
Access: r
Type: string [0…32]
Default: "unknown"

0 - L-Band beam preamble not detected

1 - L-Band beam preamble is detected

JPPP Actual Carrier Frequency

Name: /par/jppp/beam/cur/cfact
Access: r
Type: string [0…32]
Default: "unknown"

JPPP Beam Service Identifier

Name: /par/jppp/beam/cur/sid
Access: r
Type: string [0…32]
Default: "unknown"

JPPP Beam Signal Bit Error Rate

Name: /par/jppp/beam/cur/ber
Access: r
Type: string [0…32]
Default: "unknown"

JPPP Beam Signal SNR (Signal to Noise ratio)

Name: `/par/jppp/beam/cur/snr`
Access: `r`
Type: `string [0…32]`
Default: `"unknown"`

JPPP Solution Status

Name: `/par/jppp/beam/cur/sol`
Access: `r`
Type: `string [0…32]`
Default: `"unknown"`

JPPP Position Status

Name: `/par/jppp/beam/cur/pos`
Access: `r`
Type: `string [0…32]`
Default: `"unknown"`

JPPP Warning Status

Name: `/par/jppp/stat/warn`
Access: `r`
Type: `list of enumerated`
Default: `{}`

List of raised warnings, where each element is one of:

`needupd` - need update

`datupd` - datums updated

`ellupd` - ellipsoids updated

`beamupd` - global beams updated

`satupd` - satellites updated

`init` - static init mode

`spoof` - spoofing alert

`nol1corr` - no L1 corrections

`nol1map` - no L1 mapping message

`nol1dat` - no L1 datum message

`nol1alm` - no L1 almanac

`nol1stat` - no L1 stations

JPPP Error Status

Name: /par/jppp/stat/err
Access: r
Type: list of enumerated
Default: {}

List of raised errors, where each element is one of:

noppp - no PPP engine

resreject - result rejected

spoof - spoofing alert

alt - altitude

nomap - no mapping message

nostat - no stations

veloc - velocity

notime - no time

nopos - no position

noeph - no ephemeris

nomeas - no measurements

link - link

wet - wet

reg - region

expir - expiration

JPPP Subscription Start Date

Name: /par/jppp/sub/start
Access: r
Type: string [0…32]
Default: "unknown"

JPPP Subscription Expiration Date

Name: /par/jppp/sub/exp
Access: r
Type: string [0…32]
Default: "unknown"

JPPP Engine Version

Name: /par/jppp/ver
Access: r
Type: string [0…32]
Default: "unknown"

JPPP ID Number for Subscription

Name: /par/jppp/id
Access: r
Type: string [0…31]
Default: (empty string)

4.4.12 Integrated Navigation System (INS) Parameters

INS is capable to produce integrated position and velocity as well as orientation of the vehicle body frame (VBF).

VBF originates at the same point as receiver body frame (RBF) coordinate system, Y axis points forward, X - to the right, and Z - up, so that, when horizontal vehicle is heading North, VBF XYZ axes coincide with ENU (East-North-Up) axes.

RBF could be rotated at arbitrary angles with respect to VBF that are to be set by the user through the INS interface described below. Ideally, receiver should be installed on the vehicle in such a way that its RBF XYZ axes point right, forward, and up the vehicle body frame, respectively, so that RBF rotation angles are all zero.

For navigation receivers RBF originates at receiver body reference point (BRP), and is plotted along with its XYZ axes on the receiver body.

For geodetic receivers that feature built-in antenna, the RBF originates at the center of built-in IMU, directions of IMU axes match the pictogram on the receiver body, and the antenna phase center offsets are preset in the firmware for the case of non-rotated RBF.

Note: automatic computation (auto-calibration) of some or all of the needed offsets and rotations is not yet supported, so the geometry must be specified carefully by the user for proper INS operation.

INS Mode

Name: /par/pos/ins/mode
Access: rw
Type: enumerated
Values: off,ins,apc,poi
Default: off

off - INS mode is turned off

ins - INS mode is turned on, but none of generic positioning messages are affected

apc - INS mode is turned on, and generic navigation messages will contain INS-augmented coordinates and velocities of the GNSS Antenna Phase Center (APC).

poi - INS mode is turned on, and generic navigation messages will contain INS-augmented coordinates and velocities of the Point Of Interest (POI).

Heading and orientation messages, as well as INS-specific messages, will all have INS solution provided this parameter is not set to off.

INS Status

Name: /par/pos/ins/stat
Access: r
Type: enumerated
Values: off,ok,fail
Default: off

off - INS mode is off

ok - INS is producing solution and there are no errors

fail - There are errors: check /par/pos/ins/err for details. INS might still produce solution depending on particular error(s).

INS Error Flags

Name: /par/pos/ins/err
Access: r
Type: list of enumerated
Values: <see below>
Default: {}

Comma-separated list of raised errors, where each element is one of:

imu_meas - No data from IMU

sat_meas - No GNSS measurements

pos - No GNSS position

vel - Failure to compute GNSS velocity

`ins_init` - Not enough motion or no baseline fix for proper initialization

`conv` - Solution not yet converged

`gyro` - Bad gyroscope measurements

`accl` - Bad accelerometer measurements

`baseline` - No baseline fix

Note that not all the errors are fatal, so this parameter could be non-empty even when INS solution is being produced.

INS Pole Length

Name: `/par/pos/ins/layout/pole`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

This is synonym for `/par/pole/height`. If non-zero, this parameter takes precedence over `/par/pos/ins/layout/poi` parameter.

INS Layout: IMU Offset

Name: `/par/pos/ins/layout/imu`
Access: `rw`
Type: `{float,float,float} [meters]`
Values: `{[-100…100],[-100…100],[-100…100]}`
Default: `<receiver dependent>`

XYZ offset of IMU center in RBF. The default value is preset and corresponds to particular receiver housing. In particular, it is set to `{0,0,0}` for OEM boards and geodetic receivers.

INS Layout: APC Offset

Name: `/par/pos/ins/layout/apc`
Access: `rw`
Type: `{float,float,float} [meters]`
Values: `{[-100…100],[-100…100],[-100…100]}`
Default: `<receiver dependent>`

XYZ offset of antenna phase center (APC) in VBF. The default value is preset depending on particular geodetic receiver model. In particular, it is set to`{0,0,0}` for OEM and antenna-less receivers.

INS Layout: APC Reference Frame

Name: `Name: /par/pos/ins/layout/ref/apc`
Access: `Access: rw`
Type: `Type: enumerated`
Values: `Values: vbf,rbf`
Default: `Default: <receiver dependent>`

`vbf` - APC coordinates are specified with respect to vehicle body frame. Useful for external antenna. The default for receivers that lack internal antenna.

`rbf` - APC coordinates are specified with respect to receiver body frame. Useful for internal antenna. The default for receivers that feature internal antenna.

INS Layout: POI Offset

Name: `/par/pos/ins/layout/poi`
Access: `rw`
Type: `{float,float,float} [meters]`
Values: `{[-100…100],[-100…100],[-100…100]}`
Default: `{0,0,0}`

XYZ offset of point of interest (POI) in VBF.

This is only active if `/par/pos/ins/layout/pole` is zero, otherwise the POI will be the tip of the pole.

INS Layout: Baseline

Name: `/par/pos/ins/layout/bl`
Access: `rw`
Type: `{float,float,float} [meters]`
Values: `{[-100…100],[-100…100],[-100…100]}`
Default: `{0,0,0}`

XYZ offset of phase center of second antenna with respect to phase center of primary antenna, in VBF.

INS Layout: RBF Rotation Angles

Name: `/par/pos/ins/layout/rot`
Access: `rw`
Type: `{float,float,float} [degrees]`
Values: `{[-360…360],[-360…360],[-360…360]}`
Default: `{0,0,0}`

Yaw (heading), pitch, and roll of RBF with respect to VBF.

Starting from the state where RBF matches VBF, the yaw, pitch, and roll are to be applied in this exact order to the RBF to get it to receiver actual position in particular

installation. Yaw is rotation about Z axis, pitch - X, roll - Y, clockwise when looking along corresponding axis in its direction.

Note: the direction of positive yaw angle defined this way is opposite to that of navigation heading.

INS GNSS Update Rate

Name: `/par/pos/ins/msint/gnss`
Access: `rw`
Type: `integer [milliseconds]`
Values: `[20…1000]`
Default: `100`

INS IMU Update Rate

Name: `/par/pos/ins/msint/imu`
Access: `rw`
Type: `integer [milliseconds]`
Values: `[5…1000]`
Default: `20`

This is synonym for `/par/imu/dev/msint` parameter.

INS Reset

Name: `/par/pos/ins/reset`
Access: `rw`
Type: `boolean`
Values: `n,y`
Default: `n`

`n` - ignored

`y` - reset INS machinery, and set this back to 'n'

INS Resulting Orientation Angles

Name: `/par/pos/ins/hpr`
Access: `r`
Type: `{float,float,float} [degrees]`
Values: `{[0…360],[-90…90],[-180…180]}`
Default: `{}`

This parameter contains last computed heading, pitch, and roll. Empty list if no orientation is being computed. This parameter is informational: use attitude message such as `[AR]` to get definitive data.

4.4.13 Reference Parameters

Parameters described in this section specify different kinds of reference information that could be used by multiple other receiver sub-systems. For example, reference coordinates could be sent from the reference station to rovers for RTK applications, are used to calculate RTCM corrections by DGPS reference station, and are used by the Improved Timing mode; reference antenna parameters could be utilized by both RTK base and RTK rover functionality; etc.

Reference Station Coordinates

Overview

Receiver supports separate reference station coordinates for GPS and GLONASS, as these two systems use different reference datums, WGS-84 and PE-90, respectively. In addition, reference station coordinates could be specified by the user int either Cartesian or Geodetic system, that is also supported by means of separate parameters. Therefore, total of four customizable parameters are supported:

```
/par/ref/pos/gps/xyz
/par/ref/pos/gps/geo
/par/ref/pos/glo/xyz
/par/ref/pos/glo/geo
```

All these parameters should contain the coordinates of the L1 phase center of the receiver antenna.

While it's possible to specify GPS and GLONASS reference positions independently, it is recommended to use one reference position estimate for both GPS and GLONASS in most cases. To simplify this common case, receiver supports simultaneous entry of both GPS and GLONASS reference coordinates through `/par/ref/pos//xyz` and `/par/ref/pos//geo` write-only parameters (note duplicated slashes in the parameter names).

Note that "xyz" and "geo" variants of the position for the same satellite system are mutually Dependant. When one of the parameters is changed, another one is automatically re-calculated so that their values are always on the same datum and are consistent with each other.

While coordinates could be entered in any supported datum, receiver will need them in the datum used by particular satellite system, WGS-84 for GPS, and PE-90 for GLONASS, so it calculates those coordinates and makes them accessible for reading by the user in both Cartesian and Geodetic form through another four parameters:

```
/par/ref/syspos/gps/xyz
/par/ref/syspos/gps/geo
/par/ref/syspos/glo/xyz
/par/ref/syspos/glo/geo
```

With the parameters described hereafter the user specifies the location of the ARP. This location is then transmitted using RTCM 2.x message 24 and RTCM 3.x messages and is needed at the rover side in order to compute the RTK solution.

While the above parameters specify/describe the coordinates for L1 Antenna Phase Center (APC), the RTCM 2.x message 24 as well as RTCM 3.x standard requires that Antenna Reference Point (ARP) coordinates are to be transmitted from reference station to rover receivers. To meet this requirement, receiver supports additional set of parameters specifying ARP coordinates in the same way APC coordinates are specified. These parameters are:

```
/par/ref/arp/gps/xyz
/par/ref/arp/gps/geo
/par/ref/arp/glo/xyz
/par/ref/arp/glo/geo
```

and

```
/par/ref/sysarp/gps/xyz
/par/ref/sysarp/gps/geo
/par/ref/sysarp/glo/xyz
/par/ref/sysarp/glo/geo
```

The APC and ARP coordinates in the receiver are entirely independent. Receiver never calculates ARP coordinates from APC or vice versa. It's a duty of the user or corresponding application program to specify correct and consistent coordinates for ARP and APC.

Note that receiver doesn't use ARP coordinates except for the purpose of transmitting them in corresponding RTCM messages, while APC coordinates are essential to the receiver itself. Therefore, APC coordinates should always be entered for a reference station, while ARP coordinates may have arbitrary values unless you are going to transmit RTCM messages that carry ARP coordinates.

Parameters

Cartesian Reference Position for GPS

Name: `/par/ref/pos/gps/xyz`
Access: `rw`
Type: `pos_xyz`
Default: `{W84,+6378137.0000,+0.0000,+0.0000}`

Coordinates of L1 phase center of receiver antenna for GPS in Cartesian coordinate system.

Cartesian Reference Position for GLONASS

Name: /par/ref/pos/glo/xyz
Access: rw
Type: pos_xyz
Default: {W84,+6378137.0000,+0.0000,+0.0000}

Coordinates of L1 phase center of receiver antenna for GLONASS in Cartesian coordinate system.

Cartesian Reference Position for All Systems

Name: /par/ref/pos//xyz
Access: w
Type: pos_xyz

Setting this parameter will set both /par/ref/pos/gps/xyz and /par/ref/pos/glo/xyz to the specified value.

Geodetic Reference Position for GPS

Name: /par/ref/pos/gps/geo
Access: rw
Type: pos_geo
Default: {W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}

Coordinates of L1 phase center of receiver antenna for GPS in Geodetic coordinate system.

Geodetic Reference Position for GLONASS

Name: /par/ref/pos/glo/geo
Access: rw
Type: pos_geo
Default: {W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}

Coordinates of L1 phase center of receiver antenna for GLONASS in Geodetic coordinate system.

Geodetic Reference Position for All Systems

Name: /par/ref/pos//geo
Access: w
Type: pos_geo

Setting this parameter will set both /par/ref/pos/gps/geo and /par/ref/pos/glo/geo to the specified value.

WGS-84 Cartesian Reference Position (for GPS)

Name: /par/ref/syspos/gps/xyz
Access: r
Type: pos_xyz
Default: {W84,+6378137.0000,+0.0000,+0.0000}

WGS-84 Geodetic Reference Position (for GPS)

Name: /par/ref/syspos/gps/geo
Access: r
Type: pos_geo
Default: {W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}

PE-90 Cartesian Reference Position (for GLONASS)

Name: /par/ref/syspos/glo/xyz
Access: r
Type: pos_xyz
Default: {P90,+6378137.0000,+0.0000,+0.0000}

PE-90 Geodetic Reference Position (for GLONASS)

Name: /par/ref/syspos/glo/geo
Access: r
Type: pos_geo
Default: {P90,N00d00m00.000000s,E000d00m00.000000s,+0.0000}

Maximum Allowed Error in Reference Position

Name: /par/ref/limit
Access: rw
Type: float [meters]
Values: [1…10000]
Default: 1000

Should the length of the vector connecting the current position calculated by receiver with that specified for the APC by the user exceed the maximum discrepancy level specified by this parameter, the reference station will stop transmitting any RTK or DGPS messages that depend on the quality of reference position.

Cartesian ARP Position for GPS

Name: /par/ref/arp/gps/xyz
Access: rw
Type: pos_xyz
Default: {W84,+6378137.0000,+0.0000,+0.0000}

Coordinates of ARP of receiver antenna for GPS in Cartesian coordinate system.

Cartesian ARP Position for GLONASS

Name: /par/ref/arp/glo/xyz
Access: rw
Type: pos_xyz
Default: {W84,+6378137.0000,+0.0000,+0.0000}

Coordinates of ARP of receiver antenna for GLONASS in Cartesian coordinate system.

Cartesian ARP Position for All Systems

Name: /par/ref/arp//xyz
Access: w
Type: pos_xyz

Setting this parameter will set both /par/ref/arp/gps/xyz and /par/ref/arp/glo/xyz to the specified value.

Geodetic ARP Position for GPS

Name: /par/ref/arp/gps/geo
Access: rw
Type: pos_geo
Default: {W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}

Coordinates of ARP of receiver antenna for GPS in Geodetic coordinate system.

Geodetic ARP Position for GLONASS

Name: /par/ref/arp/glo/geo
Access: rw
Type: pos_geo
Default: {W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}

Coordinates of ARP of receiver antenna for GLONASS in Geodetic coordinate system.

Geodetic ARP Position for All Systems

Name: /par/ref/arp//geo
Access: w
Type: pos_geo

Setting this parameter will set both /par/ref/arp/gps/geo and /par/ref/arp/glo/geo to the specified value.

WGS-84 Cartesian ARP Position (for GPS)

Name: /par/ref/sysarp/gps/xyz
Access: r
Type: pos_xyz
Default: {W84,+6378137.0000,+0.0000,+0.0000}

WGS-84 Geodetic ARP Position (for GPS)

Name: /par/ref/sysarp/gps/geo
Access: r
Type: pos_geo
Default: {W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}

PE-90 Cartesian ARP Position (for GLONASS)

Name: /par/ref/sysarp/glo/xyz
Access: r
Type: pos_xyz
Default: {P90,+6378137.0000,+0.0000,+0.0000}

PE-90 Geodetic ARP Position (for GLONASS)

Name: /par/ref/sysarp/glo/geo
Access: r
Type: pos_geo
Default: {P90,N00d00m00.000000s,E00d00m00.000000s,+0.0000}

Reference Position Averaging

All the features depending on the receiver reference position work best when specified reference position is known in advance with high precision. However, when precise position is unknown, there are still some applications that will tolerate even not that precise reference position. Reference Position Averaging feature allows receiver to automatically calculate and set its reference position.

Reference Position Averaging Mode

Name: /par/ref/avg/mode
Access: rw
Type: boolean
Values: on,off
Default: off

`on` – receiver will compute its smoothed coordinates by averaging stand-alone position estimates over the specified interval after receiver reset or restart. The interval is defined by the `/par/ref/avg/span` parameter (see below). The absolute coordinates thus estimated will then be automatically used as the receiver's reference position.

`off` – averaging mode is turned off.

Reference Position Averaging Interval

Name: `/par/ref/avg/span`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…86400]`
Default: `180`

Provided `/par/ref/avg/mode` parameter is `on`, this parameter specifies time interval over which single-point position calculated by the receiver will be averaged before the result of averaging will be used as receiver reference position.

Reference Antenna Parameters

Marker to Antenna Phase Center (APC) Offset

Name: `/par/ref/ant/offs`
Access: `rw`
Type: `list {east,north,height}`

This parameter specifies the vector components between a surveyed point (land mark) and the APC.

`east` – east offset

`north` – north offset

`height` – height offset

East Offset of APC

Name: `/par/ref/ant/offs/east`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

North Offset of APC

Name: /par/ref/ant/offs/north
Access: rw
Type: float [meters]
Values: [-100…100]
Default: 0

Height Offset of APC

Name: /par/ref/ant/offs/height
Access: rw
Type: float [meters]
Values: [-100…100]
Default: 0

Marker to the Antenna Reference Point (ARP) Offset

Name: /par/ref/ant/arpoffs
Access: rw
Type: list {east,north,height}

This parameter specifies the vector components between a surveyed point (land mark) and the ARP.

east - east offset

north - north offset

height - height offset

East Offset of ARP

Name: /par/ref/ant/arpoffs/east
Access: rw
Type: float [meters]
Values: [-100.0…100.0]
Default: 0

North Offset of ARP

Name: /par/ref/ant/arpoffs/north
Access: rw
Type: float [meters]
Values: [-100.0…100.0]
Default: 0

Height Offset of ARP

Name: /par/ref/ant/arpoffs/height
Access: rw
Type: float [meters]
Values: [-100.0…100.0]
Default: 0

L1 APC to L2 APC Offset

Name: /par/ref/ant/l2_l1
Access: rw
Type: list {east,north,height}

This parameter specifies the vector components between L1 Antenna Phase Center (APC) and L2 APC.

east - east offset

north - north offset

height - height offset

East Offset of L2 APC

Name: /par/ref/ant/l2_l1/east
Access: rw
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

North Offset of L2 APC

Name: /par/ref/ant/l2_l1/north
Access: rw
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

Height Offset of L2 APC

Name: /par/ref/ant/l2_l1/height
Access: rw
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

Antenna type descriptor for RTCM 2.x and 3.0

Name: /par/ref/ant/id
Access: rw
Type: string [0…31]
Values: up to 31 alphanumeric characters
Default: (empty string)

Antenna Setup ID

Name: /par/ref/ant/setup
Access: rw
Type: integer
Values: [0…255]
Default: 0

This parameter is typically used by the differential service provider to inform the user about any change at the reference station that affects the antenna phase center variations.

Antenna Serial Number

Name: /par/ref/ant/sernum
Access: rw
Type: string [0…31]
Values: up to 31 alphanumeric characters
Default: (empty string)

With this parameter the user specifies the individual antenna serial number.

4.4.14 Reference Station Data on Rover

Parameters described in this section represent information about reference station being used on rover receiver. They are mostly useful for RTK operation and serve two main purposes:

1. Allow the user of the rover receiver to get information that is received from the reference station.
2. Allow the user to enter information about reference station on rover and force the rover to use entered information instead of those received from reference station.

Many of these parameters could be considered to be a reflection on the rover side of the parameters described in “Reference Parameters” on page 332.

In this section, the data received from reference station is called *got* data. The data about reference station entered by the user on the rover is called *fixed* data. Receiver will select

which data to actually use for RTK according to the values specified by the user for parameters that are described in the "Source of Data For Reference Station on Rover" below. The result of selection procedure is available through the parameters described in the "Reference Station Data for RTK" at the end of this section.

Data Received (Got) From Reference Station

Validity of Got Reference Position for GPS

Name: `/par/rover/base/pos/got/gps/valid`
Access: `r`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – rover has got valid GPS reference position from reference station.

`off` – rover didn't receive GPS reference position from reference station.

Got Reference Position (Cartesian) for GPS

Name: `/par/rover/base/pos/got/gps/xyz`
Access: `r`
Type: `pos_xyz`
Default: `{UNDEF,+6378137.0000,+0.0000,+0.0000}`

This parameter contains the base station's GPS reference coordinates received from the base station.

Got Reference Position (Geodetic) for GPS

Name: `/par/rover/base/pos/got/gps/geo`
Access: `r`
Type: `pos_geo`
Default: `{UNDEF,N00d00m00.000000s,`
`E00d00m00.000000s,+0.0000}`

This parameter contains the base station's GPS reference coordinates received from the base station.

Validity of Got Reference Position for GLONASS

Name: `/par/rover/base/pos/got/glo/valid`
Access: `r`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – rover has got valid GLONASS reference position from reference station.

`off` – rover didn't receive GLONASS reference position from reference station.

Got Reference Position (Cartesian) for GLONASS

Name: `/par/rover/base/pos/got/glo/xyz`
Access: `r`
Type: `pos_xyz`
Default: `{UNDEF,+6378137.0000,+0.0000,+0.0000}`

This parameter contains the base station's GLONASS reference coordinates received from the base station.

The datum name `UNDEF` indicates that the truth reference coordinates are undefined or unavailable.

Got Reference Position (Geodetic) for GLONASS

Name: `/par/rover/base/pos/got/glo/geo`
Access: `r`
Type: `pos_geo`
Default: `{UNDEF,N00d00m00.000000s, E00d00m00.000000s,+0.0000}`

This parameter contains the base station's GLONASS reference coordinates received from the base station.

The datum name `UNDEF` indicates that the truth reference coordinates are undefined or unavailable.

Got Antenna ID

Name: `/par/rover/base/pos/got/datum`
Access: `r`
Type: `string [0…6]`
Default: `W84`

This parameter contains the name of datum which has been used on the reference station to generate DGPS and RTK corrections.

Got Reference Station ID

Name: `/par/rover/base/stid/got`
Access: `r`
Type: `integer`
Values: `[0…4095]`
Default: `0`

This parameter contains reference station ID received from reference station.

Got Antenna ID

Name: /par/rover/base/ant/got/id
Access: r
Type: string [0…31]
Values: (alphanumeric characters)
Default: (empty string)

This parameter contains antenna ID received from reference station. If the antenna ID begins with a non-digit character, it is an RTCM antenna descriptor. Otherwise it is a CMR numerical antenna ID formatted as decimal.

Got Antenna Serial Number

Name: /par/rover/base/ant/got/sernum
Access: r
Type: string [0…31]
Default: (empty string)

This parameter contains antenna serial number received from reference station. The serial number could be transmitted in RTCM 2.x and RTCM 3.x.

Got Antenna Setup ID

Name: /par/rover/base/ant/got/setup
Access: r
Type: integer
Values: [0…255]
Default: 0

This parameter contains antenna setup ID received from reference station. The setup ID could be transmitted in RTCM 2.x message 23 and RTCM 3.x.

Got Antenna Offset

Name: /par/rover/base/ant/got/m_offs
Access: r
Type: list {type,val}

This parameter contains reference antenna offset type and value received from reference station.

Got Antenna Offset Type

Name: /par/rover/base/ant/got/m_offs/type
Access: r
Type: enumerated
Values: l1pc,arp
Default: l1pc

This parameter contains reference antenna offset type.

l1pc - offset of antenna L1 phase center (APC)

arp - offset of antenna reference point (ARP)

Got Antenna Offset Value

Name: /par/rover/base/ant/got/m_offs/val
Access: r
Type: list {east,north,height}

This parameter contains antenna vector offset from the land mark to APC or ARP depending on the offset type used at the reference station.

east - east offset

north - north offset

height - height offset

Got East Antenna Offset Value

Name: /par/rover/base/ant/got/m_offs/val/east
Access: r
Type: float [meters]
Values: [-100…100]
Default: 0

Got North Antenna Offset Value

Name: /par/rover/base/ant/got/m_offs/val/north
Access: r
Type: float [meters]
Values: [-100…100]
Default: 0

Got Height Antenna Offset Value

Name: /par/rover/base/ant/got/m_offs/val/height
Access: r
Type: float [meters]
Values: [-100…100]
Default: 0

Got L1 APC to L2 APC Offset

Name: /par/rover/base/ant/got/l2_l1
Access: r
Type: list {east,north,height}

east - east offset

north - north offset

height - height offset

Got East Offset of L2 APC

Name: /par/rover/base/ant/got/l2_l1/east
Access: r
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

Got North Offset of L2 APC

Name: /par/rover/base/ant/got/l2_l1/north
Access: r
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

Got Height Offset of L2 APC

Name: /par/rover/base/ant/got/l2_l1/height
Access: r
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

Data Entered (Fixed) For Reference Station

Fixed Cartesian Reference Position

Name: /par/rover/base/pos/fix/xyz
Access: rw
Type: pos_xyz
Default: {W84,+6378137.0000,+0.0000,+0.0000}

Fixed Geodetic Reference Position

Name: /par/rover/base/pos/fix/geo
Access: rw
Type: pos_geo
Default: {W84,N00d00m00.000000s,
E00d00m00.000000s,+0.0000}

Note: Currently only two datums, WGS-84 and PE-90, can be specified by means of this parameter.

Fixed Marker to Antenna Phase Center (APC) Offset

Name: /par/rover/base/ant/fix/offs
Access: rw
Type: list {east,north,height}
Default: {0,0,0}

east - east offset

north - north offset

height - height offset

Fixed East Antenna Offset

Name: /par/rover/base/ant/fix/offs/east
Access: rw
Type: float [meters]
Values: [-100…100]
Default: 0

Fixed North Antenna Offset

Name: /par/rover/base/ant/fix/offs/north
Access: rw
Type: float [meters]
Values: [-100…100]
Default: 0

Fixed Height Antenna Offset

Name: /par/rover/base/ant/fix/offs/height
Access: rw
Type: float [meters]
Values: [-100…100]
Default: 0

Fixed L1 APC to L2 APC Offset

Name: /par/rover/base/ant/fix/l2_l1
Access: r
Type: list {east,north,height}

east - east offset

north - north offset

height - height offset

Fixed East Offset of L2 APC

Name: /par/rover/base/ant/fix/l2_l1/east
Access: rw
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

Fixed North Offset of L2 APC

Name: /par/rover/base/ant/fix/l2_l1/north
Access: rw
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

Fixed Height Offset of L2 APC

Name: /par/rover/base/ant/fix/l2_l1/height
Access: rw
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

Source of Data For Reference Station on Rover

Clear the Reference Station Coordinates

Name: `/par/pos/pd/ref/clean`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

This parameter is used to clear the currently effective coordinates of the reference station.

`on` - RTK engine will assume there are no reference coordinates and therefore will disable differential positioning until next reference coordinates are received or entered by the user. The value of the parameter is immediately reset to `off`.

`off` - ignored.

Reference Position Source

Name: `/par/rover/base/pos/cur`
Access: `rw`
Type: `enumerated`
Values: `got,fix`
Default: `got`

`got` - reference position received from reference station will be used.

`fix` - reference position entered for reference station by the user will be used.

Reference Position Set Source

Name: `/par/ref/src`
Access: `rw`
Values: `gps,glo,any`
Default: `any`

This parameter instructs the rover which of the reference position sets received from the base to select for use by RTK.

`gps` - GPS reference position will be selected for use by RTK, GLONASS reference position will be ignored.

`glo` - GLONASS reference position will be selected for use by RTK, GPS reference position will be ignored.

`any` - any reference position received from base station will be selected for use by RTK.

Antenna Offset Source

Name: /par/rover/base/ant/cur
Access: rw
Type: enumerated
Values: got,fix
Default: got

got - antenna offsets received from reference station will be used.

fix - antenna offsets entered by the user will be used.

Both L1 APC to L2 APC offset and Marker to APC offset usage are affected by this parameter.

Reference Station Data for RTK

Reference Position (Cartesian) for RTK

Name: /par/pos/pd/ref/pos/xyz
Access: r
Type: pos_xyz
Default: {UNDEF,+6378137.0000,+0.0000,+0.0000}

This parameter contains the base station's reference position (in Cartesian form) to be used by RTK.

The datum name UNDEF indicates that the truth reference coordinates are undefined or unavailable.

Reference Position (Geodetic) for RTK

Name: /par/pos/pd/ref/pos/geo
Access: r
Type: pos_geo
Default: {UNDEF,N00d00m00.000000s,
E00d00m00.000000s,+0.0000}

This parameter contains the base station's reference position (in Geodetic form) to be used by RTK.

The datum name UNDEF indicates that the truth reference coordinates are undefined or unavailable.

Antenna Offset for RTK

Name: /par/pos/pd/ref/ant/m_offs
Access: r
Type: list {type,val}

This parameter contains reference antenna offset type and value to be used for RTK.

Antenna Offset Type for RTK

Name: `/par/pos/pd/ref/ant/m_offs/type`
Access: `r`
Type: `enumerated`
Values: `l1pc,arp`
Default: `l1pc`

This parameter contains reference antenna offset type to be used for RTK.

`l1pc` - offset of antenna L1 phase center (APC)

`arp` - offset of antenna reference point (ARP)

Antenna Offset Value for RTK

Name: `/par/pos/pd/ref/ant/m_offs/val`
Access: `r`
Type: `list {east,north,height}`

This parameter contains antenna vector offset from the land mark to APC or ARP depending on the offset type to be used for RTK.

`east` - east offset

`north` - north offset

`height` - height offset

East Antenna Offset Value for RTK

Name: `/par/pos/pd/ref/ant/m_offs/val/east`
Access: `r`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

North Antenna Offset Value for RTK

Name: `/par/pos/pd/ref/ant/m_offs/val/north`
Access: `r`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

Height Antenna Offset Value for RTK

Name: /par/pos/pd/ref/ant/m_offs/val/height
Access: r
Type: float [meters]
Values: [-100…100]
Default: 0

L1 APC to L2 APC Offset for RTK

Name: /par/pos/pd/ref/ant/l2_l1
Access: r
Type: list {east,north,height}

east - east offset

north - north offset

height - height offset

East Offset of L2 APC for RTK

Name: /par/pos/pd/ref/ant/l2_l1/east
Access: r
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

North Offset of L2 APC for RTK

Name: /par/pos/pd/ref/ant/l2_l1/north
Access: r
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

Height Offset of L2 APC for RTK

Name: /par/pos/pd/ref/ant/l2_l1/height
Access: r
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

4.4.15 Antenna Database

JAVAD GNSS receivers contain embedded antenna data-base. This database provides identification and measurement information for more than 200 antennas. More precisely, for each antenna in this database, the following entries are given:

- Antenna identifier used in RTCM standard version 3.0. This identifier or ID (as denoted below) is a string comprising up to 20 characters.
- Antenna identifier used in CMR standard. This identifier or CMR ID (as denoted below) is an integer value in the range of 0…255.
- Vector offset between the ARP and L1 phase center
- Vector offset between L1 and L2 phase centers

Antenna Type Definitions

Name: `/par/antdb`
Access: `r`
Type: `list {ver,id,cmr,ids}`

`ver` - antenna database version

`id` - parameters for antennas by antenna ID

`cmr` - parameters for antennas by CMR ID

`ids` - IDs maps. Maps of ID to CMR ID, and CMR ID to ID.

Antenna Database Version

Name: `/par/antdb/ver`
Access: `r`
Type: `string`

The antenna database version in the `M.N.K` format, where:

`M` - database major version as decimal

`N` - database minor version as decimal

`K` - patch level as decimal

Parameters for Antennas by ID

Name: `/par/antdb/id`
Access: `r`
Type: `list {[ID]}`

The list of all the antenna IDs included in the database along with their parameters.

Parameters for Antenna ID

Name: /par/antdb/id/[ID]
Access: r
Type: list {cmr,l1_arp,l2_l1}

cmr - corresponding CMR antenna identifier.

l1_arp - vector offset between L1 phase center and antenna reference point.

l2_l1 - vector offset between L2 and L1 phase centers.

Antenna CMR ID

Name: /par/antdb/id/[ID]/cmr
Access: r
Type: integer
Values: [0…255] or empty string

The CMR antenna identifier for specified antenna ID. If the selected antenna ID does not have the corresponding CMR identifier, this parameter is set to an empty string.

L1 to ARP Offset

Name: /par/antdb/id/[ID]/l1_arp
Access: r
Type: list {east, north, height} of float [meters]
Values: {[-100.0…100.0],[-100.0…100.0],[-100.0…100.0]}

Vector offset between L1 phase center and ARP.

L1 to L2 Offset

Name: /par/antdb/id/[ID]/l2_l1
Access: r
Type: list {east, north, height} of float [meters]
Values: {[-100.0…100.0], [-100.0…100.0],[-100.0…100.0]}

Vector offset between L1 and L2 phase centers.

Parameters for Antennas by CMR ID

Name: /par/antdb/cmr
Access: r
Type: list {[CMR_ID]}

The list of all antennas with assigned CMR ID along with their parameters.

Parameters for Antenna CMR ID

Name: `/par/antdb/cmr/[CMR_ID]`
Access: `r`
Type: `list {id,l1_arp,l2_l1}`

`id` - corresponding antenna ID.

`l1_arp` - vector offset between L1 phase center and antenna reference point.

`l2_l1` - vector offset between L2 and L1 phase centers.

Antenna ID for Specific CMR ID

Name: `/par/antdb/cmr/[CMR_ID]/id`
Access: `r`
Type: `string`

Antenna ID for the specified CMR ID.

L1 to ARP Offset for CMR ID.

Name: `/par/antdb/cmr/[CMR_ID]/l1_arp`
Access: `r`
Type: `list {east, north, height} of float [meters]`
Values: `{[-100.0…100.0],[-100.0…100.0],[-100.0…100.0]}`

Vector offset between L1 phase center and ARP.

L1 to L2 Offset for CMR ID

Name: `/par/antdb/cmr/[CMR_ID]/l2_l1`
Access: `r`
Type: `list {east, north, height} of float [meters]`
Values: `{[-100.0…100.0], [-100.0…100.0],[-100.0…100.0]}`

Vector offset between L1 and L2 phase centers.

IDs Maps

Name: `/par/antdb/ids`
Access: `r`
Type: `list {id,cmr}`

`id` - map from antenna ID to CMR antenna ID

`cmr` - map from CMR antenna ID to antenna ID

ID to CMR ID Map

Name: `/par/antdb/ids/id`
Access: `r`
Type: `list {[ID]}`

For every antenna ID contains its corresponding antenna ID and CMR_ID.

ID to CMR ID Map Element

Name: `/par/antdb/ids/id/[ID]`
Access: `r`
Type: `string`

For specified antenna ID, contains antenna ID together with the corresponding CMR ID in the format `{ID,CMR_ID}`.

CMR ID to ID Map

Name: `/par/antdb/ids/cmr`
Access: `r`
Type: `list{[CMR_ID]}`

For every antenna with known CMR ID contains its corresponding antenna ID and CMR_ID.

CMR ID to ID Map Element

Name: `/par/antdb/ids/cmr/[CMR_ID]`
Access: `r`
Type: `string`

For specified CMR antenna ID, contains antenna ID together with the corresponding CMR ID in the format `{ID,CMR_ID}`.

4.4.16 Base and Rover Modes

Traditionally, many GNSS receivers support the notion of *base mode* and *rover mode*. For example, when receiver is used as RTCM DGPS reference station, it's not unusual to say that receiver works in *RTCM DGPS base mode*, and when receiver is computing DGPS position, it works in so called *RTCM DGPS rover mode*. However, such model is too simplistic to be used to exactly specify the required behavior of JAVAD GNSS receivers. For example, using this terminology, a JAVAD GNSS receiver is capable to work as, say, CMR RTK base, DGPS RTCM base, and RTCM3 RTK rover simultaneously.

Due to their flexibility, JAVAD GNSS receivers have no notion of base or rover modes internally. We can say that they are *mode-less* in this sense. On one hand, this allows applications to decide exact meaning of base and rover modes themselves, if they wish to. On the other hand, this makes it somewhat more difficult to design such applications. To simplify development of applications utilizing notion of base/rover modes, JAVAD GNSS receivers support a set of parameters that:

1. Allow to check if some feature that could be considered to belong to either base or rover mode is active.
2. Allow to turn *off* some features that could be considered to belong to either base or rover mode.

Note that these parameters can't be used to turn base or rover mode *on*, because JAVAD GNSS receivers have no idea what exactly those modes are from the point of view of given application.

Example: Turn off base and rover mode:

```
⇒ set,/par/base/mode/,off
⇒ set,/par/rover/mode/,off
⇒ set,/par/pos/mode/cur,sp
```

Base Modes

Name: `/par/base/mode`
Access: `rw`
Type: `list {rtcm,cmr,jps,rtcm3}`
Values: `{on|off,on|off,on|off,on|off}`
Default: `{off,off,off,off}`

This parameter is a list of boolean values describing the status of output of messages of corresponding formats. You can turn off all of these formats by using the command `set,/par/base/mode/,off`. Refer to description of individual parameters below for details.

RTCM 2.x Base Mode

Name: `/par/base/mode/rtcm`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Setting this parameter to `off` will disable output of all the RTCM 2.x messages to all of the receiver ports. Receiver returns error if you try to set this parameter to `on`.

on - indicates that there is at least one RTCM 2.x message enabled to be output to at least one of receiver ports.

off - indicates that there are no RTCM 2.x messages enabled for output.

CMR Base Mode

Name: /par/base/mode/cmr
Access: rw
Type: boolean
Values: on,off
Default: off

Setting this parameter to off will disable output of all the CMR messages to all of the receiver ports. Receiver returns error if you try to set this parameter to on.

on - indicates that there is at least one CMR message enabled to be output to at least one of receiver ports.

off - indicates that there are no CMR messages enabled for output.

GREIS Base Mode

Name: /par/base/mode/jps
Access: rw
Type: boolean
Values: on,off
Default: off

Setting this parameter to off will disable output of all the GREIS messages to all of the receiver ports to which GREIS [BI] is enabled. Receiver returns error if you try to set this parameter to on.

on - indicates that GREIS [BI] message is enabled to be output to at least one of receiver ports.

off - indicates that GREIS [BI] message is not enabled for output.

RTCM 3.x Base Mode

Name: /par/base/mode/rtcm3
Access: rw
Type: boolean
Values: on,off
Default: off

Setting this parameter to off will disable output of all the RTCM 3.x messages to all of the receiver ports. Receiver returns error if you try to set this parameter to on.

on - indicates that there is at least one RTCM 3.x message enabled to be output to at least one of receiver ports.

off - indicates that there are no RTCM 3.x messages enabled for output.

Rover Mode

Name: /par/rover/mode
Access: rw
Type: list {rtcm,cmr,jps,rtcm3}
Values: {on|off,on|off,on|off,on|off}
Default: {off,off,off,off}

This parameter is a list of boolean values describing the status of input modes of receiver ports. You can turn all the receiver ports that are currently in rtcm, cmr, jps, or rtcm3 input modes to cmd mode by using the command set,/par/rover/mode/,off. Refer to description of individual parameters below for details.

RTCM 2.x Rover Mode

Name: /par/rover/mode/rtcm
Access: rw
Type: boolean
Values: on,off
Default: off

Setting this parameter to off will switch all of the ports running in rtcm input mode back to cmd mode. Receiver returns error if you try to set this parameter to on.

on - indicates that at least one receiver port is set to rtcm input mode.

off - indicates that none of the receiver ports are set to rtcm input mode.

CMR Rover Mode

Name: /par/rover/mode/cmr
Access: rw
Type: boolean
Values: on,off
Default: off

Setting this parameter to off will switch all of the ports running in cmr input mode back to cmd mode. Receiver returns error if you try to set this parameter to on.

on - indicates that at least one receiver port is set to cmr input mode.

off - indicates that none of the receiver ports are set to cmr input mode.

GREIS Rover Mode

Name: /par/rover/mode/jps
Access: rw
Type: boolean
Values: on,off
Default: off

Setting this parameter to off will switch all of the ports running in jps input mode back to cmd mode. Receiver returns error if you try to set this parameter to on.

on - indicates that at least one receiver port is set to jps input mode.

off - indicates that none of the receiver ports are set to jps input mode.

RTCM 3.x Rover Mode

Name: /par/rover/mode/rtcm3
Access: rw
Type: boolean
Values: on,off
Default: off

Setting this parameter to off will switch all of the ports running in rtcm3 input mode back to cmd mode. Receiver returns error if you try to set this parameter to on.

on - indicates that at least one receiver port is set to rtcm3 input mode.

off - indicates that none of the receiver ports are set to rtcm3 input mode.

4.4.17 RTCM 2.x Parameters

RTCM 2.x Reference Station Parameters

RTCM 2.x Version to Use for RTCM 2.x Messages

Name: /par/rtcm/base/ver
Access: rw
Type: enumerated
Values: v2.1,v2.2,v2.3,v2.4
Default: v2.3

This parameter allows you to use JAVAD GNSS receivers together with legacy third-party rover receivers that don't support higher versions of RTCM 2.x standard. Note that only RTK messages are affected.

Zero the Rate of Change of Pseudo-range Corrections for GPS

Name: /par/rtcm/base/zrate/gps
Access: rw
Type: boolean
Values: on,off
Default: on

on – receiver will set the rate of change of the pseudo-range corrections to zero for GPS satellites in RTCM 2.x message types 1, 9, 31, and 34. In some cases it may improve DGPS accuracy.

off – receiver will put computed values into the messages.

Zero the Rate of Change of Pseudo-range Corrections for GLONASS

Name: /par/rtcm/base/zrate/glo
Access: rw
Type: boolean
Values: on,off
Default: on

on – receiver will set the rate of change of the pseudo-range corrections to zero for GLONASS satellites in RTCM 2.x message types 1, 9, 31, and 34. In some cases it may improve DGPS accuracy.

off – receiver will put computed values into the messages.

Use Local Datum for Referencing Differential Corrections

Name: /par/rtcm/base/locdtm
Access: rw
Type: boolean
Values: on,off
Default: off

on – the datum specified by the /par/pos/datum/cur parameter will be used for referencing GPS and GLONASS differential corrections.

off – receiver will use WGS-84 and PE-90 datums for referencing GPS and GLONASS corrections, respectively.

Note: Ensure that rover uses the same setting for its /par/rtcm/rover/locdtm parameter. If use of local datum is enabled, the same local datum should be specified at both the base station and the rover (see /par/pos/datum/cur parameter on page 249).

This parameter affects RTCM 2.x message types 1, 9, 20, 21, 31 and 34.

The RTCM 2.x standard recommends using WGS-84 and PE-90 for referencing GPS and GLONASS differential corrections, respectively[1]. In some cases, however, it can be desirable to transmit corrections referenced to a local datum.

For example, in code differential, coordinates of the base station can be given in a local datum. In this case, corrections referenced to this local datum may be transmitted to the rover. At the rover side, provided that the same local datum is chosen, a user can obtain the position expressed in the same local datum. Thus, it is possible to obtain the coordinates, expressed in a local datum, without any transformations from local datum to, say, WGS-84 datum prior to transmitting the corrections. Thus, this procedure provides a comfortable method for obtaining coordinates, expressed in a local datum. However, some limitations of this procedure should be mentioned:

1. The rover should "know" that differential corrections are referenced to a local datum. If a base station serves as a reference for many rovers, each of those rovers should use the local datum specified at the base station, otherwise the rover coordinates can be distorted.
2. On long baselines, referencing the differential corrections to a local datum may introduce an additional error in the coordinates.

Satellite Constellation for RTCM 2.x Messages

Name: `/par/rtcm/base/sys`
Access: `rw`
Type: `array [0…2] of boolean`
Values: `{on|off,on|off}`
Default: `{on,on}`

This parameter instructs the base receiver to include in RTCM 2.x message types 18, 19, 20 and 21 only data associated with the specified satellite constellation. The first and the second values correspond to GPS and GLONASS, respectively. By default, all of available GPS and GLONASS satellites will be taken into account when generating these message types.

Maximum Number of Satellites for RTCM 2.x Messages

Name: `/par/rtcm/base/svm`
Access: `rw`
Type: `integer`
Values: `[0…127]`
Default: `0`

This parameter affects message types 18, 19, 20, and 21. It allows to save bandwidth of slow communication channels.

`0` – all of the available satellites will be included in corresponding messages.

1. See RTCM recommended standards for differential GNSS (Global Navigation Satellite System) service, version 2.3, August 20, 2001. (RTCM PAPER 136-2001/SC104-STD).

`[1…127]` – not more than the specified number of satellites will be included in corresponding messages. The satellites that will be excluded are those with lowest elevations.

RTCM 2.x Base Station Health

Name: `/par/rtcm/base/health`
Access: `rw`
Type: `enumerated`
Values: `good,bad,unknown`
Default: `good`

The values correspond to the following terms of the RTCM 2.x standard:

`good` – normal performance

`bad` – health status is "reference station not working"

`unknown` – health status is "reference station transmission not monitored".

RTCM 2.x Base Station Identifier

Name: `/par/rtcm/base/stid`
Access: `rw`
Type: `integer`
Values: `[0…1023]`
Default: `(receiver serial number)`

Text for RTCM 2.x Message Types 16 and 36 (GPS)

Name: `/par/rtcm/base/text/gps`
Access: `rw`
Type: `string[0…90]`
Default: `(empty string)`

Text for RTCM 2.x Message Types 16 and 36 (GLONASS)

Name: `/par/rtcm/base/text/glo`
Access: `rw`
Type: `string[0…90]`
Default: `(empty string)`

Enable CA/L1 in RTCM 2.x Message Types 18 through 21

Name: `/par/rtcm/base/meas/ca`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

Enable P/L1 in RTCM 2.x Message Types 18 through 21

Name: /par/rtcm/base/meas/p1
Access: rw
Type: boolean
Values: on,off
Default: off

Enable P/L2 in RTCM 2.x Message Types 18 through 21

Name: /par/rtcm/base/meas/p2
Access: rw
Type: boolean
Values: on,off
Default: on

Use Smoothed Pseudo-ranges in RTCM 2.x Message Types 19 through 21

Name: /par/rtcm/base/smooth
Access: rw
Type: boolean
Values: on,off
Default: off

Enable Delimiting Characters for RTCM 2.x Messages

Name: /par/rtcm/base/end
Access: rw
Type: boolean
Values: on,off
Default: off

on - receiver will insert up to two delimiting characters at the end of every RTCM 2.x message (these characters are specified by the value of /par/rtcm/base/es parameter, see below).

Delimiting Character(s) for RTCM 2.x Messages

Name: /par/rtcm/base/es
Access: rw
Type: list {integer,integer}
Values: {[-1…127],[-1…127]}
Default: {13,10}

This parameter determines up to two delimiting characters that will be added to the end of every RTCM 2.x message. The value -1 disables corresponding character, other values specify ASCII code of the character.

RTCM 2.x Rover Parameters

Use Local Datum for Referencing Differential Corrections on Rover

Name: /par/rtcm/rover/locdtm
Access: rw
Type: boolean
Values: on,off
Default: off

The value of this parameter should match those of the /par/rtcm/base/locdtm parameter on the reference station. Refer to the description of the aforementioned parameter on page 361 for details.

Use Not Monitored Reference Station

Name: /par/rtcm/rover/usenm
Access: rw
Type: boolean
Values: on,off
Default: on

This parameter enables/disables use of data received from a reference station whose health status code is set to 110. According to the RTCM 2.x standard, this code indicates that data transmitted by this station is "not monitored."

on - use data from a reference station even when its status code is 110.

off - do not use data from a reference station which status code is 110.

Check Sequence Number From the RTCM 2.x Messages

Name: /par/rtcm/rover/seqnum
Access: rw
Type: boolean
Values: on,off
Default: on

on - use message sequence numbers in computation of data link quality.

off - do not use message sequence numbers in computation of data link quality.

An RTCM 2.x message has a data field called sequence number. Sequence numbers allow the receiver to check whether any messages have been lost when receiving RTCM 2.x data. Such checking is enabled by default. If the receiver detects that a message is lost (i.e. the difference between the current and the previous sequence numbers is not equal to unity), the "bad message counter" will be incremented. The data link quality is available in GREIS [DL] message described on page 138.

RTCM 2.x Version to Assume at the Rover

Name: `/par/rtcm/rover/ver`
Access: `rw`
Type: `enumerated`
Values: `v2.1,v2.2,v2.3,v2.4`
Default: `v2.3`

This parameter allows you to use JAVAD GNSS receivers together with reference stations that transmit messages in the format specified by older versions of RTCM 2.x standard. Note that only RTK messages are affected.

Multiple Message Indicator Mode

Name: `/par/rtcm/rover/mmi`
Access: `rw`
Type: `enumerated`
Values: `def,on,off`
Default: `off`

RTCM 2.x message types 18/19 and 20/21 have a flag called "Multiple Message Indicator". This flag serves to identify the last message in a group of such messages referenced to the same time. Unfortunately, different manufacturers have interpreted this flag differently, which resulted in incompatibility between the formats used by different developers. The version 2.3 of the RTCM 2.x standard[1], unlike the version 2.2, defines this flag explicitly and unambiguously. This flag will allow a JAVAD GNSS receiver configured as a rover to be capable of using RTCM 2.3 messages transmitted by other manufacturers' base receivers.

`def` – this is the same as `on` when RTCM version (as defined by the `/par/rtcm/rover/ver` parameter) is set to either v2.1 or v2.2, and is the same as `off` when RTCM version is set to v2.3.

`on` – receiver will always verify the flag. This is expected to reduce the latency time since in this case the rover receiver needn't wait for arriving RTCM 2.x messages referenced to the next epoch. Note, however, that this will be possible only on condition that the Multiple Message Indicator behaves exactly as it is specified in version 2.3. Otherwise the data received may be interpreted incorrectly.

`off` – receiver will have to wait[2] for RTCM 2.x data corresponding to the next epoch to arrive to accept the data from the current epoch.

Note: The default value for this parameter allows rover to work reliably with any version of RTCM messages transmitted by the reference station. It is recommended to set the mode to `on` only if it

1. RTCM PAPER 136-2001/SC104-STD
2. Though this could be sometimes compensated by the *complete epoch received* logic described later.

is known that the base receiver outputs RTCM version 2.3. Otherwise selecting this mode may cause malfunction of the rover receiver in RTK.

Enable Complete Epoch Received Logic

Name: `/par/rtcm/rover/full`
Access: `rw`
Type: `enumerated`
Values: `def,on,off`
Default: `def`

This parameter specifies if the *complete epoch received* logic is applied while the receiver decoding RTCM data for RTK. The term *complete epoch received* means the receiver obtained all the data necessary for the given epoch. For example, for a GPS/GLONASS dual frequency receiver the term means the receiver has acquired the code and phase measurements on both frequencies for at least one satellite for either constellation.

`def` – the logic is either turned on or off depending on the parameter `/par/rtcm/rover/ver`, — it is turned off for the RTCM versions v2.1 and v2.2, and is turned on for version v2.3.

`on` – the logic is turned on.

`off` – the logic is turned off.

In RTCM versions earlier than 2.3, due to different interpretation of the RTCM 2.x standard by different manufacturers, it is not always possible to identify the end of epoch based on the RTCM format itself. In this case the receiver can either wait when an RTCM 2.x message with a different time arrives (thus increasing latency), or apply the *complete epoch received* logic. The latter can decrease the RTK corrections' latency by eliminating the delay required for receiving the first message referenced to the next epoch (typically 1 second).

Source of Antenna Reference Position

Name: `/par/rtcm/rover/refsrc`
Access: `rw`
Type: `enumerated`
Values: `auto,l1pc,arp`
Default: `auto`

This parameter allows to maintain compatibility between various ways of expressing the reference antenna position.

Note: JAVAD GNSS recommends that you use this parameter's default value unless you are completely sure of the message set being transmitted by the reference station.

The RTCM standard version 2.3 supports new message type 24 which provides the exact location of the reference station and the antenna height as the distance to the Antenna Reference Point (ARP). Remember that the previous versions of the standard use the message types 3 and 32 to broadcast the coordinates of the reference antenna and these messages contain the coordinates of the Antenna L1 Phase Center (APC).

`auto` – if both message sets (3/31/22 and 23/24) are transmitted in the same data stream, the rover receiver will use ARP coordinates (message type 24). If only one of the message sets is transmitted, the rover receiver will automatically extract the antenna coordinates available in the given data stream and applies them to the RTK engine.

`l1pc` – receiver will use the coordinates of the APC extracted from message types 3 or 32.

`arp` – receiver will use the coordinates of the ARP extracted from message type 24.

Reset the RTCM 2.x Decoders

Name: `/par/rtcm/rover/reset`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – receiver will reset the RTCM decoders and then will restore the value `off` of this parameter.

`off` – ignored.

For example, you may use this parameter to reset the logic associated with the value used in the `/par/rtcm/rover/refsrc` parameter.

4.4.18 RTCM 3.x Parameters

RTCM 3.x Reference Station Parameters

RTCM 3.x Reference Station Identifier

Name: `/par/rtcm3/base/stid`
Access: `rw`
Type: `integer`
Values: `[0…4095]`
Default: `(receiver serial number)`

This parameter contains the reference station ID that will be part of the RTCM 3.x correction messages. On the rover side, this ID allows easy identification of the reference station whose RTCM 3.x messages are being received by the rover.

Maximum Number of Satellites for RTCM 3.x Messages

Name: `/par/rtcm3/base/svm`
Access: `rw`
Type: `integer`
Values: `[0…127]`
Default: `0`

This parameter affects RTCM 3.x messages containing per-satellite data. It allows to save bandwidth of slow communication channels.

`0` – all of the available satellites will be included in corresponding messages.

`[1…127]` – not more than the specified number of satellites will be included in corresponding messages. The satellites that will be excluded are those with lowest elevations.

Clock Steering of RTCM3 Observations

Name: `/par/rtcm3/base/clk/steering`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

`y` – turn on clock steering for all RTCM3 observations as required by the RTCM standard

`n` – turn off clock steering for all RTCM3 observations to match those in the standard GREIS messages

Text for RTCM 3.x Message

Name: `/par/rtcm3/base/text`
Access: `rw`
Type: `string[0…127]`
Values: `arbitrary`
Default: `(empty string)`

The value of this parameter will be included into proprietary RTCM 3.x text message (Message Type 4091).

RTCM 3.x MSM Output

Enable CA/L1 in MSM

Name: /par/rtcm3/base/msm/ca
Access: rw
Type: boolean
Values: y|n
Default: y

Enable P/L1 in MSM

Name: /par/rtcm3/base/msm/p1
Access: rw
Type: boolean
Values: y|n
Default: y

Enable P/L2 in MSM

Name: /par/rtcm3/base/msm/p2
Access: rw
Type: boolean
Values: y|n
Default: y

Enable L2C in MSM

Name: /par/rtcm3/base/msm/l2c
Access: rw
Type: boolean
Values: y|n
Default: y

Enable L5 in MSM

Name: /par/rtcm3/base/msm/l5
Access: rw
Type: boolean
Values: y|n
Default: y

Enable L1C in MSM

Name: /par/rtcm3/base/msm/l1c
Access: rw
Type: boolean
Values: y|n
Default: y

RTCM 3.x Rover Parameters

Enable GLONASS Biases Application

Name: /par/rtcm3/rover/globias
Access: rw
Type: boolean
Values: n,y
Default: y

Enable application of GLONASS code-phase biases from RTCM3.2 message 1230.

RTCM 3.2 MSM Processing Compatibility

Name: /par/rtcm3/rover/msm/compat
Access: rw
Type: enumerated
Values: javad,trimble
Default: javad

Specifies compatibility mode of processing of RTCM3.2 MSM messages with particular vendor.

RTCM3 Decimated Output

RTCM3 decoder is capable to output received RTCM3 messages, optionally decimated and wrapped, into user-specified output port.

Port for Output of RTCM3 messages.

Name: /par/rtcm3/rover/out/port
Access: rw
Type: enumerated
Values: /[oport],/dev/null
Default: /dev/null

This parameter specifies output port to which RTCM3 messages being received from base should be sent.

`/[oport]` - messages will be sent to specified output port

`/dev/null` - messages won't be sent to any port

Period of Decimation of RTCM3 Messages.

Name: `/par/rtcm3/rover/out/period`
Access: `rw`
Type: `float [seconds]`
Values: `[0…86400]`
Default: `0`

This parameter specifies the period of decimation of RTCM3 messages being received. When this parameter is `0`, all the messages are output to the port specified by `/par/rtcm3/rover/out/port` parameter.

When this parameter is greater than 0, it affect only those RTCM3 messages that have time tag. Such messages are output only when their time tag modulo the value of this parameter is equal to `0`. Messages without time tag are always output to specified port.

Wrapping of RTCM3 Messages.

Name: `/par/rtcm3/rover/out/wrap`
Access: `rw`
Type: `integer`
Values: `[-1…255]`
Default: `82`

`-1` - output RTCM3 messages as is, without any wrapping

`0…255` - wrap RTCM3 messages into GREIS `[>>]` messages before output, and use specified value for the `id` field of the `[>>]` message.

Got Transformation Parameters

These parameters contains transformation data got from reference station through corresponding RTCM3 messages.

Got Datum Transformation

Name: `/par/rtcm3/rover/transform/datum`
Access: `r`
Type: `{{srcId,as,bs},{trgId,at,bt},`
`{plateId,compInd,hgtInd,horQual,vrtQual,`

```
 latOrg,lonOrg,latExt,lonExt},
{dX,dY,dZ,Scale,RX,RY,RZ,Xp,Yp,Zp}}
```

Default: {}

Datum transformation parameters got on the rover side from RTCM3 messages 1021-1022.

`srcId` - string identifier of source system

`as` - semi-major axis of source system [meters]

`bs` - semi-minor axis of source system [meters]

`trgId` - string identifier of source system

`at` - semi-major axis of target system [meters]

`bt` - semi-minor axis of target system [meters]

`plateId` - lithosphere plate number

`compInd` - transformation method to be used

`hgtInd` - height computation method to be used

`horQual` - horizontal quality of transformation

`vrtQual` - vertical quality of transformation

`latOrg` - latitude of origin, area of validity [arc-seconds]

`lonOrg` - longitude of origin, area of validity [arc-seconds]

`latExt` - N/S extension, area of validity [arc-seconds]

`lonOrg` - E/W extension, area of validity [arc-seconds]

`dX,dY,dZ` - translation of coordinate system [meters]

`Scale` - scale of transformation [ppm]

`RX,RY,RZ` - rotation of coordinate system [arc-seconds]

`Xp,Yp,Zp` - coordinates of rotation for Molodenski-Badekas method [meters]

Got Residual of Transformation

Name: `/par/rtcm3/rover/transform/resid`
Access: `r`
Type:

```
{{ELLIPS,latOrg,lonOrg,latExt,lonExt,
  latOffs,lonOffs,hgtOffs,horShft,vrtShft,
  horInterp,vrtInterp,horQual,vrtQual,MJDnum},
 {{latRes1,lonRes1,hgtRes1},…,
  {latRes16,lonRes16,hgtRes16}}}

or

{{PLANE,nrthOrg,eastOrg,nrthExt,eastExt,
```

```
            nrthOffs,eastOffs,hgtOffs,horShft,vrtShft,
            horInterp,vrtInterp,horQual,vrtQual,MJDnum},
           {{nrthRes1,eastRes1,hgtRes1},…,
            {nrthRes16,eastRes16,hgtRes16}}}
```

Default: `{}`

Residual of transformation got on the rover side from RTCM3 messages 1023-1024.

This information could be present in two distinct representations, ELLIPS and PLANE. Common fields of these two formats are:

`horShft` – flag of application of horizontal shift (0 - no shift, 1 - apply shift)

`vrtShft` – flag of application of vertical shift (0 - no shift, 1 - apply shift)

`horInterp` – horizontal interpolation method (0-bilinear, 1-biquadratic, 2-bispline)

`vrtInterp` – vertical interpolation method (0-bilinear, 1-biquadratic, 2-bispline)

`horQual` – horizontal quality of residual application

`vrtQual` – vertical quality of residual application

`MJDnum` – modified Julian day number

ELLIPS format fields are:

`ELLIPS` – identifier of residual for ellipsoidal grid representation

`latOrg` – latitude of origin, area of validity [arc-seconds]

`lonOrg` – longitude of origin, area of validity [arc-seconds]

`latExt` – N/S extension, area of validity [arc-seconds]

`lonOrg` – E/W extension, area of validity [arc-seconds]

`latOffs` – mean latitude offset for all 16 points [arc-seconds]

`lonOffs` – mean longitude offset for all 16 points [arc-seconds]

`hgtOffs` – mean height offset for all 16 points [meters]

`latResN` – residual in latitude for point N (N=1…16) [arc-seconds]

`lonResN` – residual in longitude for point N (N=1…16) [arc-seconds]

`hgtResN` – residual in height for point N (N=1…16) [meters]

PLANE format fields are:

`PLANE` – identifier of residual for plane grid representation

`nrthOrg` – northing of origin, area of validity [meters]

`eastOrg` – easting of origin, area of validity [meters]

`nrthExt` – N/S extension, area of validity [meters]

`eastOrg` – E/W extension, area of validity [meters]

`nrthOffs` – mean local northing offset for all 16 points [meters]

`eastOffs` – mean local easting offset for all 16 points [meters]

`hgtOffs` – mean height offset for all 16 points [meters]

`nrthResN` – residual in local northing for point N (N=1…16) [meters]

`eatsResN` – residual in local easting for point N (N=1…16) [meters]

`hgtResN` – residual in height for point N (N=1…16) [meters]

Got Map Projection

Name: `/par/rtcm3/rover/transform/proj`
Access: `r`
Type: `{projType,latNO,lonNO,addSNO,FE,FN}`
Default: `{}`

Map projection data got on the rover side from RTCM3 messages 1025-1027.

`projType` – type of map projection (0 - no projection)

`latNO` – latitude of natural origin [degrees]

`lonNO` – longitude of natural origin [degrees]

`addSNO` – scale factor, should be added to 993000 [ppm]

`FE` – false easting [meters]

`FN` – false northing [meters]

4.4.19 CMR Parameters

CMR Reference Station Parameters

Receiver Motion State for CMR

Name: `/par/cmr/base/motion`
Access: `rw`
Type: `enumerated`
Values: `unknown,static,kinematic`
Default: `unknown`

`unknown` – motion state is undefined. Corresponding CMR messages will contain reference coordinates entered by the user.

`static` – motion state is static (i.e., not moving). Corresponding CMR messages will contain reference coordinates entered by the user.

kinematic – motion state is moving. Corresponding CMR messages will use the current position estimate computed by the receiver (it can be an RTK, DGPS or single-point position estimate depending on which positioning mode is on) for reference station coordinates.

Data for CMR Message Type 2

Name: /par/cmr/base/desc
Access: rw
Type: {string[0…8],string[0…16],string[0…50]}
Values: {(arbitrary),(arbitrary),(arbitrary)}
Default: {(empty string),(empty string),(empty string)}

This parameter contains three strings specifying "short station ID", "COGO code", and "long station ID", in this order.

CMR Reference Station Identifier

Name: /par/cmr/base/stid
Access: rw
Type: integer
Values: [0…31]
Default: (receiver serial number)

Maximum Number of Satellites for CMR Messages

Name: /par/cmr/base/svm
Access: rw
Type: integer
Values: [0…127]
Default: 0

This parameter affects CMR messages containing per-satellite data. It allows to save bandwidth of slow communication channels.

0 – all of the available satellites will be included in corresponding messages.

[1…127] – not more than the specified number of satellites will be included in corresponding messages. The satellites that will be excluded are those with lowest elevations.

Enable P/L2 in CMR Messages

Name: /par/cmr/base/meas/p2
Access: rw
Type: boolean
Values: on, off
Default: on

Substitute P/L1 for CA/L1 in CMR Messages

Name: /par/cmr/base/pcode
Access: rw
Type: boolean
Values: on,off
Default: off

Type of CMR Message to Use for GLONASS

Name: /par/cmr/base/glo/type
Access: rw
Type: integer
Values: [3…7]
Default: 3

Since the CMR format does not allow for any predefined message type for GLONASS measurements, you must specify a message type for GLONASS raw data on your own. This is the purpose that this parameter serves.

Since some new CMR message types may appear in the future, be sure that the message type assigned to GLONASS measurements is different from all the other CMR message types. Should a conflict due to ambiguous message types occur, you may need to re-define the message type associated with GLONASS measurements (just choose any unused number within a range of [3…7]).

In addition, ensure that both the reference station and the rover receiver use the same message type for GLONASS data (see /par/cmr/rover/glo/type below).

CMR Antenna Type

Name: /par/cmr/base/ant/type
Access: rw
Type: integer
Values: [0…255]
Default: 0

This parameter contains CMR antenna numeric identifier for the type of antenna being used at the reference station.

Note: To find out relationship between your antenna type and the corresponding CMR antenna ID, use, for example, the /par/antdb/ids parameter.

CMR Receiver Type

Name: /par/cmr/base/rcv/type
Access: rw
Type: integer
Values: [0…255]
Default: 0

This parameter contains CMR receiver numeric identifier for the type of receiver being used at the reference station.

CMR Plus Reference Station Compatibility

Name: /par/cmr/base/idxset
Access: rw
Type: boolean
Values: on,off
Default: off

This parameter is used to preserve backward compatibility with the JNS firmware version 2.2 when working with CMR Plus messages.

on - instructs receiver to code CMR Plus message in accordance with the firmware version 2.2, thus, rover receivers which are uploaded with the version 2.2 can work properly.

off - use standard format for CMR Plus message.

CMR Rover Parameters

Type of CMR Message to Expect for GLONASS

Name: /par/cmr/rover/glo/type
Access: rw
Type: integer
Values: [3…7]
Default: 3

See /par/cmr/base/glo/type above for details.

CMR Plus Rover Compatibility

Name: /par/cmr/rover/idxset
Access: rw
Type: boolean
Values: on,off
Default: off

This parameter is used to preserve compatibility with the JNS firmware version 2.2 when working with CMR Plus message.

`on` – receiver will decode CMR Plus message in accordance with the firmware version 2.2, thus, rover receivers can work with the reference receiver, which is uploaded with the firmware version 2.2 and newer.

`off` – receiver will decode CMR Plus messages in a standard way.

4.4.20 Parameters of Generic GREIS Messages

Masks and Counters

Elevation Mask for Measurements Output

Name: `/par/out/elm/[oport]`
Access: `rw`
Type: `integer [degrees]`
Values: `[-90…90]`
Default: `5`

Measurements for satellites whose elevation angles are less than the specified mask won't be output to the `[oport]`.

Satellites Number Mask for Measurements Output

Name: `/par/out/minsvs/[oport]`
Access: `rw`
Type: `integer`
Values: `[0…255]`
Default: `0`

The receiver will not output measurements into the given `[oport]` as long as the number of satellites whose elevations exceed current elevation mask for measurements output is fewer than this parameter.

Output Epochs Counters

Name: `/par/out/epochs/[oport]`
Access: `r`
Type: `integer`
Default: `0`

This counter is incremented every time a new [~~](RT) message is output to the port `[oport]`. It is cleared every time the [~~](RT) message is being enabled to be output to the port, provided it is not already enabled.

Output Maximum Number Of Messages

Name: `/par/out/max`
Access: `r`
Type: `integer`

This parameter shows maximum number of messages allowed to be simultaneously enabled to any given port.

Antenna Output Masks

Name: `/par/out/ant/[oport]`
Access: `rw`
Type: `array [a…d] of boolean`
Values: `{y|n,y|n,y|n,y|n}`
Default: `{y,n,n,n}`

Each element of the array enables output of observables taken from corresponding antenna to the port `[oport]`.

Note: This parameter is only available for multi-antenna receivers.

Antenna N Output Mask

Name: `/par/out/ant/[oport]/N (N=[a…d])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y (N=0); n (N>0)`

This parameter enables output of observables taken from antenna `N` to the port `[oport]`.

Note: This parameter is only available for multi-antenna receivers.

Logging History

History logging provides statistical information on the raw data (receiver measurements) being logged to a selected stream. After *history logger* is associated with a specific output stream and data logging to this stream is enabled, *history logger* will start to collect and record corresponding information. It will record one bit of information per satellite every N seconds. This bit is set to unity if and only if at least some of the measurements have been stored in the last N seconds, and there have been no loss-of-lock events for the given satellite in the last N seconds. If either or both of these conditions are not met, a zero bit is recorded to the logging history.

The logging history has a limited capacity: a maximum of 32 satellites, 128 bits per satellite. Satellites for which all the bits are zero are not included in the logging history.

The information gathered by the logger could be obtained by means of the [LH] receiver message. For information about [LH] message, see "[LH] Logging History" on page 134.

Logging Period

Name: `/par/out/logh/period`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…86400]`
Default: `30`

The history logging period. One bit per satellite is recorded every `period` seconds.

Output Stream to be Monitored

Name: `/par/out/logh/stream`
Access: `rw`
Type: `enumerated`
Values: `/[oport],/dev/null`
Default: `/dev/null`

The name of the output stream the history logger should gather information for. If the parameter is set to `/dev/null`, history logging will be disabled.

4.4.21 Parameters of Raw Navigation Data Messages

Raw Navigation Data Output Mode

Name: `/par/raw/data/mode`
Access: `rw`
Type: `enumerated`
Values: `noerr,err,nocheck`
Default: `noerr`

Instructs receiver how to deal with raw navigation data messages when CRC, parity or any other type of error occurs.

`noerr` – receiver outputs only those raw data where there are no uncorrected errors

`err` – receiver outputs even raw data with uncorrected errors

`nocheck` – receiver outputs raw data unconditionally, without error checking. Receiver still has to find correct preamble pattern in the signal for any data to be output.

4.4.22 Parameters of NMEA messages

NMEA Standard Version

Name: /par/nmea/ver
Access: rw
Type: enumerated
Values: v2.2,v2.3,v3.0,v4.1,v4.11,v4.30
Default: v4.11

This parameter instructs the receiver to generate NMEA messages according to the specified NMEA-0183 standard[1].

Note: v4.1e legacy value is accepted as synonym for v4.11

NMEA Datum

Name: /par/nmea/locframe
Access: rw
Type: boolean
Values: on,off
Default: on

on - position referenced to the datum specified by the /par/pos/datum/cur parameter will be output in NMEA messages.

off - position referenced to the WGS84 datum will be output in NMEA messages.

Use "GP" as Talker ID in NMEA Messages

Name: /par/nmea/gp
Access: rw
Type: boolean
Values: on,off
Default: off

on - always use "GP" as talker ID, except for NMEA GSA messages.

off - use "GP", "GN", or "GL" as talker ID according to the NMEA standard, except for NMEA GGA message.

This parameter instructs the receiver to use "GP" as Talker ID in NMEA messages. This mode is implemented for compatibility with legacy equipment that may not be capable of recognizing "GN" or "GL" as Talker IDs.

NMEA GGA and GSA messages are not affected by this parameter. NMEA GSA will always use "GP", "GN", or "GL" as talker ID according to the NMEA standard. NMEA

1. NMEA-0183 Standard For Interfacing Marine Electronic Devices v.3.0. July 1, 2000.

GGA message will always use “GP” as talker ID for better backward compatibility. Newer software could utilize more modern NMEA GNS message instead of GGA.

Enable NMEA Messages When UTC Time is Unavailable.

Name: /par/nmea/notime
Access: rw
Type: boolean
Values: on,off
Default: on

off – receiver will not output any NMEA messages (excluding GSV) when UTC time is not available.

on – receiver will output NMEA messages when UTC time is not available, with zero UTC time.

Limit the Total Number of Satellites in GGA by 12

Name: /par/nmea/ggalim
Access: rw
Type: boolean
Values: on,off
Default: off

on – no more than 12 satellites will be reported in GGA message.

off – actual number of satellites will be reported in GGA even when it exceeds 12.

In accordance with the NMEA-0183 standard, the total number of satellites in a GGA sentence is limited to 12. In practice, however, there may be more than 12 GPS satellites in sight. This parameter serves for compatibility with any software that strictly follows the NMEA-0183 standard.

Output Mode for HDT and ROT Messages

Name: /par/nmea/head/fixed
Access: rw
Type: boolean
Values: on,off
Default: off

on – HDT and ROT messages will be output in RTK with fixed ambiguities position computation mode only.

off – HDT and ROT messages will be output with any position computation mode where corresponding values are calculated.

Enable GRS and GSA to Contain More Than 12 Satellites

Name: /par/nmea/grsgsa
Access: rw
Type: boolean
Values: on,off
Default: off

on - GRS and GSA messages may contain more than 12 GNSS satellites.

off - GRS and GSA messages will never contain more than 12 satellites.

Mantissa Length for Fractional Seconds of UTC Time

Name: /par/nmea/frac/sec
Access: rw
Type: integer
Values: [0…3]
Default: 2

This parameter specifies the number of digits in the fractional part of the seconds of UTC time.

Mantissa Length of Fractional Minutes for GGA Message

Name: /par/nmea/frac/min/GGA
Access: rw
Type: integer
Values: [1…7]
Default: 7

This parameter specifies the length of mantissa for representation of fractional minutes of latitude and longitude for GGA message.

Mantissa Length of Fractional Minutes for GGAs Message

Name: /par/nmea/frac/min/GGAs
Access: rw
Type: integer
Values: [1…7]
Default: 3

This parameter specifies the length of mantissa for representation of fractional minutes of latitude and longitude for GGAs message. GGAs is the same GGA message but with its own mantissa length.

Mantissa Length of Fractional Minutes for GLL Message

Name: /par/nmea/frac/min/GLL
Access: rw
Type: integer
Values: [1…7]
Default: 7

This parameter specifies the length of mantissa for representation of fractional minutes of latitude and longitude for GLL message.

Mantissa Length of Fractional Minutes for GNS Message

Name: /par/nmea/frac/min/GNS
Access: rw
Type: integer
Values: [1…7]
Default: 7

This parameter specifies the length of mantissa for representation of fractional minutes of latitude and longitude for GNS message.

Mantissa Length of Geoidal Separation and Orthometric Height

Name: /par/nmea/frac/alt
Access: rw
Type: integer
Values: [1…4]
Default: 4

This parameter specifies the number of digits in the fractional meters for both geoidal separation and orthometric height (altitude above the geoid).

Mantissa Length of Fractional Degrees for VTG Message

Name: /par/nmea/frac/deg/VTG
Access: rw
Type: integer
Values: [1…3]
Default: 3

Mantissa Length of Fractional Speed for VTG Message

Name: /par/nmea/frac/speed/VTG
Access: rw
Type: integer
Values: [1…4]
Default: 4

Mantissa Length of Fractional Residuals for GRS Message

Name: /par/nmea/frac/res/GRS
Access: rw
Type: integer
Values: [0…4]
Default: 3

Mantissa Length of Fractional Degrees for HDT and ROT Messages

Name: /par/nmea/frac/deg/HDT
Access: rw
Type: integer
Values: [1…3]
Default: 3

Offset for True Heading from HDT message

Name: /par/nmea/head/offset
Access: rw
Type: float [degrees]
Values: (-360…360)
Default: 0

This parameter defines the direction with respect to which the True Heading will be calculated. The direction is defined as specified number of degrees clockwise from the True North.

NMEA GSV Verbose Mode

Name: /par/nmea/GSV/verbose
Access: rw
Type: boolean
Values: n,y
Default: n

n – output single GSV message for every SV, with maximum SNR value of all the signals.

y – output separate GSV message for every signal. WARNING: this produces very lengthy output, but matches NMEA recommendations better.

4.4.23 Parameters of BINEX Messages

BINEX Site Name

Name: /par/binex/site
Access: rw
Type: string[0…127]
Values: (any string)
Default: (empty string)

The value of this parameter will be output into the field 0x04 of the BINEX record 0x00-00.

BINEX Data Identifier

Name: /par/binex/data_id
Access: rw
Type: string[0…4]
Values: (any string)
Default: (empty string)

The value of this parameter will be output into the field 0x0f of the BINEX record 0x00-00. If the length of the string is less than 4 characters, the value to be output to the field 0x0f will be padded on the right to 4 characters. The padding is performed using spaces.

Enable Fields of BINEX Record 0x00-00

Name: /par/binex/00_00
Type: list {04,0f,17,19,1a,1b,1d,1f} of boolean
Values: {on|off,…,on|off}
Default: {on,…,on}

Each element of this parameter enables output of corresponding field of BINEX record 0x00-00. When an element is on, the output of corresponding field is enabled, when an element is off, the output of corresponding field is disabled.

Note: To turn all the fields on or off, use set,/par/binex/00_00/,on, or set,/par/binex/00_00/,off command, respectively. Use separate field parameters described below to control separate fields.

Enable Field "F" of BINEX Record 0x00-00

Name: /par/binex/00_00/F (F=[04,0f,17,19,1a,1b,1d,1f])
Access: rw
Type: boolean
Values: on,off
Default: on

4.4.24 File Management

Existing Files

List of Existing Files

Name: /log
Access: r
Type: list {…}
Default: {}

For each existing file, this list includes an entry called after the name of the file containing the file attributes. File attributes have the format {size,mtime} where:

size - the size of the file in bytes

mtime - the time of the last modification of the file in the format:

YYYYYMMMDDDhhhmmmsss

where YYYY is year, MM is month, DD is day, hh is hours, mm is minutes and ss is seconds.

Attributes of the File NAME

Name: /log/NAME
Access: r
Type: {size,mtime}

For an existing file NAME, this parameter contains its attributes (see description of the /log parameter). In addition to print and list, you can use remove commands with this parameter to remove the file NAME.

Size of the File NAME

Name: /log/NAME&size
Access: r
Type: integer [bytes]

Contents of the File NAME

Name: /log/NAME&content
Access: r

When you print this parameter, receiver will output raw contents of the file, i.e., not wrapped into [RE] messages. You can terminate the output any time by sending "#" character to the receiver. If you use command identifier, receiver will output [RE] message containing the identifier before the contents of the file.

Examples

Example: Print list of all the file attributes along with their names; remove one of the files; then print the list again:

```
⇒ print,/log:on
⇐ RE032/log={log1127b={      12910,2006Y11M27D13h21m25s},
  RE02D log1127a={      21465,2006Y11M27D13h21m12s}}
⇒ remove,/log/log1127b
⇒ print,/log:on
⇐ RE032/log={log1127a={      21465,2006Y11M27D13h21m12s}}
```

Example: List only the names of all existing files; remove all the files; list the names again:

```
⇒ list
⇐ RE013{log0113a,log1127a}
⇒ remove,/log/
⇒ list
⇐ RE002{}
```

Current Log-files

In this section, the notation `[a…e]` indicates particular log-file; the user should specify single letter in the range.

Current Log-file

Name:	`/cur/file/[a…e]`
Access:	`rw`
Type:	`string`
Values:	`(any existing file name)`
Default:	`(empty string)`

These parameters contain the names of the current log-files, if such files exist, or empty strings, otherwise.

If there is no current log-file, then setting this parameter to an existing file's name will instruct the receiver to open this file for data appending thus making it the new “current” file.

If the current log-file exists, then changing this parameter will instruct the receiver to close the existing “current” file and then open a new “current” file.

If the command refers to a file that does not exist, receiver will create new file with given name and will switch current log-file to it.

To stop data logging and close current log-file use the dm command described on page 41.

Note: /cur/file/a has a synonym, /cur/log, that is provided for compatibility with older firmware versions that didn't have support for multiple log-files.

Current Log-file Size

Name: /cur/file/[a…e]&size
Access: r
Type: integer [bytes]

The size of corresponding log-file, if any. An error message will be reported if there is no current log-file.

Note: /cur/file/a&size has a synonym, /cur/log&size, that is provided for compatibility with older firmware versions that didn't have support for multiple log-files.

Log-files Management Parameters

Parameters described in this section define the rules for automatic and implicit management of the receiver log-files. The automatic and implicit management is performed by automatic file rotation mode (AFRM), and by TriPad interface. *Implicit* in this context means that file management in these cases is performed not through regular GREIS commands, but by internal receiver algorithms.

Automatic File Rotation Mode (AFRM)

JAVAD GNSS receiver has capability to rotate log-files automatically. The term *file rotation* means that the receiver closes previous current file and opens a new one according to the user-defined schedule.

File rotation schedule is independent for each log-file, and rotation happens when AFRM reference time modulo period is equal to phase modulo period:

T_{ref} mod period = phase mod period

If rotation is to be performed at given epoch, a new log-file is opened immediately before the scheduled epoch, so that all of the data tagged with this epoch will be recorded into the new log-file.

For each file:

- T_{ref} – is specified by combination of /par/log/rot/sc/scale common parameter and the file-specific one /par/log/rot/sc/scales/N.
- period – is equal to common /par/log/rot/sc/period parameter multiplied by file-specific /par/log/rot/sc/factors/N factor.

`phase` – is defined by combination of common `/par/log/rot/sc/phase` and the file-specific one `/par/log/rot/sc/phases/N`.

In addition, AFRM uses a counter that is decremented on every epoch where at least one file is rotated, until its value becomes zero. Once the counter is zero, file rotation automatically stops. This feature allows to create as many log-files as necessary and then stop data logging. The counter is initialized simultaneously with AFRM, and its initial value is set through the parameter `/par/log/rot/sc/count`.

Note that a log-file is also opened right after turning AFRM on, but AFRM counter is not decremented on such initial file creation.

When opening a new log-file, the receiver enables output of file-specific set of messages with file-specific output period (update rate). The set of messages and the output periods are programmable.

The JAVAD GNSS receiver is capable of removing the “oldest” log-files, if there is no free memory left to continue data logging. This feature is off by default. Even if you turn it on, your receiver will not delete the files with the earliest file creation times unless AFRM is also on, that minimizes the risk of inadvertent file removal.

Example: Configure AFRM to log 10-minute files at 20Hz and 1-hour files at 5Hz:

```
⇒ %%set,/par/log/rot/mode,off
⇒ %%set,/par/log/rot/sc/period,600     # Base AFRM period: 10 minutes
⇒ %%set,/par/log/rot/sc/factors,{1,6}  # 6 will bring 6×10 minutes = 1h
⇒ %%set,/par/log/a/name/pre,"a20_10_"  # give meaningful name prefix
⇒ %%set,/par/log/a/sc/period,0.05      # 20Hz
⇒ %%set,/par/log/b/name/pre,"b05_60_"  # give meaningful name prefix
⇒ %%set,/par/log/b/sc/period,0.2       #  5Hz
⇒ %%set,/par/log/imp/,n                # Manage…
⇒ %%set,/par/log/imp,{y,y}             # … just 2 files
⇒ %%set,/par/log/rot/mode,on
```

File Rotation Mode

Name: `/par/log/rot/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – enable AFRM.

`off` – disable AFRM.

Turning this parameter from `off` to `on` starts data logging to the log-files enabled by the `/par/log/imp` parameter, copies the value of the `/par/log/rot/sc/count` parameter to `/par/log/rot/count`, and enables files rotation according to their schedules.

The names of the created files are generated automatically as if the `create` command without arguments has been issued. The set of messages specified by `/par/log/[a…e]/msgs` parameter is enabled to be output to the file with the period specified by corresponding `/par/log/[a…e]/sc/period` parameter.

Turning this parameter from `on` to `off` disables files rotation and closes log-files enabled by the `/par/log/imp` parameter.

Enable Oldest File Removal

Name: `/par/log/rot/rmold`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – enable automatic removal of the oldest log-file while AFRM is on and there is not enough free space for files to continue data logging. It doesn't enable automatic removal when AFRM is not active.

`off` – disable automatic removal of files.

Files which names start with dot (.) are never subject for automatic file removal.

Prefer Prefix for Oldest File Removal

Name: `/par/log/rot/rm/pre`
Access: `rw`
Type: `string [0…20]`
Values: `(any string)`
Default: `(empty string)`

If set to empty string, receiver will simply remove oldest file when automatic removal of the oldest file is turned on. If set to a non-empty string, this string is taken as a file name prefix. In this case receiver will preferably remove files which names begin with this prefix, unless number of such files will get less than those specified by `/par/log/rot/rm/keep` parameter. In the latter case, or if there are no such files, receiver will resort to simple removal of oldest file.

Number of Preferably Removable Files to Keep.

Name: /par/log/rot/rm/keep
Access: rw
Type: integer
Values: [0…0x7fffffff]
Default: 24

Number of files matching /par/log/rot/rm/pre that we should try to keep.

Force File Rotation

Name: /par/log/rot/force
Access: rw
Type: boolean
Values: on, off
Default: off

on - provided AFRM is turned on, will force the *rotation event* to be generated at the closest internal receiver epoch.

off - has no effect.

File Rotation Running Counter

Name: /par/log/rot/count
Access: r
Type: integer
Values: [0…2^{31}-1]

This parameter indicates how many *rotation events* remains to happen before the file rotation stops. Zero means an unlimited number of files.

Refer to /par/log/rot/sc/count below for the method of changing this parameter.

File Rotation Scheduling Parameters

Name: /par/log/rot/sc
Access: r
Type: list {period,phase,count,scale,factors,phases,scales}

Files Rotation Period

Name: /par/log/rot/sc/period
Access: rw
Type: integer [seconds]
Values: [60…86400]
Default: 3600

File rotation period for a file is defined as the value of this parameter multiplied by the file-specific factor /par/log/rot/sc/factors/N.

Per-file Rotation Factors

Name: /par/log/rot/sc/factors
Access: rw
Type: array [0…N-1] of integer
Values: {[1…1440],…}
Default: {1,…}

Array of rotation factors for each file. Element 0 corresponds to /cur/file/a, element 1 - to /cur/file/b, etc. The values specify decimation of AFRM events for given file. I.e., effective period of rotation of given file is equal to the AFRM period multiplied by the file-specific factor provided by this parameter.

Files Rotation Phase

Name: /par/log/rot/sc/phase
Access: rw
Type: integer [seconds]
Values: [-86400…86400]
Default: 0

Per-file Rotation Phases

Name: /par/log/rot/sc/phases
Access: rw
Type: array [0…N-1] of integer
Values: {[-86400…86400],…}
Default: {0,…}

Array of phases for each file. Element 0 corresponds to /cur/file/a, element 1 - to /cur/file/b, etc.

If 0, use the value of common /par/log/rot/sc/phase parameter for file rotation phase, otherwise use specified value.

Files Rotation Counter

Name: /par/log/rot/sc/count
Access: rw
Type: integer
Values: [0…231-1]
Default: 0

This parameter specifies the total number of files that will be created before file rotation is turned off, 0 meaning unlimited number of files. The value of this parameter is copied

to /par/log/rot/count whenever /par/log/rot/mode is turned from off to on. In addition, when /par/log/rot/mode is on, setting this parameter will update the value of /par/log/rot/count.

File Rotation Time Scale

Name: /par/log/rot/sc/scale
Access: rw
Type: enumerated
Values: ref,utc,cggtts
Default: ref

Common time scale to use to drive AFRM rotation events for all the files, unless different time scale is specified by /par/log/rot/sc/scales/N parameter for given file.

ref - use receiver reference time T_r

utc - use UTC time scale

cggtts - use CGGTTS specific time scale

Per-file Rotation Time Scales

Name: /par/log/rot/sc/scales
Access: rw
Type: array [0…N-1] of enumerated
Values: {[ref,utc,cggtts],…}
Default: {ref,…}

Array of AFRM rotation time scales for each file. Element 0 corresponds to /cur/file/a, element 1 - to /cur/file/b, etc.

ref - use the value of /par/log/rot/scale parameter

utc,cggtts - use particular time scale

File Name Reuse

Name: /par/log/[a…e]/name/reuse
Access: rw
Type: enumerated
Values: never,start,always
Default: never

never - never reuse file names, i.e., always create unique file name when AFRM generates file names automatically.

`start` - on receiver startup, if AFRM finds that a file with the name it wants to use already exists, AFRM will open this file and append data to it.

`always` - if AFRM finds that file with the name it wants to use already exists on any AFRM event, AFRM will open this file and append data to it.

Enable Implicit Management of Current Log-files

Name: `/par/log/imp`
Access: `rw`
Type: `array [0…1] of boolean`
Values: `{y|n,…,y|n}`
Default: `{y,n}`

The first element of the array corresponds to the `/cur/file/a`, and the second element – to the `/cur/file/b`.

Enable Implicit Management of Specific Current Log-file

Name: `/par/log/imp/N (N=[0…4])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y for N=0, n for N=1`

`N=0` corresponds to the `/cur/file/a`, and `N=1` – to the `/cur/file/b`, …

`y` - corresponding current log-file will be controllable via AFRM and TriPad.

`n` - corresponding current log-file will be ignored by AFRM and TriPad.

Automatically Generated File Names

File name is generated automatically when one creates a file using `create` command without arguments, or through AFRM, or using the TriPad <FN> button to start data recording. Parameters in this section govern automatic file names generation.

An automatically generated file name has the following format:

`P…P<DATE>S…S`

where `P…P` is prefix specified by the value of the receiver parameter `/par/log/[a…e]/name/pre (X=log)`, format of the `<DATE>` is specified by parameters `/par/log/name/date` and `/par/log/[a…e]/name/hour`, and `S…S` is a suffix to make unique names for the files created on the same day.

The first suffix in a day is single letter `'a'` that is changed to the next letter in alphabetical order for every next file created on the same day. After the process gets to `'z'` yet another letter is appended to the suffix, etc. Therefore, the suffixes go in sequence like

this: a,b,…,z,za,zb,…zz,zza,zzb,…. For example, an automatically generated file name for a file created September, 17 may look like this: log0917zzy.

File Name Date Format

Name: /par/log/name/date
Access: rw
Type: enumerated
Values: mmdd,day,yymmdd
Default: mmdd

This parameter specifies the format of the date to be used in automatically generated file names.

mmdd - month (2 digits) and day inside month (2 digits)

day - day inside year (3 digits)

yymmdd - year since 2000 (2 digits), month (2 digits), and day inside month (2 digits)

File Name Prefix

Name: /par/log/[a…e]/name/pre
Access: rw
Type: string[0…20]
Default: log

This parameter determines the file name prefix used as part of automatically generated file names. The string may only contain characters valid for file names, refer to the description of create command on page 45.

File Name Suffix

Name: /par/log/[a…e]/name/suf
Access: rw
Type: string[0…10]
Default: <empty string>

This parameter determines the file name suffix used as part of automatically generated file names. The string may only contain characters valid for file names, refer to the description of create command on page 45.

Example: set file name suffix to ".jps":

⇒ set,/par/log/a/name/suf,".jps"

File Name Use Hour

Name: /par/log/[a…e]/name/hour
Access: rw
Type: boolean
Values: on,off
Default: off

on - append letter 'a' to 'x' denoting hour inside day to the date.

off - do not append hour letter.

Implicit Logging Parameters

These parameters are used when files are implicitly created by either AFRM or TriPad.

Implicit Message Output Period

Name: /par/log/[a…e]/sc/period
Access: rw
Type: float [seconds]
Values: [0…86400]
Default: 1

This parameter specifies the interval of outputting messages into the log-file when data logging is activated with the TriPad or through the AFRM.

Example: Suppose we want to program AFRM to simultaneously store 2 files, 24-hours long each, one with epochs every 1 second, and another one with epochs every 15 seconds. Suppose also that we want to keep only up to 10 1-second files and use the rest of the disk for as many 15-second files as possible. To achieve this, we can specify different prefixes for naming of 1-second and 15-second files, and then use preferable prefix for files removal. Here are commands:

```
⇒ set,/par/log/rot/mode,off
⇒ set,/par/log/rot/rmold,on
⇒ set,/par/log/rot/sc/period,86400
⇒ set,/par/log/a/sc/period,1
⇒ set,/par/log/a/name/pre,"log01_"
⇒ set,/par/log/rot/rm/pre,"log01_"
⇒ set,/par/log/rot/rm/keep,10
⇒ set,/par/log/b/sc/period,15
⇒ set,/par/log/b/name/pre,"log15_"
⇒ set,/par/log/imp,{y,y}
⇒ set,/par/log/rot/mode,on
```

Implicit Message Set to Enable

Name: /par/log/[a…e]/msgs
Access: rw
Type: string <message set name>
Values: [def,usr/0,usr/1,rtk/jps/min,rtk/jps/max]
Default: def

This parameter specifies the name of particular message set that will be enabled for corresponding log-file when data logging is activated with TriPad or through AFRM.

File Push Parameters

File push feature allows to automatically put log-files to external server(s) called *destinations*. Currently FTP and SFTP servers are supported as *destinations* by the file push feature.

When file push feature is turned on, every time a file is closed on receiver, its name is appended to the end of the list of files to be pushed. The file push feature takes file name from the head of the list and tries to push corresponding file to external server. If successful, the file name is removed from the list and next name is processed (if any).

At normal operation as soon as file is added to the list, the file is pushed to the server, and the name is removed from the list, so the list of files to be pushed is usually empty or contains single file name. However, when receiver has problem(s) pushing files, this list will grow. This list is stored in the file called "`.push_files`" in the receiver and receiver reads this list at startup, so no files created while FTP push is active are forgotten to be pushed.

On some of receiver models this feature has also support for automatic conversion of log-files to RINEX, and either just keeping them on the file-system, or pushing them to an external server instead of the original files.

Example: Convert files to RINEX files and leave them there for download

```
⇒ set,/par/log/push/mode,off
⇒ set,/par/log/push/dest/a/host,"rinex:"
⇒ set,/par/log/push/mode,on
```

Example: Convert files to RINEX, tar-gzip them, and send resulting archives out over FTP to the `fp/rinex` directory on the host `192.168.2.10`. Use given credentials (user name and password) to login to the destination host over FTP:

```
⇒ set,/par/log/push/mode,off
⇒ set,/par/log/push/dest/a/host,"rin.tar.gz:192.168.2.10"
⇒ set,/par/log/push/dest/a/workdir,"fp/rinex"
⇒ set,/par/log/push/dest/a/user,"fp-guest"
⇒ set,/par/log/push/dest/a/passwd,"fp-guest-secret"
⇒ set,/par/log/push/mode,on
⇐
```

If you replace `/par/log/push/dest/a/host` setting by:

```
⇒ set,/par/log/push/dest/a/host,"192.168.2.10"
```

receiver will be pushing original files to the host and won't convert them to RINEX.

●

The following parameters constitute configuration of the file push feature.

File Push Mode

Name: `/par/log/push/mode`
Access: `rw`
Type: `boolean`
Values: `off,on`
Default: `off`

`off` - file push mode is off

`on` - file push mode in on

File Push Suspend

Name: `/par/log/push/susp`
Access: `rw`
Type: `boolean`
Values: `off,on`
Default: `off`

`off` - enable pushing

`on` - suspend pushing files to the destination while still collecting file names to be pushed.

Useful, e.g., to create file push schedules using session programming, when one needs to suspend transfers for some time periods.

File Push Reset

Name: /par/log/push/reset
Access: rw
Type: boolean
Values: off,on
Default: off

Setting this parameter to on clears the list of files to be pushed and then the value of the parameter is set back to off.

File Push Host

Name: /par/log/push/dest/D/host D=[a…f]
Access: rw
Type: string[0…128]
Values: <destination designator>
Default: "192.168.2.1"

This parameter specifies destination for file push. The syntax of the parameter is:

```
[prefix:][scheme://][host]
```

where

prefix is one of:

- rin.tar.gz - convert file to be pushed to RINEX files, put the files into tar archive, and gzip it, resulting in a single file named <file>.rin.tar.gz. If <host> is non-empty, push resulting file to the specified <host>, as if <host> were the entire value of the parameter.
- rin.zip - convert file to be pushed to RINEX files, put the files into ZIP archive, resulting in a single file named <file>.rin.zip. If <host> is non-empty, push resulting file to the specified <host>, as if <host> were the entire value of the parameter.
- rinex - convert file to be pushed to separate RINEX files. No actual push anywhere will be performed. The files will be available for download though.
- cggtts.tar.gz - convert file to be pushed to CGGTTS file(s), put the files into tar archive, and gzip it, resulting in a single file named <file>.cggtts.tar.gz. If <host> is non-empty, push resulting file to the specified <host>, as if <host> were the entire value of the parameter.
- cggtts - convert file to be pushed to separate CGGTTS files. No actual push anywhere will be performed. The files will be available for download though.

scheme is one of:

- ftp - use FTP protocol (default)

`sftp` – use SFTP protocol

`host`: if it could be parsed as an IP address, the address is used, otherwise the value is considered to be host name and DNS lookup using the name is performed to get destination IP address.

If `prefix` is omitted, no file conversion is performed.

If `scheme` is omitted, FTP protocol is assumed.

If `host` is omitted, no actual file transfer is performed, only conversion(s), if any, are performed.

File Push IP Port

Name: `/par/log/push/dest/D/port D=[a…f]`
Access: `rw`
Type: `integer`
Values: `[1…65535]`
Default: `26`

This parameter specifies destination IP port for file push.

File Push User

Name: `/par/log/push/dest/D/user D=[a…f]`
Access: `rw`
Type: `string[0…32]`
Values: `(any string)`
Default: `"anonymous"`

User name to login to the destination host. When set to empty string, no attempt to send user name is performed.

File Push Password

Name: `/par/log/push/dest/D/passwd D=[a…f]`
Access: `rw`
Type: `string[0…32]`
Values: `(any string)`
Default: `(empty string)`

Password to login to the destination host. When set to empty string, no attempt to send password is performed.

Note: This parameter is never printed implicitly.

File Push Working Directory

Name: /par/log/push/dest/D/workdir D=[a…f]
Access: rw
Type: string[0…128]
Values: (any string)
Default: (empty string)

Working directory on the destination server to push files to. When set to empty string, no attempt to send working directory to server is performed.

File Push Timeout

Name: /par/log/push/dest/D/timeout D=[a…f]
Access: rw
Type: integer [seconds]
Values: [1…86400]
Default: 600

The period of inactivity of connection after which the connection is terminated and another attempt to push file is preformed.

Options for RINEX Converter

Name: /par/log/push/dest/D/rinex/opts D=[a…f]
Access: rw
Type: string[0…128]
Values: <any string>
Default: "-v3.03"

The `--mts` option, if present in the string, will tune conversion to RINEX to use MTS-specific naming conventions, and will not be passed to RINEX conversion utility.

Enable Hatanaka RINEX Compression

Name: /par/log/push/dest/D/rinex/crx D=[a…f]
Access: rw
Type: bool
Values: off,on
Default: off

Options for CGGTTS Converter

Name: /par/log/push/dest/D/cggtts/opts D=[a…f]
Access: rw
Type: string [0…128]
Values: ""

File Push Log-files Mask

Name: /par/log/push/dest/D/files D=[a…f]
Access: rw
Type: array [a…e] of boolean
Values: {y|n,…,y|n}
Default: {y,…,y}

y - closing corresponding /log/file/X will cause pushing the file to this destination

n - closing corresponding /log/file/X will be ignored by this push destination

Version of jps2rin

Name: /par/log/push/ver/jps2rin
Access: r
Type: string[0…32]
Values: "unknown"

Version of CGGTTS Converter

Name: /par/log/push/ver/cggtts
Access: r
Type: string [0…128]
Values: "unknown"

File Push State

Name: /par/log/push/state
Access: r
Type: list {stage,queued,pushed,rate,progress,file,cnt,err}

These read-only parameters aid in monitoring and troubleshooting of File Push operation.

File Push Stage

Name: /par/log/push/state/stage
Access: r
Type: enumerated
Values: off,susp,idle,pause,convert,lookup,ccmd,negotiate,cdata,send

Number of Files Queued for Push

Name: /par/log/push/state/queued
Access: r
Type: integer

Number of Files Pushed So Far

Name: /par/log/push/state/pushed
Access: r
Type: integer

Last Transfer Rate

Name: /par/log/push/state/rate
Access: r
Type: integer [bytes/second]

Transfer Progress

Name: /par/log/push/state/progress
Access: r
Type: list {size,sent,rate}

Transfer Progress Current File Size

Name: /par/log/push/state/progress/size
Access: r
Type: integer [bytes]

Transfer Progress Sent So Far

Name: /par/log/push/state/progress/sent
Access: r
Type: integer [bytes]

Transfer Progress Rate

Name: /par/log/push/state/progress/rate
Access: r
Type: integer [bytes/second]

Last Pushed File Name

Name: /par/log/push/state/file/last
Access: r
Type: string [0…64]

Current File Name Being Pushed

Name: /par/log/push/state/file/cur
Access: r
Type: string [0…64]

Number of Files Dropped From Push

Name: /par/log/push/state/cnt/drop
Access: r
Type: integer

Number of Retry Attempts To Push

Name: /par/log/push/state/cnt/retry
Access: r
Type: integer

Number of Rescheduled Files Due To Errors

Name: /par/log/push/state/cnt/resched
Access: r
Type: integer

Number of Files Dropped During Reschedule

Name: /par/log/push/state/cnt/redrop
Access: r
Type: integer

Last System Error Code Number

Name: /par/log/push/state/err/num
Access: r
Type: integer

Last System Error Code Description

Name: /par/log/push/state/err/str
Access: r
Type: string

VPN Parameters

To configure OpenVPN, copy OpenVPN client configuration file to receiver using "scp" utility, then turn /par/net/vpn/a/mode to on, and restart receiver, e.g., using

⇒ set,/par/net/reboot,on

command.

On Windows copying of the file could be achieved, say, using:

```
> pscp -P 22 -pw <PASSWORD> openvpn.conf openvpn@<HOSTNAME>:
```

and on Linux:

```
$ scp opnevpn.conf openvpn@<HOSTNAME>:
```

Where <HOSTNAME> is host name or IP address of receiver, and <PASSWORD> is password that by default is set to the same value as default for /par/net/passwd parameter, and could be changed by login to receiver over ssh using user name user and the same password, then changing password for user openvpn, by:

```
$ sudo passwd openvpn
```

VPN Mode

Name: /par/net/vpn/a/mode
Access: rw
Type: boolean
Values: off,on
Default: off

Enable VPN connection

VPN Config Available

Name: /par/net/vpn/a/config
Access: r
Type: boolean
Values: n,y
Default: n

n - no configuration file for access to VPN server

y - configuration file is found

VPN Current State

Name: /par/net/vpn/a/state
Access: r
Type: enumerated
Values: down,up
Default: down

down - VPN connection is down

up - VPN connection is up and running

VPN Enable Log File

Name: /par/net/vpn/a/log
Access: rw
Type: boolean
Values: n,y
Default: n

n - VPN will store no log-file

y - VPN will store log-file `/log/.openvpn.log`

NetFilter (IP Filter) Parameters

Network filter allows to filter incoming IP traffic based on source IP address, protocol of IP packet, as well as on destination port of TCP and UDP network packets.

The filter consists of a set of rules, every of which may either accept or drop the packet when it matches the rule.

NetFilter Policy

Name: `/par/net/filt/policy`
Access: `rw`
Type: `enumerated`
Values: `off,accept,drop`
Default: `off`

`off` - disable NetFilter

`accept` - accept packets that are not matched by any of the rules

`drop` - drop packets that are not matched by any of the rules

NetFilter Rule #N

Name: `/par/net/filt/rule/N (N=[0…4])`
Access: `rw`
Type: `list {policy,add,arange,port,span,proto}`
Default: `{off,,,-1,0,any}`

Entire filter #N. See below for description of individual elements.

This parameter is useful to set multiple elements at once, like this:

⇒ `%%set,/par/net/filt/rule/2,{allow,192.168.2.4,/8,23,0,tcp}`

NetFilter Rule #N Policy

Name: `/par/net/filt/rule/N/policy (N=[0…4])`
Access: `rw`
Type: `enumerated`
Values: `off,accept,drop`
Default: `off`

`off` - the rule is inactive and won't match any packets

`accept` - accept packets that are matched by this rule

drop - drop packets that are matched by this rule

NetFilter Rule #N Source IP Address

Name: /par/net/filt/rule/N/addr (N=[0…4])
Access: rw
Type: string
Values: (any valid IP address or <empty>)
Default: <empty>

<empty> - the rule will not check for source IP address of incoming packets, i.e., any IP will potentially match the rule

<IP> - the rule will match according to the rules below, depending on the value of /par/net/filt/rule/N/arange.

Case value of /par/net/filt/rule/N/arange:

empty - the rule will match if source IP address of incoming packet is equal to specific IP, e.g., 192.168.2.8.

IP address - the rule will match if source IP address lays within [addr-arange] range, e.g., [192.168.2.8-192.168.3.97].

subnet bits - the rule will match if source IP address belongs to specific subnet, e.g., 192.168.2.8/24.

NetFilter Rule #N Source IP Address Range

Name: /par/net/filt/rule/N/arange (N=[0…4])
Access: rw
Type: string
Values: (any valid IP address, or <empty>, or /<bits>)
Default: <empty>

<empty> - match single IP address specified by /par/net/filt/rule/N/addr, e.g., 192.168.2.8

<IP> - match range from /par/net/filt/rule/N/addr to /par/net/filt/rule/N/arange, inclusive, e.g., [192.168.2.8-192.168.3.97]

/<bits> - match subnet ADDR/<bits>, e.g., 192.168.2.8/24

NetFilter Rule #N Destination Port

Name: /par/net/filt/rule/N/port (N=[0…4])
Access: rw
Type: integer
Values: [-1…65535]
Default: -1

-1 - the rule will not check TCP/UDP destination port of the packet

`[0…65535]` – match only packets which TCP/UDP destination port number in the range `[port…port+span]`, where span is specified by `/par/net/filt/rule/N/span` parameter

NetFilter Rule #N Destination Port Span

Name: `/par/net/filt/rule/N/span (N=[0…4])`
Access: `rw`
Type: `integer`
Values: `[0…65535]`
Default: `0`

If `/par/net/filt/rule/N/port` is `-1`, this parameter is unused, otherwise it defines the span of the interval for TCP/UDP destination port check.

NetFilter Rule #N Protocol

Name: `/par/net/filt/rule/N/proto (N=[0…4])`
Access: `rw`
Type: `enumerated`
Values: `any,tcp,udp,all`
Default: `any`

`any` – match any protocol

`tcp` – match TCP packets only

`udp` – match UDP packets only

`all` – match both TCP and UDP packets

Note: if particular packet destination port is specified by `/par/net/filt/rule/N/port`, the '`any`' and '`all`' values will behave exactly the same, selecting both UDP and TCP packets, otherwise '`any`' will match any type of packet at all, whereas '`all`' will match TCP or UDP only.

Timing Parameters

CGGTTS Parameters

CGGTTS stands for “CGGTTS-Version 2E: an extended standard for GNSS Time Transfer”.

CGGTTS support in JAVAD GNSS receivers is achieved by combining multiple receiver features to get suitable JPS files and then converting them to CGGTTS format by either built-in or external JAVAD-to-CGGTTS conversion utility.

Example: To create suitable CGGTTS logging configuration, first setup antenna parameters, clock steering and improved timing modes:

```
⇒ set,ref/ant/id,"JAV_GRANT-G3T   NONE"
⇒ set,ref/pos//geo,{W84,N55d47m55.267261s,E37d31m14.352703s,380}
⇒ set,ref/arp//geo,{W84,N55d47m55.267261s,E37d31m14.352703s,380}
⇒ set,raw/time/sync,steady
⇐ set,pos/clk/fixpos,y
```

Turn off AFRM and File Push

```
⇒ set,log/push/mode,off
⇒ set,log/rot/mode,off
```

Setup parameters for CGGTTS logging using receiver file B (leaving file A for other purposes). Essentials are to use special CGGTTS time scale, 30-seconds epochs, log-file rotation being multiple of 16-minutes being shifted a bit to start every log file a little bit earlier than corresponding CGGTTS time interval:

```
⇒ set,log/rot/sc/period,60  # Will use factors
⇒ set,log/rot/sc/scales/1,cggtts
⇒ set,log/rot/sc/factors/1,16 # 16 minutes
⇒ set,log/rot/sc/phases/1,-120 # 2 minutes
⇒ set,log/b/sc/period,30
```

We need [TD](cgg) message, so add it to the default set of messages

```
⇒ em,/msg/def,/msg/jps/cgg:{0,0,1,0x1c}
⇒ set,log/push/dest/a/host,"cggtts:" #".tar.gz:"
⇒ set,log/push/dest/a/files,0x2 # File B only
⇒ set,timing/cggtts/hdrs/1,"COMMENTS=FOR TEST,REF=XXX"
```

Finish configuration and turn AFRM and FilePush ON

```
⇒ set,log/b/name/pre,cgg
⇒ set,out/elm/cur/file/b,0
⇒ set,log/imp,0x2 # File B only
⇒ set,log/rot/rmold,on
⇒ set,log/push/mode,on
⇒ set,log/rot/mode,on
```

CGGTTS Headers

Name: /par/timing/cggtts/hdrs
Access: rw
Type: array [0…4] of string [0…128]
Default: {"FY=2005,REF=UTC(USNO),CH=816(UNIVERSAL)","",…}

Each element should be comma-separated list of fields in the form NAME=VALUE:

```
NAME=VALUE,…,NAME=VALUE
```

Every element will be output in a separate [TD](cgg) message and then corresponding headers will be output to CGGTTS files by JPS-to-CGGTTS conversion utility.

Two fields have predefined meaning for the naming of CGGTTS files:

LL=xx - is the two alphabetical character code for the laboratory

RR=yy - are two receiver identification characters selected by laboratory

In addition, the FY field has special meaning as well:

FY=yyyy - First Year of Operation

The rest of fields (if any) will be treated by JPS-to-CGGTTS conversion utility in accordance with CGGTTS format specification for the headers. Unknown fields will be ignored, though a warning to the user may be produced.

Example of specifying headers and resulting [TD](cgg) messages:

Example: If we keep /par/timing/cggtts/hdrs/0 at default value, set 1 and 2 like this:

```
⇒ set,/par/timing/cggtts/hdrs/1,"LL=TL,RR=09,LAB=Test CGGTTS Laboratory"
⇒ set,/par/timing/cggtts/hdrs/2,"COMMENTS=Example Comment"
```

and then let receiver output the [TD](cgg) message:

```
⇒ out,/cur/term,/msg/jps/cgg
```

we will get the following output:

```
⇐ TD036,CGGTTS,{FY=2005,REF=UTC(USNO),CH=816 (UNIVERSAL)},@7C
⇐ TD034,CGGTTS,{LL=TL,RR=09,LAB=Test CGGTTS Laboratory},@E6
⇐ TD026,CGGTTS,{COMMENTS=Example Comment},@80
```

Notification Parameters

Mail Notification Parameters

For now, notification e-mails are sent only for file push failures.

Mail SMTP Host

Name: /par/notify/mail/smtp/host
Access: rw
Type: string[0…128]
Values: ""

Mail SMTP User

Name: /par/notify/mail/smtp/user
Access: rw
Type: string[0…32]
Values: ""

Mail SMTP Port

Name: /par/notify/mail/smtp/port
Access: rw
Type: integer [0…65535]
Values: 587

Mail SMTP Password

Name: /par/notify/mail/smtp/passwd
Access: rw
Type: string[0…32]
Values: ""

Mail To Header

Name: /par/notify/mail/to
Access: rw
Type: string[0…128]
Values: ""

Mail From Header

Name: /par/notify/mail/from
Access: rw
Type: string[0…128]
Values: "no-reply@<host-name>"

Internal Disk Parameters

Blocks Count

Name: `/dev/blk/a&blocks`
Access: `r`
Type: `integer`

Number of blocks on the internal block device. The internal block device is used by the receiver file-system to store receiver files.

Block Size

Name: `/dev/blk/a&block_size`
Access: `r`
Type: `integer [bytes]`

The size of a single block on the internal block device.

File-system Parameters

Available Memory

Name: `/par/dev/blk/a/size`
Access: `r`
Type: `integer [bytes]`

Free Memory

Name: `/par/dev/blk/a/free`
Access: `r`
Type: `integer [bytes]`

Block Size

Name: `/par/dev/blk/a/block_size`
Access: `r`
Type: `integer [bytes]`

Maximum Number of Files

Name: `/par/dev/blk/a/max_files`
Access: `r`
Type: `integer`

The file-system will refuse to create a new file if the current number of files on the file-system is greater or equal to the value of this parameter.

Number of Files

Name: /par/dev/blk/a/files
Access: r
Type: integer

Current number of files on the file-system.

Number of Bad Blocks

Name: /par/dev/blk/a/bad_blocks
Access: r
Type: integer [bytes]

Verification of Writing

Name: /par/dev/blk/a/verify
Access: rw
Type: enumerated
Values: off,fast,slow
Default: fast

File-system Version

Name: /par/dev/blk/a/ver
Access: r
Type: integer

The version of file-system the block device was formatted by.

Format Time

Name: /par/dev/blk/a/tfmt
Access: r
Type: integer [seconds]

The time-tag of the last initialization (format) of the file system. The time is measured in seconds since 00:00:00 Jan 1, 1986.

Amount of Data

Name: /par/dev/blk/a/data
Access: r
Type: integer

Number of bytes of data stored on the file-system. This number is usually less than number of used blocks multiplied by block size as every file may waste up to one block of data due to the file-system requirement that every file occupies integer number of blocks.

The value of this parameter is what the _MEM option is limiting since firmware version 2.5p2, instead of the old behavior, when _MEM limited the number of used blocks multiplied by block size.

Show Removed Files

Name: `/par/dev/blk/a/removed`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`off` - receiver file system behaves as usual

`on` - receiver file system allows to view only removed files, and is in read-only mode.

This parameter allows the user to recover inadvertently deleted files.

The receiver's file system will be remounted every time this parameter is set to either `on` or `off`. The alternative way to force the file system to remount is clearing the NVRAM, but this may be unacceptable in many cases.

Note: The command `init,/par/` won't set this parameter to the default `off` value.

Note: This parameter is not stored in the receiver's NVRAM, so it will always be set to `off` after the receiver is powered on.

Memory to Use for Data Storage

Name: `/par/dev/blk/a/mem`
Access: `rw`
Type: `enumerated`
Values: `int,ext`
Default: `int`

`int` - use internal memory for data storage.

`ext` - use external (e.g., SDCARD) memory for data storage.

This parameter will take effect after next receiver reboot. Current memory being used for data storage is available through `/par/dev/blk/a/curmem` parameter.

Memory Currently in Use for Data Storage

Name: `/par/dev/blk/a/curmem`
Access: `r`
Type: `enumerated`
Values: `int,ext`
Default: `int`

To change the value of this parameter, set `/par/dev/blk/a/mem` parameter and reboot receiver.

File-system Initialization Progress

Name: `/par/stat/fsinit`
Access: `r`
Type: `list {total,processed}`

These two fields allow monitoring of the file system initialization or remount progress. When initialization or remount is not in progress, these two values are the same.

`total` - the total number of blocks used for file storage.

`processed` - the number of blocks that are already processed.

File System Buffer Allocation

Name: `/par/stat/fsb`
Access: `r`
Type: `list {sz,max,cnt}`

This parameter is intended for the JAVAD GNSS firmware developers and is subject to change at any time.

`sz` - the size of the file system buffer.

`max` - maximum number of bytes in the file system buffer since the last receiver start-up.

`cnt` - current number of bytes in the file system buffer.

External Disk

Some of receivers support attachment of external disk to the system and automatic downloading of files from internal disk to the external one.

For automatic downloading to start, the attached disk should have its first partition formatted as FAT32, and on this partition it should have a folder with the name that matches the receiver ID. After attachment of such a disk, receiver will automatically download files that are not yet there in this folder from the receiver internal file storage. The files which name, time, and size match those on the internal file storage downloading won't happen.

The parameters below provide both status of external disk and information on the progress of the file downloading procedure.

External Disk Update Mode

Name: /par/edisk/mode
Access: rw
Type: enumerated
Values: once,regular
Default: once

once - copy files from internal to external disk once after external disk is inserted.

regular - copy files to external disk regularly as they appear on the internal disk

File Downloading Progress

Name: /par/edisk/progress
Access: r
Type: list {total, downloaded}

total - total number of kilobytes to be downloaded in this session.

downloaded - number of kilobytes already downloaded in this session.

These two fields allow to monitor the file downloading progress. When an external disk is not inserted both fields are zero. After downloading process finished downloaded field is equal to the total field.

File Downloading State

Name: /par/edisk/stat
Access: rw
Type: enumerated
Values: error,nocard,wait,downloading,finished

error - error occurred. Refer to /par/edisk/err for details.

nocard - no external disk inserted.

wait - waiting for receiver file-system to become ready.

downloading - download in progress.

finished - downloading has been successfully finished.

File Downloading Error

Name: /par/edisk/err
Access: r
Type: enumerated
Values: none,ediskfs,rcvfs,reading,writing,nospace

none - no error.

ediskfs - error occurred while mounting the external disk or opening a file on the external disk for writing.

`rcvfs` - error occurred while opening a file for reading on receiver file system.

`reading` - file reading error occurred.

`writing` - file writing error occurred.

`nospace` - there is not enough space on external disk to download all the data.

4.4.25 Session programming

Overview

Session programming means specifying one or more *jobs* for *session scheduler*. Basically, a job is a set of receiver commands executed at the specified time(s). The session scheduler can handle a fixed number of jobs. Each job has a unique integer identifier, counting from zero. Each job specification comprises:

`spec` - time specification

`cmds` - index of the command set to be executed

`count` - counter

`port` - output port

`offset` - offset

`period` - period

Besides the above mentioned four fields, every job has an *activity flag*. Jobs whose activity flags are not set, will be ignored by the session scheduler. A job with the activity flag set is called *active job*. An active job will be executed by the session scheduler as soon as the current time decremented by `offset` matches the value of the `spec` field. It will be also executed if `period` field is non-zero and the adjusted time modulo `period` is zero. As an exception, if `spec` field has only wild-cards and `period` is non-zero, the job will be run only according to the `period`, not every second as `spec` suggests.

If two or more jobs are programmed to be executed at the same time, the jobs will be executed in the order specified in the scheduler, e.g., if the jobs with identifiers `0` and `1` are to be started at the same time, job `0` will be executed before job `1`.

When it's time to execute a job, the scheduler takes index from the `cmds` field of the job specification and executes command(s) found at this index in the array of command strings. Using index to identify command string allows multiple jobs to share the same command string. The string found is executed the same way if it were received through input port specified in the `port` field of the job. As a consequence of this rule, should command generate some output, the output is sent to the `port`. Current input mode of the `port` doesn't matter though, – the command is executed as if the port is in the command mode anyway.

Every active job also serves as a wake-up point for the receiver. Therefore if session scheduler is active and receiver is in sleep mode, the next job to be executed will first wake-up receiver and only then corresponding commands will be executed. This feature also makes it useful to have empty command string as a job specification. Such specification will just wake-up receiver without execution of any commands.

When session scheduler is being turned on or is being restarted, the scheduler makes a copy of *activity flag* and `count` field of every job. Let's call these copies `stat/active` and `stat/count`, respectively. Every time the job is executed and `stat/count` is non-zero, the `stat/count` is decremented. Should `stat/count` become zero as a result of decrement, the `stat/active` flag is turned off. The job is thus deactivated and stays inactive until scheduler mode is changed or scheduler is restarted. This allows to limit number of executions of a job by setting its `count` field to a non-zero value. The `stat/mode` and `stat/count` fields of every job could be read from the receiver for reference but couldn't be directly changed by user.

Time specification `spec` is a template containing four fields, namely day of week `day`, hour of day `hour`, minute of hour `minute`, and second of minute `second`. Each field can either be set to an integer value in corresponding range, or to a special value that serves as a wild-card that matches any value. In order for a job to be executed at given time `T`, job's `spec` should match given time `T`. Scheduler compares current time against `spec` by first decomposing current time into the set of corresponding fields, then comparing every field of decomposed time against corresponding field of `spec`. Time matches `spec` if and only if every field of time matches corresponding field of `spec`. Fields of `spec` having special value match any value of corresponding field of time. For example, specification `2d17h__m30s`, where "`__`" (double underscore) is special value, matches `2:17:00:30`, `2:17:01:30`,..., `2:17:59:30`. It means that job having such `spec` will be executed at the middle of every minute during one hour starting on Tuesday, 17:00:00. Due to the fact that week number couldn't be specified, the job will in fact be executed every Tuesday at 17:00:30, 17:01:30,…, and 17:59:30 (60 times). To limit executions of such job to single (next) Monday, job counter should be set to 60 before activation of the session.

In fact session scheduler doesn't check `spec` of every active job against current time every second. Instead it determines job that should be executed next (along with its next execution time) whenever scheduler mode is changed, scheduler is restarted, or job is executed. The identifier of the next job to be run and corresponding execution time along with current time could be read from the receiver.

Parameters

Session Mode

Name: /par/sess/mode
Access: rw
Type: enumerated
Values: on,off,susp
Default: off

This parameter specifies the current session scheduler mode.

off - session scheduler is disabled

on - session scheduler is active

susp - session scheduler is active and works almost as usual, but does not actually execute commands.

Restart Session Scheduler

Name: /par/sess/restart
Access: rw
Type: boolean
Values: on,off
Default: off

on - restarts session scheduler. This is similar to setting /par/sess/mode to off then back to on. The value of the parameter is returned back to off after scheduler restart.

off - ignored.

Job Activity Flags

Name: /par/sess/active
Access: rw
Type: array [0…15] of boolean
Values: {y|n,…,y|n}
Default: {n,…,n}

The value of this parameter is copied to the /par/sess/stat/active one whenever session scheduler mode is changed or scheduler is restarted.

Job #N Activity Flag

Name: /par/sess/active/N (N=[0…15])
Access: rw
Type: boolean
Values: y,n
Default: n

y - job is enabled.

n - job is disabled.

Job Specifications

Name: /par/sess/job
Access: rw
Type: array [0…15] of job specification

Job #N Specification

Name: /par/sess/job/N (N=[0…15])
Access: rw
Type: list {spec,cmds,count,port,offset,period}
Default: {__d__h__m__s,0,0,"",0,0}

Job #N Time Specification

Name: /par/sess/job/N/spec (N=[0…15])
Access: rw
Type: timespec
Default: __d__h__m__s

Job is executed whenever current time matches its time specification. "__" in time specification matches any value.

The day field of a timespec is in the range [0…6], where 0 is Sunday, and 6 - Friday.

Job #N Index of Command String

Name: /par/sess/job/N/cmds (N=[0…15])
Access: rw
Type: integer
Values: [0…7]
Default: 0

This parameter designates index into the array /par/sess/cmds, which corresponding element in turn contains the command(s) to be executed.

Job #N Execution Counter

Name: /par/sess/job/N/count (N=[0…15])
Access: rw
Type: integer
Values: [0…2147483647]
Default: 0

If non-zero, designates number of times the job should be executed by the scheduler before deactivation. If zero, execution is not limited.

The value is copied to the corresponding /par/sess/stat/count/N parameter every time session scheduler mode is changed or scheduler is restarted.

Job #N Current Terminal

Name: /par/sess/job/N/port (N=[0…15])
Access: rw
Type: enumerated
Values: /[oport],(empty string)
Default: (empty string)

The commands assigned to the job are executed as if they were received from the port specified by this parameter. Should the commands generate receiver replies, they will be sent to corresponding output port.

Job #N Reference Time Offset

Name: /par/sess/job/N/offset
Access: rw
Type: integer [seconds], or 'utc'
Values: [-864000…86400],utc
Default: 0

[-864000…86400] – offset in seconds to the job reference time

utc – use current UTC offset as the offset to the job reference time (see /par/sess/stat/utc for current UTC offset)

Job #N Execution Period

Name: /par/sess/job/N/period
Access: rw
Type: integer [seconds]
Values: [0…86400]
Default: 0

0 – do not execute this job periodically, i.e., only consider 'spec'.

`[1…86400]` – execute the job with specified period, in addition to what `'spec'` says. If `'spec'` has no fields set, it's ignored and only `'period'` is considered.

Command Strings

Name: `/par/sess/cmds`
Access: `rw`
Type: `array [0…15] of strings`
Default: `{(empty string),…,(empty string)}`

Command String #N

Name: `/par/sess/cmds/N, where N= 0…15`
Access: `rw`
Type: `string [0…64]`
Default: `(empty string) (empty string)`

Session Scheduler Status

Name: `/par/sess/stat`
Access: `r`
Type: `list {active,count,job,time}`

Running Activity Flags

Name: `/par/sess/stat/active`
Access: `r`
Type: `array [0…15] of boolean`
Values: `{y|n,…,y|n}`

This parameter is set to the value of `/par/sess/active` parameter whenever session scheduler mode is changed or session scheduler is restarted.

Running Activity Flag for Job #N

Name: `/par/sess/stat/active/N (N=[0…15])`
Access: `r`
Type: `boolean`
Values: `y,n`

Job Execution Down-counters

Name: `/par/sess/stat/count`
Access: `r`
Type: `array [0…15] of integer`

Job #N Execution Down-counter

Name: /par/sess/stat/count/N (N=[0…15])
Access: r
Type: integer
Values: [0…2147483647]

Whenever session scheduler mode is changed or scheduler is restarted, the value of corresponding /par/sess/job/N/count parameter is copied to this variable. If the variable is greater than zero, it is decremented every time corresponding job is executed. Should the variable became zero as a result of decrement, corresponding current activity flag /par/sess/stat/active/N is set to "n", thus the job is deactivated.

Next Job to be Executed

Name: /par/sess/stat/job
Access: r
Type: integer
Values: [-1…15]
Default: -1

-1 - there are no jobs to be executed

[0…15] - job index to be executed next

Timing Information

Name: /par/sess/stat/time
Access: r
Type: list {next,curr,delta}

Time of Execution of Next Job

Name: /par/sess/stat/time/next
Access: r
Type: timespec

If there is no next job, the value will be __d__h__m__s.

Current Time

Name: /par/sess/stat/time/curr
Access: r
Type: timespec

Current time in timespec format that is running no matter what scheduler mode is.

Time Left to Next Job Execution

Name: /par/sess/stat/time/delta
Access: r
Type: timespec

If there is no next job, the value will be __d__h__m__s.

Session UTC Offset

Name: /par/sess/stat/utc
Access: r
Type: integer [seconds]

Current UTC to receiver time offset, to be used when /par/sess/job/N/offset is set to 'utc'.

Examples

Example: Suppose we need to program receiver for a single session that will begin next Wednesday, 12:30:00 and end on Thursday, 10:00:00. During this time receiver should write file "ses.log" containing the default set of messages into its internal memory at 1Hz and simultaneously output NMEA GGA message to its serial port B at 10Hz. Except this time period, receiver should be turned off. In fact there are multiple ways to achieve this, here is one of them. Let's define two jobs for the session:

```
job 0:
  spec  = 4d12h30m00s
  cmd   = "create,ses.log;em,/cur/log,def:1;em,/dev/ser/b,nmea/GGA:0.1"
  count = 1

job 1:
  spec  = 5d10h00m00s
  cmd   = "dm,/dev/ser/b;dm,/cur/log;set,power,off"
  count = 1
```

After programming these two jobs and turning session scheduler on, we can put receiver into sleep mode (using MINTER or set,sleep,on command). Receiver will wake-up at the time specified by the job a and execute corresponding commands. When the time for the job b comes, receiver will execute corresponding commands turning receiver off as a result. Note that we turn power off in the job b as opposed to putting receiver into sleep mode. This way we don't need to set counters for jobs because once receiver is turned off, it can't wake-up anymore. This however, has a side effect that both jobs will remain active and thus may trigger corresponding actions next time we turn receiver power on. If we don't want that, we will need to set count field for both jobs to 1 while programming.

Note also that if we activate such session after Wednesday, 12:30:00 but before Thursday, 10:00:00, the receiver will not do what we meant. The first job that will be run in this case is job `b`, not job `a`, so job `a` won't be executed at all.

Here are actual commands to program the above jobs (recall that '#' is comment character):

```
⇒ # Turn off scheduler and deactivate all jobs
  set,/par/sess/mode,off
  set,/par/sess/active,n
  # Define 'spec', 'cmds', 'count', and 'port' fields for jobs 0 and 1
  set,/par/sess/job/0,{4d12h30m0s,0,1,/dev/null}
  set,/par/sess/job/1,{5d10h0m0s,1,1,/dev/null}
  # Define commands 0 and 1
  set,/par/sess/cmds/0,"create,ses.log;em,/cur/log,def:1;em,/dev/ser/b,
    nmea/GGA:0.1"
  set,/par/sess/cmds/1,"dm,/dev/ser/b;dm,/cur/log;set,power,off"
  # Activate jobs 0 and 1
  set,/par/sess/active,{y,y}
  # Turn scheduler on
  set,/par/sess/mode,on
```

Example: Suppose we need to setup a permanent station that will work from 8:00:00 to 20:00:00 every day from Monday to Friday and will be turned off during weekend (Saturday and Sunday). During work-time the station should transmit differential correction through one of its serial ports and write measurement files to its internal memory. The files should be automatically rotated every hour and oldest file should be deleted when there is not enough memory to write latest data.

To achieve this we first program receiver to be base station to send corrections and configure AFRM mode to handle logging stuff (not shown in this example).

Then program the following 4 jobs:

```
job 0: spec = _d08h00m00s cmd = ""
job 1: spec = _d20h00m00s cmd = "set,sleep,on"
job 2: spec = 7d08h00m00s cmd = "set,sleep,on"
job 3: spec = 0d08h00m00s cmd = "set,sleep,on"
```

Jobs `a` and `b` will turn receiver on and off at specified times every day. But that is not exactly what we want as receiver then will work on Saturday and Sunday as well. To prevent this, we configure jobs `c` and `d` that will put receiver into sleep mode immediately after wake-up on Saturday and Sunday, respectively. There will be short periods of

time when receiver is turned on Saturday and Sunday, but who cares? Note that we don't need jobs similar to `c` and `d` to be run at 20:00:00 on Saturday and Sunday as job `b` will put receiver into sleep mode anyway.

Here are the commands (base station configuration and AFRM programming not shown):

```
⇒ # Turn off scheduler and deactivate all jobs
  set,/par/sess/mode,off
  set,/par/sess/active,n
  # Define 'spec', 'cmds', 'count', and 'port' fields for jobs 0…3
  set,/par/sess/job/0,{  8h0m0s,0,0,/dev/null}
  set,/par/sess/job/1,{ 20h0m0s,1,0,/dev/null}
  set,/par/sess/job/2,{7d8h0m0s,1,0,/dev/null}
  set,/par/sess/job/3,{0d8h0m0s,1,0,/dev/null}
  # Define commands 0 and 1
  set,/par/sess/cmds/0,""
  set,/par/sess/cmds/1,"set,sleep,on"
  # Activate jobs
  set,/par/sess/active,{y,y,y,y}
  # Turn scheduler on
  set,/par/sess/mode,on
```

Example: Suppose we need to output the size of current log-file to receiver port A every 10 seconds. There is no predefined message that contains this information, but there is corresponding parameter that we can print, so we program single job to achieve the required result:

```
job a:
  spec =  d h m s
  cmd  = "%job_a : size=%print,/cur/file/a&size"
  port = /dev/ser/a
  period = 10
```

When this job is active, receiver will output RE message to the port A every 10 seconds in the form:

```
⇐ REXXX%job_a : size=% SIZE
```

where `SIZE` is current file size. We've put "`%job_a : size=%`" into the command to be able to identify the reply got from the job.

Here are actual commands:

```
⇒ # Turn off scheduler and deactivate all jobs
  set,/par/sess/mode,off
  set,/par/sess/active,n
```

```
# Define 'spec', 'cmds', 'count', and 'port' fields for job 0
set,/par/sess/job/0,{"",0,0,/dev/ser/a,0,10}
# Define command 0
set,/par/sess/cmds/0,"%job_a : size=%print,/cur/file/a&size"
# Activate job 0
set,/par/sess/active/0,y
# Turn scheduler on
set,/par/sess/mode,on
```

Note: This example demonstrates one use of the `port` field of job specification. Another one would be monitoring of execution of session commands. By setting `port` to one of receiver ports it is possible, e.g., to see if any errors occur as a result of job execution.

4.4.26 Notebook

Notebook allows the user to store arbitrary information into the receiver and retrieve it later if necessary. In addition, as this information is output in the [PM] message along with other parameters, it could be used to pass arbitrary text data from controller application to a post-processing one[1].

Notes

Name: `/par/note`
Access: `r`
Type: `list {str,nv}`

Note Strings

Name: `/par/note/str`
Access: `r`
Type: `array[0…7] of strings`

Note String #N

Name: `/par/note/str/N (N=[0…7])`
Access: `rw`
Type: `string [0…64]`

When setting a string for index N, the corresponding flag `/par/note/nv/N` is cleared (set to `n`) to indicate that this string is set by the user, not read from the NVRAM.

1. See also "event" on page 50 for another way to communicate to post-processing applications.

Change Indicators

Name: /par/note/nv
Access: r
Type: array [0…7] of boolean

When receiver starts up, it reads all the note strings from its NVRAM and sets all the values in this array to y. When user sets some of strings, corresponding NVRAM flag is cleared (set to n).

4.4.27 Generic Communication Parameters

The parameters described in this section are common for all the supported ports.

Current Terminal

Current Terminal

Name: /cur/term
Access: r
Type: string
Values: /[port]

The name of the current terminal, i.e., the name of the input stream the command requesting the value was received through.

Basic Operation Modes

Input Mode

Name: /par/[port]/imode
Access: rw
Values: cmd,echo,jps,rtcm,rtcm3,cmr,omni,sisnet,gbas,auto, none,dtp,term
Default: cmd

With this parameter the user specifies what type of incoming data to accept on the selected receiver port.

- cmd – command mode. Being in this mode, the receiver's port recognizes GREIS commands sent by the user.
- echo – echo mode. This mode is the same as cmd mode unless /par/[port]/echo is set to a value different from /dev/null, see /par/[port]/echo parameter description below for details.

`jps` – GREIS input mode. In this mode receiver is capable to recognize both standard and non-standard GREIS messages.

`rtcm` – RTCM 2.x input mode.

`rtcm3` – RTCM 3.x input mode.

`cmr` – CMR/CMR+ input mode. For more information on CMR format, please refer to ftp://ftp.trimble.com/pub/survey/cmr.

`omni` – unsupported.

`sisnet` – SISNeT input mode.

`gbas` – GBAS/GRAS input mode. For more information see RTCA document DO-246D.

`auto` – automatically handle any supported format of differential data. All the formats handled by `jps`, `rtcm`, `rtcm3`, `cmr`, `sisnet`, or `gbas` modes will be recognized. This mode leads to increased CPU load compared to any of specific modes, so please prefer to use particular mode once you know it.

`none` – means that the port will ignore any incoming data.

`dtp` – the port is currently attached to the Data Transfer Protocol (DTP), so all the input goes there. This mode could be set only by the `get` GREIS command. The mode will return to `cmd` as soon as DTP terminates.

`term` – the PPP data link is currently established over this port, so all the input goes there. This mode could be set only implicitly by the PPP stack. When parameter is implicitly set to this mode, attempts to change the mode will fail.

Output Mode

Name: `/par/[port]/omode`
Access: `r`
Values: `std,dtp,term`
Default: `std`

`std` – standard output mode.

`dtp` – GREIS Data Transfer Protocol (DTP) is active on the port. This mode could be set only by the `get` GREIS command. The mode will return to `std` as soon as DTP terminates. In this mode all the usual messages output to the given port is temporarily suppressed.

`term` – the PPP data link is currently established over this port. The mode will return to `std` as soon as PPP terminates. In this mode all the usual messages output to the given port is temporarily suppressed.

Output Duplication

Name: /par/[port]/dup
Access: rw
Type: enumerated
Values: /[oport],/dev/null
Default: /dev/null

This parameter specifies an output port to which all the data being output to the [port] should be sent (duplicated).

/[oport] - the outgoing data will be sent (duplicated) to specified output port. The outgoing data that will be duplicated include regular message output, the data echoed (see "Echo Parameters" below) from another port, and the data duplicated from another port; if any.

/dev/null - no data duplication will be performed.

Besides troubleshooting, this feature is useful if you need to request the output of exactly the same messages into multiple ports. Instead of programming each port output individually, the messages could be enabled to one of the ports, and then duplicated to another one, and then duplicated from those one to yet another one, etc. Not only it's simpler to program, but it will also reduce processor load as CPU won't spend time to re-generate the same messages multiple times.

Echo Parameters

Echo Port

Name: /par/[port]/echo
Access: rw
Type: enumerated
Values: /[oport],/dev/null
Default: /dev/null

This parameter specifies an output port to which all the incoming data being received from the [port] should be sent.

/[oport] - the incoming data will be sent (echoed) to specified output port. Echoing the data doesn't prevent interpretation of them according to currently selected input mode, unless the input mode is set to echo, in which case the data will be otherwise ignored.

/dev/null - the incoming data won't be sent to any of output ports.

Given the above description, to achieve just echoing of the data from an input port to an output one (*pure echo*), one needs to set the input mode to echo, and the echo port to the name of the required output port. If the input port that is to be turned into the pure echo

mode is the current port, the sequence of commands shown in the examples below is recommended.

Note: To program a feature sometimes referred to as "daisy-chain", i.e., bidirectional virtual channel between two ports, it's required to set pure echo mode on both ports participating in the daisy chain.

Echo-off Sequence

Name: `/par/[port]/eoff`
Access: `rw`
Type: `string [4…32]`
Values: `(any string)`
Default: `#OFF#`

The sequence of characters that will reset `/par/[port]/echo` to its default `/dev/null` value once it is received through the `[port]`.

Note that receiving of the echo-off sequence doesn't change the input mode of the current port. Instead, to conveniently support turning from pure echo to command mode, the `echo` input mode behaves exactly like `cmd` input mode when `/par/[port]/echo` parameter is set to `/dev/null`.

Note: The default value for this parameter is intentionally chosen so that it will be considered to be just a comment by the GREIS language parser. It makes it safe to send the default echo-off sequence even when corresponding port is currently in command mode.

Warning: *The echo-off sequence will be echoed to the current echo port before the current echo port is turned to /dev/null. If you setup a daisy-chain between your controller and some device connected to receiver port, make sure to first program echo-off sequence to a value that will do no harm to the device. On the other hand, don't change the echo-off sequence from its default value without necessity, as other applications trying to establish communication with the receive will likely to fail provided your leave the port in the pure echo mode.*

Enable Wrapped Echo

Name: `/par/[port]/ewrap`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – when this parameter is `on` and `/par/[port]/echo` is other than `/dev/null`, data echoing is carried out in the "wrapped" mode, i.e. all of the characters received from `[port]` are combined into a corresponding [>>] message (see "[>>] Wrapper" on page 132) before being output to the output port.

`off` – no wrapping of the data will occur.

For the purposes of wrapping, the data are stored in the internal receiver buffer until either timeout in the receiving of data occurs, or the amount of data in the buffer exceeds the wrapping threshold. The timeout is currently not customizable and set to 100 milliseconds, and the threshold could be changed by the `/par/[port]/wsize` parameter.

Wrapping Threshold

Name: `/par/[port]/wsize`
Access: `rw`
Type: `integer`
Values: `[1…128], bytes`
Default: `128`

This parameter specifies the threshold value for wrapping input data when in the wrapped mode.

Examples

Example: Setup pure echo from the current port to the serial port B. The commands below will do the job even if current port is already in the echo mode to the same or some other port, provided echo off sequence was not changed from its default value. The commands also make sure none of them are echoed to the destination port:

```
⇒ #OFF#
⇒ set,/cur/term/imode,cmd
⇒ set,/cur/term/echo,/dev/null
⇒ set,/cur/term/imode,echo
⇒ set,/cur/term/echo,/dev/ser/b
```

Example: Setup *daisy chain* between current port and serial port B. The commands below will do the job even if current port is already in the echo mode to the same or some other port, provided echo off sequence was not changed from its default value:

```
⇒ #OFF#
⇒ set,/cur/term/imode,cmd
⇒ set,/cur/term/echo,/dev/null
⇒ set,/par/dev/ser/b/imode,echo
⇒ set,/par/dev/ser/b/echo,/cur/term
⇒ set,/cur/term/echo,/dev/ser/b
⇒ set,/cur/term/imode,echo
```

Advanced Input Mode

Overview

JAVAD GNSS receivers support advanced input mode. This mode allows to use single input port (being set to `jps` input mode) to feed receiver with data in multiple formats, as well as dispatch different data to different decoders. In this mode the receiver will decode the [>>] messages and non-standard messages and will pass the data decoded from these messages to a specified decoder (e.g., commands interpreter or RTCM decoder), or will send the decoded data to a specified output port. To provide backward compatibility with the earlier firmware versions, advanced input mode is turned off by default. In this case the [>>] messages and non-standard messages are ignored when the input mode is set to `jps`.

Note: Remember that the primary purpose of the `jps` input mode is to receive and decode those messages in GREIS format that carry information suitable for phase-differential mode of position computation.

There is a set of parameters through which advanced input mode can be controlled. This set is represented by an array of three[1] elements. Each element of the array is a *specification* that consists of the following fields: `mode`, `id`, `skip`, and `port`. When either [>>] or non-standard message is received, its contents is checked against every specification in turn according to the rules described below. The first specification that matches will govern the execution of the message contents. If no matching specification is found, the contents of the message is ignored. In the *matching stage*, the receiver uses the `mode` and `id` fields. In the *execution stage*, the receiver uses `mode`, `skip`, and `port` fields.

Matching stage

On this stage, the receiver compares the `mode` and `id` fields with the contents of the received message.

Depending on the value of the `mode` field, the specification is allowed to match a message as follows:

`none` – the specification never matches any message.

`cmd, echo, jps, rtcm, rtcm3, cmr` – the specification could match the [>>] message, but never matches a non-standard message.

`nscmd, nsecho` – the specification could match a non-standard message but never matches the [>>] message.

Once the value in the `mode` field allows to match given message, the receiver will compare the value in the `id` field of the specification with the *identifier* of the message. For

1. The number of elements is somewhat arbitrary and can be expanded in the future (if required).

[>>] message, its identifier is the value of its `id` field. For the non-standard messages, the first byte of the message is taken as its identifier. The message matches the specification in two cases:

1. The value of the message identifier exactly matches the value of the `id` field of specification.
2. The value of the message identifier doesn't matter if the `id` field of the specification has a special value `-1`.

Execution stage

Once the message passes the matching stage, i.e., there is a specification that matches the message, the contents of the message is executed according to the first matching specification as follows:

If the `skip` field is set to `y`, the first byte of the message is skipped before executing the message contents. Otherwise the first byte is considered as the part of the message contents. Note that for [>>] message the contents never includes its generic header, i.e., the contents is its body.

After optional skipping of the first byte, the message contents is executed according to the value of the `mode` field in the matched specification as follows:

`jps, rtcm, rtcm3, cmr` - the receiver will pass the contents to the corresponding type of decoder as if the message has been received through the port specified in the `port` field of the specification. The input mode of the corresponding port (`/par/[port]/imode`) should be set accordingly for execution to actually take place. The empty value of the `port` field denotes the current port (i.e., the port the initial message is received through). Bear in mind that the empty value of the `port` field for `jps` mode is meaningless (as well as explicit setting the port parameter to the port matching those the parameter is being set for) as the contents would pass to the same decoder the initial message came from. This, in turn, would break the decoder logic, so the receiver will protect itself by just throwing away the contents of the message.

`cmd, nscmd` - the receiver will pass the contents to the command interpreter as if the command has been received through the port `port`. The empty value of `port` denotes the current port.

`echo, nsecho` - the receiver will either send the contents to the output port specified in the `port` field of the matched specification.

Parameters Description

Input Specifications

Name: /par/[port]/jps
Access: rw
Type: array [0…2] of input specification

Input Specification

Name: /par/[port]/jps/N (N=[0…2])
Access: rw
Type: list {mode, id, skip, port}

Matching/Execution Mode

Name: /par/[port]/jps/N/mode (N=[0…2])
Access: rw
Type: enumerated
Values: none,cmd,echo,jps,rtcm,rtcm3,cmr,nscmd,nsecho
Default: none

In the *matching stage*, the values have the following meaning:

`none` - the specification will never match any message.

`cmd, echo, jps, rtcm, rtcm3, cmr` - the specification may possibly match the [>>] message. The matching criterion is the product of comparison between the `id` field of the [>>] message and the `id` field of the specification.

`nscmd, nsecho` - the specification may possibly match a non-standard message. The matching criterion is the product of comparison between the first byte of the non-standard message and the `id` field of the specification.

In the *execution stage*, the values have the following meaning:

`none` - never appears in the execution stage as no message could match this mode.

`cmd` - execute the message contents as command(s) received from the port specified in the `port` field of the input specification.

`echo` - send the message contents to the port specified in the `port` field of the input specification.

`jps` - process the message contents as data in GREIS format received from the port specified in the `port` field of the input specification.

`rtcm` - process the message contents as data in RTCM 2.x format received from the port specified in the `port` field of the input specification.

`rtcm3` - process the message contents as data in RTCM 3.0 format received from the port specified in the `port` field of the input specification.

`cmr` - process the message contents as data in CMR format received from the port specified in the `port` field of the input specification.

`nscmd` - the same as `cmd`.

`nsecho` - the same as `echo`.

Message Identifier Matcher

Name: `/par/[port]/jps/N/id (N=[0…2])`
Access: `rw`
Type: `integer`
Values: `[-1…255]`
Default: `-1`

`-1` - matches any message.

`[0…255]` - matches a message which first byte is equal to the value of the field. For the [>>] message, its first byte is its `id` field. For non-standard message, the first byte is the message is its first character (lying between "!" and "/" in ASCII).

Skip the First Byte

Name: `/par/[port]/jps/N/skip (N=[0…2])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

`y` - the first byte of the message is skipped before executing the message contents.

`n` - the first byte of the message is not skipped.

Source/Destination Port

Name: `/par/[port]/jps/N/port (N=[0…2])`
Access: `rw`
Type: `enumerated`
Values: `/[oport],(empty string)`
Default: `(empty string)`

If the value of the `mode` field is set to either `echo` or `nsecho` (see the parameter `/par/[port]/jps/N/mode` on page 437), this parameter specifies the output port to which the message contents should be sent.

For all other values of the `mode` field, this parameter specifies the input port so that the message contents is executed as if the data have been received through the specified port.

The default value, empty string, designates the current port, i.e., the port through which the initial message has been received.

Examples

Example: Suppose we have a controller connected to the serial A port of the receiver and we need to send both commands and CMR differential messages to the receiver using this port. Suppose also that the controller is capable to wrap CMR data into the [>>] messages and uses the `id` field of the [>>] message as part of CMR data; and we wish to pass these CMR data to the receiver as if they were received from serial C port. Suppose also that the controller is capable to send GREIS commands to the receiver prefixing them by the '`!`' character. The receiver could be then configured using the following commands:

```
⇒ set,/par/dev/ser/a/jps/0,{nscmd,33,y,""}
⇒ set,/par/dev/ser/a/jps/1,{cmr,-1,n,/dev/ser/c}
⇒ set,/par/dev/ser/a/jps/2/mode,none
⇒ set,/par/dev/ser/a/imode,jps
⇒ set,/par/dev/ser/c/imode,cmr
```

Note: In the first command, 33 is decimal value of the ASCII code of the '`!`' character that our example controller will be prefixing to GREIS commands after the port is configured as shown.

Note: Note that in the last command the port (`/dev/ser/c` in our example) should match those one specified in the second command as a virtual source of CMR data.

Example: Suppose that unlike previous example, the controller always sends GREIS commands prefixed by command identifier (containing any string surrounded by '`%`' characters). Then we would instead wish to configure the 0-th specification as follows:

```
⇒ set,/par/dev/ser/a/jps/0,{nscmd,37,n,""}
```

where 37 is ASCII code of the '%' character, and we've changed the value of `skip` from `y` to `n`.

Example: Suppose that unlike previous examples, the controller sets `id` field of generated [>>] messages to decimal value 43, and puts CMR data only into the `data` field of the message. Then we will program the 1-th specification as follows:

```
⇒ set,/par/dev/ser/a/jps/1,{cmr,43,y,/dev/ser/c}
```

●

4.4.28 Serial Port Parameters

In this section, `[sport]` denotes a serial (RS232) port, – any of `/dev/ser/X (X=[a…d])`.

Hardware Settings

Baud Rate

Name: /par/[sport]/rate
Access: rw
Type: integer
Values: 460800,230400,153600,115200,57600,38400,19200,9600,4800, 2400,1200,600,300
Default: &def

The attribute /par/[sport]/rate&def specifies the default value for this parameter.

Name: /par/[sport]/rate&def
Access: rw
Type: enumerated
Values: 460800,230400,153600,115200,57600,38400,19200,9600,4800, 2400,1200,600,300
Default: 115200

RTS/CTS Handshake

Name: /par/[sport]/rtscts
Access: rw
Type: boolean
Values: on,off
Default: off

RTS State

Name: /par/[sport]/rts
Access: rw
Type: boolean
Values: on,off
Default: on

CTS State

Name: /par/[sport]/cts
Access: r
Type: boolean
Values: on,off

Number of Data Bits

Name: /par/[sport]/bits
Access: rw
Values: 5,6,7,8
Default: 8

Number of Stop Bits

Name: /par/[sport]/stops
Access: rw
Type: integer
Values: 1,2
Default: 1

Parity

Name: /par/[sport]/parity
Access: rw
Type: enumerated
Values: N,odd,even,fodd,feven
Default: N

N - no parity

odd - odd parity

even - even parity

fodd - forced odd parity (logical 1)

feven - forced even parity (logical 0)

Infrared Mode

Name: /par/[sport]/ir
Access: rw
Type: boolean
Values: on,off
Default: off

Note that the JAVAD GNSS receiver may have either one infrared port, which is always port D, or no infrared port.

Serial Port Type

Name: /par/[sport]/type
Access: r
Type: enumerated
Values: rs232,rs422

This parameter contains current type of corresponding serial port. Port type is either set through PRTT option or is set in hardware, depending on board type.

Output Time-frames

Overview

This feature allows to use the receiver in a time-sharing network where every receiver in a network is only allowed to send its data during specific time intervals.

Output time-frame is a periodic time interval of a specified length. Time-frames are entirely specified by three parameters: `period`, `length`, and `delay`. By definition, a time-frame begins the `delay` seconds after the receiver time modulo `period` becomes zero, and lasts for the `length` seconds.

While no time-frame is active, the data to be output to the port is buffered inside the receiver. As soon as next time-frame begins, receiver starts to output data to the port and keeps output allowed till the end of the time-frame. At the end of the time-frame receiver clears its internal buffers not to allow the data that reminded buffered to be output at the subsequent time frame.

Time-frame Mode

Name: `/par/[sport]/oframe/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` - corresponding port will output data only at the specified time-frames.

`off` - data output to corresponding port will be enabled all the time.

Time-frame Period

Name: `/par/[sport]/oframe/period`
Access: `rw`
Type: `float [seconds]`
Values: `[0…86400]`
Default: `1`

This parameter specifies the frequency at which output time-frames will occur. Time-frames will be scheduled to be started at the moments when the receiver time modulo `period` is equal to zero. Note however that a time-frame is actually started the delay seconds later.

Time-frame Length

Name: /par/[sport]/oframe/length
Access: rw
Type: float [seconds]
Values: [0…86400]
Default: 1

This parameter specifies how long output will be enabled once a time-frame has been started.

Time-frame Delay

Name: /par/[sport]/oframe/delay
Access: rw
Type: float [seconds]
Values: [0…86400]
Default: 0

This parameter specifies how much time passes between the moment the output time-frame is scheduled, and when the time-frame actually begins.

4.4.29 Network Parameters

Note: Changes to some of these parameters may take effect only after receiver reboot.

Host Name

Custom Network Host Name

Name: /par/net/host/name
Access: rw
Type: string[0…31]
Values: <hostname>
Default: ""

Changing this parameter will take effect after receiver reboot. If empty (the default) host name is generated automatically at startup, otherwise host name is set at startup to specified value.

Current active host name could be read from /par/net/host/curname parameter.

Current Network Host Name

Name: /par/net/host/curname
Access: r
Type: string[0…31]
Values: <hostname>
Default: <receiver dependent>

Current active host name. Could be changed by setting /par/net/host/name parameter followed by receiver reboot.

DNS Parameters

Default DNS Server

Name: /par/net/dns
Access: rw
Type: string
Values: (any valid IP address)
Default: 0.0.0.0

The address of the DNS server to use by default. The 0.0.0.0 value means do not perform DNS lookups. The default DNS server will be used whenever receiver doesn't determine DNS server to use automatically.

Current DNS Server

Name: /par/net/curdns
Access: r
Type: string
Values: (any valid IP address)
Default: 0.0.0.0

The address of the current DNS server. The value of this parameter could differ from /par/net/dns value when DNS server to use is determined automatically. The 0.0.0.0 value means do not perform DNS lookups.

DNS Service Discovery (DNS-SD)

DNS-SD Mode

Name: /par/net/dnssd/responder/mode
Access: rw
Type: boolean
Values: on,off
Default: on

`on` - DNS-SD responder is active.

`off` - DNS-SD responder is turned off.

Dynamic DNS (DynDNS) Client Parameters

Dynamic DNS client provides capability to associate static DNS name with (dynamic) IP address (automatically) assigned to the receiver. This is achieved by automatically registering assigned IP address on a server that provides dynamic DNS services.

Note: the only dynamic DNS server currently supported is *http://www.dyndns.org*. To detect current receiver IP address, the *checkip.dyndns.org:80* server will be used.

Note: For DynDNS client to work properly, the DNS service should be available and should be able to resolve the names of dynamic DNS hosts.

Note: changes of these parameters take effect after receiver reboot.

DynDNS Mode

Name: `/par/net/ddns/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`off` - the DynDNS client is turned off.

`on` - the DynDNS client is turned on. Receiver will try to register IP address at the dynamic DNS service whenever IP address changes.

DynDNS State

Name: `/par/net/ddns/state`
Access: `r`
Type: `enumerated`
Values: `off,active,abort`
Default: `off`

`off` - DynDNS client is turned off.

`active` - DynDNS client is active.

`abort` - DynDNS client is inactive due to unrecoverable error. Receiver reboot is required to pull the client from this state.

DynDNS User

Name: /par/net/ddns/user
Access: rw
Type: string[0…32]
Values: (any string)
Default: "user"

The user name of the account on the dynamic DNS server.

DynDNS Password

Name: /par/net/ddns/passwd
Access: rw
Type: string[0…32]
Values: (any string)
Default: "passwd"

The password of the account on the dynamic DNS server.

Note: This parameter is never printed implicitly.

DynDNS Receiver DNS Name (Alias)

Name: /par/net/ddns/alias
Access: rw
Type: string[0…32]
Values: (any string)
Default: "user.dyndns.org"

The host name to be assigned to the receiver. When registered, receiver will be accessible through Internet using this particular name.

DHCP Client Configuration

DHCP client, when turned on, is capable to automatically assign LAN parameters to the receiver.

DHCP Client Mode

Name: /par/net/dhcp/client/mode
Access: rw
Type: boolean
Values: on,off
Default: off

on – DHCP client is turned on. LAN/WLAN configuration parameters will be selected automatically on receiver start-up. User-provided parameters will only be used if DHCP can't figure them.

`off` – DHCP client is turned off. Static LAN/WLAN configuration parameters will be used.

LAN Configuration

These parameters aid in configuration of your receiver to be part of a TCP/IP local area network (LAN). Only static configuration defined by the parameters is currently supported (i.e., there is no support for DHCP). Note that all these parameters are sticky and are not reset to their default values when receiver NVRAM is cleared or parameters are initialized by the `init` command.

Receiver IP Address

Name: `/par/net/ip/addr`
Access: `rw`
Type: `ip_address`
Values: `(any valid IP address)`
Default: `192.168.2.2`

This parameter identifies the receiver on a TCP/IP network.

Network mask

Name: `/par/net/ip/mask`
Access: `rw`
Type: `ip_address`
Values: `(any valid IP address)`
Default: `255.255.255.192`

This parameter specifies the network mask of the local network the receiver is connected to.

Default Gateway

Name: `/par/net/ip/gw`
Access: `rw`
Type: `string`
Values: `(any valid IP address)`
Default: `192.168.2.1`

The default gateway to use for packets that don't belong to the local network.

Maximum Transmission Unit (MTU)

Name: `/par/net/ip/mtu`
Access: `rw`
Type: `integer`
Values: `[128…16384]`
Default: `1500`

The MTU for the interface.

MAC Address

Name: `/par/net/mac/addr`
Access: `rw`
Type: `string`
Values: `(any valid MAC address)`
Default: `(automatically generated unique value)`

This parameter specifies receiver's unique hardware number on a network. High part of the MAC address is fixed and is equal to `00:18:D7`. Low part is generated automatically from the receiver ID. The user usually does not need to change the MAC address.

IPv6 Support on Ethernet

Name: `/par/net/ipv6/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` - enable IPv6 on Ethernet interface

`off` - disable IPv6 on Ethernet interface

WLAN (WiFi) Configuration

These parameters aid in configuration of your receiver to be part of a TCP/IP wireless local area network (WLAN). Note that all these parameters are sticky and are not reset to their default values when receiver NVRAM is cleared or parameters are initialized by the `init` command.

WLAN Mode

Name: `/par/net/wlan/mode`
Access: `rw`
Type: `boolean`
Values: `on,off,adhoc,hs`
Default: `on`

`off` - WLAN is turned off

`on` - WLAN client mode

`adhoc` - WLAN Ad-Hoc access point mode

`hs` - WLAN Hotspot access point mode

WLAN Connection State

Name: `/par/net/wlan/state`
Access: `r`
Type: `enumerated`
Values: `off,on,connecting,disconnecting,associated,error`
Default: `on`

WLAN Error

Name: `/par/net/wlan/error`
Access: `r`
Type: `string[0…128]`
Default: `"none"`

Human-readable error string describing the last error happened.

WLAN Mode

Name: `/par/net/wlan/ap/mode`
Access: `rw`
Type: `string[3]`
Values: `wep,wpa`
Default: `wep`

`wep` - receiver will either use WEP encryption or non-encrypted AP.

`wpa` - receiver will use WPA-PSK/WPA2-PSK encryption. For this mode to work, parameters `/par/net/wlan/ap/passphrase` and `/par/net/wlan/ap/ssid` should be set appropriately.

WLAN WPA Pass-phrase

Name: `/par/net/wlan/ap/passphrase`
Access: `rw`
Type: `string[8…63]`
Values: `(any string)`
Default: `(empty string)`

WLAN Access Point ID

Name: /par/net/wlan/ap/id
Access: rw
Type: string
Values: (any valid MAC address)
Default: "00:00:00:00:00:00"

WLAN Access Point SSID

Name: /par/net/wlan/ap/ssid
Access: rw
Type: string[0…32]
Values: (any string)
Default: (empty string)

This parameter will be used to identify the access point to use when /par/net/wlan/ap/id parameter is set to all zeros (the default value).

WLAN Access Point RSSI

Name: /par/net/wlan/ap/rssi
Access: r
Type: integer
Values: [0…255]
Default: 0

Access point Received Signal Strength Indication (RSSI). Indicates the amount of power present in a received radio signal.

WLAN Access Point Keys

Name: /par/net/wlan/ap/keyN (N=[1…4])
Access: w
Type: string [0…32]
Values: (any string)
Default: (empty string)

The N'th key string for the access point.

To increase security, these parameters are write-only and can't be read back.

WLAN Access Point Current SSID

Name: /par/net/wlan/ap/ssidcur
Access: r
Type: string [0…64]

WLAN Access Point Current Frequency

Name: /par/net/wlan/ap/freq
Access: r
Type: string [0…8], [MHz]

WLAN Connection List

Name: /par/net/wlan/ap/con/list
Access: r
Type: list of strings
Default: {}

List of available WLAN SSIDs.

Example: {"MyWlan","NET-4"}

Note: This parameter obsoletes /par/net/wlan/ap/list/con

WLAN Clear Connection List

Name: /par/net/wlan/ap/con/clear
Access: rw
Type: boolean
Values: y,n
Default: n

y - WLAN connection list will be cleared and the value of this parameter will be set back to 'n'

n - ignored

Note: This parameter obsoletes /par/net/wlan/ap/list/con/clear

WLAN Receiver IP Address

Name: /par/net/wlan/ip/addr
Access: rw
Type: ip_address
Values: (any valid IP address)
Default: 192.168.2.2

This parameter identifies the receiver on a TCP/IP network. This address is not used when DHCP client is turned on. In the latter case the address will be assigned by DHCP client automatically.

WLAN Network mask

Name: /par/net/wlan/ip/mask
Access: rw
Type: ip_address
Values: (any valid IP address)
Default: 255.255.255.192

This parameter specifies the network mask of the wireless network the receiver is connected to.

WLAN Default Gateway

Name: /par/net/wlan/ip/gw
Access: rw
Type: string
Values: (any valid IP address)
Default: 192.168.2.1

The default gateway to use for packets that don't belong to the wireless network.

WLAN Maximum Transmission Unit (MTU)

Name: /par/net/wlan/ip/mtu
Access: rw
Type: integer
Values: [128…16384]
Default: 1500

The MTU for the interface.

WLAN MAC Address

Name: /par/net/wlan/mac/addr
Access: rw
Type: string
Values: (any valid MAC address)
Default: (automatically generated unique value)

This parameter specifies receiver's unique hardware number on a network. High part of the MAC address is fixed and is equal to 00:18:D7. Low part is generated automatically from the receiver ID. The user usually does not need to change the MAC address.

GPRS/DIALUP (PPP) Configuration

These parameters aid in establishing of either GPRS or dial-up connection to a provider of Internet services using point-to-point protocol (PPP).

To create PPP link through internal or external modem connected to receiver serial port, the user should set the mode of corresponding modem (`/par/modem/X/mode, X=[a…d]`) to either `gprs` (for GSM modem) or `dial-up` (for GSM or analog modem), depending on the kind of required connection (GPRS or dial-up). Receiver scans modem mode parameters for each modem from `a` to `d` in order and selects the first one with the `mode` equal to `gprs` or `dial-up`. It then creates PPP link over corresponding modem port.

Should the `mode` parameter of the modem port on which PPP link is active be set to a value that differs from the current setting, the PPP connection for this port will be terminated and the firmware will repeat search for modem mode equal to `gprs` or `dial-up` among modems (starting from `/par/smodem/a`).

Examples

Example: Establish GPRS link through GSM modem connected to receiver serial port C. The example assumes BEELINE cellular operator and PIN code `1234`.

```
⇒ set,/par/net/ppp/gprs/pdp/apn,internet.beeline.ru
⇒ set,/par/net/ppp/gprs/pdp/id,1
⇒ set,/par/net/ppp/gprs/passwd,beeline
⇒ set,/par/net/ppp/gprs/user,beeline
⇒ set,/par/net/ppp/gprs/dial,"×99×××1#"
⇒ set,/par/modem/c/type,gsm
⇒ set,/par/modem/c/pin,1234
⇒ set,/par/modem/c/mode,gprs
```

Example: Establish dial-up PPP link over PSTN modem connected to receiver serial port C.

```
⇒ set,/par/net/ppp/dial-up/dial,96007000
⇒ set,/par/net/ppp/dial-up/user,mtd0633877@dlp
⇒ set,/par/net/ppp/dial-up/passwd,abc
⇒ set,/par/modem/c/type,pstn
⇒ set,/par/modem/c/mode,dialup
```

PPP Configuration Parameters

PPP Connection State

Name: `/par/net/ppp/state`
Access: `r`
Type: `enumerated`
Values: `off,connecting,connected,disconnecting`
Default: `off`

`off` - PPP connection is inactive.

`connecting` - receiver tries to establish PPP connection.

`connected` - PPP connection is up and running.

`disconnecting` - receiver is disconnecting from PPP peer.

PPP Error

Name: `/par/net/ppp/error`
Access: `r`
Type: `string[0…256]`
Values: `(any string)`
Default: `"none"`

This parameter contains PPP error message(s), or `"none"` if there are no errors.

PPP Baud Rate

Name: `/par/net/ppp/speed`
Access: `r`
Type: `enumerated`
Values: `9600,19200,38400,57600,115200`

This parameter contains baud rate at which receiver talks to the peer of PPP connection or to the modem.

PPP Set Default Route

Name: `/par/net/ppp/route`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` - receiver will add default route to the system routing table using the PPP peer as the default gateway. Note that if receiver is simultaneously connected to the LAN, this will switch routing from the LAN default gateway to the PPP peer.

`off` - receiver won't add the PPP peer as the default route to the system routing table, unless `/par/net/tcp/addr` is set to `0.0.0.0`.

PPP Debugging

Name: `/par/net/ppp/debug`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – PPP connection debugging facilities are enabled. The debugging information will be output to the receiver serial Port A.

`off` – PPP connection debugging facilities are disabled.

Enable PAP Authentication

Name: `/par/net/ppp/auth/pap`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – enable use of unencrypted password authentication protocol (PAP).

`off` – receiver will refuse to authenticate itself to the peer using PAP.

Enable CHAP Authentication

Name: `/par/net/ppp/auth/chap`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – enable use of challenge handshake authentication protocol (CHAP).

`off` – receiver will refuse to authenticate itself to the peer using CHAP.

Enable Van Jacobson Compression

Name: `/par/net/ppp/comp/vj`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – Van Jacobson style TCP/IP header compression (VJ) is enabled in both the transmit and receive directions.

`off` – the VJ compression is disabled.

Enable Connection-ID Compression

Name: `/par/net/ppp/comp/vjc`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – receiver will omit the connection-ID byte from VJ-compressed headers and will ask the peer to do so.

`off` – receiver will not omit the connection-ID byte, nor ask the peer to do so.

GPRS Configuration

GPRS Dial Number

Name: `/par/net/ppp/gprs/dial`
Access: `rw`
Type: `string[0…32]`
Default: `"×99×××1#"`

This parameter specifies the dial number for GPRS connection.

GPRS User Name

Name: `/par/net/ppp/gprs/user`
Access: `rw`
Type: `string[0…32]`
Default: `(empty string)`

This parameter specifies GPRS user name.

GPRS Password

Name: `/par/net/ppp/gprs/passwd`
Access: `rw`
Type: `string[0…32]`
Default: `(empty string)`

This parameter specifies GPRS password.

Note: This parameter is never printed implicitly.

GPRS PDP Context Identifier

Name: `/par/net/ppp/gprs/pdp/id`
Access: `rw`
Type: `integer`
Values: `[1…4]`
Default: `1`

This parameter specifies Packet Data Protocol (PDP) context identifier.

GPRS PDP Access Point Name

Name: /par/net/ppp/gprs/pdp/apn
Access: rw
Type: string[0…32]
Values: (any string)
Default: (empty string)

This parameter specifies Packet Data Protocol (PDP) access point name.

GPRS QoS Requested Precedence Class

Name: /par/net/ppp/gprs/at/cgqreq/prcd
Access: rw
Type: integer
Values: [0…3]
Default: 0

This parameter specifies the precedence class for Quality of Service Profile Requested (for AT+CGQREQ command).

GPRS QoS Requested Delay Class

Name: /par/net/ppp/gprs/at/cgqreq/delay
Access: rw
Type: integer
Values: [0…4]
Default: 0

This parameter specifies the delay class for Quality of Service Profile Requested (for AT+CGQREQ command).

GPRS QoS Requested Reliability Class

Name: /par/net/ppp/gprs/at/cgqreq/relb
Access: rw
Type: integer
Values: [0…5]
Default: 0

This parameter specifies the reliability class for Quality of Service Profile Requested (for AT+CGQREQ command).

GPRS QoS Requested Peak Throughput Class

Name: /par/net/ppp/gprs/at/cgqreq/peak
Access: rw
Type: integer
Values: [0…9]
Default: 0

This parameter specifies the peak throughput class for Quality of Service Profile Requested (for AT+CGQREQ command).

GPRS QoS Requested Mean Throughput Class

Name: /par/net/ppp/gprs/at/cgqreq/mean
Access: rw
Type: integer
Values: [0…31]
Default: 0

This parameter specifies the mean throughput class for Quality of Service Profile Requested (for AT+CGQREQ command).

GPRS QoS Minimum Precedence Class

Name: /par/net/ppp/gprs/at/cgqmin/prcd
Access: rw
Type: integer
Values: [0…3]
Default: 0

This parameter specifies the precedence class for Quality of Service Profile Minimum acceptable (for AT+CGQMIN command).

GPRS QoS Minimum Delay Class

Name: /par/net/ppp/gprs/at/cgqmin/delay
Access: rw
Type: integer
Values: [0…4]
Default: 0

This parameter specifies the delay class for Quality of Service Profile Minimum acceptable (for AT+CGQMIN command).

GPRS QoS Minimum Reliability Class

Name: /par/net/ppp/gprs/at/cgqmin/relb
Access: rw
Type: integer
Values: [0…5]
Default: 0

This parameter specifies the reliability class for Quality of Service Profile Minimum acceptable (for AT+CGQMIN command).

GPRS QoS Minimum Peak Throughput Class

Name: /par/net/ppp/gprs/at/cgqmin/peak
Access: rw
Type: integer
Values: [0…9]
Default: 0

This parameter specifies the peak throughput class for Quality of Service Profile Minimum acceptable (for AT+CGQMIN command).

GPRS QoS Minimum Mean Throughput Class

Name: /par/net/ppp/gprs/at/cgqmin/mean
Access: rw
Type: integer
Values: [0…31]
Default: 0

This parameter specifies the mean throughput class for Quality of Service Profile Minimum acceptable (for AT+CGQMIN command).

DIALUP Configuration

DIALUP Dial Number

Name: /par/net/ppp/dial-up/dial
Access: rw
Type: string[0…32]
Values: (any string)
Default: (empty string)

This parameter specifies dial number for dial-up Internet provider.

DIALUP User Name

Name: /par/net/ppp/dial-up/user
Access: rw
Type: string[0…32]
Values: (any string)
Default: (empty string)

This parameter specifies user (login) name for dial-up Internet provider.

DIALUP Password

Name: /par/net/ppp/dial-up/passwd
Access: rw
Type: string[0…32]
Values: (any string)
Default: (empty string)

This parameter specifies password for dial-up Internet provider.

Note: This parameter is never printed implicitly.

DIALUP Modem Initialization Script

Name: /par/net/ppp/dial-up/init
Access: rw
Type: string[0…64]
Values: (any string)
Default: "@AT@OK@ATI@OK@ATT@OK@"

This parameter contains chat script to initialize dial-up modem. The chat script defines a conversational exchange between the receiver and the modem. The syntax and semantics of the chat script used by the receiver matches those of the widely used “chat” program (see, e.g., *http://docs.freebsd.org/info/uucp/uucp.info.Chat_Scripts.html* for documentation), except the '@' character is used instead of cartridge return to separate chat commands.

Network Servers Parameters

Receiver implements FTP (in read-only mode), TCP , HTTP, TCP output, UDP output, NTRIP Caster, PTP, and NTP servers.

FTP server is suitable do download files from receiver using standard FTP client(s).

TCP server allows to establish bidirectional TCP connections to receiver and use the connections as regular data input-output streams. This is suitable both for controlling and for monitoring of the receiver.

HTTP server allows to establish bidirectional connections to receiver on top of HTTP protocol and to use the connections as data input-output streams. This is suitable both for controlling and for monitoring of the receiver through external program running in a WWW browser.

TCP and UDP output server is suitable for feeding of multiple rovers with data streams of different kinds (RTCM, CMR, etc.).

NTRIP Caster is an implementation of corresponding standards, both v1 and v2.

PTP and NTP servers are two different kinds of time transfer and synchronization services.

TCP/FTP Password

Name: `/par/net/passwd`
Access: `rw`
Type: `string[0…15]`
Values: `(any string)`
Default: `(automatically generated value unique for each receiver)`

By using this parameter the user sets a password for FTP and TCP connections.

Note: For security, this parameter is never printed implicitly.

Note: There are two values that are treated specially to bypass login/password authentication for TCP server connections: `_INSECURE`, and `xINSECURE`. See below for details.

TCP Server Configuration

TCP server provides ability for client to establish raw TCP connection to receiver and use GREIS interface over resulting communication channel. The maximum number of simultaneous connections supported is equal to the supported number of TCP streams (`dev/tcp/a`,…,`dev/tcp/e`). The procedure to establish TCP connection is as follows.

After client connects to the port specified by the `/par/net/tcp/port` parameter, receiver first outputs a few lines of information. Every line of information is started by the '#' character (hex value `0x23`) and ended by `<CR><LF>` sequence (hex values `0x0D 0x0A`). This information contains receiver and board names, firmware version, and receiver electronic ID. Here is an example of the information lines:

```
# TRE_G3TH receiver ready
# /par/rcv={id=3VXX6K1QC6N7S3VLWTZ3K2UW03,ver={main="3.2.0 Jun,23,2010",
    board=TRE_G3TH_5}}
```

After information lines are output, receiver outputs the `login:` prompt. Explicit `a`, `b`,…, `e` reply followed by new line to the login prompt will make connection to corresponding TCP stream (`dev/tcp/a`,…,`dev/tcp/e`, respectively). Entering just new line (i.e., empty

login string) means that receiver needs to automatically select the first available TCP stream to make connection to.

After getting reply to the login prompt, receiver will issue the `Password:` prompt. The string specified in `/par/net/passwd` followed by new line should be entered by the client in reply to the password prompt. If the string entered does not match, connection will be denied, otherwise connection to the selected TCP port will be established.

If login and/or password information is wrong, or requested stream is already in use, or all the streams are already in use in case of automatic stream selection, receiver will output the

```
Login incorrect.
```

message and repeat the login/password procedure after a few seconds of pause. The pause is implemented for security reasons. Receiver will allow up to 3 consecutive attempts to enter login and password. After third attempt fails, receiver will close connection.

Finally, after authentication succeeds, receiver outputs the following line:

```
RE019%Logged in on /dev/tcp/X%
```

where `X` is substituted by the actual TCP stream letter the connection was made to (i.e, `a`, `b`, …, `e`).

You can configure receiver to bypass the login/password authentication altogether by setting `/par/net/passwd` parameter to the special value `_INSECURE`. In this case only information lines will be output and then connection will be established to the first available TCP stream. If you set this parameter to `xINSECURE` instead, receiver will output login and password prompts but won't wait for client to enter login and password; it will establish connection to the first available TCP stream without need to enter login and password.

Example: Configure receiver to listen for TCP connections on port 7890 and accept connections without login and password:

```
⇒ set,/par/net/tcp/port,7890
⇒ set,/par/net/passwd,_INSECURE
```

TCP Port

Name:	`/par/net/tcp/port`
Access:	`rw`
Type:	`integer`
Values:	`[1…65535]`
Default:	`8002`

IP port receiver is listening on for incoming TCP connections.

TCP Connection Timeout

Name: /par/net/tcp/timeout
Access: rw
Type: integer [seconds]
Values: [1…0x7FFFFFFF]
Default: 600

The period of TCP connection inactivity after which the connection will be terminated.

TCP Server TLS/SSL Mode.

Name: /par/net/tcp/tls
Access: rw
Type: boolean
Values: on,off
Default: off

off - only unencrypted TCP connections will be allowed by TCP server.

on - only secure TLS/SSL connections will be allowed by TCP server.

HTTP Server Configuration

HTTP Port

Name: /par/net/http/port
Access: rw
Type: integer
Values: [1…65535]
Default: 80

IP port receiver is listening on for incoming HTTP connections.

HTTP Connection Timeout

Name: /par/net/http/timeout
Access: rw
Type: integer [seconds]
Values: [1…0x7FFFFFFF]
Default: 10

The period of HTTP connection inactivity after which corresponding socket will be closed.

HTTP Server TLS/SSL Mode

Name: /par/net/http/tls
Access: rw
Type: boolean
Values: on,off
Default: off

off - only unencrypted HTTP connections will be allowed by HTTP server.

on - only secure TLS/SSL connections will be allowed by HTTP server.

TCP Output Server Configuration

TCP output server listens for connections on a few consecutive IP ports starting from those specified by the /par/net/tcpo/port parameter. Once TCP client connects to one of these ports, it will receive data enabled to be output to corresponding /dev/tcpo/X port. Multiple TCP clients connected to the same IP port managed by the TCP output server (see /par/net/tcpo/port parameter) will all receive the same data stream. For example, when /par/net/tcpo/port parameter is set to 8010 and client connects to IP port 8012, client will get data being output to the /dev/tcpo/c receiver output stream.

TCP Output Base Port

Name: /par/net/tcpo/port
Access: rw
Type: integer
Values: [1…65535]
Default: 8010

Base port for TCP output streams. This port corresponds to the /dev/tcpo/a output stream. Next consecutive ports correspond to next consecutive streams.

TCP Output Connection Idle Timeout

Name: /par/net/tcpo/timeout
Access: rw
Type: integer [seconds]
Values: [1…0x7FFFFFFF]
Default: 600

The period of inactivity of TCP output connection after which the connection will be terminated by the server.

TCP Output Connections Count

Name: /par/net/stat/tcpod/count
Access: r
Type: integer
Default: 0

TCP Output Specific IP Port

Name: /par/dev/tcpo/X/port, X=[a…e]
Access: rw
Type: integer
Values: [0…65535]
Default: 0

0 - IP port for this output stream is determined by the value of /par/net/tcpo/port

[1…65535] - specific IP port for this particular TCP output stream

TCP Output Current IP Port

Name: /par/dev/tcpo/X/curport, X=[a…e]
Access: r
Type: integer
Values: [1…65535]
Default: (as defined by "/par/net/tcpo/port")

This parameter is equal to the current IP port being used for this TCP output stream.

UDP Output Server Configuration

Receiver supports data output using UDP packets. You will use receiver port name dev/udp/X (X=[a…e]) with em or out commands to request UDP output. Parameters described in this section specify destination of UDP packets generated by the receiver as well as the rules of splitting of output data stream of bytes into stream of UDP packets.

UDP Destination Address

Name: /par/dev/udp/X/addr, X=[a…e]
Access: rw
Type: string
Values: (any valid IP address)
Default: 255.255.255.255

This parameter specifies destination address for UDP packets. 255.255.255.255 is considered to be LAN broadcast address. Other kinds of broadcast and multicast addresses are also supported.

UDP Destination Port

Name: /par/dev/udp/X/port, X=[a…e]
Access: rw
Type: integer
Values: [1…65535]
Default: 8004

This parameter specifies destination port for UDP packets.

Output Size Margin for UDP Packets

Name: /par/dev/udp/X/omargin, X=[a…e]
Access: rw
Type: integer [bytes]
Values: [0…1200]

This parameter specifies output size margin for UDP packets. As soon as number of bytes to be output exceeds the value of this parameter, UDP packet containing the data will be sent. Note that receiver will never split single GREIS message between multiple UDP packets, so typical sizes of UDP packets will be greater than the value of this parameter. The end of epoch will cause sending of UDP packet provided the number of bytes remaining to be output is greater than zero, therefore the last packet in an epoch will be typically shorter than the value of this parameter.

UDP Multicast TTL

Name: /par/dev/udp/X/mcttl, X=[a…e]
Access: rw
Type: integer [bytes]
Values: [0…255]
Default: 1

This parameter specifies TTL for UDP multicast packets.

NTRIP Caster Configuration

NTRIP Caster output streams are called /dev/ntrip/X X=[a…e]. An NTRIP client that connects to mountpoint with the name matching /par/net/ntrip/X/mp, will receive the messages enabled to corresponding /dev/ntrip/X output stream.

NTRIP Caster Mode

Name: /par/net/ntrip/mode
Access: rw
Type: boolean
Values: off,on
Default: off

off - caster is turned off and is inactive

on - caster is turned on

NTRIP Caster IP Port

Name: /par/net/ntrip/port
Access: rw
Type: integer
Values: [1…65535]
Default: 2101

NTRIP Caster Connection Timeout

Name: /par/net/ntrip/timeout
Access: rw
Type: integer [seconds]
Values: [1…0x7FFFFFFF]
Default: 600

The period of inactivity of NTRIP output connection after which the connection will be terminated.

NTRIP Caster Login User Name

Name: /par/net/ntrip/user
Access: rw
Type: string [0…32]
Values: (any string)
Default: (empty string)

NTRIP Caster Password

Name: /par/net/ntrip/passwd
Access: rw
Type: string [0…32]
Values: (any string)
Default: (empty string)

NTRIP Caster Protocol Version

Name: /par/net/ntrip/ver
Access: rw
Type: integer
Values: 1,2
Default: 1

NTRIP Caster Mountpoint of Source Entry X

Name: /par/net/ntrip/str/X/mp X=[a…e]
Access: rw
Type: string [0…32]
Values: (any string)
Default: "<RCV>_<SN>_X"

where:

<RCV> - receiver model

<SN> - serial number

NTRIP Caster Source Entry X String N

Name: /par/net/ntrip/str/X/N X=[a…e] N=[0…4]
Access: rw
Type: string [0…100]
Values: (arbitrary string)
Default: ";;;0;;;;;;0;0;;;N;N;0;none" (for N=0)
<empty string> (for N>0)

Mountpoint source entry sent in the NTRIP sourcetable is concatenation of mountpoint name and then all the N strings for particular mountpoint X.

FTP Server Configuration

FTP Port

Name: /par/net/ftp/port
Access: rw
Type: integer
Values: [1…65535]
Default: 21

IP port the receiver is listening on for incoming FTP connections. Only 1 simultaneous connection is supported.

FTP Connection Timeout

Name: /par/net/ftp/timeout
Access: rw
Type: integer [seconds]
Values: [1…0x7FFFFFFF]
Default: 600

The period of FTP connection inactivity after which the connection will be terminated.

FTP Password

Name: /par/net/ftp/passwd
Access: rw
Type: string[0…15]
Values: (any string)
Default: ""

If non-empty string, use it for FTP access instead of /par/net/passwd.

NTP Server Configuration

For receivers that support Network Time Protocol (NTP), the following parameters are available:

NTP Port

Name: /par/net/ntp/port
Access: rw
Type: integer
Values: [1…65535]
Default: 123

IP port number the receiver is listening on for incoming NTP requests.

NTP Error

Name: /par/net/ntp/error
Access: r
Type: string

A string describing NTP error, or NONE when there is no error.

NTP Requests

Name: /par/net/ntp/rcvd
Access: r
Type: integer
Values: [1…2147483647]

Number of NTP requests received since startup.

NTP Replies

Name: /par/net/ntp/sent
Access: r
Type: integer
Values: [1…2147483647]

Number of NTP replies sent since startup.

PTP Server Configuration

For receivers that support Precision Time Protocol (IEEE Std 1588, versions 1 and 2), this section describes parameters that govern PTP server behavior and report its state.

Note: Receiver behaves as PTP grand-master clock only.

PTP Mode

Name: /par/net/ptp/mode
Access: rw
Type: boolean
Values: off,on
Default: off

off - PTP server is turned off.

on - PTP server is turned on.

Maximum Time RMS for PTP Output

Name: /par/net/ptp/max_rms
Access: rw
Type: float [seconds]
Values: [0…86400]
Default: 10^{-7}

Set this to value higher than 1000 for PTP to be allowed to output even before first lock to SVs (see /par/net/ptp/check_sync as well).

Check Time Synchronization for PTP Output

Name: /par/net/ptp/check_sync
Access: rw
Type: boolean
Values: on,off
Default: on

Set this to `off` for PTP to be allowed to output even before first lock to SVs (see `/par/net/ptp/max_rms` as well).

PTP Version

Name: `/par/net/ptp/ver`
Access: `rw`
Type: `integer`
Values: `1,2`
Default: `1`

The PTP version specified by this parameter will be activated on the next reboot of receiver.

PTP One Step

Name: `/par/net/ptp/one_step`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

Note:: for receivers with no PTP-aware Ethernet PHY this parameter is set to `off` by default and can't be changed.

`on` - PTP one-step mode is active. I.e., `SYNC` packet has its own time-stamp, and no `FOLLOW-UP` packets are sent.

`off` - PTP two-step mode is active. I.e., `FOLLOW_UP` packet has time stamp for previous `SYNC` packet.

PTP Domain Number

Name: `/par/net/ptp/domain`
Access: `rw`
Type: `integer`
Values: `[0…255]`
Default: `0`

This parameter specifies PTPv2 `domainNumber` clock property.

PTP Transport

Name: `/par/net/ptp/trans`
Access: `rw`
Type: `enumerated`
Values: `ipv4,ether`
Default: `ipv4`

PTP Announce Message Interval

Name: /par/net/ptp/int/ann
Access: rw
Type: integer [log2(seconds)]
Values: [-7…7]
Default: 1

Interval of output of ANNOUNCE messages. Interval in seconds is equal 2 power this parameter. E.g., -1=0.5 seconds, 2=4 seconds.

PTP Sync Message Interval

Name: /par/net/ptp/int/sync
Access: rw
Type: integer [log2(seconds)]
Values: [-7…7]
Default: 0

Interval of output of SYNC messages. Interval in seconds is equal 2 power this parameter. E.g., -1=0.5 seconds, 2=4 seconds.

PTP Delay_Req Message Interval

Name: /par/net/ptp/int/dreq
Access: rw
Type: integer [log2(seconds)]
Values: [-7…7] log2(sec)
Default: 0

Desired interval of output of Delay_Req messages that PTP Master puts into its Delay_Resp message for PTP Slaves to follow. Interval in seconds is equal 2 power this parameter. E.g., -1=0.5 seconds, 2=4 seconds.

PTP Priority1 and Priority2 Fields

Name: /par/net/ptp/pri/N (N=1,2)
Access: rw
Type: integer
Values: [0…255]
Default: 128

These parameters specify the values of priority1 and priority2 fields of the ANNOUNCE message that are used for Best Master Clock selection.

PTP Unicast Mode

Name: /par/net/ptp/unicast/mode
Access: rw
Type: boolean
Values: off,on
Default: off

off - use PTP profile multicast mode

on - use unicast mode

PTP Unicast Negotiation

Name: /par/net/ptp/unicast/negotiate
Access: rw
Type: boolean
Values: off,on
Default: off

on - enable unicast message delivery and interval negotiation using signaling messages, as used by the Telecom profile.

off - disable negotiation, and use the list of unicast destinations specified by /par/net/ptp/unicast/dests parameter.

PTP Unicast Destinations

Name: /par/net/ptp/unicast/dests
Access: rw
Type: string[0…255]
Values: <comma-separated list of destination hosts>
Default: ""

Comma-separated list of destination host names or IP addresses for unicast mode when used without negotiation.

PTP Additional Options for PTPd

Name: /par/net/ptp/opts
Access: rw
Type: string[0…255]
Values: <arbitrary string>
Default: ""

Refer to PTPd manual for description of accepted command-line options.

PTP Port State

Name: /par/net/ptp/stat/port
Access: r
Type: string
Values: <human-readable string>

PTP port state according to PTP specification.

PTP Clock Synchronization State

Name: /par/net/ptp/stat/clock
Access: r
Type: enumerated
Values: no_sync,sync

no_sync - PTP clock is not synchronized to GNSS time scales.

sync - PTP clock is synchronized to GNSS time scales.

PTP Version Currently in Use.

Name: /par/net/ptp/stat/ver
Access: r
Type: integer
Values: 1,2

PTP version currently in use. This could differ from /par/net/ptp/ver when those was changed, but receiver was not reboot yet.

PTP V1 Clock Stratum

Name: /par/net/ptp/stat/v1/stratum
Access: r
Type: integer
Values: <PTP Clock Stratum>

PTP V1 Clock Preferred

Name: /par/net/ptp/stat/v1/pref
Access: r
Type: boolean
Values: y,n

PTP V1 Clock Identifier

Name: /par/net/ptp/stat/v1/id
Access: r
Type: enumerated
Values: DFLT,GPS

PTP V2 Clock Class

Name: /par/net/ptp/stat/v2/class
Access: r
Type: integer
Values: <PTP Clock Class>

PTP V2 Clock Accuracy

Name: /par/net/ptp/stat/v2/acc
Access: r
Type: integer
Values: <PTP Clock Accuracy>

PTP V2 Time Source

Name: /par/net/ptp/stat/v2/src
Access: r
Type: integer
Values: <PTP Time Source>

TCP Client Parameters

Receiver is capable to operate as TCP client for different kinds of TCP servers. Depending on receiver model, up to 2 independent connections could be established at any given time, using separate TCP client instances, called tcpcl/a, and tcpcl/b.

The rcv mode of TCP Client is capable to operate in UDP mode as well, see /par/net/tcpcl/X/proto parameter.

Note: Originally, when firmware supported only single TCP client, all the parameters in this section were called /par/net/tcpcl/x. When multiple clients support has been implemented, the parameters for each client were called /par/net/tcpcl/X/x (X=[a,b]) instead. The original /par/net/tcpcl/x set of parameters was left as synonyms for /par/net/tcpcl/a/x new parameters, but they became obsolete and are not described in this manual anymore.

Suspend TCP Clients

Name: /par/net/tcpcl/suspend
Access: rw
Type: boolean
Values: on, off
Default: off

off - TCP clients will work according to the rest of configuration

on - TCP clients will close all connections and suspend themselves

TCP Client Mode

Name: /par/net/tcpcl/X/mode X=[a,b]
Access: rw
Type: enumerated
Values: off,rcv,ntrip,ntrips
Default: off

off - TCP client is inactive.

rcv - use TCP connection to another receiver (RCV server), or another generic TCP server possibly featuring login/password access control.

ntrip - connect to NTRIP caster as NTRIP client

ntrips - connect to NTRIP caster as NTRIP server

When using rcv or ntrip mode, corresponding TCP client port of the receiver, dev/tcpcl/X, should be configured to receive corresponding type of corrections by setting the port input mode accordingly. Refer to "Input Mode" on page 430 for instructions on setting input mode of a port. In addition, We recommend to use extrap RTK mode (refer to "RTK Position Computation Mode" on page 293) due to potentially large delays on the Internet/GPRS.

When using ntrips mode, enable required messages to corresponding dev/tcpcl/X port.

TCP Client Connection Timeout

Name: /par/net/tcpcl/X/timeout X=[a,b]
Access: rw
Type: integer [seconds]
Values: [1…0x7FFFFFFF]
Default: 10

The period of TCP client connection inactivity after which corresponding socket will be closed.

TCP Client Protocol

Name: /par/net/tcpcl/X/proto X=[a,b]
Access: rw
Type: enumerated
Values: tcp,udp,udpm,udpb
Default: tcp

tcp - use TCP to connect to server

udp - use UDP unicast to receive data from server

udpm - use UDP multicast to receive data from server

udpb - use UDP broadcast to receive data from server

This parameter is supported by RCV mode of TCP client only.

TCP Client Connection State

Name: /par/net/tcpcl/X/state X=[a,b]
Access: r
Type: enumerated
Values: off,connecting,connected,disconnecting,error

`off` - TCP client is inactive.

`connecting` - TCP client is connecting to the server.

`connected` - TCP client is connected to the server.

`disconnecting` - TCP client is disconnecting from the server.

`error` - TCP client was unable to connect to the server. In this case the parameter `/par/net/tcpcl/X/error` will contain the reason of the error.

TCP Client Error

Name: /par/net/tcpcl/X/error X=[a,b]
Access: r
Type: string[0…64]
Values: (any string)
Default: "none"

This parameter will contain human-readable description of the reason of TCP client failure (if any).

RCV Mode Parameters

The parameters below are useful to provide a method to establish TCP connection to another (remote) JAVAD GNSS receiver, request data from the remote receiver, and then use the data as RTK/DGPS corrections.

Example: Configure receiver to connect to the reference receiver's TCP port B and receive RTCM corrections from the reference receiver:

```
⇒ set,/par/net/tcpcl/a/rcv/addr,172.17.0.34
⇒ set,/par/net/tcpcl/a/rcv/port,8002
⇒ set,/par/net/tcpcl/a/rcv/login,b      # login to TCP port B
⇒ set,/par/net/tcpcl/a/rcv/passwd,abc
⇒ set,/par/dev/tcpcl/a/imode,rtcm       # expect RTCM corrections
⇒ set,/par/net/tcpcl/a/mode,rcv
```

For this example to work, the reference receiver should be configured something like this:

```
⇒ set,/par/net/ip/addr,172.17.0.34
⇒ set,/par/net/ip/mask,255.255.255.0
⇒ set,/par/net/ip/gw,172.17.0.1
⇒ set,/par/net/tcp/port,8002
⇒ set,/par/net/passwd,abc
⇒ em,/dev/tcp/b,/msg/rtcm/{18,19,22,3}
⇒ set,/par/reset,y                  # for network changes to take effect
```

IP Address of TCP Server

Name: `/par/net/tcpcl/X/rcv/addr X=[a,b]`
Access: `rw`
Type: `string`
Values: `(any valid IP address)`
Default: `0.0.0.0`

The value of this parameter specifies IP address of the remote receiver, provided `/par/net/tcpcl/X/rcv/name` is empty.

Host Name of TCP Server

Name: `/par/net/tcpcl/X/rcv/name X=[a,b]`
Access: `rw`
Type: `string[0…64]`
Values: `(any string)`
Default: `(empty string)`

The value of this parameter specifies network name of the remote receiver, to be resolved using DNS.

IP Port of TCP Server

Name: `/par/net/tcpcl/X/rcv/port X=[a,b]`
Access: `rw`
Type: `integer`
Values: `[0…65535]`
Default: `0`

The value of this parameter specifies IP port of the remote receiver.

Establish Raw Connection to TCP Server

Name: /par/net/tcpcl/X/rcv/raw X=[a,b]
Access: rw
Type: boolean
Values: on,off
Default: off

off – receiver will establish connections to TCP server, will wait for login and password prompts, and answer them using the values specified by /par/net/tcpcl/X/rcv/login and /par/net/tcpcl/X/rcv/passwd parameters, respectively.

on – receiver will establish raw connection to TCP server, i.e., it won't expect any login or password prompts and won't send login or password to the TCP server.

Login Name for TCP Server

Name: /par/net/tcpcl/X/rcv/login X=[a,b]
Access: rw
Type: string[0…32]
Values: (any string)
Default: (empty string)

This parameter specifies the login name to be sent to the remote receiver as the reply to the login prompt. When /par/net/tcpcl/rcv/raw parameter is set to on, this parameter is not used.

Refer to the description of the /par/net/tcp/port parameter on page 462 for details of suitable values for connecting to receiver TCP server.

Login Password for TCP Server

Name: /par/net/tcpcl/X/rcv/passwd X=[a,b]
Access: rw
Type: string[0…32]
Default: (empty string)

This parameter specifies the password to be sent to the remote receiver as the reply to the password prompt. The value of this parameter should match the value of the parameter /par/net/passwd of the remote receiver.

When /par/net/tcpcl/X/rcv/raw parameter is set to on, this parameter is not used.

Note: This parameter is never printed implicitly.

RCV Client Identifier

Name: /par/net/tcpcl/X/rcv/id X=[a,b]
Access: rw
Type: string[0…63]
Values: (any string)
Default: (empty string)

This string will be sent to the server once, right after connection is established.

NTRIP Client Parameters

The parameters below are useful to provide a method to establish connection to an NTRIP caster, request data from particular mount point, and then receive and use the data as RTK/DGPS corrections.

Example: Configure receiver to connect to the NTRIP caster at specific IP address and port, to request data from the mountpoint REF1 (that we know is sending RTCM corrections), and to receive RTCM corrections from the mount point. Suppose also that this NTRIP caster requires NMEA GGA message to be sent to it periodically.

Note: You can obtain information about endpoints from the NTRIP table. To request NTRIP table, use /par/net/tcpcl/X/ntrip/table parameter (see below).

```
⇒ set,/par/net/tcpcl/a/ntrip/addr,87.236.81.134
⇒ set,/par/net/tcpcl/a/ntrip/port,80
⇒ set,/par/net/tcpcl/a/ntrip/mountpt,REF1
⇒ set,/par/net/tcpcl/a/ntrip/user,abc
⇒ set,/par/net/tcpcl/a/ntrip/passwd,abc
⇒ set,/par/net/tcpcl/a/ntrip/nmea,10          # send GGA every 10 seconds
⇒ set,/par/dev/tcpcl/a/imode,rtcm           # mountpoint sends RTCM
⇒ set,/par/net/tcpcl/a/mode,ntrip
```

IP Address of NTRIP Caster

Name: /par/net/tcpcl/X/ntrip/addr X=[a,b]
Access: rw
Type: string
Values: (any valid IP address)
Default: 0.0.0.0

The value of this parameter specifies IP address of the NTRIP caster to use, provided /par/net/tcpcl/X/ntrip/name is empty.

Host Name of NTRIP Caster

Name: /par/net/tcpcl/X/ntrip/name X=[a,b]
Access: rw
Type: string[0…64]
Values: (any string)
Default: (empty string)

The value of this parameter specifies network name of the NTRIP caster to use, to be resolved using DNS.

IP Port of NTRIP Caster

Name: /par/net/tcpcl/X/ntrip/port X=[a,b]
Access: rw
Type: integer
Values: [0…65535]
Default: 0

The value of this parameter specifies IP port of the NTRIP caster to use.

NTRIP Mount Point

Name: /par/net/tcpcl/X/ntrip/mountpt X=[a,b]
Access: rw
Type: string[0…15]
Values: (any string)
Default: (empty string)

This parameter specifies the mount point of the NTRIP caster to get data from.

NTRIP User Name

Name: /par/net/tcpcl/X/ntrip/user X=[a,b]
Access: rw
Type: string[0…32]
Values: (any string)
Default: (empty string)

This parameter specifies user ID for the protected space of the requested mount point. Only basic authentication scheme is supported. If empty, no user or password values will be sent to the NTRIP caster.

NTRIP Password

Name: /par/net/tcpcl/X/ntrip/passwd X=[a,b]
Access: rw
Type: string[0…32]
Values: (any string)
Default: (empty string)

This parameter specifies the password for the protected space of the requested mount point. Only basic authentication scheme is supported.

Note: This parameter is never printed implicitly.

NTRIP Host Name

Name: /par/net/tcpcl/X/ntrip/host X=[a,b]
Access: rw
Type: string[0…64]
Values: (any string)
Default: (empty string)

This parameter specifies the host name of the request. It is required for virtual hosts (multiple different instances of an NTRIP Caster at one server with same port). Informational on receiving side, but required for virtual casters.

NTRIP Protocol Version

Name: /par/net/tcpcl/X/ntrip/ver X=[a,b]
Access: rw
Type: integer
Values: 1,2
Default: 1

NMEA GGA Period for NTRIP

Name: /par/net/tcpcl/X/ntrip/nmea X=[a,b]
Access: rw
Type: integer
Values: [-1…86400], seconds
Default: 0

-1 – receiver will not send NMEA GGA messages to NTRIP caster.

0 – receiver will send NMEA GGA message to NTRIP caster only once after connection to the caster is established.

[1…86400] – receiver will send NMEA GGA messages to the NTRIP caster periodically, every specified number of seconds.

NTRIP Source Table

Name: /par/net/tcpcl/X/ntrip/table X=[a,b]
Access: r
Type: string

Printing this parameter forces receiver to request NTRIP source table from the NTRIP caster and output the table in the reply. Every line of the NTRIP source table will be output in a separate [RE] message.

Example:

```
⇒ print,/par/net/tcpcl/a/ntrip/table
⇐ RE014 SOURCETABLE 200 OK
⇐ RE020 Server: NTRIP Caster 1.5.8/1.0
⇐ RE01A Content-Type: text/plain
⇐ RE017 Content-Length: 11366
⇐ RE002
⇐ RE056 CAS;www.euref-ip.net;2101;EUREF-IP;BKG;0;DEU;50.12;8.69;
  http://www.euref-ip.net/home
⇐ RE092 NET;EUREF;EUREF;B;N;http://www.epncb.oma.be/euref_IP;
  http://www.epncb.oma.be:80/stations/log/skl;
  http://igs.bkg.bund.de/index_ntrip_reg.htm;none
⇐ RE07A STR;ACOR0;Coruna;RTCM 2.3;1(1),3(60),16,18(1),19(1);2;GPS;
  EUREF;ESP;43.36;351.60;0;0;LEICA GRX1200PRO;none;B;N;3000;IGNE
⇐ RE091 STR;ALAC0;Alicante;RTCM
  2.1;1(1),3(10),16,18(1),19(1),22(10),23(10),24(10),59;2;
  GPS;EUREF;ESP;38.34;359.52;0;0;TRIMBLE NETRS;none;B;N;5000;IGNE
⇐ RE091 STR;ALME0;Almeria;RTCM 2.3;1(1),3(10),18(1),19(1),22(10),23(10),
  24(10),59(10);2;GPS;EUREF;ESP;36.85;357.54;
  0;0;TRIMBLE NETRS;none;B;N;4000;IGNE
⇐ […]
⇐ RE010 ENDSOURCETABLE
```

NTRIP Server Parameters

IP Address of NTRIP Caster

Name: /par/net/tcpcl/X/ntrips/addr X=[a,b]
Access: rw
Type: string
Values: (any valid IP address)
Default: 0.0.0.0

Host Name of NTRIP Caster

Name: /par/net/tcpcl/X/ntrips/name X=[a,b]
Access: rw
Type: string [0…64]
Values: (any string)
Default: (empty string)

IP Port of NTRIP Caster

Name: /par/net/tcpcl/X/ntrips/port X=[a,b]
Access: rw
Type: integer
Values: [0…65535]
Default: 0

NTRIP Protocol Version

Name: /par/net/tcpcl/X/ntrips/ver X=[a,b]
Access: rw
Type: integer
Values: [1…2]
Default: 1

Mount Point on NTRIP Caster

Name: /par/net/tcpcl/X/ntrips/mountpt X=[a,b]
Access: rw
Type: string [0…15]
Values: (any string)
Default: (empty string)

User Name on NTRIP Caster

Name: /par/net/tcpcl/X/ntrips/user X=[a,b]
Access: rw
Type: string [0…32]
Values: (any string)
Default: (empty string)

This parameter specifies user ID for the protected space of the requested mount point. Only basic authentication scheme is supported. If empty, no user or password values will be sent to the NTRIP caster.

Password on NTRIP Caster

Name: /par/net/tcpcl/X/ntrips/passwd X=[a,b]
Access: rw
Type: string [0…32]
Values: (any string)
Default: (empty string)

This parameter specifies the password for the protected space of the requested mount point. Only basic authentication scheme is supported.

Host Name for NTRIP Caster

Name: /par/net/tcpcl/X/ntrips/host X=[a,b]
Access: rw
Type: string [0…64]
Values: (any string)
Default: (empty string)

This parameter specifies host name of the request. It is required for virtual hosts (multiple different instances of an NTRIP Caster at one server with same port). Informational on receiving side, but required for virtual casters.

Source Table Fields

Source Table Identifier

Name: /par/net/tcpcl/X/ntrips/table/id X=[a,b]
Access: rw
Type: string [0…32]
Values: (any string)
Default: (empty string)

Source identifier, e.g. name of city next to source location.

Source Table Data Format

Name: /par/net/tcpcl/X/ntrips/table/format X=[a,b]
Access: rw
Type: string [0…32]
Values: (any string)
Default: (empty string)

Data format RTCM, RAW, etc.

Source Table Format Description

Name: /par/net/tcpcl/X/ntrips/table/formatd X=[a,b]
Access: rw
Type: string [0…64]
Values: (any string)
Default: (empty string)

E.g., RTCM message types or RAW data format etc., update periods in parenthesis in seconds.

Source Table Carrier Phase

Name: /par/net/tcpcl/X/ntrips/table/carrier X=[a,b]
Access: rw
Type: integer
Values: [0…2]
Default: 0

0 – no carrier phase. E.g. for DGPS

1 – L1 carrier phase. E.g. for RTK

2 – L1&L2 carrier phase. E.g. for RTK

Source Table Navigation Systems

Name: /par/net/tcpcl/X/ntrips/table/nav X=[a,b]
Access: rw
Type: string [0…32]
Values: (any string)
Default: (empty string)

Source Table Network

Name: /par/net/tcpcl/X/ntrips/table/net X=[a,b]
Access: rw
Type: string [0…32]
Values: (any string)
Default: (empty string)

Source Table Country

Name: /par/net/tcpcl/X/ntrips/table/country X=[a,b]
Access: rw
Type: string [0…3]
Values: (any string)
Default: (empty string)

Three–character country code in ISO 3166.

Source Table Position Latitude

Name: /par/net/tcpcl/X/ntrips/table/lat X=[a,b]
Access: rw
Type: string [0…16]
Values: (any string)
Default: (empty string)

Position, latitude, north (approximate position in case of nmea=1).

Source Table Position Longitude

Name: /par/net/tcpcl/X/ntrips/table/lon X=[a,b]
Access: rw
Type: string [0…16]
Values: (any string)
Default: (empty string)

Position, longitude, east (approximate position in case of nmea=1).

Source Table NMEA Required

Name: /par/net/tcpcl/X/ntrips/table/nmea X=[a,b]
Access: rw
Type: integer
Values: [0…1]
Default: 0

0 – Client should not send NMEA message with its approximate position to Caster

1 – Client should send NMEA GGA message with approximate position to Caster

Source Table Stream Type

Name: /par/net/tcpcl/X/ntrips/table/sol X=[a,b]
Access: rw
Type: integer
Values: [0…1]
Default: 0

0 – Stream is generated from single base

1 – Stream is generated from networked reference stations

Source Table Stream Generator

Name: /par/net/tcpcl/X/ntrips/table/gen X=[a,b]
Access: rw
Type: string [0…32]
Values: (any string)
Default: (empty string)

Hardware or software that generates data stream.

Source Table Stream Compression

Name: /par/net/tcpcl/X/ntrips/table/compr X=[a,b]
Access: rw
Type: string [0…32]
Values: (any string)
Default: (empty string)

Compression/Encryption algorithm applied.

Source Table Authentication

Name: /par/net/tcpcl/X/ntrips/table/auth X=[a,b]
Access: rw
Type: enumerated
Values: N,B,D
Default: N

N - No access protection

B - Basic access protection

D - Digest access protection

Source Table Fee

Name: /par/net/tcpcl/X/ntrips/table/fee X=[a,b]
Access: rw
Type: enumerated
Values: N,Y
Default: N

N - No user fee

Y - Usage is charged

Source Table Bit Rate

Name: /par/net/tcpcl/X/ntrips/table/bitrate X=[a,b]
Access: rw
Type: integer [bits/s]
Values: [0…65535]
Default: 0

Source Table Stream Miscellaneous

Name: `/par/net/tcpcl/X/ntrips/table/misc X=[a,b]`
Access: `rw`
Type: `string [0…64]`
Values: `(any string)`
Default: `(empty string)`

Miscellaneous information, the last data field in source table record.

Network Statistics

Note: The parameters described below are mostly intended for the use by receiver firmware developers and are subject to change at any time.

TCP/IP Network Statistics

Name: `/par/net/stat`
Access: `r`
Type: `list {tcpd,mbuf,tcp,udp,icmp,if,drv,mem}`

`tcpd` - a list of active TCP connections.

`mbuf, tcp, udp, icmp, if` - internal statistics of the TCP/IP stack. The description of these fields exceeds the scope of this document[1].

`drv` - statistics from low-level Ethernet driver.

`mem` - memory usage statistics for network subsystem memory pool.

List of Active TCP Connections

Name: `/par/net/stat/tcpd`
Access: `r`
Type: `list`

For every active TCP connection this list contains an entry with a name that is a number in the range `[0…4]`.

Active TCP Connection

Name: `/par/net/stat/tcpd/N (N=[0…4])`
Access: `r`
Type: `list {ip,port,dev}`

`ip` - IP address of a peer of the selected TCP connection.

`port` - IP port of a peer of the selected TCP connection.

`dev` - TCP device allocated for the selected TCP connection.

1. Details can be found in the FreeBSD documentation.

TCP Peer IP Address

Name: /par/net/stat/tcpd/N/ip (N=[0…4])
Access: r
Type: ip_address

TCP Peer IP Port

Name: /par/net/stat/tcpd/N/port (N=[0…4])
Access: r
Type: integer
Values: [0…65635]

TCP Device Allocated for TCP Connection

Name: /par/net/stat/tcpd/N/dev (N=[0…4])
Access: r
Type: string
Values: /dev/tcp/X (X=[a…e]), (empty string)

An empty string indicates that the connection has been established but the user has not yet been logged in, i.e., receiver waits for login and/or password to be entered by the peer.

TLS/SSL Memory Pool Statistics

Name: /par/net/stat/tls
Access: r
Type: list {arena,ordblks,uordblks,fordblks,maxfree}

This parameter describes memory allocation statistics for the SSL/TLS memory pool. This parameter is intended for the JAVAD GNSS firmware developers and is subject to change at any time.

Ethernet Driver Statistics

Name: /par/net/stat/drv
Access: r
Type: list {rxints,txints,errints,rxskips,state}

rxints - number of receive interrupts.

txints - number of transmit interrupts.

errints - number of error interrupts.

rxskips - number of receive packets the driver lost.

state - driver state flags.

Receive Interrupts Count

Name: /par/net/stat/drv/rxints
Access: r
Type: integer

Transmit Interrupts Count

Name: /par/net/stat/drv/txints
Access: r
Type: integer

Error Interrupts Count

Name: /par/net/stat/drv/errints
Access: r
Type: integer

Lost Packets Count

Name: /par/net/stat/drv/rxskips
Access: r
Type: integer

Driver State Flags

Name: /par/net/stat/drv/state
Access: r
Type: integer

Driver state flags in hexadecimal representation as the logical OR product of the following flags:

- 0x01 - waiting for receive event
- 0x02 - waiting for transmit event
- 0x04 - waiting for packet transmit acknowledge
- 0x08 - waiting for transmitter ready
- 0x10 - sleeping

4.4.30 GSM, UHF, and FH Modem Parameters

Currently receivers support up to four GSM, UHF, or FH modems, called a, b, c, and d.

Note: In JAVAD GNSS receivers, the internal GSM, UHF, or FH modem is usually hardware-wise connected to Port C. As for external GSM radio modems, it is common practice to connect such modems to the receiver's Port B or Port D, if available.

Modem Mode

Name: /par/modem/X/mode (X=[a…d])
Access: rw
Type: enumerated
Values: off,master,slave,gprs,dialup,uhf,fh,auto,scan,lband
Default: off

This parameter specifies the mode the modem connected to corresponding receiver port will use to communicate with the remote modem at the other end of the radio link.

off - modem is off.

master - receiver will try to dial in to the remote (slave) GSM modem using the number specified by the parameter /par/modem/X/dial to call. Could be used for rover receiver only.

slave - GSM modem will wait for incoming calls from a master modem. Could be used for base receiver (reference station) only.

gprs - receiver will try to establish GPRS connection over GSM modem.

dial-up - receiver will try to establish dial-up connection over GSM or analog modem.

uhf - receiver will use UHF modem. Only parameters containing uhf in their name are applicable when this mode is set.

fh - receiver will use FH modem. Only parameters containing fh in their name are applicable when this mode is set.

auto - receiver will automatically detect modem and use it.

scan - receiver will automatically detect modem (UHF or FH) and will get spectrum from modem

lband - receiver will use LBAND receiver.

beacon - receiver will use BEACON receiver.

Current Modem Mode

Name: /par/modem/X/curmode (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Receiver Port the Modem is Connected to

Name: /par/modem/X/port (X=[a…d])
Access: rw
Type: string
Values: /dev/ser/x or /dev/blt/x

This parameter specifies the name of the serial port to which the modem is connected.

Note: for most receivers this parameter is read-only.

TX Buffer Bytes

Name: `/par/modem/X/bport/txbyte (X=[a…d])`
Access: `r`
Type: `integer`
Values: `[0…4294967295]`
Default: `0`

The number of bytes in TX buffer of modem port.

RX Buffer Bytes

Name: `/par/modem/X/bport/rxbyte (X=[a…d])`
Access: `r`
Type: `integer`
Values: `[0…4294967295]`
Default: `0`

The number of bytes in RX buffer of modem port.

GSM Modem Parameters

GSM Hotstart Mode

Name: `/par/modem/X/hotstart (X=[a…d])`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` - modem will be initialized at receiver startup and therefore will quickly become ready for operation after it is turned on due to change of its mode.

`off` - modem will be turned off until user turns it on by changing its mode. This mode saves power and is more reliable as it re-initializes modem entirely on every turning it on, but results in longer modem initialization.

GSM Registration Mode

Name: `/par/modem/X/rmode (X=[a…d])`
Access: `rw`
Type: `enumerated`
Values: `auto,2g,3g,4g,best`
Default: `auto`

`auto` - GSM modem will automatically register in the first available network, in the following order: 4g, 3g, 2g

`2g` - register in 2g network

`3g` - register in 3g network

`4g` - register in 4g network

`best` - GSM modem will automatically register on the best signal

SIM Card Number

Name: `/par/modem/X/sim (X=[a…d])`
Access: `rw`
Type: `integer`
Values: `[1…3]`
Default: `1`

This parameter specifies the SIM card number to use for receivers that support multiple SIM cards.

PIN Code

Name: `/par/modem/X/pin (X=[a…d])`
Access: `rw`
Type: `string[4]`
Values: `four decimal digits`
Default: `0000`

This parameter specifies the SIM card's PIN code for a GSM modem.

Dial Number

Name: `/par/modem/X/dial (X=[a…d])`
Access: `rw`
Type: `string[0…32]`
Values: `up to 32 decimal digits`
Default: `(empty string)`

This parameter specifies the phone number that the GSM modem will dial when in master mode.

Modem Control State

Name: `/par/modem/X/state (X=[a…d])`
Access: `r`
Type: `enumerated`
Values: `off,detect,detected,init,registration,gregistration, ready,ring,dialling,hot,connect,discon,err`

This parameter shows the current modem control state.

`off` – modem control is turned off.

`detect` – searching for a modem on the corresponding port.

`detected` – modem has been detected. Modem initialization is in progress.

`init` – modem is being initialized

`registration` – modem is being registered in the network.

`gregistration` – modem is being registered in the GPRS network.

`ready` – modem has been initialized and registered in the GSM network. If the modem is in slave mode, it is ready to receive an incoming call. If the modem is in master mode, it is ready to dial in to the slave modem.

`dialing` – modem is dialing the selected phone number as specified by the parameter `/par/modem/X/dial` (in master mode only).

`ring` – an incoming call is being received (in slave mode only).

`hot` – modem is hot and will be quickly turned into `connect` state as soon as its mode is turned on. For details, refer to the parameter "GSM Hotstart Mode" .

`connect` – connection has been established.

`discon` – connection has been broken ("disconnecting").

`err` – fatal error has occurred. In this case the user will need to change the parameter `/par/modem/X/mode` to `off`, fix the problem and then retry setting the required modem mode. See the parameter `/par/modem/X/err` for what specifically might have caused the error.

Last Detected Modem Error

Name: `/par/modem/X/err (X=[a…d])`
Access: `r`
Type: `enumerated`
Values: `none,detect,init,pin,net,busy,no_carrier,no_answer, gnet,gservice,gdenial,prot,freq,fw,chan`
Default: `none`

This parameter shows the last of the errors identified by the modem driver provided the value of `/par/modem/X/state` is `err`.

`none` – no errors have been detected.

`detect` – cannot find a modem on the port.

`init` – an initialization error has occurred.

`pin` – wrong PIN code.

`net` – a network error has occurred.

busy – the phone number is busy. To rectify this temporary problem, just call again at a later time.

no_carrier – cannot detect the carrier signal. This temporary error can occur if the second modem (at the other end of the radio link) has not been initialized or if there are some problems with the GSM network. The given GSM modem will continue to dial in to the modem on the other side of the radio link until the carrier signal is detected.

no_answer – no hang up is detected after a fixed network time-out.

gnet – GPRS network error.

gservice – error attaching to GPRS service.

gdenial – GPRS registration denied.

prot – incorrect protocol.

freq – incorrect frequency.

fw – incompatible version of modem firmware.

chan – wrong channel number.

syn – frequency synthesizer error.

ant – antenna detect error.

Note: The modem control will attempt to automatically fix a detected error in case the parameter /par/modem/X/state takes a value other than err or off. No user intervention is needed unless the parameter /par/modem/X/state turns out equal to err, which means that the modem control has not been able to fix the problem on its own.

Data Wait Timeout

Name: /par/modem/X/rcvtimeout (X=[a…d])
Access: rw
Type: integer [seconds]
Values: [0…1000]
Default: 5

If the receiver has not received any data from the modem for rcvtimeout seconds, the modem will be disconnected and then re-initialized. If the parameter is set to 0, such control will be disabled.

Service Word Repeat Period

Name: /par/modem/X/sndtime (X=[a…d])
Access: rw
Type: integer [seconds]
Values: [0…1000]
Default: 2

This parameter, which specifies a time interval, is used to ensure reliable communication between the pair of modems (master - slave) and avoid unnecessary modem reinitialization.The transmit modem will send the service word to the receive modem every `sndtime` seconds. Note that the service word will not affect the differential corrections (RTCM or CMR messages) in any way. If the parameter is set to zero, the service word will not be used in data transmission.

Note: To ensure reliable and secure modem communication, the parameter `/par/modem/X/sndtime` must be larger then the period of transmitting differential corrections. Also, care should be taken that the time `/par/modem/X/rcvtimeout` is greater than the service word repeat period by 2 to 3 seconds.

Network Type

Name: `/par/modem/X/type (X=[a…d])`
Access: `rw`
Type: `enumerated`
Values: `gsm,pstn`
Default: `gsm`

This parameter specifies the type of the network to use.

`gsm` - GSM network (for GSM modem).

`pstn` - Public Switched Telephone Network (for analog modem).

Cellular Operator Name

Name: `/par/modem/X/inf/cell/oper (X=[a…d])`
Access: `r`
Type: `string[0…50]`
Default: `"unknown"`

GSM/GPRS/EDGE coverage

Name: `/par/modem/X/inf/cell/cover (X=[a…d])`
Access: `r`
Type: `string[0…50]`
Default: `"none"`

GSM Signal Quality.

Name: `/par/modem/X/inf/cell/sq (X=[a…d])`
Access: `r`
Type: `string[0…50]`
Default: `"unknown"`

AT+CBST Data Rate

Name: /par/modem/X/at/cbst/speed (X=[a…d])
Access: rw
Type: integer
Values: [0…255]
Default: 71

AT+CBST Bearer Service Name

Name: /par/modem/X/at/cbst/name (X=[a…d])
Access: rw
Type: integer
Values: [0…32]
Default: 0

AT+CBST Connection Element

Name: /par/modem/X/at/cbst/ce (X=[a…d])
Access: rw
Type: integer
Values: [0…32]
Default: 1

Modem Vendor

Name: /par/modem/X/inf/dev/vendor (X=[a…d])
Access: r
Type: string[0…50]
Default: "unknown"

Modem Model

Name: /par/modem/X/inf/dev/model (X=[a…d])
Access: r
Type: string[0…50]
Default: "unknown"

Modem Revision

Name: /par/modem/X/inf/dev/rev (X=[a…d])
Access: r
Type: string[0…50]
Default: "unknown"

Modem Serial Number

Name: /par/modem/X/inf/dev/sn (X=[a…d])
Access: r
Type: string[0…50]
Default: "unknown"

Modem FCC ID

Name: /par/modem/X/inf/dev/fccid (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Modem IC

Name: /par/modem/X/inf/dev/ic (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Modem Equipment Identity

Name: /par/modem/X/inf/dev/id (X=[a…d])
Access: r
Type: string[0…50]
Default: "unknown"

This parameter displays the IMEI (International Mobile station Equipment Identity) for G24, H24, GE864 modems, and the MEID (Mobile Equipment Identifier) for C24 modem.

UHF Modem Parameters

UHF Modem Output Power

Name: /par/modem/X/uhf/link/pwr (X=[a…d])
Access: rw
Type: integer [dBm]
Values: [15…46]
Default: 30

UHF Modem Frequency

Name: /par/modem/X/uhf/link/freq (X=[a…d])
Access: rw
Type: integer [Hz]
Values: [138000000…470000000], with step 10000
Default: 440000000

UHF Modem RX Frequency

Name: /par/modem/X/uhf/link/rfreq (X=[a…d])
Access: rw
Type: integer [Hz]
Values: 0 or [138000000…470000000], with step 6250 or 10000
Default: 0

If this parameter is 0, the RX frequency will be taken from /par/modem/X/uhf/link/freq parameter.

UHF Modem TX Frequency

Name: /par/modem/X/uhf/link/tfreq (X=[a…d])
Access: rw
Type: integer [Hz]
Values: 0 or [138000000…470000000], with step 6250 or 10000
Default: 0

If this parameter is 0, the TX frequency will be taken from /par/modem/X/uhf/link/freq parameter.

Call Sign of Transceiver

Name: /par/modem/X/uhf/link/csign (X=[a…d])
Access: rw
Type: string[0…10]
Values: [A-Z,0-9]
Default: ""

UHF Protocol Type

Name: /par/modem/X/uhf/link/prot (X=[a…d])
Access: rw
Type: enumerated
Values: simrx,simtx,simtr,tmorx,tmotx,tmotr,
trmrx,trmtx,trmtr,stlrx,stltx,stltr,simrtr,
tt450shwrx,tt450shwtx,tt450shwtr,
trmm3rx,trmm3tx,trmm3tr,trmm2rx,trmm2tx,trmm2tr
Default: simrx

simrx - Simplex Receiver Protocol
simtx - Simplex Transmitter Protocol
simtr - Simplex Transceiver Protocol
tmorx - Transparent w/EOT Receiver Protocol
tmotx - Transparent w/EOT Transmitter Protocol
tmotr - Simplex Transceiver Protocol
trmrx - TRMB Receiver Protocol
trmtx - TRMB Transmitter Protocol
trmtr - Trimtalk 450S Transceiver Protocol
stlrx - STL Receiver Protocol
stltx - STL Transmitter Protocol
stltr - STL Transceiver Protocol
simrtr - Simplex Repeater Protocol
tt450shwrx - TT450S(HW) Receiver Protocol
tt450shwtx - TT450S(HW) Transmitter Protocol
tt450shwtr - TT450S(HW) Transceiver Protocol
trmm3rx - Trimmark3 Receiver Protocol
trmm3tx - Trimmark3 Transmitter Protocol
trmm3tr - Trimmark3 Transceiver Protocol
trmm2rx - Trimmark ll/lle Receiver Protocol
trmm2tx - Trimmark ll/lle Transmitter Protocol
trmm2tr - Trimmark ll/lle Transceiver Protocol

Modulation Type for Simplex Protocol

Name: /par/modem/X/uhf/link/protst/sim/mod (X=[a…d])
Access: rw
Type: enumerated
Values: dbpsk,dqpsk,d8psk,d16qam
Default: dqpsk

Channel Spacing for Simplex Protocol

Name: /par/modem/X/uhf/link/protst/sim/space (X=[a…d])
Access: rw
Type: enumerated
Values: 6250,12500,20000,25000
Default: 25000

Scrambling for Simplex Protocol

Name: /par/modem/X/uhf/link/protst/sim/scr (X=[a…d])
Access: rw
Type: integer
Values: [0…255]
Default: 255

Forward Error Correction for Simplex Protocol

Name: /par/modem/X/uhf/link/protst/sim/fec (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

Transmission of Sync Packets for Simplex Protocol

Name: /par/modem/X/uhf/link/protst/sim/snrm (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: off

When enabled, synchronization packets will be transmitted to avoid possible data damage.

Number of Repeaters for Simplex Protocol

Name: /par/modem/X/uhf/link/protst/sim/mode/base/rtrnum (X=[a…d])
Access: rw
Type: integer
Values: [0…1]
Default: 0

This parameter should contain the number of repeaters used in a system of data transmission from base to rover for Simplex Protocol.

Data Source for Simplex Protocol

Name: /par/modem/X/uhf/link/protst/sim/mode/rover/source (X=[a…d])
Access: rw
Type: enumerated
Values: auto,base,rtr1
Default: auto

This parameter contains the source of data transmission for rover (for Simplex Protocol).

Echo Port for Simplex Protocol Repeater

Name: /par/modem/X/uhf/link/protst/sim/mode/rtr/echo (X=[a…d])
Access: rw
Type: enumerated
Values: off,a
Default: a

Modulation Type for Transparent Protocol

Name: /par/modem/X/uhf/link/protst/tmo/mod (X=[a…d])
Access: rw
Type: enumerated
Values: gmsk
Default: gmsk

Channel Spacing for Transparent Protocol

Name: /par/modem/X/uhf/link/protst/tmo/space (X=[a…d])
Access: rw
Type: enumerated
Values: 6250,12500,20000,25000
Default: 25000

Scrambling for Transparent Protocol

Name: /par/modem/X/uhf/link/protst/tmo/scr (X=[a…d])
Access: rw
Type: integer
Values: [0…255]
Default: 255

Forward Error Correction for Transparent Protocol

Name: /par/modem/X/uhf/link/protst/tmo/fec (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

Modulation Type for TRMB Protocol

Name: /par/modem/X/uhf/link/protst/trm/mod (X=[a…d])
Access: rw
Type: enumerated
Values: gmsk
Default: gmsk

Channel Spacing for TRMB Protocol

Name: /par/modem/X/uhf/link/protst/trm/space (X=[a…d])
Access: rw
Type: enumerated
Values: 6250,12500,20000,2500
Default: 25000

Scrambling for TRMB Protocol

Name: /par/modem/X/uhf/link/protst/trm/scr (X=[a…d])
Access: rw
Type: integer
Values: [0…255]
Default: 255

Forward Error Correction for TRMB Protocol

Name: /par/modem/X/uhf/link/protst/trm/fec (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

Modulation Type for STL Protocol

Name: /par/modem/X/uhf/link/protst/stl/mod (X=[a…d])
Access: rw
Type: enumerated
Values: 4fsk
Default: 4fsk

Channel Spacing for STL Protocol

Name: /par/modem/X/uhf/link/protst/stl/space (X=[a…d])
Access: rw
Type: enumerated
Values: 6250,12500,20000,2500
Default: 25000

Forward Error Correction for STL Protocol

Name: /par/modem/X/uhf/link/protst/stl/fec (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

Satel Model Compatibility

Name: /par/modem/X/uhf/link/protst/stl/cmpt (X=[a…d])
Access: rw
Type: enumerated
Values: 3as,easy
Default: 3as

Clock Correction for STL Protocol

Name: /par/modem/X/uhf/link/protst/stl/clkcrr (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

Modulation Type for TT450S(HW) Protocol

Name: /par/modem/X/uhf/link/protst/tt450shw/mod (X=[a…d])
Access: rw
Type: enumerated
Values: gmsk
Default: gmsk

Channel Spacing for TT450S(HW) Protocol

Name: /par/modem/X/uhf/link/protst/tt450shw/space (X=[a…d])
Access: rw
Type: enumerated
Values: 6250,12500,20000,2500
Default: 25000

Modulation Type for Trimmark3 Protocol

Name: /par/modem/X/uhf/link/protst/trmm3/mod (X=[a…d])
Access: rw
Type: enumerated
Values: gmsk
Default: gmsk

Channel Spacing for Trimmark3 Protocol

Name: /par/modem/X/uhf/link/protst/trmm3/space (X=[a…d])
Access: rw
Type: enumerated
Values: 6250,12500,20000,2500
Default: 25000

Modulation Type for Trimmark II/IIe Protocol

Name: /par/modem/X/uhf/link/protst/trmm2/mod (X=[a…d])
Access: rw
Type: enumerated
Values: gmsk
Default: gmsk

Channel Spacing for Trimmark II/IIe Protocol

Name: /par/modem/X/uhf/link/protst/trmm2/space (X=[a…d])
Access: rw
Type: enumerated
Values: 6250,12500,20000,25000
Default: 25000

TX Delay

Name: /par/modem/X/uhf/link/txdelay (X=[a…d])
Access: rw
Type: integer, [milliseconds]
Values: [0…650]
Default: 20

Delay from receiving of the first byte of data from modem port to the start of data transmission.

Temperature

Name: /par/modem/X/uhf/inf/link/tx/temp (X=[a…d])
Access: r
Type: string[0…32], [Celsius]
Default: "unknown"

Output Power

Name: /par/modem/X/uhf/inf/link/tx/pwr (X=[a…d])
Access: r
Type: string[0…32], [dBm]
Default: "unknown"

Power Supply Voltage

Name: /par/modem/X/uhf/inf/link/tx/volt (X=[a…d])
Access: r
Type: string[0…32], [Volts]
Default: "unknown"

System Power Supply Voltage

Name: /par/modem/X/uhf/inf/link/tx/svolt (X=[a…d])
Access: r
Type: string[0…32], [Volts]
Default: "unknown"

Number of Transferred Frames

Name: /par/modem/X/uhf/inf/link/tx/cnt/frame (X=[a…d])
Access: r
Type: integer
Values: [0…4294967295]

Number of Transferred Bytes

Name: /par/modem/X/uhf/inf/link/tx/cnt/byte (X=[a…d])
Access: r
Type: integer
Values: [0…4294967295]

Current Number of Transferred Bytes

Name: /par/modem/X/uhf/inf/link/tx/cnt/curbyte (X=[a…d])
Access: r
Type: integer
Values: [0…4294967295]

Byte Count in the UART Buffer

Name: /par/modem/X/uhf/inf/link/tx/cnt/ubyte (X=[a…d])
Access: r
Type: integer
Values: [0…4294967295]

Voltage Standing Wave Ratio

Name: /par/modem/X/uhf/inf/link/tx/ant/vswr (X=[a…d])
Access: r
Type: string [0…32]
Default: "unknown"

Reflected Power Indicator

Name: /par/modem/X/uhf/inf/link/tx/ant/rflpwr (X=[a…d])
Access: r
Type: string [0…32]
Values: "unknown"

Received Signal Strength Indication

Name: /par/modem/X/uhf/inf/link/rx/rssi (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Bit Error Rate

Name: /par/modem/X/uhf/inf/link/rx/ber (X=[a…d])
Access: r
Type: float
Default: "unknown"

UHF Modem Model

Name: /par/modem/X/uhf/inf/model (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

UHF Modem FCC ID

Name: /par/modem/X/uhf/inf/fccid (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

UHF Modem IC

Name: /par/modem/X/uhf/inf/ic (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

UHF Modem Product ID

Name: /par/modem/X/uhf/inf/id (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Internal UHF Modem Product ID

Name: /par/modem/X/uhf/inf/id2 (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

UHF Modem Serial Number

Name: /par/modem/X/uhf/inf/sn (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

UHF Modem Hardware Revision

Name: /par/modem/X/uhf/inf/hw (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

UHF Modem Software Version

Name: /par/modem/X/uhf/inf/sw (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

UHF Modem Bootloader Version

Name: /par/modem/X/uhf/inf/bl (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

UHF Modem MCU Firmware Version

Name: /par/modem/X/uhf/inf/mcu (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

UHF Modem Power Board MCU Firmware Version

Name: /par/modem/X/uhf/inf/mcu2 (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Spectrum Start Frequency

Name: /par/modem/X/uhf/scan/freq/start (X=[a…d])
Access: rw
Type: integer [Hz]
Values: [138000000…470000000], step 6250
Default: 406000000

Spectrum Stop Frequency

Name: /par/modem/X/uhf/scan/freq/stop (X=[a…d])
Access: rw
Type: integer [Hz]
Values: [138000000…470000000], step 6250
Default: 470000000

Spectrum Frequency Step

Name: /par/modem/X/uhf/scan/freq/step (X=[a…d])
Access: rw
Type: integer [Hz]
Values: [5000…999000]
Default: 12500

Spectrum Scanning Mode

Name: /par/modem/X/uhf/scan/mode (X=[a…d])
Access: rw
Type: enumerated
Values: scan,rssi
Default: scan

Spectrum Scanning Timeout

Name: /par/modem/X/uhf/scan/timeout (X=[a…d])
Access: rw
Type: integer [msec]
Values: [250…25000]
Default: 1500

The signal timeout for scanning in RSSI mode.

Save Configuration File in Internal UHF Modem

Name: /par/modem/X/uhf/save/int (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on (off for TRIUMPH-LS)

Save Configuration File in External UHF Modem

Name: /par/modem/X/uhf/save/ext (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

Antenna Detection Procedure

Name: /par/modem/X/uhf/ctrl/ant (X=[a…d])
Access: rw
Type: enumerated
Values: auto,manual,ignore
Default: ignore

Provides control for antenna detection procedure. Applicable tor LMR400 (ID:741, ID:931).

Voltage Standing Wave Ratio

Name: /par/modem/X/uhf/ctrl/vswr (X=[a…d])
Access: rw
Type: integer
Values: [3…6]
Default: 3

Applicable to LMR400 (ID:741, ID:931).

Voltage Standing Wave Ratio Update Time

Name: /par/modem/X/uhf/ctrl/utime (X=[a…d])
Access: rw
Type: integer, [seconds]
Values: [0…50]
Default: 0

Applicable to LMR400 (ID:741, ID:931).

Automatic Gain Control

Name: /par/modem/X/uhf/ctrl/agc (X=[a…d])
Access: rw
Type: enumerated
Values: on,off
Default: on

Applicable to LMR400 (ID:741, ID:931).

Attenuation Control Level

Name: /par/modem/X/uhf/ctrl/acl (X=[a…d])
Access: rw
Type: integer, [dBm]
Values: [-60…0]
Default: -50

Applicable to UHFSSRX (ID:111).

Switch Control for Attenuation

Name: /par/modem/X/uhf/ctrl/sw (X=[a…d])
Access: rw
Type: integer, [dBm]
Values: [-40…0]
Default: 0

Provides FH/UHF switch control for attenuation. Applicable to UHFSSRX (ID:111).

FH Modem Parameters

FH Modem Repeater Mode

Name: /par/modem/X/fh/link/rtr/mode (X=[a…d])
Access: rw
Type: enumerated
Values: off,on
Default: off

FH Modem Zone

Name: /par/modem/X/fh/link/zone (X=[a…d])
Access: rw
Type: enumerated
Values: usa,aus,eu
Default: usa

The zone of FH radio operation.

usa - USA

aus - Australia

eu - Europe

FH Modem Power for USA and Australia

Name: /par/modem/X/fh/link/usa/pwr (X=[a…d])
Access: rw
Type: integer [dBm]
Values: [15…30]
Default: 30

FH Modem Protocol for USA and Australia

Name: /par/modem/X/fh/link/usa/prot (X=[a…d])
Access: rw
Type: enumerated
Values: simrx,simtx
Default: simrx

simrx - Simplex Receiver Protocol

simtx - Simplex Transmitter Protocol

FH Modem FH Rule for USA and Australia

Name: /par/modem/X/fh/link/usa/frule (X=[a…d])
Access: rw
Type: integer
Values: [0…9]
Default: 0

The rule of frequency hopping of FH radio for USA and Australia zones.

FH Modem Simplex Modulation for USA and Australia

Name: /par/modem/X/fh/link/usa/protst/sim/mod (X=[a…d])
Access: rw
Type: enumerated
Values: gmsk
Default: gmsk

The modulation type for Simplex Protocol of FH radio for USA and Australia zones.

FH Modem Simplex Scrambling for USA and Australia

Name: /par/modem/X/fh/link/usa/protst/sim/scr (X=[a…d])
Access: rw
Type: integer
Values: [0…1]
Default: 1

The scrambling for Simplex Protocol of FH radio for USA and Australia zones.

FH Modem Simplex FEC for USA and Australia

Name: /par/modem/X/fh/link/usa/protst/sim/fec (X=[a…d])
Access: rw
Type: enumerated
Values: cnv,off
Default: cnv

cnv - convolution code

off - none

The Forward Error Correction for Simplex Protocol of FH radio for USA and Australia zones.

FH Modem Redundancy of Data Packets for USA and Australia

Name: /par/modem/x/fh/link/usa/chan/redund
Access: rw
Type: enumerated
Values: single,double
Default: double

single - Data Packet is transmitted once at the current frequency.

double - Data Packet is transmitted twice: first time at the current frequency, second time at the next frequency. Best Data Packet will be then selected on receiving end,

FH Modem Power for Europe

Name: /par/modem/X/fh/link/eu/pwr (X=[a…d])
Access: rw
Type: integer [dBm]
Values: [7.…27]
Default: 27

RX FH Modem Frequency for Europe

Name: /par/modem/X/fh/link/eu/rfreq (X=[a…d])
Access: rw
Type: integer [Hz]
Values: [868000000…870000000], with 6250 Hz step
Default: 869000000

TX FH Modem Frequency for Europe

Name: /par/modem/X/fh/link/eu/tfreq (X=[a…d])
Access: rw
Type: integer [Hz]
Values: [868000000…870000000], with 6250 Hz step
Default: 869000000

FH Modem Protocol for Europe

Name: /par/modem/X/fh/link/eu/prot (X=[a…d])
Access: rw
Type: enumerated
Values: simrx,simtx
Default: simrx

simrx - Simplex Receiver Protocol

simtx - Simplex Transmitter Protocol

FH Modem Simplex Modulation for Europe

Name: /par/modem/X/fh/link/eu/protst/sim/mod (X=[a…d])
Access: rw
Type: enumerated
Values: gmsk
Default: gmsk

FH Modem Simplex Channel Spacing for Europe

Name: /par/modem/X/fh/link/eu/protst/sim/space (X=[a…d])
Access: rw
Type: enumerated
Values: 12500,20000,25000
Default: 25000

FH Modem Simplex Scrambling for Europe

Name: /par/modem/X/fh/link/eu/protst/sim/scr (X=[a…d])
Access: rw
Type: integer
Values: [0…1]
Default: 1

FH Modem Simplex FEC for Europe

Name: /par/modem/X/fh/link/eu/protst/sim/fec (X=[a…d])
Access: rw
Type: enumerated
Values: cnv,off
Default: cnv

cnv - convolution code

off - none

The Forward Error Correction for Simplex Protocol of FH radio for Europe zone.

Number of Transferred Bytes

Name: /par/modem/X/fh/inf/link/tx/cnt/byte (X=[a…d])
Access: r
Type: integer
Values: [0…4294967295]

Current Number of Transferred Bytes

Name: /par/modem/X/fh/inf/link/tx/cnt/curbyte (X=[a…d])
Access: r
Type: integer
Values: [0…4294967295]

FH Modem Temperature in Celsius

Name: /par/modem/X/fh/inf/link/tx/temp (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Output Power of FH Modem in dBm

Name: /par/modem/X/fh/inf/link/tx/pwr (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Power Supply Voltage of FH Modem in Volt

Name: /par/modem/X/fh/inf/link/tx/volt (X=[a…d])
Access: r
Type: string[0…32], [volt]
Default: "unknown"

System Power Supply Voltage of FH Modem

Name: /par/modem/X/fh/inf/link/tx/svolt (X=[a…d])
Access: r
Type: string[0…32], [volt]
Default: "unknown"

Received Signal Strength Indication

Name: /par/modem/X/fh/inf/link/rx/rssi (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Number of Received Bad Sub-packets

Name: /par/modem/X/fh/inf/link/rx/bsp (X=[a…d])
Access: r
Type: integer
Values: [0…4294967295]
Default: 0

Number of Received Sub-packets

Name: /par/modem/X/fh/inf/link/rx/rsp (X=[a…d])
Access: r
Type: integer
Values: [0…4294967295]
Default: 0

Sub-packets Error Rate

Name: /par/modem/X/fh/inf/link/rx/per (X=[a…d])
Access: r
Type: float, or "unknown"
Default: "unknown"

Reset Statistics Information

Name: /par/modem/X/fh/statreset (X=[a…d])
Access: w
Type: boolean
Values: yes,no
Default: no

FH Modem FCC ID

Name: /par/modem/X/fh/inf/fccid (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

FH Modem IC

Name: /par/modem/X/fh/inf/ic (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

FH Modem Model

Name: /par/modem/X/fh/inf/model (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

FH Modem Product ID

Name: /par/modem/X/fh/inf/id (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Internal FH Modem Product ID

Name: /par/modem/X/fh/inf/id2 (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

FH Modem Power Board MCU Firmware Version

Name: /par/modem/X/fh/inf/mcu2 (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

FH Modem Serial Number

Name: /par/modem/X/fh/inf/sn (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

FH Modem Hardware Revision

Name: /par/modem/X/fh/inf/hw (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

FH Modem Firmware Version

Name: /par/modem/X/fh/inf/fw (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

FH Modem Bootloader Version

Name: /par/modem/X/fh/inf/bl (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Spectrum Start Frequency

Name: /par/modem/X/fh/scan/freq/start (X=[a…d])
Access: rw
Type: integer [Hz]
Values: [850000000…970000000], step 6250
Default: 900000000

Spectrum Stop Frequency

Name: /par/modem/X/fh/scan/freq/stop (X=[a…d])
Access: rw
Type: integer [Hz]
Values: [850000000…970000000], step 6250
Default: 930000000

Spectrum Frequency Step

Name: /par/modem/X/fh/scan/freq/step (X=[a…d])
Access: rw
Type: integer [Hz]
Values: [1000…999000]
Default: 10000

Save Configuration File in Internal FH915 Modem

Name: /par/modem/X/fh/save/int (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on (off for TRIUMPH-LS)

Save Configuration File in External FH915 Modem

Name: /par/modem/X/fh/save/ext (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

Auto Modem Parameters

Prefer LBAND Over BEACON

Name: /par/modem/X/auto/prefer/lband (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

on - prefer LBAND receiver when LBAND/BEACON board is detected.

off - prefer BEACON receiver when LBAND/BEACON board is detected.

Prefer UHF Receiver When UHFSSRX Board is Detected.

Name: /par/modem/X/auto/prefer/uhf (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

Auto Modem Model

Name: /par/modem/X/auto/inf/model (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Auto Modem Product ID

Name: /par/modem/X/auto/inf/id (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Internal Auto Modem Product ID

Name: /par/modem/X/auto/inf/id2 (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Auto Modem Serial Number

Name: /par/modem/X/auto/inf/sn (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Auto Modem Hardware Revision

Name: /par/modem/X/auto/inf/hw (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Auto Modem Software Version

Name: /par/modem/X/auto/inf/sw (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Auto Modem Bootloader Version

Name: /par/modem/X/auto/inf/bl (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Auto Modem MCU Firmware Version

Name: /par/modem/X/auto/inf/mcu (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

Auto Modem Power Board MCU Firmware Version

Name: /par/modem/X/auto/inf/mcu2 (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Receiver Parameters

LBAND Frequency Channel

Name: /par/modem/X/lband/link/chan (X=[a…d])
Access: rw
Type: integer
Values: [1…32]
Default: 1

LBAND Link Rate

Name: /par/modem/X/lband/link/lrate (X=[a…d])
Access: rw
Type: enumerated [bps]
Values: 4800,2400,1200,600,300
Default: 1200

LBAND Scrambling Initial Seed

Name: /par/modem/X/lband/link/scr (X=[a…d])
Access: rw
Type: string [1…4]
Values: (hexadecimal string)
Default: 1

The 16-bit scrambling initial seed in hex format.

LBAND Unique Word

Name: /par/modem/X/lband/link/uw (X=[a…d])
Access: rw
Type: string [16]
Values: (hexadecimal string)
Default: E15AE893E15AE893

The 8-byte Unique Word in hex format.

LBAND Antenna Power Switcher

Name: /par/modem/X/lband/ant/pwr (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

LBAND Channels Frequencies

Name: /par/modem/X/lband/map/freq (X=[a…d])
Access: rw
Type: array [1…32] of integer [Hz]
Values: {[1525000000…1559000000],…}
Default: {1525000000,0,…}

The frequencies list will be loaded into LBAND receiver at startup. The first zero value in the list terminates loading.

LBAND Model

Name: /par/modem/X/lband/inf/model (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Product ID

Name: /par/modem/X/lband/inf/id (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Receiver Product ID

Name: /par/modem/X/lband/inf/id2 (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Power Board MCU Firmware Version

Name: /par/modem/X/lband/inf/mcu2 (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Serial Number

Name: /par/modem/X/lband/inf/sn (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Hardware Revision

Name: /par/modem/X/lband/inf/hw (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Firmware Version

Name: /par/modem/X/lband/inf/sw (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Boot-loader Version

Name: /par/modem/X/lband/inf/bl (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Service Identifier Within Frame

Name: /par/modem/X/lband/inf/link/rsid (X=[a…d])
Access: r
Type: string[0…32] (in hex format)
Default: "unknown"

LBAND Received Signal Strength Indication

Name: /par/modem/X/lband/inf/link/rssi (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Synchronization Status

Name: /par/modem/X/lband/inf/link/sync (X=[a…d])
Access: r
Type: enumerated
Values: 0,1,unknown
Default: unknown

0 - no synchronization.

1 - synchronization established.

LBAND Bit Error Rate

Name: /par/modem/X/lband/inf/link/ber (X=[a…d])
Access: r
Type: float
Default: "unknown"

LBAND Current Channel Frequency

Name: /par/modem/X/lband/inf/link/freq (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Tuned Frequency Offset

Name: /par/modem/X/lband/inf/link/freqoffset (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

LBAND Current Symbol Rate

Name: /par/modem/X/lband/inf/link/symrate (X=[a…d])
Access: r
Type: float
Default: "unknown"

BEACON Receiver Parameters

BEACON Frequency Channel

Name: /par/modem/X/beacon/link/chan (X=[a…d])
Access: rw
Type: integer
Values: [1…32]
Default: 1

BEACON Link Rate

Name: /par/modem/X/beacon/link/lrate (X=[a…d])
Access: rw
Type: enumerated [bps]
Values: 50,100,200
Default: 100

BEACON Antenna Power Switcher

Name: /par/modem/X/beacon/ant/pwr (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

BEACON J300 Connector

Name: /par/modem/X/beacon/ant/bcn (X=[a…d])
Access: rw
Type: boolean
Values: on,off
Default: on

off - select J302 connector.

on - select J300 connector;

BEACON Channels Frequencies

Name: /par/modem/X/beacon/map/freq (X=[a…d])
Access: rw
Type: array [1…32] of integer [Hz]
Values: {[283500…325000],…}
Default: {283500,0,…}

The frequencies list will be loaded into BEACON receiver at startup. The first zero value in the list terminates loading.

BEACON Model

Name: /par/modem/X/beacon/inf/model (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

BEACON Product ID

Name: /par/modem/X/beacon/inf/id (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

BEACON Receiver Product ID

Name: /par/modem/X/beacon/inf/id2 (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

BEACON Power Board MCU Firmware Version

Name: /par/modem/X/beacon/inf/mcu2 (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

BEACON Serial Number

Name: /par/modem/X/beacon/inf/sn (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

BEACON Hardware Revision

Name: /par/modem/X/beacon/inf/hw (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

BEACON Firmware Version

Name: /par/modem/X/beacon/inf/sw (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

BEACON Boot-loader Version

Name: /par/modem/X/beacon/inf/bl (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

BEACON Received Signal Strength Indication

Name: /par/modem/X/beacon/inf/link/rssi (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

BEACON Synchronization Status

Name: /par/modem/X/beacon/inf/link/sync (X=[a…d])
Access: r
Type: enumerated
Values: 0,1,unknown
Default: unknown

0 - no synchronization.

1 - synchronization established.

BEACON Current Channel Frequency

Name: /par/modem/X/beacon/inf/link/freq (X=[a…d])
Access: r
Type: string[0…32]
Default: "unknown"

BEACON Tuned Frequency Offset

Name: `/par/modem/X/beacon/inf/link/freqoffset (X=[a…d])`
Access: `r`
Type: `string[0…32]`
Default: `"unknown"`

BEACON Current Symbol Rate

Name: `/par/modem/X/beacon/inf/link/symrate (X=[a…d])`
Access: `r`
Type: `string[0…32]`
Default: `"unknown"`

Beacon Scanning Mode

Name: `/par/modem/X/beacon/scan/mode (X=[a…d])`
Access: `rw`
Type: `enumerated`
Values: `manual,auto`
Default: `manual`

`manual` - set by link chan, link sigtype

`auto` - automatic scannig

Beacon Frequency Criterion

Name: `/par/modem/X/beacon/scan/freq (X=[a…d])`
Access: `rw`
Type: `enumerated`
Values: `full,map`
Default: `full`

`full` - full range with 500 Hz step

`map` - channel map scanning

Beacon Scanning Criterion

Name: `/par/modem/X/beacon/scan/sigtype (X=[a…d])`
Access: `rw`
Type: `enumerated [bps]`
Values: `100, 200, 50, all`
Default: `100`

`100,200,50` - scan specific signal type

`all` - scan all of the above

4.4.31 Bluetooth Parameters

Bluetooth Mode

Name: /par/blt/mode
Access: rw
Type: enumerated
Values: off,on
Default: on

off - Bluetooth is turned off

on - Bluetooth is turned on

Bluetooth Reset

Name: /par/blt/reset
Access: w
Type: boolean
Values: yes,no
Default: no

yes - reset Bluetooth module and Bluetooth stack

no - ignored

Bluetooth Automatic Name Generation

Name: /par/blt/nauto
Access: rw
Type: enumerated
Values: on,off
Default: on

on - use automatically generated name

off - use the value of the parameter /par/blt/name

Automatically generated name format is:

```
MODEL SN (ID1,ID2,ID3)
```

where:

MODEL - receiver model

SN - receiver serial number

ID1 - internal modem product ID

ID2 - external modem product ID

ID3 - product ID of internal radio module of external modem

Bluetooth Name

Name: /par/blt/name
Access: rw
Type: string[0…32]
Values: (any string)
Default: "MODEL SN"

where:

MODEL - receiver model

SN - receiver serial number

User-defined Bluetooth name. Only active when /par/blt/nauto is off.

Bluetooth PIN

Name: /par/blt/pin
Access: rw
Type: string [0…4]
Values: (any string)
Default: 1234

Bluetooth Enable PIN Request

Name: /par/blt/pinreq
Access: rw
Type: boolean
Values: on,off
Default: off

Remote Bluetooth Device Address

Name: /par/dev/blt/X/chan/addr (X=[a,b])
Access: rw
Type: string[12]
Values: (any string)
Default: "000000000000"

Address of remote Bluetooth device that allowed to connect to this port. If address is set to 000000000000, then any Bluetooth device can connect to this port.

Current Remote Bluetooth Device Address

Name: /par/dev/blt/X/chan/curaddr (X=[a,b])
Access: r
Type: string[0…12]
Values: (any string)
Default: "unknown"

Address of remote Bluetooth device that is currently connected to this port.

Current Remote Bluetooth Device RSSI

Current Remote Bluetooth Device RSSI

Name: /par/dev/blt/X/chan/rssi (X=[a,b])
Access: r
Type: string[0…15]
Values: (RSSI in dB)
Default: "unknown"

RSSI value of Bluetooth connection for this port, measured in dB, relative to Golden Device Power Range.

4.4.32 Advanced Power Management

Primary Control Points

External Power Voltage

Name: /par/pwr/ext
Access: r
Type: float [volts]

Receiver Board Voltage

Name: /par/pwr/board
Access: r
Type: float [volts]

This voltage is measured directly on the board excluding the voltage drop across the "power switching circuitry".

External Antenna Control Points

External Antenna Voltage

Name: /par/pwr/extant
Access: r
Type: float [volts]

Provided /par/ant/curinp parameter's value is ext, this parameter contains the voltage of the antenna power supply.

External Antenna Current

Name: /par/pwr/extantdc
Access: r
Type: float [milliamperes]

Provided /par/ant/curinp parameter's value is ext, this parameter contains the DC the antenna draws from the antenna power supply.

Batteries Status and Charging

Battery Parameters

Battery Power

Name: /par/pwr/cell/pwr
Access: r
Type: float [watts]

If the value is negative, the battery is being discharged, otherwise it is being charged

Battery Voltage

Name: /par/pwr/cell/[a|b]/v
Access: r
Type: float [volts]

Battery DC

Name: /par/pwr/cell/[a|b]/dc
Access: r
Type: integer [milliamperes]

If the value is negative then the current flows from battery to system else the current flows from charger to battery.

Battery Charge Level

Name: /par/pwr/cell/[a|b]/soc
Access: r
Type: float [percents]
Values: [0…100]

Battery Temperature

Name: /par/pwr/cell/[a|b]/thermo
Access: r
Type: float [Celsius]
Values: [-100…200]

Temperature inside of the battery pack

Battery Full Capacity

Name: /par/pwr/cell/[a|b]/fullcap
Access: r
Type: integer [mAh]

Battery Time To End

Name: /par/pwr/cell/[a|b]/tte
Access: r
Type: string [0…32]

Time to end in hours and minutes, for example, 21h04m

Battery Power-off Level

Name: /par/pwr/cell/level/off
Access: rw
Type: float [volts]
Values: [0…7]
Default: 6.5

This parameter specifies voltage level of battery at which rcv will be turned off

Charger Parameters

Charger Mode

Name: /par/pwr/charge/mode
Access: rw
Type: enumerated
Values: auto,off
Default: auto

`auto` - automatic charging mode

`off` - charging off

Charger Voltage

Name: `/par/pwr/charge/v`
Access: `r`
Type: `float [volts]`

Charger DC

Name: `/par/pwr/charge/dc`
Access: `r`
Type: `integer [milliamperes]`

Charger Power

Name: `/par/pwr/charge/pwr`
Access: `r`
Type: `float [watts]`

Charger State

Name: `/par/pwr/charge/[a|b]/state`
Access: `r`
Type: `enumerated`
Values: `off,error,susp,charging,charged`

`off` - charging is off

`error` - charging failure

`susp` - charging suspended

`charging` - charging in progress

`charged` - charging complete

Charger HW Status

Note: For `TRIUMPH_1MP rev.2` and above only.

Name: `/par/pwr/charge/[a|b]/stat`
Access: `r`
Type: `integer`
Values: `0,1`

`0` - external power less than `8V`, or malfunction

`1` - operation

Fuel Gauge Parameters

Fuel Gauge Connection Status

Name: `/par/pwr/fgauge/[a|b]/conn`
Access: `r`
Type: `boolean`
Values: `y,n`

`y` – battery connected to the gauge

`n` – no battery connected to the gauge

Fuel Gauge Battery Voltage

Name: `/par/pwr/fgauge/[a|b]/v`
Access: `r`
Type: `float [volts]`

Fuel Gauge Battery DC

Name: `/par/pwr/fgauge/[a|b]/dc`
Access: `r`
Type: `integer [milliamperes]`

Negative values indicate power being drawn from the battery, positive – battery being charged.

Fuel Gauge Battery Charging State 1 and 2

Name: `/par/pwr/fgauge/[a|b]/stat1`
Access: `r`
Type: `integer`
Values: `0,1`

Name: `/par/pwr/fgauge/[a|b]/stat2`
Access: `r`
Type: `integer`
Values: `0,1`

The meaning of these two bits is hardware-dependent, and makes sense only when external power is applied.

Fuel Gauge Charging Alert

Name: /par/pwr/fgauge/[a|b]/alert
Access: r
Type: integer
Values: 0,1

0 - no alert

1 - charging alert raised

Fuel Gauge Charger Override Flag

Name: /par/pwr/fgauge/a/override
Access: r
Type: integer
Values: 0,1

0 - charger override logic is off

1 - charger override logic is in progress

Charger override logic is used to force charging when fuel gague is not functional due to lack or deep discharge of the battery.

Fuel Gauge Force Init

Name: /par/pwr/fgauge/init
Access: w
Type: boolean
Values: y,n
Default: n

y - fuel gauge will be initialized

n - ignored

This pseudo-parameter is for internal use. Doesn't work for Triumph2.

External Power Output

Enable Power Output

Name: /par/pwr/out/ab
Access: rw
Type: boolean
Values: on,off
Default: off

on - power will be output onto external connectors

off - there will be no power on external connectors

Output Voltage

Name: /par/pwr/extports
Access: r
Type: float [volts]

This parameter contains the voltage of the power being output onto external connectors.

Secondary Control Points

Digital Part 3 Volt Power

Name: /par/pwr/d3v
Access: r
Type: float [volts]

Modem Control Points

GSM Modem Voltage

Name: /par/pwr/mdm/gpwr
Access: r
Type: float [volts]

Radio Modem Voltage

Name: /par/pwr/mdm/mpwr
Access: r
Type: float [volts]

Radio Modem Current

Name: /par/pwr/mdm/mdc
Access: r
Type: float [milliamperes]

GSM Modem Current

Name: /par/pwr/mdm/gdc
Access: r
Type: float [milliamperes]

Second Radio Modem Current

Name: /par/pwr/mdm/mdc2
Access: r
Type: float [milliamperes]

3V3 Modem Voltage

Name: /par/pwr/mdm/3v3
Access: r
Type: float [volts]

Temperature Control Points

RF Temperature

Name: /par/pwr/temp/rf
Access: r
Type: float [Celsius]
Values: [-40…+125]

Digital/ASIC Temperature

Name: /par/pwr/temp/dig
Access: r
Type: float [Celsius]
Values: [-40…+125]

CPU Temperature

Name: /par/pwr/temp/cpu
Access: r
Type: float [Celsius]
Values: [-40…+125]

RF (NT1066) Chip Temperature

Name: /par/pwr/temp/nt/[a|b]
Access: r
Type: float [Celsius]
Values: [-40…+125]

Battery Cell Temperature

Name: /par/pwr/temp/cell/[a|b]
Access: r
Type: float [Celsius]
Values: [-40…+125]

RTC Temperature

Name: /par/pwr/temp/rtc
Access: r
Type: float [Celsius]
Values: [-40…+125]

OCXO Temperature

Name: /par/pwr/temp/ocxo
Access: r
Type: float [Celsius]
Values: [-40…+125]

Temperature near the Oven Controlled Crystal Oscillator.

IMU Temperature

Name: /par/pwr/temp/imu
Access: r
Type: float [Celsius]
Values: [-40…+105]

Modems Temperature

Name: /par/pwr/temp/mdm
Access: r
Type: float [Celsius]
Values: [-40…+125]

4.4.33 TriPad Parameters

The following parameters allow the user to set/query the receiver configuration data responsible for the TriPad FN button's functionality.

Appending data to a specific file

Name: /par/button/file
Access: rw
Type: string[20]
Default: (empty string)

This parameter instructs the receiver to append new data to a specific existing file (unless the receiver finds no file with this name) when starting data recording via the FN button. This parameter can be set to a string comprising up to 20 valid characters. This string designates the name of the file you have selected for data appending.

If you have specified an empty name, the receiver will assign the current log-file an "automatically created name" every time you use TriPad to start data recording. (Note that this automatically created file name will depend on both the file creation time (month and day) and some additional "letter suffices". The latter are used to avoid confusion between files created on the same day).

Alternatively, suppose you have specified a non-empty file name, say NAME. If there is no log-file with this name in the receiver memory, pushing the FN key will instruct the receiver to create a new file named /log/NAME. Otherwise, the receiver will be using the existing file /log/NAME for appending new data.

TriPad <FN> Button Click Action

Name: /par/button/action
Access: rw
Type: enumerated
Values: led,dyn,tcpcl
Default: led

This allows the user to specify the <FN> button short click[1] functionality .

led - <FN> button click will do nothing (this is historical value).

dyn - <FN> button click will toggle between the static and dynamic receiver modes, provided data recording is active. Every time dynamic mode is changed, receiver will output an appropriate free form event to the current log-file.

tcpcl - <FN> button short click will suspend/resume TCP clients (See "Suspend TCP Clients" on page 475.).

When data recording is active, you can easily distinguish between static and dynamic visually. If the <REC> LED blinks green, the current mode is dynamic, if it blinks yellow, the mode is static.

Initial Dynamic Mode

Name: /par/button/dyn
Access: rw
Type: enumerated
Values: static,dynamic
Default: static

When /par/button/action is set to dyn, this parameter will specify the initial dynamic mode for all of the new files opened through TriPad.

1. Press the button and release it in less than one second.

Toggle Automatic File Rotation Mode (AFRM) via TriPad

Name: /par/button/rot
Access: rw
Type: boolean
Values: on,off
Default: off

off - TriPad <FN> button will turn data logging on and off.

on - TriPad <FN> button will turn AFRM on and off.

Turn Data Recording On at Startup

Name: /par/button/auto
Access: rw
Type: enumerated
Values: on,off,always
Default: off

on - should a power failure occur in the course of data recording, the receiver will then automatically open a new file and resume data recording when power is on again. From a functional point of view, this is equivalent to pushing the <FN> button to start data logging once the receiver is powered on again.

always - this case is similar to the previous one except that the auto-start mechanism will be launched at receiver start time irrespective of whether the power failure occurred while data recording or not.

off - receiver will not resume data logging after power failure.

Note: Setting this parameter to either on or always will not make the receiver itself automatically start when power is restored after a power failure, though recent receivers will remember their on or off status, and therefore will turn on when power is restored, provided power failure occurred when they were turned on.

Position Calculated Externally

Name: /par/button/fine
Access: rw
Type: integer [seconds]
Values: [0…300]
Default: 0

When this pseudo-parameter is set to non-zero value, <POS> or <SAT> LED will start to blink specially to indicate to the user that fine position has been calculated externally for this unit. This special blinking will continue until specified number of seconds expires, or until <FN> button is pressed.

When printed, this parameter always returns value 0.

4.4.34 CAN Ports Parameters

In this section, [cport] denotes a CAN port - any of `/dev/can/X (X=[a,b])`.

CAN messages that receiver accepts are specified by the two parameters: `/par/[cport]/sid/in/first` and `/par/[cport]/sid/in/cnt`. To be accepted by the receiver, the CAN message Standard Identifiers (SIDs) of the input CAN messages must be in the range [`first…first+cnt-1`]. In addition, receiver will use the received SIDs to establish relative order of received the CAN messages.

CAN messages that receiver generates have programmable SIDs. The SID starts with the value specified by the `/par/[cport]/sid/out/first` parameter and is incremented by one for every CAN message being output until number of SIDs in the sequence exceeds the value of parameter `/par/[cport]/sid/out/cnt`. Then the SID is reset to its first value and the process continues. For example, if `first` is set to 0x710 and `cnt` is set to 3, the output CAN messages will have the following SIDs:

```
0x710,0x711,0x712,0x710,0x711,…
```

CAN Baud Rate

Name: `/par/[cport]/rate`
Access: `rw`
Type: `integer [kbit/s]`
Values: `1000, 500, 250, 125`
Default: `125`

CAN bus baud rate in kilobits per second.

First SID for Input CAN Messages

Name: `/par/[cport]/sid/in/first`
Access: `rw`
Type: `integer`
Values: `[0x000…0x7FF]`
Default: `0x700`

The Number of SIDs for Input CAN Messages.

Name: `/par/[cport]/sid/in/cnt`
Access: `rw`
Type: `integer`
Values: `[1…8]`
Default: `8`

First SID for Output CAN Messages.

Name: /par/[cport]/sid/out/first
Name: rw
Type: integer
Values: [0x000…0x7FF]
Default: 0x700

The Number of SIDs for Output CAN Messages.

Name: /par/[cport]/sid/out/cnt
Access: rw
Type: integer
Values: [1…8]
Default: 8

4.4.35 IRIG Modulator Parameters

Enable IRIG Signal Output

Name: /par/dev/irig/out
Access: rw
Type: boolean
Values: on,off
Default: off

IRIG Reference Time

Name: /par/dev/irig/time
Access: rw
Type: enumerated
Values: utcusno,gps
Default: utcusno

IRIG Signal Offset

Name: /par/dev/irig/offs
Access: rw
Type: integer [ns]
Values: [-500000…500000]
Default: 0

This parameter specifies IRIG signal offset in nanoseconds. Positive value will delay the signal with respect to the reference time.

IRIG Signal Amplitude

Name: /par/dev/irig/ampl
Access: rw
Type: integer
Values: [0…255]
Default: 170

IRIG Time Code Format

Name: /par/dev/irig/format
Access: rw
Type: enumerated
Values: a,b
Default: b

IRIG Code Control Function Bits

Name: /par/dev/irig/cntrl
Access: rw
Type: integer
Values: [0…0x3ffff]
Default: 0

This parameter contains 18 bits of IRIG code control function.

4.4.36 GPIO Parameters

Receivers may have general-purpose IO pins. The parameters in this section allow user to configure and use these pins. The number of available pins depends on particular board type and is denoted below by letter "P".

GPIO Pins Output

Name: /par/dev/gpio/out
Access: rw
Type: array {0…P} of boolean
Values: {y|n,y|n,…,y|n}
Default: {n,n,…,n}

n – corresponding pin is configured as input pin.

y – corresponding pin is configured as output pin.

GPIO Pin #N Output

Name: /par/dev/gpio/out/N N=[0…P]
Access: rw
Type: boolean
Values: y,n
Default: n

n - pin number N is configured as input pin.

y - pin number N is configured as output pin.

GPIO Pins Value

Name: /par/dev/gpio/value
Access: rw
Type: array {0…P} of boolean
Values: {y|n,y|n,…,y|n}
Default: {n,n,…,n}

This parameter contains voltage levels of corresponding pins.

n - low voltage level on corresponding pin.

y - high level on corresponding pin.

For pins configured as input, the value of corresponding element on setting is ignored, and printing returns current state of externally applied voltage. For pins configured as output, the setting will drive output voltage to corresponding state.

GPIO Pin #N Value

Name: /par/dev/gpio/value/N N=[0…P]
Access: rw
Type: boolean
Values: y,n
Default: n

This parameter contains voltage level of pin number P.

n - voltage level is low.

y - voltage level is high.

When pin configured as input, setting of this parameter is ignored, and the level of external voltage applied to this pin is displayed on print. When pin configured as output, setting this parameter will drive voltage output of the pin according to the new value.

4.4.37 Spectrum Parameters

To receive spectrum data from the receiver, first enable either `[sp]` or `[sP]` GREIS message, then set `/par/spectr/out` parameter to `y`.

Note: Please do not change `/par/spectr/coher_aver`, `/par/spectr/frq_step` and `/par/spectr/resampl` parameters from their default values if you are not an expert in the spectrum stuff as some combinations of these parameters may give unexpected results.

Note: Some of parameters in this section are available only for some receivers. Use `list,/par/spectr` to get the actual list for your receiver/firmware version.

The `N` in the descriptions of parameters below denotes the number of supported RF bands for given receiver.

Enable Spectrum Acquisition

Name: `/par/spectr/out`
Access: `w`
Type: `enumerated`
Values: `n,y,always`
Default: `n`

`y` - (re)start run-once acquisition of data for spectrum output.

`always` - enable permanent acquisition of data for spectrum output.

Enable Spectrum Acquisition for Spectrum #N

Name: `/par/spectr/out/N, N=[0…K], K - firmware dependent`
Access: `rw`
Type: `enumerated`
Values: `off,on,always`
Default: `off`

`on` - (re)start run-once acquisition of data for spectrum #N

`always` - enable permanent acquisition of data for spectrum #N

Time Interval of Coherent Spectrum Data Accumulation

Name: `/par/spectr/coher_aver`
Access: `rw`
Type: `integer [s]`
Values: `[1…20]`
Default: `1`

Time Interval of Incoherent Spectrum Data Accumulation

Name: /par/spectr/n_aver
Access: rw
Type: integer [s]
Values: [1…100]
Default: 1

Spectrum Frequency Step

Name: /par/spectr/frq_step
Access: rw
Type: integer [Hz]
Values: [1…10000]
Default: 1000

Spectrum Re-sampling Number

Name: /par/spectr/resampl
Access: rw
Type: integer
Values: [1…10]
Default: 10

Internally, spectrum data are computed with /par/spectr/frq_step interval. Then, the mean, maximum, or minimum (depending on /par/spectr/m_aver) of all the /par/spectr/resampl resulting values is taken to give the final values for [sp] and [sP] messages. Therefore, the frequency step in [sp] and [sP] messages is equal to the multiple of /par/spectr/frq_step and /par/spectr/resampl values.

Spectrum Averaging Mode

Name: /par/spectr/m_aver
Access: rw
Type: enumerated
Values: mean,min,max
Default: mean

Spectrum Mean Values

Name: /par/spectr/mean
Access: r
Type: array [0…N] of float [dB]
Values: {[0…127],…}

This parameter contains mean spectrum value for each of RF band for the time interval the spectrum has been measured over.

Mean AGC Values for Spectrum

Name: /par/spectr/agc/mean
Access: r
Type: array [0…N] of float
Values: {[0…127],…}

This parameter contains mean value of AGC for each of RF band for the time interval the spectrum has been measured over.

RMS of AGC Values for Spectrum

Name: /par/spectr/agc/rms
Access: r
Type: array [0…N] of float
Values: {[0…127],…}

This parameter contains RMS value of AGC for each of RF band for the time interval the spectrum has been measured over.

Names of Spectrum Bands

Name: /par/spectr/name
Access: r
Type: array [0…N] of enumerated
Values: {gps1|gps2|gps5|glo1|glo2|glo3,…}

Array of names of spectrum bands for each RF band.

Spectrum First Points

Name: /par/spectr/frq0
Access: r
Type: array [0…N] of float
Values: {[-1600…1600],…}
Default:

Frequency of the first (left) point of spectrum data for each RF band.

If the value is negative, frequency of the first (left) point of corresponding spectrum is equal to its absolute (positive) value, but frequency has descending order.

Spectrum/AGC Calibration Parameters

Name: /par/spectr/calib
Access: rw
Type: array [0…N] of integer
Values: {[-1000000000…+1000000000],…}
Default: {0,…}

These parameters are not used by receiver. External software may read/write them for its own purpose. These parameters are preserved over clear NVRAM and parameters initialization procedures.

Spectrum/AGC Calibration Parameters for External Antenna

Name: `/par/spectr/calibext`
Access: `rw`
Type: `array [0…N] of integer`
Values: `{[-1000000000…+1000000000],…}`
Default: `{0,…}`

These parameters are not used by receiver. External software may read/write them for its own purpose. These parameters are preserved over clear NVRAM and parameters initialization procedures.

Number of Points in Each Spectrum Message

Name: `/par/spectr/n_out`
Access: `rw`
Type: `integer`
Values: `[1…1000]`
Default: `100`

The number of points in each spectrum message. For high speed interfaces this number may be increased to get all the data faster.

This parameter could be absent for older receiver types.

L-band Spectrum Frequency Step

Name: `/par/spectr/lband_frq_step`
Access: `rw`
Type: `integer [Hz]`
Values: `[1…10000]`
Default: `1`

L-band spectrum frequency step. L-band spectrum is much narrower than other spectra, it has its own setting.

Spectrum Type

Name: `/par/spectr/log_point`
Access: `rw`
Type: `enumerated`
Values: `chan, filt, ajm`
Default: `filt`

`chan` - spectrum inside DSP channel processing

`filt` - spectrum after digital filter

`ajm` - spectrum after anti-jamming filter (if 'ajm' mode is turned on)

Current Spectrum Step

Name: `/par/spectr/curstep`
Access: `r`
Type: `float`
Values: `[0.1…100000]`
Default: -

After setting of spectrum parameters (and `/par/spectr/frq_step` in particular) receiver calculates effective spectrum step that, as number of samples must be power of 2, will usually end up being different from requested `frq_step`. Receiver selects the nearest possible value.

Knowledge of `curstep` is needed for making sense of the spectrum data. This value is also output in the `extData` field of the `[Sp]` message.

Current Spectrum Size

Name: `/par/spectr/cursize`
Access: `r`
Type: `integer`
Values: `[1…1000000]`
Default: -

After setting of spectrum parameters receiver calculates total number of points in resulting spectrum.

Knowledge of `cursize` is needed for making sense of the spectrum data. This value is also output in the `extData` field of the `[Sp]` message.

Antenna Spectrum Mask

Name: `/par/spectr/ant`
Access: `rw`
Type: `array [a,b,c,d] of boolean`
Values: `{y|n,y|n,y|n,y|n}`
Default: `{y,n,n,n}`

Each element of the array enables spectrum recording on corresponding antenna.

4.4.38 Magnetometer Parameters

Magnetometer Identifier

Name: /par/mag/id
Access: r
Type: enumerated
Values: MEMSIC48

Magnetometer Mode

Name: /par/mag/mode
Access: rw
Type: boolean
Values: off,on
Default: off

off - magnetometer is turned off

on - magnetometer is turned on

Magnetometer Update Rate

Name: /par/mag/msint
Access: rw
Type: integer [milliseconds]
Values: [5…1000], multiple of 5
Default: 100

Period in milliseconds of update rate of magnetometer measurements.

Magnetometer Reset

Name: /par/mag/reset
Access: rw
Type: boolean
Values: on,off
Default: off

on - magnetometer will be reset and the value of this parameter will be set back to off

off - ignored

Magnetometer Error Status

Name: /par/mag/stat/error
Access: r
Type: enumerated
Values: none,init,irq,bus
Default: none

none - no errors

init - error occurred during magnetometer initialization

irq - no interrupt requests from magnetometer

bus - error occurred while reading magnitudes from magnetometer

Magnetometer Raw Magnitude Measurements

Name: /par/mag/stat/mag/A, A=[x,y,z]
Access: r
Type: integer
Values: [-131072…131071]
Default: 0

A - X/Y/Z-axis magnitude output

4.4.39 IMU Parameters

IMU Identifier

Name: /par/imu/dev/id
Access: r
Type: enumerated
Values: ADIS16505

IMU Mode

Name: /par/imu/dev/mode
Access: rw
Type: boolean
Values: auto,off,on
Default: auto or off (receiver dependent)

on - IMU is turned on

off - IMU is turned off

auto - IMU is turned on/off automatically as needed, depending on other modes of operation.

IMU Update Rate

Name: /par/imu/dev/msint
Access: rw
Type: integer [milliseconds]
Values: [5…1000], multiple of 5
Default: 100 or 20

Period in milliseconds of update rate of IMU measurements.

IMU Filter Size

Name: /par/imu/dev/filt/B
Access: rw
Type: integer
Values: [0…6]
Default: 6

Control of the Bartlett window FIR filter. The number of taps in each stage is:

$N = 2^B$

IMU Monitor

Name: /par/imu/dev/monitor
Access: rw
Type: boolean
Values: on,off
Default: on

Monitor the state of the IMU and resets it if something goes wrong.

IMU Reset

Name: /par/imu/dev/reset
Access: rw
Type: boolean
Values: on,off
Default: off

on - IMU will be reset and the value of this parameter will be set back to off

off - ignored

IMU Reset Status

Name: /par/imu/dev/stat/reset
Access: r
Type: integer
Values: <non-negative integer>
Default: 0

Number of sensor restarts since IMU mode is turned `on`.

IMU Diag Status

Name: `/par/imu/dev/stat/diag`
Access: `r`
Type: `integer`
Values: `<hex integer>`
Default: `0`

IMU system status/error flag indicators formatted as hexadecimal.

IMU Temperature

Name: `/par/imu/dev/stat/temp`
Access: `r`
Type: `float [deg]`
Values: `[-40…105]`
Default: `0.0`

Temperature of the IMU unit.

IMU Error Status

Name: `/par/imu/dev/stat/error`
Access: `r`
Type: `enumerated`
Values: `none,init,irq,bus,checksum,diag`
Default: `none`

`none` - no errors

`init` - error occurred during IMU initialization

`irq` - no interrupt requests from IMU

`bus` - error occurred while reading raw measumerents from IMU

`checksum` - error in checking the checksum of raw measurements from IMU

`diag` - error of IMU indicator flag

IMU Raw Accelerometer Measurements

Name: `/par/imu/dev/stat/meas/accl/A, A=[x,y,z]`
Access: `r`
Type: `float [m/sec^2]`
Values: `[-78.3…78.3]`
Default: `0`

A - X/Y/Z-axis (accelerometer) output

IMU Raw Gyroscope Measurements

Name: /par/imu/dev/stat/meas/gyro/A, A=[x,y,z]
Access: r
Type: float [deg/sec]
Values: [-500…500]
Default: 0

A - X/Y/Z-axis (gyroscope) output

IMU Firmware Version

Name: /par/imu/ver
Access: r
Type: string

IMU Board Revision

Name: /par/imu/rev
Access: r
Type: string

4.4.40 Tilt-Compensated Position Parameters

Receivers equipped with built-in Inertial Measurement Unit (IMU), such as Triumph-3, are capable to provide position of the bottom end of the pole, so called *tilt-compensated* position.

To configure receiver to provide tilt-compensated position, in addition to general RTK configuration, use the following commands, assuming the start from default receiver settings:

⇒ set,/par/pos/pd/mode,extrap
⇒ set,/par/pole/height,<HEIGHT>
⇒ set,/par/pole/mode,on

Where:

<HEIGHT> = <pole length> + <adapter length> + <ARP to APC offset>,

in meters.

To turn this mode off, use:

⇒ set,/par/pole/mode,off

To get results of computation in this mode, use GREIS[pg] messasge ("[pg] Pole Tip Geodetic Position" on page 126), e.g.:

⇒ em,/dev/ser/a,/jps/msg/pg:0.1

Note: No other messages are affected by this mode, so one can gather, say, RTK coordinates of APC simultaneously.

In addition, before starting survey/stakeout, it is necessary to provide enough motion for the system to complete initialization. Initialization needs a few seconds of receiver being stationary, and then sufficient horizontal motion of the antenna, where horizontal velocity changes at least from 0 to 1 [m/s]. Here is the suggested procedure:

1. After configuring receiver as described above (or turning ON already configured receiver), set receiver approximately vertically on a point and wait for RTK fix, so that /par/pole/err has no other values than ins_init or conv (or error field of [pg] message has no other bits than 0x10 or 0x20).
2. Keeping the end of the pole stationary on the point, tilt or swing receiver actively but smoothly in different directions, until /par/pole/stat becomes ok ([pg] error becomes 0 and position appears).
3. Receiver is producing pole tip position and is ready for continuous operation.

Warning: *try to avoid jerks during operation. They may cause loss of initialization and the need to repeat initialization procedure.*

The following parameters are defined to govern this mode of operation.

Tilt Compensation Mode

Name: /par/pole/mode
Access: rw
Type: boolean
Values: off,on
Default: off

Tilt Compensation Pole Length

Name: /par/pole/height
Access: rw
Type: float [meters]
Values: [-100…100]
Default: 0

Offset of the pole tip measured from antenna phase center (APC).

Tilt Compensation Status

Name: /par/pole/stat
Access: r
Type: enumerated
Values: off,ok,fail
Default: off

off - mode is off

ok - receiver provides tilt-compensated solution

fail - no tilt-compensated solution: check /par/pole/err for the cause(s)

Tilt Compensation Error

Name: /par/pole/err
Access: r
Type: list of enumerated
Values: <see below>
Default: {}

Comma-separated list of raised warnings, where each element is one of:

Value	Description
imu_meas	No data from IMU
sat_meas	No GNSS measurements: tracking issue
pos	No GNSS position: no RTK fix
vel	Failure to compute GNSS velocity
ins_init	Not enough motion for proper initialization
conv	Solution not yet converged

4.4.41 Messages

Message Groups

There are several message groups supported by the receiver. Each group comprises several related messages. Each entry in a group specifies default scheduling parameters for corresponding message. In addition to using message groups and message names in the list and print commands, individual message names could be used in the em, out, and dm commands.

Example: List the names of all the supported NMEA messages:

```
⇒ list,/msg/nmea
⇐ RE03F{GGA,GBS,GLL,GMP,GNS,GRS,GSA,GST,GSV,HDT,RMC,ROT,VTG,ZDA,P_ATT}
```

Example: Print default scheduling parameters of the GREIS [GA] message:

```
⇒ print,/msg/jps/GA
⇐ RE011{0.00,0.00,0,0x2}
```

Example: Enable output of the NMEA GGA message into the current terminal using default scheduling parameters:

```
⇒ em,/cur/term,/msg/nmea/GGA
```

●

GREIS Message NAME

Name: /msg/jps/NAME
Access: r
Type: sched_params

This parameter contains default scheduling parameters for GREIS message NAME. In general, message NAME matches the two-letter message identifier (see "Standard Messages" on page 64). However, for messages that have non-alphanumeric identifiers, the names used differ from their identifiers (see "Standard Predefined Messages" on page 67).

NMEA Message NAME

Name: /msg/nmea/NAME
Access: r
Type: sched_params

This parameter contains default scheduling parameters for NMEA message NAME. Standard NMEA messages are called after their three-letters identifiers (e.g., GGA). Proprietary NMEA messages are called by their three-letters identifiers and using P_ prefix (e.g., P_ATT).

RTCM 2.x Message NAME

Name: /msg/rtcm/NAME
Access: r
Type: sched_params

This parameter contains default scheduling parameters for RTCM 2.x message NAME. RTCM messages are called after their decimal identifiers.

RTCM 3.x Message NAME

Name: /msg/rtcm3/NAME
Access: r
Type: sched_params

This parameter contains default scheduling parameters for RTCM 3.x message NAME. RTCM 3.x messages are called after their decimal identifiers.

CMR Message NAME

Name: /msg/cmr/NAME
Access: r
Type: sched_params

This parameter contains default scheduling parameters for CMR message NAME. CMR messages are called after their decimal identifiers.

Message Sets

The main purpose of supporting message sets is to provide ability to enable output of multiple messages and specify their scheduling parameters using single object name, the name of a message set.

Unlike message groups, message sets may contain "unrelated" messages, i.e., messages taken from different message groups. To avoid name clashes, the entries in the message sets have names comprising both message group name and message name inside its group.

Also unlike message groups, message sets are customizable. You may use em and dm commands on message sets the same way you use them on output streams. You may also add and remove messages to/from message sets, and you may change scheduling parameters of the messages in the message sets using remove, create, and set commands, but em and dm are usually more convenient.

Note that the contents of a message set is only relevant at the moment of enabling the output of the message(s) from the message set, and has no impact on the currently enabled messages.

Below are some examples of using the message sets. Refer to description of corresponding commands for details and more examples.

Example: Remove GREIS [EL] message from the default set of messages:

⇒ remove,/msg/def/jps/EL

or, the same thing, using dm

⇒ dm,/msg/def,/msg/jps/EL

Example: Remove all the messages from the default set of messages:

⇒ remove,/msg/def/

or

⇒ dm,/msg/def

Example: Reconfigure the default set of messages to contain only NMEA GGA and ZDA messages:

⇒ remove,/msg/def/
⇒ create,/msg/def/nmea/GGA
⇒ create,/msg/def/nmea/ZDA

or, using em/dm:

⇒ dm,/msg/def
⇒ em,/msg/def,/msg/nmea/{GGA,ZDA}

Example: Change scheduling parameters for GREIS [SI] message in the default set of messages:

⇒ set,/msg/def/jps/SI,{1,0,0,0x2}

or

⇒ em,/msg/def,/msg/jps/SI:{1,0,0,0x2}

Note that if [SI] is not in the default message set, the former will cause error reply, while the latter one will add [SI] to the end of the default set of messages, and will set specified scheduling parameters.

Example: Add GREIS [SI] message to the default set of messages with specific scheduling parameters:

⇒ create,/msg/def/jps/SI
⇒ set,/msg/def/jps/SI,{1,0,0,0x2}

or, using em:

⇒ em,/msg/def,/msg/jps/SI:{1,0,0,0x2}

Example: Enable all the messages currently in the default set of messages to be output to the current terminal using scheduling parameters specified for the messages in the default set:

⇒ em,/cur/term,/msg/def

Example: Restore the default value for the default set of messages:

⇒ init,/msg/def

Default Set of Messages

Name: `/msg/def`
Access: `rw`
Type: `list {sched_params,…,sched_params}`
Default: `(receiver dependent)`

This parameter contains the default set of messages. The default value of this parameter is designed to contain GREIS messages suitable for wide range of post-processing software.

This message set is implicitly used by default when receiver log-files are created using TriPad or through AFRM. You can change what message set will be used instead of default one using `/par/log/[a…e]/msgs` parameter described on page 399.

Warning: *Remember that some of the message types are critical for JAVAD GNSS post-processing software to be able to import and process GREIS files correctly. Care should be taken when customizing the default set of messages.*

Minimum Set of GREIS Messages for RTK

Name: `/msg/rtk/jps/min`
Access: `rw`
Type: `list {sched_params,…,sched_params}`
Default: `(receiver dependent)`

This parameter contains minimum required set of GREIS messages suitable for RTK.

Maximum Set of GREIS Messages for RTK

Name: `/msg/rtk/jps/max`
Access: `rw`
Type: `list {sched_params,…,sched_params}`
Default: `(receiver dependent)`

This parameter contains maximum set of GREIS messages suitable for RTK.

User Sets of Messages

Name: `/msg/usr/N [N=0,1]`
Access: `rw`
Type: `list {sched_params,…,sched_params}`
Default: `(empty)`

These parameters contain user sets of messages.

Message Output Lists

For every output port, there is corresponding message output list that holds the messages being enabled to be output to this port along with their current scheduling parameters.

Similar to message sets, message output lists may contain "unrelated" messages, i.e., messages taken from different message groups. To avoid name clashes, the entries in the message output lists have names comprising both message group name and message name inside its group.

The contents of these output lists are implicitly modified by the `em`, `out`, and `dm` commands.

Unlike message sets, there is no way to use message lists as the second parameter of `em` and `dm` commands, nor could they be modified by `remove`, `create`, or `set` commands.

Message Output List for a Port

Name: `/par/out/[oport]`
Access: `r`
Type: `list {sched_params,…,sched_params}`
Default: `{}`

This parameter contains the list of messages enabled for output to the `[oport]` along with their scheduling parameters. Messages are output in the same order in which they appear in the message output list.

Number of Messages Enabled for Output to a Port

Name: `/par/out/[oport]&count`
Access: `r`
Type: `integer`
Values: `[0…49]`
Default: `0`

This parameter contains number of messages currently being enabled to be output to the `[oport]`.

4.4.42 Condition Indication Mode Parameters

Condition indication feature allows external software to use one of receiver LEDs to visually communicate some condition to receiver operator.

When the mode is turned on, condition will be indicated according to the value of `/par/ind/state` parameter. The value of `/par/ind/state` is in turn governed by setting of `/par/ind/v` parameter and the `/par/ind/timeout` parameter.

Condition Indication Mode

Name: /par/ind/mode
Access: rw
Type: enumerated
Values: off,bt,wf
Default: off

off - turn indication off

bt - use Bluetooth LED for condition indication

wf - use WiFi LED for condition indication

Condition Indication Value

Name: /par/ind/v
Access: rw
Type: boolean
Values: y,n
Default: n

y - turn /par/ind/state to 'on'

n - turn /par/ind/state to 'off'

This parameter is not stored in NVRAM.

Condition Indication Timeout

Name: /par/ind/timeout
Access: rw
Type: integer [seconds]
Values: [0…0x7FFFFFFF]
Default: 10

Once time passed since the latest setting of /par/ind/v parameter exceeds the value of this parameter, the /par/ind/state will be reset to 'none'. When set to 0, timeout will never trigger.

Condition Indication State

Name: /par/ind/state
Access: r
Type: enumerated
Values: none,on,off
Default: none

none - corresponding LED will be yellow.

on - corresponding LED will be green.

out - corresponding LED will be red.

This parameter is not stored in NVRAM and is set to `'none'` at receiver startup.

4.4.43 Miscellaneous parameters

Processor's Clock Frequency

Name: `/par/cpu/frq`
Access: `r`
Type: `integer [MHz]`

Returns the processor's clock frequency.

DSP Sampling Frequency

Name: `/par/asic/curfrq`
Access: `rw`
Type: `integer [MHz]`
Values: `[40,50,60,70,80]`
Default: `50 or 60, depending on board model`

This parameter holds the value that will be set as current DSP sampling frequency (see `/par/asic/curfrq` below) after receiver reboot. User does not need to change this parameter except for very specific timing applications. Refer to "*Receiver Clock Synchronizing onfiguration Example*" on our WWW site for details.

DSP Current Sampling Frequency

Name: `/par/asic/curfrq`
Access: `r`
Type: `integer [MHz]`
Values: `[40,50,60,70,80]`

Processor Load Statistics

Name: `/par/load`
Access: `r`
Type: `list`

This parameter contains statistics of processor load gathered since last request of the value of this parameter.

All the elements of the list but the last one have the format:

```
{NAME,load,min,max}
```

where:

`NAME` – thread name

`load` - computed processor load (in percents) associated with this thread

`min` - computed minimum time (in milliseconds) per thread execution cycle

`max` - computed maximum time (in milliseconds) per thread execution cycle

The last element of the list is of special interest to the user. It has the following format:

```
{load,soft_err,hard_err}
```

where:

`load` - average processor load in percents

`soft_err` - number of detected soft real-time errors

`hard_err` - number of detected hard real-time errors

The average processor load is supposed to be less than 90, and the number of detected hard real-time errors must be zero.

Reboot on Exception Mode on Errors

Name: `/par/except`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`off` - receiver will enter "exception mode" if an unrecoverable error in program execution is detected.

`on` - receiver will reboot itself and continue running when an unrecoverable error in program execution is detected. Here "reset" means a hardware reset similar to power cycle.

Receiver Board Temperature

Name: `/par/dev/thermo/out`
Access: `r`
Type: `float [Celsius degrees]`

Communication Board Temperature

Name: `/par/dev/thermo/mdm`
Access: `r`
Type: `float [Celsius degrees]`

Size of the Receiver's NVRAM

Name: `/par/dev/nvm/a/size`
Access: `r`
Type: `integer [bytes]`

Free Space in the Receiver's NVRAM

Name: `/par/dev/nvm/a/free`
Access: `r`
Type: `integer [bytes]`

Time From the Receiver's Battery-Backed RTC

Name: `/par/dev/rtc/time`
Access: `r`
Type: `{sec,min,hour,day,month,year}`

Memory Allocation Statistics

Name: `/par/stat/mem`
Access: `r`
Type: `list {sz,sf,min,bf,max,bu,su,ac,fc,ls,ms}`

This parameter describes memory allocation statistics for the main memory pool. This parameter is intended for the JAVAD GNSS's firmware developers and is subject to change at any time.

`sz` – pool size in bytes

`sf` – free memory

`min` – minimum free memory

`bf` – number of free blocks

`max` – maximum number of free blocks

`bu` – number of blocks used

`su` – memory currently in use

`ac` – number of allocations

`fc` – number of deallocations (frees)

`ls` – number of iterations in the longest block search

`ms` – mean number of block search iterations

4.4.44 Receiver Options

Options Overview

Among the many capabilities of your JAVAD GNSS receiver there is a special class of capabilities which are referred to as “receiver options”. By default, receiver options are disabled so you have to take special measures to activate them. It can be done by uploading an Option Authorization File (OAF) to the receiver using desktop receiver control software.

Each option is characterized by the following descriptors:

- Option name
- Purchased value
- Leased value
- Expiration date of leased value
- Current value

A receiver option can be “purchased”, “leased”, or “purchased” and “leased” at the same time. A leased option is characterized by the leased value and the expiration date.

A purchased option is characterized only by the purchased value because such an option has no expiration date. Since any option can be “purchased” and “leased” at the same time, we should take into account all of the “purchased” and “leased” descriptors as a whole. It is the numerical descriptor “current” that serves this purpose. This descriptor indicates the value currently effective for the given option.

Note that “current” will either coincide with the larger of the purchased and leased values or be set to -1.

- If “current” equals -1, this means that the corresponding receiver option is not supported by the firmware version you use.
- If “current” equals zero, the corresponding receiver option is disabled.
- If “current” equals a positive integer, the option is enabled.

Options Parameters

With the following read-only parameters, you can retrieve information about the receiver options.

Complete Information About the Receiver Options

Name: /par/opts
Access: r
Type: list {cind,_GPS,_GLO,…}

The first entry in this list is the /par/opts/cind parameter described above. The rest are entries for every existing JAVAD GNSS option. Refer to "Supported Options" on page 570 for the list of options.

Complete Information About the Option NAME

Name: /par/opts/NAME
Access: r
Type: list {cur,purchased,leased,date}

Current Value of the Option NAME

Name: /par/opts/NAME/cur
Access: r
Type: integer
Values: [-1…511]

This parameter contains currently effective option value. It will either be set according to the larger of the /par/opts/NAME/purchased and /par/opts/NAME/leased values or will be set to -1.

-1 – the option NAME is not supported either by the firmware version you use, or by the receiver hardware.

0 – the option NAME is disabled.

[1…511] – option NAME is enabled. Refer to "Supported Options" on page 570 for the meaning of particular values for given option.

Purchased Value of the Option NAME

Name: /par/opts/NAME/purchased
Access: r
Type: integer
Values: [0…511]

Leased Value of the Option NAME

Name: /par/opts/NAME/leased
Access: r
Type: integer
Values: [0…511]

Expiration Date of the Option NAME

Name: /par/opts/NAME/date
Access: r
Type: string
Default: 0

This parameter contains either 0 if no leased value is loaded, or expiration date of the leased value in the format "ddmmyy", where:

dd - decimal day of month [01…31]

mm - decimal month number [01…12]

yy - decimal year [00…99]

Supported Options

The following table describes currently supported receiver options:

Table 4-2. Receiver Options

Name	Description
_GPS	Allow GPS satellites
_GLO	Allow GLONASS satellites
_L1_	Allow CA/L1 measurements
_L2_	Allow P/L2 and P/L1 measurements
_POS	Maximum allowed position update rate for single point and code differential positioning in Hz. [0…100]
_RAW	Maximum measurement update rate in Hz. [0…100]
_MEM	Maximum memory space for raw data files. Allows to store at least N megabytes of data, where: N = _MEM (_MEM=[0…128]) N = 128 + (_MEM - 128) × 16 (_MEM=[128…248]) N = 2048 + (_MEM - 248) × 1024 (_MEM=[248…511])
COOP	(unsupported)
_PPS	Enable PPS signals. 1 - one of the available PPS signals, PPS A, is enabled; 2 - both PPS signals are enabled.
EVNT	Enable Event signals. 1 - one of the available Event signals, Event A, is enabled; 2 - both Event signals are enabled.
_AJM	Enable Jamming Suppressor.

Table 4-2. Receiver Options

Name	Description
_MPR	Enable Code and Carrier Phase Multipath Suppressors.
_FRI	Enable external frequency input.
_FRO	Enable 20 MHz stable frequency output.
RS_A	Maximum allowed baud rate for serial port A (in kilo-baud). If this option is not loaded, the current value of this option will be set to 115, thus enabling the port and limiting its speed by 115200 baud.
RS_B	Maximum allowed baud rate for serial port B (in kilo-baud)
RS_C	Maximum allowed baud rate for serial port C (in kilo-baud)
RS_D	Maximum allowed baud rate for serial port D (in kilo-baud)
INFR	(unsupported) Enable the infrared port.
_PAR	(unsupported) Enable the parallel port.
_GSM	Enable GSM support. This is a bit-field option. bit#0 - enable internal GSM modem. bit#1 - enable generic GSM driver.
_UHF	Enable UHF support. This is a bit-field option. bit#0 - enable internal UHF modem. bit#1 - enable generic UHF driver.
RAIM	Enable RAIM.
_DTM	Enable datums other than WGS84 and PE90.
MAGN	Enable magnetic declination.
_GEO	Enable geoid model. Refer to “Use Fixed Geoidal Separation” on page 260 for details.
_WPT	(unsupported)
WAAS	Enable SBAS satellites.
OMNI	(unsupported)
CDIF	Enable code differential positioning mode. [0,1]
PDIF	Maximum allowed RTK position update rate in Hz. [0…100]
RTMO	Enable RTCM messages. This is a bit-field option. If bit#0 is set, RTCM messages relating to code differential are enabled. If bit#1 is set, RTCM messages relating to carrier phase differential are enabled. Other bits are reserved.

Table 4-2. Receiver Options

Name	Description
RTMI	Maximum number of ports that could be simultaneously set to the `rtcm` input mode. [0…5]
CMRO	Enable CMR messages. This is a bit-field option. If bit#0 is set, the whole range of CMR messages are enabled. Other bits are reserved.
CMRI	Maximum number of ports that could be simultaneously set to the `cmr` input mode. [0,1]
_LIM	Disable height and/or velocity limitation. This is a bit-field option. bit#0 - disable both height and velocity limitation. bit#1 - disable height limitation only. bit#2 - disable velocity limitation only.
_CPH	Enable true carrier phase. 0 - integral doppler is output instead of true carrier phase. In this case the option PDIF will not be fully available because only float solutions can be obtained when RTK using integral doppler for true carrier phase. 1- true carrier phase is output. Note: this option is currently always enabled.
ETHR	Enable Ethernet. [0,1]
_USB	Enable USB device interface. [0,1]
OCTO	Enables heading and attitude modes. [0,1,2] 1 - heading mode is enabled 2 - both heading and attitude modes are enabled
AUTH	Authorization for external programs. This is a bit-field option with the bits having the following definition [TBD]: bits #0…#7 – reserved.
JPSO	Enable GREIS messages. This is a bit-field option. If bit#0 is set, the GREIS message `[BI]` is enabled, making it possible to setup static RTK reference station using GREIS messages .
JPSI	Maximum number of ports that could be simultaneously set to the `jps` input mode. [0…5]
_TCP	Maximum number of simultaneous TCP connections. [0…5]
_FTP	Maximum number of simultaneous FTP connections. [0,1]
_BTL	Enable Bluetooth
DIST	Maximum allowed range for RTK

Table 4-2. Receiver Options

Name	Description
CORI	Enable differential corrections input by port. This is a bit-field option. bit#0 - enable input on Serial A. bit#1 - enable input on Serial B. bit#2 - enable input on Serial C. bit#3 - enable input on Serial D.
LAT1	Specifies the latitude of the upper left corner of the rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to x90, where x = 0 stands for N – North hemisphere (positive numbers), x = 1 stands for S – South hemisphere (negative numbers)
LON1	Specifies the longitude of the upper left corner of the rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to 360.
LAT2	Specifies the latitude of the lower right corner of the rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to x90, where x = 0 stands for N – North hemisphere (positive numbers), x = 1 stands for S – South hemisphere (negative numbers)
LON2	Specifies the longitude of the upper left corner of the rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to 360.

L_CS	Checksum of the LAT1, LON1, LAT2, LON2 options. This checksum is computed according to the following algorithm:
	`L_CS = LAT1 ^ LON1 ^ LAT2 ^ LON2`
	`if(L_CS == 0) L_CS = 1`
	
	If the checksum mismatches, then its current value is set to 0 and the receiver will not compute its position and output raw data measurements within corresponding rectangle.

LAT3	Specifies the latitude of the upper left corner of the second rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to x90, where x = 0 stands for N – North hemisphere (positive numbers), x = 1 stands for S – South hemisphere (negative numbers)
LON3	Specifies the longitude of the upper left corner of the second rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to 360.

Table 4-2. Receiver Options

Name	Description
LAT4	Specifies the latitude of the lower right corner of the second rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to x90, where x = 0 stands for N – North hemisphere (positive numbers), x = 1 stands for S – South hemisphere (negative numbers)
LON4	Specifies the longitude of the upper left corner of the rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to 360.

LCS2	Checksum of the LAT3, LON3, LAT4, LON4 options. This checksum is computed according to the following algorithm:
	`LCS2 = LAT3 ^ LON3 ^ LAT4 ^ LON4`
	`if(LCS2 == 0) LCS2 = 1`
	
	If the checksum mismatches, then its current value is set to 0 and the receiver will not compute its position and output raw data measurements within corresponding rectangle.

RM3I	Maximum number of ports that could be simultaneously set to the `rtcm3` input mode. [0…5]
RM3O	Enable RTCM3 messages. This is a bit-field option. If bit#0 is set, RTCM3 messages relating to code differential are enabled. If bit#1 is set, RTCM3 messages relating to carrier phase differential are enabled. Other bits are reserved.
_CAN	Number of enabled CAN interface ports. [1,2]
_PPP	Enable point-to-point protocol (PPP) support.
TCCL	Enable TCP clients. This is a bit-field option. bit#0 - enable RCV client. bit#1 - enable NTRIP client. bit#2 - enable SISNET client. bit#3 - enable NTRIP Server client. bit#4 - enable File Push client.
IRIG	Enable IRIG timing interface.
UDPO	Enable messages output support over UDP/IP.
_IMU	Enable Inertial Measurement Unit support.
_L2C	Allow L2C measurements.
_L5_	Allow L5 measurements.
_GAL	Allow Galileo satellites.
WIFI	Enable WiFi support.

Table 4-2. Receiver Options

Name	Description
RCVT	Receiver type. 0 - OEM board. Other values are board-dependent: - For TR_XXX boards: 1 - ALPHA 2 - TYRANT 3 - ALPHA2 4 - MCANT - For TRE_XXX, DUO_XXX, and QUA_XXX boards: 1 - DELTA 2 - SIGMA 3 - JLINK 4 - DELTA_S
GCLB	Enable GLONASS calibrator.
COPN	Enable CANOpen support.
PRTT	Port type. Enable RS422 mode by port. This is a bit-field option. bit#0 - enable RS422 on Serial A. bit#1 - enable RS422 on Serial B. bit#2 - enable RS422 on Serial C. bit#3 - enable RS422 on Serial D. bit#4 - enable decoding of bit#3 on TYRANT. bit#5 - combine SerialB and SerialD Rx/Tx pins into single SerialD output pins. Useful for MCANT, to fit 2 RS232 port into 4 pins of single connector. If bit#4 is not set for TYRANT, bit#3 is ignored and port D is always set into RS422 mode.
DEVS	Installed external devices. This is a bit-field option. bit#0 - external frequency range and output power. bit#1 - PPS level
_PTP	Enable Precision Time Protocol support.
TCPO	Maximum number of TCP output ports [0…5]
_TLS	Enable TLS/SSL support
HTTP	Enable HTTP server
QZSS	Enable QZSS support
_L1C	Enable L1C signal support
COMP	Enable BeiDou (COMPASS) system support
GBAI	Enable GBAS input support
GBAO	Enable GBAS output support
SPEC	Enable spectrum measurements support

Table 4-2. Receiver Options

Name	Description
_E5B	Enable E5B signal support
EDEV	Connected external devices. This is a bit-field option. bit#0 - Inclinometer and electronic compass. bit#1 - Video camera. bit#2 - LBAND/BEACON receiver. bit#3 - Oscillator type. 0 - built-in, 1 - external.
_GEN	Unit repair generation (for internal use of JAVAD GNSS)
IRNS	Enable IRNSS support
_SPF	Enable spoofing/jamming detection.
NTRP	Number of enabled NTRIP Caster mountpoints. [0…5]

APPENDICES

A.1 Computing Checksums

For messages, the checksum is computed starting with the first byte of the message identifier and ending with the byte immediately preceding the checksum field, inclusive.

For commands, the checksum is computed starting with the command's first non-blank byte and ending with the character '@', inclusive.

A.1.1 Computing 8-bit Checksum

Provided 'count' bytes of data are put into a buffer 'src', the 8-bit checksum could be computed according to the following algorithm:

```
typedef unsigned char u1;

enum {
  bits   = 8,
  lShift = 2,
  rShift = bits - lShift
};

#define ROT_LEFT(val) ((val << lShift) | (val >> rShift))

u1 cs(u1 const* src, int count)
{
  u1 res = 0;
  while(count--)
    res = ROT_LEFT(res) ^ *src++;
  return ROT_LEFT(res);
}
```

A.1.2 Computing CRC16

Provided 'count' bytes of data are put into a buffer 'src', the CRC16 checksum could be computed according to the following algorithm:

```
typedef unsigned short Crc16;

enum {
  WIDTH = 16,    // Width of poly
  POLY = 0x1021, // Poly. Bit #16 is set and hidden
  BYTE_BITS = 8, // Number of bits in byte
```

```
  TABLE_SIZE = 1 << BYTE_BITS, // Size of table
  MSB_MASK = 1 << (WIDTH - 1)  // Mask for high order bit in a word
};

// Table (generated by 'crc16init()'.
static Crc16 table[TABLE_SIZE];

// Initializes the table. Should be called once before the first
// call to 'crc16()'
void crc16init(void)
{
  Crc16 i;
  for(i = 0; i < TABLE_SIZE; ++i) {
    Crc16 val = i << (WIDTH - BYTE_BITS);
    int j;
    for(j = 0; j < BYTE_BITS; ++j)
      val = (val << 1) ^ ((val & MSB_MASK) ? POLY : 0);
    table[i] = val;
  }
}

// Calculates CRC16 of 'cnt' bytes from 'src' and returns result.
// Initial value of CRC16 is supplied by caller in 'crc'.
Crc16 crc16(Crc16 crc, void const* src, int cnt)
{
  unsigned char const* s = (unsigned char const*)src;
  while(cnt--) {
    crc = (crc << BYTE_BITS)
      ^ table[(crc >> (WIDTH - BYTE_BITS)) ^ *s++];
  }
  return crc;
}
```

When the crc16() function is used to calculate checksum of a receiver message, the initial value of CRC16 should be set to zero.

A.2 Data Transfer Protocol

The GREIS data transfer protocol (DTP) is primarily designed for downloading measurement files from JAVAD GNSS receivers to a host computer and to upload new firmware to the JAVAD GNSS receivers. The process of downloading or uploading should be initiated by corresponding GREIS command(s) sent to the JAVAD GNSS receiver. After a transfer is initiated, parties should use the protocol described here.

In this section the terms “transmitter” and “receiver” are used to denote the data source and destination, respectively, of the data transfer protocol. We will call the JAVAD GNSS board “JAVAD GNSS receiver” to distinguish it from the receiving end of the protocol.

A.2.1 Protocol Description

The protocol is a fixed-size block protocol with checksum and the ability to re-send a block multiple times should a transmission error occur.

For the purposes of transmission, the stream of bytes to be transferred is divided into stream of blocks of fixed size, except the size of the last block that could be smaller. The last block of data will be transferred in full size protocol block of special type. The size of data blocks is negotiated between parties in advance, before the protocol starts. Each block of data is assigned its number starting from zero for the first block. The blocks of data are then transferred from transmitter to receiver using the following formats and procedures.

The format of a single block of transmission could be represented by the following C structure, where all multi-byte fields are sent in the "least significant byte first" (LSB) order:

```
struct Block {
  u1 type;         // Block type:
                   //  ORDINAL (SOB, 0x02)
                   //  LAST (EOT, 0x04)
                   //  ABORT (#, 0x23)
  // The following fields do not exist for block of type ABORT.
  union {
    u2 number;     // Block number for block of type
                   // ORDINAL (0 - based).
    i2 count;      // Number of bytes of data in the
                   // block of type LAST. Value -1
                   // indicates transfer error. In this
                   // case 'data' field of the block
                   // contains a zero-terminated string
                   // describing the error type.
  };
  u1 data[size];   // Data block. In the block of type LAST
                   // only the first 'count' bytes are
                   // valid. The rest of the bytes are filled
                   // with zeroes.
  u2 cs;           // CRC16 checksum of bytes starting from
                   // the 'type' field up to, but not
                   // including, 'cs' field. It is calculated
                   // through all blocks starting from block 0.
  u1 eob;          // End Of Block marker (EOB, 0x03).
};
```

The transmitter sends blocks of this format in response to the receiver's requests. Requests are single byte values. There are three types of requests:

1. NACK, negative acknowledge, code 0x15. Request to re-send the current block.
2. ACK, positive acknowledge, code 0x06. Request to send the next block.

3. ABORT, abort transfer. One of the characters in the range [!–/] (ASCII codes [0x21–0x2F]). Particular value sent could be used to denote the kind of error occurred.

Any other value received when request is expected is ignored by the transmitter.

The NACK request is sent in the following cases:

- After the transfer protocol is initiated, to ask the transmitter to send the first block (block number zero).
- When receiving error occurs[1], to ask the transmitter to re-send the last block.

The ACK request is sent after the block with matching block number is successfully received and passed the test for data integrity, to ask the transmitter to send the next block.

The ABORT request could be sent by the receiver instead of ACK or NACK request to terminate the transfer protocol.

To deal with the possibility of losing synchronization between the transmitter and receiver, receiver should keep track of the expected block number. Only if actual block number of successfully received block is equal to the expected block number, receiver should send the ACK request. If successfully received block has the number that is less than those expected, receiver should silently ignore the block as it's just another copy of previously received block. If received block number is greater than those expected[2], receiver should stop the protocol by sending the ABORT request due to unrecoverable loss of synchronization.

The protocol can be terminated by the following events:

1. Transmitter sends a block of type LAST with a positive "count" field. This is a normal end of transfer. In this case the last "count" bytes of data are in the "data" field of the block. The rest of the bytes of the "data" field are filled with zeroes.
2. Transmitter sends a block of type LAST with a "count" field equal to "-1" (minus one). This means that an error on the transmitter end has occurred. The "data" field of the block contains a zero-terminated ASCII string describing the error.
3. Transmitter sends a block of type ABORT (i.e. sends "#" instead of SOB or EOT). This means that the receiver should immediately terminate its operation.

1. Including the case when no data is received in reasonable time after previous ACK or NACK request.
2. Though unlikely, this could still happen should NACK sent by the receiver be received as ACK by the transmitter due to transmission error.

4. Receiver sends the ABORT request instead of ACK or NACK request. This means that an error on the receiver end has occurred. In this case the particular value sent denotes the error code.

A.2.2 Checksum Calculation

The CRC16 checksum is calculated from the bytes of blocks starting from the field "type" up to but not including the field "cs". The initial value of "cs" is set to zero (0) at the beginning of the transfer session. Each additional block uses the initial value for "cs" obtained from the previous successfully sent and received block.

Assuming the received block is stored into the buffer named "block", the checksum validation could be done as follows (see "Computing CRC16" on page 577 for implementation of the crc16() function):

```
unsigned char block[1 + 2 + 512 + 2 + 1];
Crc16 crc = 0;
…
// NOTE: crc isn't reset to 0 at each block.
crc = crc16(crc, block, 1 + 2 + 512);
if(crc == crcReceived)
  // Checksum is correct
else
  // Checksum is wrong
```

To achieve this result, the transmitter calculates the "cs" field as follows:

```
unsigned char block[1 + 2 + 512 + 2 + 1];
Crc16 crc = 0;
…
/* NOTE: crc isn't reset to 0 at each block. */
crc = crc16(crc, block, 1 + 2 + 512);
/* Resulting 'crc' is output by transmitter LSB first. */
```

A.3 Compensating for Phase Rollovers

Carrier phases from receiver messages [pc], [p1], and [p2] will have discontinuities due to the periodic rollovers of the 32-bit word used to represent the carrier phase in these messages. Such rollovers don't occur very frequently (approximately once every 15 minutes or more). If you wish to use these messages and want to remove this kind of discontinuities from the carrier phases, use the following correction technique.

The C function *phase()* below recovers full phase:

```
typedef unsigned long u32;

/*
Return full carrier phase in cycles provided 'prev' contains short carrier
phase from previous epoch, 'cur' contains short carrier phase at current
epoch, and 'rollovers' points to the variable containing number of detected
rollovers so far.
Updates the number of detected rollovers.
*/
double phase(u32 curr, u32 prev, long* rollovers)
{
  if(curr < (1 << 30) && prev > (1 << 30) * 3)
    ++*rollovers;
  if(curr > (1 << 30) * 3 && prev < (1 << 30))
    --*rollovers;
  return ((1 << 16) * (double)(*rollovers) * (1 << 16) + curr) / 1024.;
}
```

Pseudo-code for a program utilizing the above function is:

```
long rollovers = 0;
u32 curr;
u32 prev;

SEEK_TO_THE_FIRST_EPOCH;
prev = GET_SHORT_PHASE_FORM_CURRENT_EPOCH;
loop {
  curr = GET_SHORT_PHASE_FORM_CURRENT_EPOCH;
  truePhase = phase(curr, prev, &rollovers);
  prev = curr;
  OUTPUT_PHASE(truePhase);
  SEEK_TO_THE_NEX_EPOCH;
}
```

A.4 Obsolete Messages

Warning: ***Messages described in this section have know problems and are supported for backward compatibility only. Please don't use these messages for new projects!***

A.4.1 Integrated Messages (obsolete)

Warning: *Use generic standard messages for new projects instead of integrated messages*

For the users that prefer to have different yet logically related data in a single message, the JAVAD GNSS receiver supports a set of integrated messages. For example, the message [rM] may contain all of the code and carrier phase measurements available in the receiver for the given epoch, though this is achieved in exchange for much more complex internal message structure. Integrated messages are also somewhat optimized for real-time applications, so we also sometimes call them "real-time messages". The exact

contents of these messages are defined by corresponding receiver parameters[1] that are not local to particular output stream. It means that using the integrated messages, one can't have different variants of these messages to be enabled to be output to different output streams simultaneously.

In the integrated messages, the field "sample" serves two main purposes. First, it allows the user to preserve data integrity since messages referenced to a specific epoch will all have the same sample number. Second, this field allows the user to keep track of the number of lost messages issued through a given port since the sample number is incremented when the next epoch starts.

Integrated messages can be received by the user in an arbitrary order of precedence. Before decoding a message its CRC16 checksum must be checked. Remember that the checksum is computed for all the bytes starting from the first byte of the header of the message up to but not including the checksum itself (for more information about CRC16 algorithm, please refer to "Computing CRC16" on page 577).

The following tables, which are given for user reference, will explain the relationships between the integrated messages, [rE], [rM], [rV], and the basic JPS messages.

[rM] can be used in place of the following messages:

Pseudo-range measurements	[RC], [rc], [R1], [r1], [1R], [1r], [R2], [r2], [2R], [2r]
Carrier Phase measurements	[PC], [pc], [CP], [cp], [P1], [p1], [1P], [1p], [P2], [p2], [2P], [2p]
Doppler	[DC], [D1], [D2]
Signal Lock Loop Flags	[FC], [F1], [F2]
Carrier to Noise ratio	[EC], [E1], [E2]
Satellite navigation status	[SS]
Satellite Indices	[SI]
GLONASS Satellite System Numbers	[NN]
Receiver Date and Receiver Time	[RD], [~~]
Time since Last Loss-of-Lock[1]	[TC]
Receiver Reference Time to Receiver Time Offset[2]	[TO]

1. Note that there is a limitation on the maximum tracking time reported in [rM] (102.3 seconds).
2. Remember that there is a limitation on the clock offset resolution (125 ns).

[rV] can be used in place of the following messages:

Position/Velocity messages	[PO], [VE], [PV]
Solution time tag	[ST]

1. See "Parameters of NMEA messages" on page 382.

[rE] can be used in place of the following messages:

Receiver Date and Receiver Time	[RD], [~~]

You can govern the structure of your [rM] message by means of parameters from section “Parameters of NMEA messages” on page 382. Also note that the format of [rM] allows addition of new fields to the structure if necessary.

In the event of new fields showing up in the message, its version number is incremented of course. Note that lengths of the structures “Header” and “SlotRec” are specified in the message explicitly, which makes it possible to maintain backward compatibility with any older software using the message.

The message [rE], which was conceived as a time tag for any other message type, is reserved for future use. The field “sample”, which will exist in any integrated message, is intended to maintain data integrity, i.e., all messages associated with a given epoch must have identical “sample” numbers.

[rE] Reference Epoch (obsolete)

Warning: *This message is obsolete! User corresponding generic messages instead.*

```
struct RefEpoch {10} {
  u2 sample;    // Sample number [dimensionless]
  u2 scale;     // Time scale ID, leap second status and
                // week/day part of epoch representation [bitfield]
                //   15…13: time scale ID:
                //          0 - GPS, 1 - GLONASS, 2 - UTC;
                //   12…11: leap second status:
                //          0 - no leap second epoch;
                //          1 - positive leap second;
                //          2 - negative leap second;
                //          3 - leap second status is unknown;
                //       this flag shows whether a leap second
                //       occurred at the current epoch;
                //    10…0: week/day representation:
                //   (a) if time scale ID is GPS:
                //       week number [0…1023],
                //       1024 indicates unknown week number;
                //   (b) if time scale ID is GLONASS:
                //       day number within 4-year period [1…1461],
                //       0 indicates unknown day number
                //   (c) if time scale ID is UTC:
                //     day number within the year [1…366],
                //     0 indicates unknown day number;
  u4 reftime;   // Milliseconds part of epoch representation [ms]:
                //   (a) if time scale ID is GPS:
                //       milliseconds of GPS week;
                //   (b) if time scale ID is GLONASS:
                //       milliseconds of GLONASS day;
```

```
                //   (c) if time scale ID is UTC,
                //       milliseconds of UTC day;
  u2 crc16;     // 16-bit CRC
};
```

[rM] Raw Measurements (obsolete)

Warning: *This message is obsolete! User corresponding generic messages instead.*

```
struct RawMeas {N×((14|10|6)×M+6)+14} {
  u2 sample;    // Sample number []
  u2 scale;     // See [rE] for description
  u4 reftime;   // See [rE] for description
  i2 clock;     // Clock offset:
                // 15…2: Clock offset
                //       -2^13 /+(2^13-1) [125 nanoseconds]:
                //  1…0: Clock offset ID:
                //       0 - clock offset is unavailable
                //       1 - [GPS - Receiver time]
                //       2 - [GLONASS - Receiver time]
                //       3 - reserved
  u2 flags;     // Flags [bitfield]:
                //   15…13: message version [0…7]
                //    12…8: total number of "svd" records (N)
                //     7…5: this value plus 6 makes the length
                //          of the structure "Header" in bytes
                //     4…0: this value plus 10 (for "version" 0 and 1),
                //          or 6 (for "version" [2…7]) makes the length
                //          of the structure "SlotRec" in bytes
  SvData svd[N]; // SVs data (see below)
  u2 crc16;      // 16-bit CRC
};

struct SvData {(14|10|6)×M+6} {
  Header header;    // Header (see below)
  SlotRec slot[M];  // Slot records (see below)
};

struct Header {6} {
  u4 refrange;      // Reference pseudo-range [0.02 meters]
  u1 usi;           // USI (see [SI] message)
  u1 num;           // Number of slot records (M) [bitfield]:
                    //   7…3: reserved
                    //   2…0: number of slot records minus one (M)
};

struct SlotRec {14|10|6} {
  // Note: The zeroth element of the array Slot[i],i=0,…,M-1,
  // unlike the other elements, does not contain corrections
  // to the reference pseudo-range from the Header structure.
  // To provide the user with additional information, the flag
  // 'svst' is used for 'delrange' in the zeroth slot.
  i2 svstOrDelrange;  // SV status [bitfield], or
                      // Delta pseudo-range [0.02 meters].
     // SV status [bitfield]:
     //   15…11: GLONASS slot number (for GPS SV the field
     //          is undefined), [0…32], 0 - unknown
     //    10…6: Channel number [0…31], 31 - unavailable
```

```
        //     5…0: Satellite navigation status
        //  Delta pseudo-range [0.02 meters]:
        //    [full pseudo-range for given slot]-[refrange]
    u4 word1;  // Packed data 1 [bitfield]:
               //   31…12: [carrier phase] - [refrange]
               //          [-2^19…(2^19-1)] [0.0005 meters]
               //    11…9: slot ID:
               //          0 - C/A L1; 1 - P1; 2 - P2; 3 - C/A L2;
               //          4 - L5; 5,6,7 - reserved
               //       8: reserved
               //       7: signal lock loop flags are available
               //       6: lock time is available
               //     5…0: Signal-to-noise ratio [dB×Hz]
    u2 flags;  // Signal lock loop flags (see [FC] message)
    u2 lock;   // Packed data 2 [bitfield]:
               //   15…12: fractional part of Signal-to-noise
               //          ratio [0.1 dB×Hz]
               //   11…10: reserved
               //     9…0: lock time [0.1 second]. Tracking time since
               //          last loss of lock. Varies between 0 and
               //          102.3 seconds. "Gets stuck" at 102.3s after
               //          the actual tracking time exceeds this value
               //          (until another loss of lock occurs).
    u4 word2;  // Packed data 3. Only present for "version" 0 [bitfield]:
               //   31…7: Doppler [-2^24…(2^24-1)], [0.001 Hz]
               //    6…0: reserved
};
```

When handling [rM] message, the following rules must be observed:

1. The user should retrieve from the message its version number and the lengths of the structures `Header` and `SlotRec`. These fields are necessary to maintain compatibility with older software in case the message structure is modified in the future. At present, there are three versions, 0, 1, and 2. Versions 1 and 2 are intended for RTK applications. In version 1, the field `word2` is removed from the structure `SlotRec` altogether, which results in a more compact data set as compared against version 0. In version 2, the fields `flags`, `lock`, and `word2` are removed from the structure `SlotRec`.
2. Next, the user should retrieve the "total number of `svd` records" field. Although it is possible to decode the message by using the message length from the message's header, taking into account the "total number of `svd` records" field simplifies the decoding.
3. Field `refrange` from the structure `SvData` serves as a reference for all the other code and carrier phase measurements available for the given satellite. In other words, all the measurements other than the reference pseudo-range, are represented as deltas referenced to a common reference value. Such an approach allows the reduction of message length. The first field in the structure `SlotRec` should be handled depending on whether the structure's "slot number" is zero or not.

4. Field num from the structure Header shows the total number of slot records (see the structure "SlotRec"). For example, if only C/A measurements are enabled in the message, the field num will be zero.

[rV] Receiver's Position and Velocity (obsolete)

Warning: *This message is obsolete! User corresponding generic messages instead.*

```
struct PosVelVector {42} {
  u2 sample;  // Sample number []
  u2 delta;   // [bitfield]:
              //   15…5: Difference between the raw measurement time
              //         (available from either [rM] or [rE] message)
              //         and the position time tag [-1024…1023], [5 ms]
              //    4…0: reserved
  u4 word1;   // 32 MSB of Position X-component;
  u4 word2;   // [bitfield]:
              //   31…24: 8 LSB of Position ECEF X-component [10^-4 m]
              //          or Latitude [10^-11 radians]
              //          or Grid (Local) X-component [10^-4 m];
              //      23: 1 - indicates that Position is valid
              //   22…21: 0 - Position is given in ECEF system
              //          1 - Position is given in geodetic coordinates
              //              (latitude, longitude, height above
              //              ellipsoid)
              //          2 - Position is given in grid (or local)
              //              coordinates
              //          3 - reserved
              //   20…16: Number of GPS SVs used in computation;
              //      15: 1 - indicates that Velocity is valid
              //   14…13: reserved
              //    12…8: Number of GLONASS SVs used in computation;
              //     7…4: Position computation mode
              //          (see Table 3-3 on page 70);
              //     3…0: Velocity computation mode
              //          (see Table 3-3 on page 70);
  u4 word3;   // 32 MSB of Position Y-component;
  u4 word4;   //   31…24: 8 LSB of Position ECEF Y-component [10^-4 m]
              //          or Longitude [10^-11 radians]
              //          or Grid (Local) Y-component [10^-4 m];
              //   23…15: PDOP × 10 [];
              //    14…0: RMS velocity error [0.001 m/s];
  u4 word5;   // 32 MSB of position Z-component;
  u4 word6;   // [bitfield]:
              //   31…24: 8 LSB of position ECEF Z-component
              //          or Height above ellipsoid or geoid[1] [10^-4 m];
              //   23…20: reserved;
              //    19…0: RMS Position error [0.001 m];
  u4 word7;   // [bitfield]:
              //   31…4: velocity X-component [10^-4 m/s]
              //         or East component (if types 1 and 2 are
              //         selected in bits 22…21 of the 'word2' field);
              //    3…2: reserved;
              //    1…0: 2 MSB of GREIS datum number (see note below);
```

1. Depending on the value of /par/raw/rtm/geoid parameter

```
  u4 word8;   // [bitfield]:
              //   31…4: velocity Y-component [10^-4 m/s]
              //         or North component (if types 1 and 2 are
              //         selected in bits 22…21 of the 'word2' field);
              //    3…0: bits 7…4 of datum number;
  u4 word9;   // [bitfield]:
              //   31…4: velocity Z-component [10^-4 m/s]
              //         or Height component (if types 1 and 2 are
              //         selected in bits 22…21 of the 'word2' field);
              //    3…0: 4 LSB of GREIS datum number;
  u2 crc16;   // 16-bit CRC;
};
```

Note: For GREIS datum numbers, please refer to *"Reference Ellipsoids and Local Datums"* available from *http://www.javad.com*. Currently GREIS datum numbers range between zero and 221.

[rT] Receiver Clock Offsets (obsolete)

Warning: *This message is obsolete! User corresponding generic messages instead.*

```
struct ClockOffsets {var} {
  u2 sample;    // Sample number []
  u2 reserved;  // Reserved for future extensions
  u1 recSize;   // Size of data block, in bytes, that corresponds to
                // the given satellite system (8 bytes currently);
  ClkOffs Offs[N];  // Clock offsets (see below).
        // 'N' can be derived from the following expression:
        // N = (len - 7) / recSize, where 'len' is message body
        // length taken from message header
  u2 crc16;         // 16-bit CRC
};

struct ClkOffs {
  u4 word1;  // [bitfield]:
             //     31: reserved;
             //     30: if set, improved timing mode is turned on;
             //   29…0: clock offset [10^-4 meters], bit combination
             //         0x20000000 means the clock offset is
             //         unavailable or exceeds ±536870911;
u4 word2;    // [bitfield]:
             //   31…29: reserved;
             //   28…26: navigation system (0 - GPS, 1 - GLN);
             //    25…0: derivative of clock offset [10^-4 m/s],
             //          bit combination 0x2000000 means that clock
             //          offset is unavailable or exceeds ±33554431;
};
```

A.4.2 Generic Messages (obsolete)

[SI] Satellite Indices (obsolete)

Warning: *This message is obsolete. [SX] should be preferred. Refer to "Backward Compatibility Considerations" on page 90 for further discussion*

```
struct SatIndex {nSats+1} {
  u1 usi[nSats];  // USI array []
  u1 cs;          // Checksum
};
```

The [SI] message contains an array of USIs for every satellite in SvsIdx. [SI] was the way to build and update the SvsIdx from the GREIS message stream before [SX] has been introduced. Even newer software may need to decode the [SI] to be able to process files generated by old versions of receiver firmware.

A.4.3 Raw Navigation Data (obsolete)

[GD] GPS Raw Navigation Data (obsolete)

Warning: *This format is obsolete. Use [gd] message instead.*

```
struct GpsNavData {N×recSize+2} {
  u1     recSize;  // Size of satellite data record (currently 42)
  SvData dat[N];   // Satellite data. "N" can be derived from the
                   // following expression:
                   // N=([Message Length] - 2) / recSize)
  u1     cs;       // Checksum
};

struct SvData {recSize} {
  i1 prn;          // Pseudo-Range Number (PRN)
  u1 cnt;          // Counter which is updated upon receiving a
                   // new sub-frame for given satellite.
  u4 data[10];     // GPS sub-frame contents. Every 4-bytes word
                   // contains 30 LSB of the GPS navigation data.
};
```

When decoding this message it's essential to remember the value of the cnt field of the last received data block for every satellite and ignore consecutive sub-frames with matching value of the cnt field, if any.

[QD] QZSS Raw Navigation Data (obsolete)

Warning: *This format is obsolete. Use [qd] message instead.*

```
struct QzssNavData {N×recSize+2} {
  GpsNavData   data;
};
```

[LD] GLONASS Raw Navigation Data (obsolete)

Warning: *This format is obsolete. Use [lD] message instead.*

```
struct GloNavData {N×recSize+2} {
  u1     recSize;  // Size of satellite data record (currently 18)
  SvData dat[N];   // Satellite data. "N" can be derived from the
                   // following expression:
                   // N=([Message Length] - 2) / recSize)
  u1     cs;       // Checksum
};

struct SvData {recSize} {
  i1 fcn1;         // Frequency Channel Number plus 1(FCN+1)
  u1 cnt;          // Counter which is updated upon receiving a
                   // string of a GLONASS sub-frame for given FCN.
  u4 data[4];      // GLONASS string contents. Every 4-bytes word contains
                   // 25 LSB of the string of GLONASS sub-frame.
};
```

When decoding this message it's essential to remember the value of the `cnt` field of the last received data block for every satellite and ignore consecutive sub-frames with matching value of the `cnt` field, if any.

A.4.4 Text Messages (obsolete)

[MS] RTCM 2.x Status (obsolete)

Warning: *This format is obsolete. Use [DL] instead.*

This message describes the status of RTCM rover station.

#	Format	Description
1	RTCMST	Message title
2	%3D	Time [in seconds] elapsed since last message was received (maximum value = 999). Estimated with an accuracy of ±1 second.
3	%4D	Number of received messages (between 0001…9999). If no message has been received, this data field contains zero.
4	%4D	Number of corrupt messages (between 0001…9999). If there are no corrupt messages detected, this data field is set to zero.
5	%.2F	Data link quality in percent (0…100).
6	@%2X	Checksum

[QQ] IMU Attitude Angles

This message contains IMU attitude angles.

#	Format	Description
1	INSATT	Message title
2	%C	INS solution indicator: "V" means solution is valid "N" means there is no solution
3	%6.2F	UTC time of the INS solution (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%.3F	Roll angle [degrees]
5	%.3F	Pitch angle [degrees]
6	%.3F	Heading angle [degrees]
7	@%2X	Checksum

[WW] IMU Measurements

This message contains IMU measurements converted into physical units with compensated misalignments, scale factors, and temperature drifts.

#	Format	Description
1	INSDAT	Message title
2	%6.2F	UTC time of the INS solution (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
3	%.3F	X acceleration [m/sec^2]
4	%.3F	Y acceleration [m/sec^2]
5	%.3F	Z acceleration [m/sec^2]
6	%.3F	X angular velocity [deg/sec]
7	%.3F	Y angular velocity [deg/sec]
8	%.3F	Z angular velocity [deg/sec]
9	@%2X	Checksum

[ZZ] IMU Integrated Antenna Velocities

This message contains GNSS+INS NED integrated velocities.

#	Format	Description
1	INSVEL	Message title
2	%C	INS solution indicator: "V" means solution is valid "N" means there is no solution
3	%6.2F	UTC time of the INS solution (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%.3F	X (North) local velocity [m/sec]
5	%.3F	Y (East) local velocity [m/sec]
6	%.3F	Z (Down) local velocity [m/sec]
7	@%2X	Checksum

A.4.5 IMU Measurements Messages (obsolete)

[fA] Accelerometer Raw Data

```
struct fA  {7} {
  i2 ax; // ax multiplied by 588 [g]
  i2 ay; // ay multiplied by 588 [g]
  i2 az; // az multiplied by 588 [g]
  u1 cs; // Checksum
};
```

[lA] Accelerometer Raw Data

```
struct lA {7} {
  i2 ax; // ax multiplied by 588 [g]
  i2 ay; // ay multiplied by 588 [g]
  i2 az; // az multiplied by 588 [g]
  u1 cs; // Checksum
};
```

[tA] Accelerometer and Gyroscope calibrated Data

```
struct tA {25} {
f4 ax; //(ax-ox)×Kx [ m/sec^2]
f4 ay; //(ay-oy)×Ky [ m/sec^2]
f4 az; //(az-oz)×Kz [ m/sec^2]
f4 wx; //wx-gox [rad/sec]
```

```
f4 wy; //wy-goy [rad/sec]
f4 wz; //wz-goz [rad/sec]
u1 cs; // Checksum
};
```

`Kx,Ky,Kz` - scale factors

`ox,oy,oz` - offsets

[aV] Gyroscope Raw Data

Angular velocities in the body-frame.

```
struct aV {13} {
  f4 wx; // x-component multiplied by 2800 [rad/sec]
  f4 wy; // y-component multiplied by 2800 [rad/sec]
  f4 wz; // z-component multiplied by 2800 [rad/sec]
  u1 cs; // Checksum
};
```

[mA] Magnetometer Raw Data

Magnetic field in the body-frame.

```
struct mA {7} {
i2 bx; // x-component
i2 by; // y-component
i2 bz; // z-component
u1 cs; // Checksum
};
```

[dV] Acceleration in ENU

```
struct dV {13} {
  f4 dvx; // Derivative of the eastern velocity [m/sec^2]
  f4 dvy; // Derivative of the northern velocity [m/sec^2]
  f4 dvz; // Derivative of the up velocity [m/sec^2]
  u1 cs;  // Checksum
};
```

[IM] Inertial Measurements

```
struct InertialMeasurements {25} {
  f4 accelerations[3];        // ax,ay,az [m/sec^2]
  f4 angularVelocities[3];    // wx,wy,wz [rad/sec]
  u1 cs;
};
```

This message contains measurements from the IMU converted into physical units with compensated misalignments, scale factors, and temperature drift. The measurements are provided in the local coordinate system.

[MA] Accelerometer and Magnetometer Measurements

```
struct AccMag  {38} {
  u4 time;              // receiver time [ms]
  f4 accelerations[3]; // ax, ay, az [cm/sec^2]
  f4 induction[3];      // bx, by, bz
  f4 magnitude;         // Value of magnetic field
  f4 temperature;       // Temperature of magnetic sensor [deg C]
  u1 calibrated;        // 1 - calibrated, 0 - not calibrated
  u1 cs;                // Checksum
};
```

This message contains compensated measurements from accelerometer and magnetometer.

A.4.6 Tilt-compensated Solution Messages (obsolete)

[PE] WGS84 Coordinates of the Pole End

```
struct PE {25} {
  f8 x;  // x-coordinate of the end of the pole in WGS84 [m]
  f8 y;  // y-coordinate of the end of the pole in WGS84 [m]
  f8 z;  // z-coordinate of the end of the pole in WGS84 [m]
  u1 cs; // Checksum
};
```

[pV] Local Coordinates of Antenna Phase Center

```
struct pV {37} {
  f8 x;  // Eastern coordinate of APC in ENU [m]
  f8 y;  // Northern coordinate of APC in ENU [m]
  f8 z;  // Up-coordinate of APC in ENU [m]
  f4 vx; // Eastern velocity in ENU [m/s]
  f4 vy; // Northern velocity in ENU [m/s]
  f4 vz; // Up velocity in ENU [m/s]
  u1 cs; // Checksum
};
```

[pE] Tilt-compensated Solution (full)

```
struct pE {104} {
  u1 status; // Status (see below)
  u1 error;  // Error (see below)
  f8 lat;    // Latitude of the end of the pole in WGS84 [rad]
  f8 lon;    // Longitude of the end of the pole in WGS84 [rad]
  f8 h;      // Height of the end of the pole above WGS84 ellipsoid [m]
  f4 rms_xy; // RMS of position in local plane [m]
  f4 rms_h;  // RMS of position of local hight [m]
  f4 len;    // Length of time interval [sec];
  f4 crit0;  // First value of the criterion
  f4 crit1;  // Final value of the criterion
  f4 x;      // Eastern component of the end of the pole in ENU [m]
  f4 y;      // Nothern component of the end of the pole in ENU [m]
```

```
    f4 z;       // Up-component of the end of the pole in ENU [m]
    f4 heading; // Heading angle of the device [rad]
    f4 pitch;   // Pitch angle of the device [rad]
    f4 roll;    // Roll angle of the device [rad]
    f4 crit0_dir; // Initial value of dir angle [rad]
    f4 crit1_dir; // Final value of dir angle [rad]
    f4 min_dir;   // Dir angle [rad]
    f4 incline;   // Incline angle [rad]
    f4 rotation;  // Rotation angle [rad]
    f4 biasWX;    // Auto bias for x-component of the angular velocity
    f4 biasWY;    // Auto bias for y-component of the angular velocity
    f4 biasWZ;    // Auto bias for z-component of the angular velocity
    u1 fast_status; // Status of continuous tilt-compensated solution
    u1 cs;      // Checksum
  };
```

Status:

Value	Description
0	Ready
1	Error
2	Waiting for immobility
3	Waiting for motion
4	In motion
5	Immobile
6	Ready to count
7	Counting
8	Fixed

Error:

Value	As string
0	No errors
1	Start undefined
2	T0 > history T1
3	T0 < history T0
4	N > history length
5	No position
6	No fixed position
7	No ENU coordinates
8	No IMU data
9	No gyroscope data
10	Unable to minimize
11	Pole end moved, or calibration fails

Value	As string
12	Motion amplitude is not enough to detect heading
13	Incorrect value of /par/pos/curmsint
14	No velocity measurements are available
15	Command available only when /par/imu/mode is fixed_tip
16	Integration time exceeded
17	Motion not detected before stop
18	Disabled when auto_count is on
19	Uncalibrated accelerometer and/or gyroscope

[pe] Tilt-compensated Solution (short)

```
struct pe {54} {
  f8 x; // Current x-coordinate of the end of the pole in WGS84 [meters]
  f8 y; // Current y-coordinate of the end of the pole in WGS84 [meters]
  f8 z; // Current z-coordinate of the end of the pole in WGS84 [meters]
  u1 error;           // see Table 2
  i4 posMsInt;        // Position update rate [ms]
  i4 headingMode;     // Mode of operation
  f4 stopInterval;    // Stillness duration [sec]
  f4 motionInterval; // Duration of motion  [sec]
  f4 poleLength;      // Length of the pole [meters]
  u1 autoCount;       // Automatic detection of stillness
  u1 autoGyrBiases;  // Automatic estimation of gyroscope bias
  u1 gyroCalibEpochs;// IMU calibration epochs
  u1 calibState;      // IMU calibration state
  f4 doppsmi;         // Doppler smoothing bandwidth [Hz]
  u1 cs; // Checksum
};
```

A.5 Obsolete Receiver Objects

In this section you will find a list of receiver objects that are considered obsolete, along with recommended substitutions. Some of obsolete objects listed in the table below are not supported anymore. Others are still supported for backward compatibility but are subject for removal without notice. Obsolete objects that are still supported are only recognized when explicitly adressed in the GREIS commands, i.e., they are not output by print and list commands when containing group of objects is being output.

Table A-1. Obsolete Receiver Objects

Obsolete Object	Use Instead
/par/version/main	/par/rcv/ver/main
/par/rcvid	/par/rcv/id

Table A-1. Obsolete Receiver Objects

Obsolete Object	Use Instead
/par/raw/elm	/par/out/elm/cur/file/a
/par/button/period	/par/log/sc/period
/par/rtcm/rover/maxage	/par/pos/cd/maxage
/par/rtcm/mode	/par/base/mode/rtcm /par/rover/mode/rtcm
/par/pos/meas	/par/pos/sp/meas
/par/pos/iono	/par/pos/sp/iono
/par/pos/tropo	/par/pos/sp/tropo
/par/hd/mode	/par/pos/pd/hd/mode
/par/hd/uselen	/par/pos/pd/hd/uselen
/par/hd/len/0	/par/pos/pd/hd/len/0
/par/opts/all/NAME	/par/opts/NAME
/par/opts/cur/NAME	/par/opts/NAME/cur
/par/hist/out	/par/pos/pd/hist/out /par/pos/pd/hist/num /par/pos/pd/hist/bad
/par/out/elm/cur/log	/par/out/elm/cur/file/a
/par/out/minsvs/cur/log	/par/out/minsvs/cur/file/a
/par/out/epochs/cur/log	/par/out/epochs/cur/file/a
/cur/log	/cur/file/a
/par/out/cur/log	/par/out/cur/file/a
/par/log/sc/period	/par/log/a/sc/period
/par/cmd/create/prefix	/par/log/a/name/pre
/par/rover/base/ant/got/offs	/par/rover/base/ant/got/m_offs
/log/&[name,size,time]	/log (i.e., print,/log:on)
/par/blt/mode	/par/dev/blt/a/mode
/par/blt/name	/par/dev/blt/a/name
/par/blt/pin	/par/dev/blt/a/pin
/par/blt/pinreq	/par/dev/blt/a/pinreq
/par/net/tcpcl/×	/par/net/tcpcl/a/×
/msg/jps/GD	/msg/jps/gd
/msg/jps/QD	/msg/jps/qd
/msg/jps/LD	/msg/jps/lD
/msg/jps/MS	/msg/jps/DL

Table A-1. Obsolete Receiver Objects

Obsolete Object	Use Instead
/par/cmd/create/date	/par/log/name/date
/par/cmd/create/pre/[a,b]	/par/log/[a|b]/name/pre
/par/lock/gal/giove	removed since firmware 3.7.0

Table A-2. Obsolete Receiver Objects Since Firmware 3.7.5

Obsolete Object	Use Instead
/par/lock/×/sat	/par/lock/sat/×
/par/lock/glo/fcn	/par/lock/sat/glo
/par/lock/gal/e5a	/par/lock/sig/gal/e5a
/par/lock/gal/e5b	/par/lock/sig/gal/e5b
/par/lock/gal/altboc	/par/lock/sig/gal/aboc
/par/lock/gal/e6	/par/lock/sig/gal/e6
/par/lock/glo/l3	/par/lock/sig/glcdma/l3
/par/lock/pcode	/par/lock/sig/gps/p1 /par/lock/sig/gps/p2 /par/lock/sig/glo/p1 /par/lock/sig/glo/p2
/par/lock/gps/pcode	/par/lock/sig/gps/p1 /par/lock/sig/gps/p2
/par/lock/glo/pcode	/par/lock/sig/glo/p1 /par/lock/sig/glo/p2
/par/lock/l2c	/par/lock/sig/gps/l2c /par/lock/sig/glo/l2c /par/lock/sig/qzss/l2c
/par/lock/l5	/par/lock/sig/gps/l5 /par/lock/sig/waas/l5 /par/lock/sig/qzss/l5
/par/lock/qzss/lex	/par/lock/sig/qzss/l6

Table A-3. Obsolete Receiver Objects Since Firmware 3.7.6

Obsolete Setting	Use Instead
set,/par/lock/mode,old	set,/par/lock/adv/guide,y
set,/par/lock/mode,new	set,/par/lock/adv/guide,n

Table A-4. Obsolete Receiver Objects Since Firmware 4.3.00

Obsolete Object	Use Instead
/par/log/push/host /par/log/push/port /par/log/push/passwd /par/log/push/timeout /par/log/push/rinex/opts /par/log/push/rinex/crx /par/log/push/cggtts/opts	/par/log/push/dest/a/host /par/log/push/dest/a/port /par/log/push/dest/a/passwd /par/log/push/dest/a/timeout /par/log/push/dest/a/rinex/opts /par/log/push/dest/a/rinex/crx /par/log/push/dest/a/cggtts/opts
/par/pwr/bat/[a|b] /par/pwr/bat/[a_dc|b_dc] /par/pwr/bat/level/off	/par/pwr/cell/[a|b]/v /par/pwr/cell/[a|b]/dc /par/pwr/cell/level/off
/par/pwr/charge/bat /par/pwr/charger /par/pwr/chdc	/par/pwr/charge/mode /par/pwr/charge/v /par/pwr/charge/dc
/par/pwr/fgauge/[a|b]/soc	/par/pwr/cell/[a|b]/soc

Table A-5. Obsolete Receiver Objects Since Firmware 4.4.00

Obsolete Object	Use Instead
/par/net/wlan/ap/list/con/clear	/par/net/wlan/ap/con/clear
/par/net/wlan/ap/list/con	/par/net/wlan/ap/con/list
/par/imu/height	/par/pole/height
/par/imu/mode	/par/pole/mode

Table A-6. Obsolete Receiver Objects Since Firmware 4.5.00

Obsolete Object	Use Instead
/par/pos/pd/period	/par/pos/pd/per/ref
/par/pos/pd/afperiod	/par/pos/pd/per/af
/par/pos/pd/experiod	/par/pos/pd/per/ex
/par/pos/pd/dblchk	none

A.5.1 Parameters of Integrated Messages (obsolete)

These parameters allow you to tailor the integrated messages to your particular needs. For more information about the integrated messages, please see “Integrated Messages (obsolete)” on page 582.

Include CA/L1 Measurements into [rM] Message

Name: /par/raw/rtm/meas/ca
Access: rw
Type: boolean
Values: on,off
Default: on

Include P1/L1 Measurements into [rM] Message

Name: /par/raw/rtm/meas/p1
Access: rw
Type: boolean
Values: on,off
Default: on

Include P2/L2 Measurements into [rM] Message

Name: /par/raw/rtm/meas/p2
Access: rw
Type: boolean
Values: on,off
Default: on

Version of Format of [rM] Message

Name: /par/raw/rtm/ver
Access: rw
Type: integer
Values: 0,1,2
Default: 0

0 - the format is the default one.

1 - the field word2 (doppler) is excluded from the contents of the [rM] message.

2 - the fields flags, lock, and word2 are excluded from the contents of the [rM] message. In addition, clock field is reserved and contains zeros.

Time Scale for Integrated Messages

Name: /par/raw/rtm/tscale
Access: rw
Type: enumerated
Values: gps,glo,utc
Default: gps

Type of Coordinates for [rV] Message

Name: /par/raw/rtm/coord
Access: rw
Type: enumerated
Values: xyz,geo,grid
Default: xyz

xyz - use Cartesian coordinates in [rV] message.

geo - use geodetic coordinates in [rV] message.

grid - use grid coordinates in [rV] message.

Enable Geoidal Height for [rV] Message

Name: /par/raw/rtm/geoid
Access: rw
Type: boolean
Values: on,off
Default: off

off - message [rV] will contain ellipsoidal height when parameter /par/raw/rtm/coord is set to geo or grid.

on - message [rV] will contain orthometric height when parameter /par/raw/rtm/coord is set to geo or grid.

A.5.2 RTPK Parameters (Obsolete)

Some receiver models may support running phase-differential post-processing engine and gathering its results. Only static occupations could be meaningfully processed this way.

Besides particular support in the receiver, current requirement is availability of suitable RTCM3 corrections.

This processing is performed by storing all the needed data into a file on receiver, processing of the file by RTPK engine tool, monitoring the process and result, and finally downloading RTPK report file from the receiver.

Note: In addition to the features described here, RTPK could be run through file push feature, by using "rtpk:" prefix in host name for file push. This would interfere with these parameters, so make sure both modes are not in use simultaneously. Fortunately, it's unlikely true rover will need file push to be turned on in the first place.

The following parameters are defined to govern RTPK execution and gathering of RTPK results.

RTPK Job Identifier

Name: /par/rtpk/job
Access: rw
Type: integer
Values: [-1…16535]
Default: -1

This is pseudo-parameter that serves both to start RTPK processing and to wait for its termination.

To start RTPK processing, this parameter should be set to its current value. Setting this parameter to any other value but its current one will cause error message, and RTPK won't be started.

The value of this parameter is not changed as the result of matching setting, instead RTPK process is started, and the value of this parameter will change some time later, once the RTPK process is finished. If RTPK process is already running, the 'set' command will return error.

If '/par/rtpk/file' is an empty string or is set to one of /cur/file/X, and there is no corresponding file open, the command will return error message.

As an exception, setting this parameter to -1 will start RTPK processing (provided it's in 'idle' state) no matter what current value of /par/rtpk/job is. This feature is supported for ease of manual operation and, being less robust, should not be used by automated tools.

RTPK File Name

Name: /par/rtpk/file
Access: rw
Type: string [0…63]
Values: (any string)
Default: (empty string)

This specifies file to be processed by RTPK. Either file name or output stream name ("/cur/file/[a…e]") are supported. The default empty string is synonym for "/cur/file/a".

Note: To be useful for RTPK, the file content should be specially crafted to have both rover measurements and corrections from reference station. See "Example" below for a way to configure receiver in a suitable manner.

Options for RTPK Engine

Name: /par/rtpk/opts
Access: rw
Type: string[0…128]
Values: ""

Version of RTPK Engine

Name: /par/rtpk/ver
Access: r
Type: string[0…32]
Values: "unknown"

RTPK State

Name: /par/rtpk/state
Access: r
Type: enumerated
Values: idle,starting,running
Default: idle

idle - RTPK is idle and could be started

starting - request to start RTPK is accepted, but RTPK is not started yet. You will rarely see this state in practice as it quickly changes to 'running'.

running - RTPK is running. No requests to run it are accepted when in this state.

RTPK Last Report File

Name: /par/rtpk/last/report
Access: r
Type: string
Values: (file name or empty)
Default: (empty string)

The name of the last report file generated by RTPK. The name is generated from the name of the input file automatically and won't change from run to run if input file name is the same, so previous report file will be overwritten in this case.

RTPK Last Error

Name: /par/rtpk/last/error
Access: r
Type: integer [0…255]
Values: 0

0 - no error. Last RTPK run was successful and generated report

1…29 - RTPK returned with this error and no report

30 - RTPK returned with an error but still generated report

31 - RTPK tool does not exist or is not executable

32 - input file for RTPK does not exist or is not readable

33 - error creating working environment for RTPK (such as failure to create dedicated working directory or changing there)

34 – error creating template database

35 – RTPK returned exit code in range [30…39], and generated no report file.

36…39 – reserved

40…50 – exit code of the RTPK tool

RTPK Example

Common Configuration

Configure receiver to obtain RTCM3 corrections and to forward them to a file, e.g.:

```
⇒ %imode% set,/par/dev/ser/b/imode,rtcm3
⇒ %corr%  set,/par/rtcm3/rover/out/port,/cur/file/a
```

Tell receiver to pass `/cur/file/a` to RTPK:

```
⇒ %file%  set,/par/rtpk/file,/cur/file/a
```

Site Configuration

Now on a site called, say, `SITE01`, do the following once:

```
⇒ %close% dm,/cur/file/a      # ignore [ER], if any
⇒ %rm%    remove,/log/SITE01  # ignore [ER], if any
⇒ %cr%    create,/log/SITE01
⇒ %em%    em,/cur/file/a,/msg/def:1
```

Site Operation

Monitor the number of epochs written to the file so far using:

```
⇒ %ep%    print,/par/out/epochs/cur/file/a
```

and when you want to process gathered data, do:

```
⇒ %pj%    print,/par/rtpk/job
```

and get result. Let's suppose we got `37`. Now start processing by sending:

```
⇒ %sj%    set,/par/rtpk/job,37
```

Ensure your get proper `[RE]` reply. Optionally, use

```
⇒ %ps%    print,/par/rtpk/state
```

to ensure the state is `idle` before starting processing.

Now, once processing is started, again using:

```
⇒ %pj%    print,/par/rtpk/job
```

wait for the moment the job identifier changes from `37` to different value. This indicates that processing is finished and you can gather results.

Check the resulting error code using:

```
⇒ %pe%    print,/par/rtpk/last/error
```

and get results by downloading the file identified by

```
⇒ %pr%    print,/par/rtpk/last/report
```

if this is not empty string. To download resulting file, either use the get command (see “get” on page 52), or `print` command, like this:

```
⇒ %results%    print,/log/SITE01.json&content
```

If results are not satisfactory, repeat from the start of "Site Operation" to process more data.

End of Site Operation

If results are satisfactory or can't be achieved, stop logging for this site:

```
⇒ %dm% dm,/cur/file/a
```

and optionally remove the files, both the log you've created, and report file that has been generated by RTPK.

1731 Technology Drive, San Jose, CA 95110 USA
Phone: +1(408)573-8100
Fax: +1(408)573-9100
www.javad.com

GREIS (GNSS Receiver External Interface Specification)