SNMP' Technical Note

Version 1.2
March 10, 2008

Septentrio Receivers – SNMP' Technical Note

Septentrio Satellite Navigation
Ubicenter, Philipssite 5
B-3001 Leuven, Belgium

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

Table of contents

1 Foreword

1.1 Scope

The objective, of this document, is to give an overview of the SNMP' binary interface. The SNMP' interface is a binary Septentrio proprietary interface that's loosely based on SNMP (**S**imple **N**etwork **M**anagement **P**rotocol).

This technical note is meant for programmers wanting to implement/integrate an application with the SNMP' binary interface. The reader is expected to have a sufficient knowledge of the ASN.1 and SNMP protocols. This document focuses on the specificities of the Septentrio implementation.

There's a one to one relationship between the formal MIB-description (**M**anagement **I**nformation **B**ase) and the ASCII command-line interface for all *exe*, *get* and *set*-commands. Contrary to the SNMP' format the actual MIB description used is fully SNMPv2 compliant.

It is not within the scope of this document to give the actual MIB description of every Septentrio receiver platform. The receivers are able to list their formal MIB description though.

1.2 Applicability

The SNMP' interface has the following advantages:

- Smaller footprint needed.
- Less lines of code equals faster time to market.
- Less lines of code equals less %CPU time needed.

- Applications that are already network managed, can easily be modified/adapted to act as an SNMP proxy-server for the Septentrio receiver(s).

- By listing the MIB description out of the receiver one has the possibility to make ones application auto-adaptable to different Septentrio receiver platforms and/or SW-releases.

As such the Septentrio receiver control applications, such as RxControl, are the first clients of the SNMP' protocol and take full advantage of the auto adaptive possibilities. Some of the free format fields in the MIB-description and command groupings are specifically there for the Septentrio control applications. They are mentioned in the document but are only to be considered as possible hints to build a user interface.

In this document, the command names, texts and OID's are to be seen as illustrative examples. They are syntactically correct, but they may not reflect the actual status of the MIB in your receiver and as such can't be used for actual implementation of SNMP' messages.

1.3 Document Overview

This document includes a description of:

- The SNMP' protocol;
- The SNMPv2 MIB description language through '**`lstCommandHelp`**' and '**`lstMIBDescription`**' command.

Meanwhile the attention is also on following points:

- Interpreting SNMPv2 MIB description in order to build SNMP' messages;
- The one to one relation between *exe*, *get* and *set*-command lines and the MIB-description.

2 The SNMP' protocol

```
/*!

  \file "snmptypes.h"
  \brief (S)imple (N)etwork (M)anagement (P)rotocol type description
  \ingroup SNMPMIB_SNMP

  snmptypes.h gives the basic typedefs for:
  - The SNMP protocol
  \verbatim
  +---------+----------------------------------+
  | Message | PDU = (P)rotocol (D)ata (U)nit   |
  | Header  |                                  |
  +---------+----------------------------------+
  \endverbatim
  - The PDU
  \verbatim
  +------+-------+------+-----+ +-------+-------------+-------+---
  | PDU  |Request|Error |Error| |OID,   |OID,         |OID,   |...
  | type |  ID   |Status|Index| |payload|p a y l o a d|payload|...
  +------+-------+------+-----+ +-------+-------------+-------+---
  \endverbatim


  \par Note:
  These are pseudo-SNMP definitions that are propriatary to Septentrio. They
  match the SNMP build-up of the messages but are optimised amongst others
  to match native byte order of our receivers.

  \par Origin:
  General

  \par Author:
  Freddy Voorspoels
  \par Backup:
  Philipe Jacobs

  \par Copyright:
  (c) 2006 Septentrio Satellite Navigation nv/sa, Belgium

*/

#ifndef SNMPTYPES_H
#define SNMPTYPES_H 1                  /*!< To avoid multiple inclusions */


/* Includes ==(needed for self-contained / -consistent)======================*/

#include "ssntypes.h"

/* Declarations ============================================================*/

/* Constants & Macros ------------------------------------------------------*/

#define SNMP_VERSION1  1      /*!< Septentrio's 1st version of the interface */

#define SNMP_SET       'S'
#define SNMP_GET       'G'
#define SNMP_RESP      'R'

#define SNMPERROR_MSGTYPE    1   /*!< message of unknown type received */
#define SNMPERROR_OID        2   /*!< OID doesn't exist */
#define SNMPERROR_SETACTION  3   /*!< SET Action not available for this OID */
#define SNMPERROR_GETACTION  4   /*!< GET Action not available for this OID */
#define SNMPERROR_SIZE       5   /*!< Buffer size not correct for this OID */
#define SNMPERROR_VALUE      6   /*!< Value not within syntax for this OID */
#define SNMPERROR_EXE        7   /*!< Couldn't execute callback for this OID */
```

```
#define SNMP_MAX_SIZE        2048 /*!< Maximum allowed size for SNMP messages */

/* Types ---------------------------------------------------------------------*/

/** SNMP Message Header.
  The SNMP Message header is mainly used for synchronisation.
*/
typedef struct snmpHeader_s
{
  char           preamble[2];  /*!< { '$', '&' } */
  uint8_t        version;      /*!< propriatary version */
  uint8_t        checksum;     /*!< modulo 2 checksum = XOR (header excluded)*/
  uint16_t       length;       /*!< Length of message (header not included) */
  char           community[2]; /*!< Not used for now (readable password) */
} snmpHeader_t;


/** SNMP (P)rotocol (D)ata (U)nit header.
  The PDU is a data-container for type of action and reserved room for
  reply status.
*/
typedef struct snmpPDUHeader_s
{
  char           type;         /*!< 'S'=Set, 'G'=Get, 'R'=Response */
  uint8_t        requestID;    /*!< sequence-number of request */
  uint8_t        errorStatus;  /*!< error occured */
  uint8_t        errorIndex;   /*!< at xth variable binding */
} snmpPDUHeader_t;

/** SNMP (O)bject (ID)entification of variable binding.
  The OID gives size and OID of the variable binding
*/
typedef struct snmpOID_s
{
  uint8_t        size;           /*!< size of payload */
  uint8_t        appl;           /*!< Application (eg IO, Navigation) */
  uint8_t        group;          /*!< CMD group */
  uint8_t        command;        /*!< CMD */
  uint8_t        arg_tableEntry; /*!< Argument or table entry (case of table)*/
  uint8_t        ind_tableArg;   /*!< 0 or Table Argument */
  uint8_t        tableInd;       /*!< index in table */
  uint8_t        nil;            /*!< always 0 */
} snmpOID_t;

/** SNMP 4Byte variable binding (OID included)
  This variable binding is only usefull for FInt32 and Enum32
*/
typedef struct snmpBinding4_s
{
  snmpOID_t      oid;         /*!< OID of size 4 byte binding */
  int32_t        payload;     /*!< int32_t payload for FInt32, Enum32 */
} snmpBinding4_t;             /*   BitS and Str really have variable binding */

/* External Variables/Structs ------------------------------------------------*/

/* External Procedures -------------------------------------------------------*/




#endif                                /* SNMPTYPES_H */
/* END OF HEADER FILE ========================================================*/
```

Figure 2-1: SNMPTYPES_H

SNMP' messages are limited to a maximum size of 2kByte. Each message consists of:

- Message Header: Contains data needed for synchronization on the binary data stream
- PDU (=**P**rotocol **D**ata **U**nit) : Contains the actual message in:
    - PDU Header: contains type of message + possible errors

- A number of variable bindings: each variable binding represents an entry in the MIB. Each entry is identified by the OID (**O**bject**ID**) followed by a variable size buffer containing the actual data.

Within one SNMP' message any number (from 1 till 255) of variable bindings in any order can be used as long as the total length of the SNMP' message doesn't exceed the maximum size (in other words: arguments from different set- or get- commands from the command-line interface can be combined into one SNMP' message and the variable bindings may come in a random order).

For exe-commands the correct behaviour isn't guaranteed when the variable bindings are intermixed with other commands. In case SNMP' commands are used for exe-commands it is advised to use all arguments (at least the ones with 'STATUS current') and to use them in the same order as in the command (i.e. variable binding one equals the first argument of exe-Command, variable binding 2 equals second argument of exe-command, …). Failing to do so for exe-commands may result in unexpected behaviour.

The message header contains the following info:

Item	Contains	Representation
Preamble	$&	24h 26h
Version	1	01h
Checksum	Modulo 2 checksum (XOR) over the PDU only.	1 byte
Length	Length of the PDU	2 bytes little endian
Community	Reserved	2 bytes: 00h 00h

Table 2-1: SNMP' message Header

There are 3 types of SNMP' messages defined:

- Set
- Get
- Response

Set messages contain the variable bindings to be set in the MIB. The response contains the variable bindings as set in the MIB up to the first variable binding that resulted into an error. If there was an error the entire payload of the last variable binding is replaced with 0xFF. The rest of the variable bindings (after the erroneous) aren't replied.

Get messages contain the variable bindings to be retrieved from the MIB (including the placeholders for the data!). The response contains the variable bindings as retrieved from the MIB up to the first variable binding that resulted into an error. If there was an error the entire payload of the last variable binding is replaced with 0xFF. The rest of the variable bindings (after the erroneous) aren't replied.

Item	Contains	Representation
Type	S for Set / G for get / R for response	53h / 47h / 52h
Request ID	0 .. 255	00h .. FFh
Error Status	0 for no error	00h
	1 for message type received is unknown	01h
	2 for OID doesn't exist within MIB	02h
	3 for SET action not available for this OID	03h
	4 for GET action not available for this OID	04h

	5 for buffer size is not correct for this OID 6 for value not within syntax for this OID 7 for callback for this OID gives error	05h 05h 07h
Error Index	0 on no error else nr of erroneous variable binding (counting from 1 for the first variable binding)	00h .. FFh

Table 2-2: PDU Header

The replied request ID's are equal to the request ID's of the set or get requests. There's no difference between a response from a set or a get request. It's up to the client to use the request ID to differentiate between the replies. Most often a sequence number is used as request ID. But if the client decided to do so she or he could predefine a number of separate get and set request ID's for her/his application that only uses a predefined and limited number of different SNMP' messages.

Within the Septentrio receivers there are two main groups of OID's to be distinguished:

- Non tabular
- Tabular (e.g. an array of settings per external port)

The OID's of the non tabular MIB entries consist of:

.mibnumber.groupnumber.commandnumber.argument

The OID's of tabular MIB entries consist of:

.mibnumber.groupnumber.commandnumber.1.argument.index

Mibnumber: The commands are grouped in sub-mibs that are identified above as mibnumber. The mibnumber can be seen as a hint for the items within the main menu of a GUI. (It's to say the menu items you permanently see on Windows® GUI's at the top of your application. See Figure 3-2: Suggested menu structure.)

Groupnumber: The groupnumber forms subgroups within the mibnumber. The groupnumber can be seen as a hint for the grouping of the commands within submenus on a GUI. (It's to say the list of items that drop down when on Windows® GUI's when you select an item in the main menu. See Figure 3-2: Suggested menu structure.)

Commandnumber: The commandnumber is an identification for an individual ASCII command within the subgroup.

ArgumentThe argument gives the Xth argument out of the command in the ASCII commandline interface. For non-tabular commands this starts from 1. For tabular commands this starts from 2. The argument 1 is reserved for the argument that lists the possible items in the table. There's no actual values to be stored in the MIB for this argument.

.1.: The 1 between commandnumber and argument indicates that argument 1 of the command is going to be used as list of possible items within the table. (Please note that the ASCII interface only allows 1 to be used here).

Index: For tabular commands the index indicates the Xth row out of the table that is actually used (NOT the actual Enum32 value!).

Non Tab.	Tabular	Contains	Representation

Size		Size of the payload in bytes.	00h .. FFh
Mibnumber		Main menu item identification.	01h .. FAh
Groupnumber		Submenu item identification.	01h .. FAh
Commandnumber		Command number identification.	01h .. FAh
Argument	**1**	Xth argument / Table entry.	01h .. FAh
0	**Argument**	End of OID / Xth argument	00h / 02h .. FAh
0	**Index**	Xth item out of the table	00h / 01h .. FAh
Payload		Nr of bytes as specified in Size	See Table 2-4: Payload

Table 2-3: Variable Binding

The following types of arguments/payloads exist:

Type	Comment	Size (Bytes)	Representation
Enum32	List of discrete values each representing a different textual counterpart.	4	2's complement 32 bit integer little endian
Integer32	Range of values (from X till Y presicion=1, 0.1,0.01,0.001, …) The precision is used to get a fixed point representation for floats/doubles!	4	2's complement 32 bit integer little endian
Integer64	Range of values (from X till Y presicion=1, 0.1,0.01,0.001, …) The precision is used to get a fixed point representation for doubles!	8	2's complement 64 bit integer little endian
String	Character buffer	1..255	0 terminated as long as length < size
BITS	Bitfields representable by a textual counterpart. With highest bit of byte[0] equals bit 0 and highest bit of byte[1] equals bit 8.	1..255	Reverse bit order

Table 2-4: Payload

For instance the Enum32 or Integer32 value:

- Decimal : 15
- Hexadecimal : 0000000Fh
- Little Endian : 0Fh 00h 00h 00h

For instance the Eum32 or Integer32 value:

- Decimal : -70829
- Hexadecimal : FFFEEB53h
- Little Endian : 53h EBh FEh FFh

For instance the Integer64 value:

- Decimal : 15
- Hexadecimal : 000000000000000Fh
- Little Endian : 0Fh 00h 00h 00h 00h 00h 00h 00h

For instance the Integer64 value:

- Decimal : -70829
- Hexadecimal : FFFFFFFFFFFEEB53h
- Little Endian : 53h EBh FEh FFh FFh FFh FFh FFh

For instance the BITS (size2) = bit0 + bit6 +bit9 = 10000010 01000000 is represented by:

- 82h 40h

3 The SNMPv2 MIB description language

The formal language is explained with the ltsCommandHelp (help) and lstMIBDescription (lmd) command replies (as generated by the receiver) as a guide.

Three types of replies exist:

- "**`lstCommandHelp, Overview`**": lists existing commands (in Septentrio proprietary format);
- "**`lstMIBDescription, Overview`**": lists Command OID's (in SNMPv2 compliant format);
- "**`lstMIBDescription, CommandName`**": lists Argument definitions (in SNMPv2 compliant format).

Please note that the "**`lstCommandHelp, CommandName`**" gives a brief explanation of the given command in free format.

3.1 lstCommandHelp, Overview

```
COM1>help, Overview
$R; lstCommandHelp, Overview
$-- BLOCK 1 / 0
MENU: I/O
  GROUP: ioSelection
    sdio, setDataInOut
    gdio, getDataInOut
  GROUP: ioOutput
    SUB: subSBFOut
      TAB: tabSBFGroup
        ssgp, setSBFGroups
        gsgp, getSBFGroups
      TAB: tabSBFOnce
        esoo, exeSBFOutputOnce
        gsoo, getSBFOutputOnce
    SUB: subNMEAOut
      TAB: tabNMEAGroup
        sngp, setNMEAGroups
        gngp, getNMEAGroups
      TAB: tabNMEAOnce
        enoo, exeNMEAOutputOnce
        gnoo, getNMEAOutputOnce
      TAB: tabNMEAPrecision
        snp , setNMEAPrecision
        gnp , getNMEAPrecision
    SUB: subDifCorOut
      TAB: tabDifCorOutRTCM2
        srdf, setRTCMv2DataForm
        grdf, getRTCMv2DataForm
        srdi, setRTCMv2DataInterv
        grdi, getRTCMv2DataInterv
        srdo, setRTCMv2DataOutput
        grdo, getRTCMv2DataOutput
        srmx, setRTCMv2Message16
        grmx, getRTCMv2Message16
      TAB: tabDifCorOutRTCM3
        stdf, setRTCMv3DataForm
        gtdf, getRTCMv3DataForm
        stdi, setRTCMv3DataInterv
        gtdi, getRTCMv3DataInterv
        stdo, setRTCMv3DataOutput
        gtdo, getRTCMv3DataOutput
      TAB: tabDifCorOutCMR2
        scdf, setCMRv2DataForm
        gcdf, getCMRv2DataForm
        scdi, setCMRv2DataInterv
        gcdi, getCMRv2DataInterv
        scdo, setCMRv2DataOutput
        gcdo, getCMRv2DataOutput
        scmx, setCMRv2Message2
        gcmx, getCMRv2Message2

---->
$-- BLOCK 2 / 0
  GROUP: ioInput
    SUB: subDifCorIn
      TAB: tabDifCorInRTCM2
        srdu, setRTCMv2DataUsage
        grdu, getRTCMv2DataUsage
      TAB: tabDifCorInRTCM3
        stdu, setRTCMv3DataUsage
        gtdu, getRTCMv3DataUsage
      TAB: tabDifCorInCMR2
        scdu, setCMRv2DataUsage

        …
```

Figure 3-1: **lstCommandHelp, Overview**

MENU: Gives the name for the main menu item (with corresponding mibnumber as defined in Table 2-3: Variable Binding).
GROUP: Gives the name of the submenu-item under the MENU item (with corresponding groupnumber as defined in Table 2-3: Variable Binding).
SUB: Gives a suggestion for the GUI. These are typically the menu items that appear on your screen (left or right) of a selection in a submenu.
TAB: Gives the suggestion to group the commands underneath the item in a separate tab on the window.

mnemonic, fullCommandName: The actually available commands on the receiver. The mnemonic gives a 3 a 4 character equivalent to the full command name. Both can be used in the ASCII command-line interface.

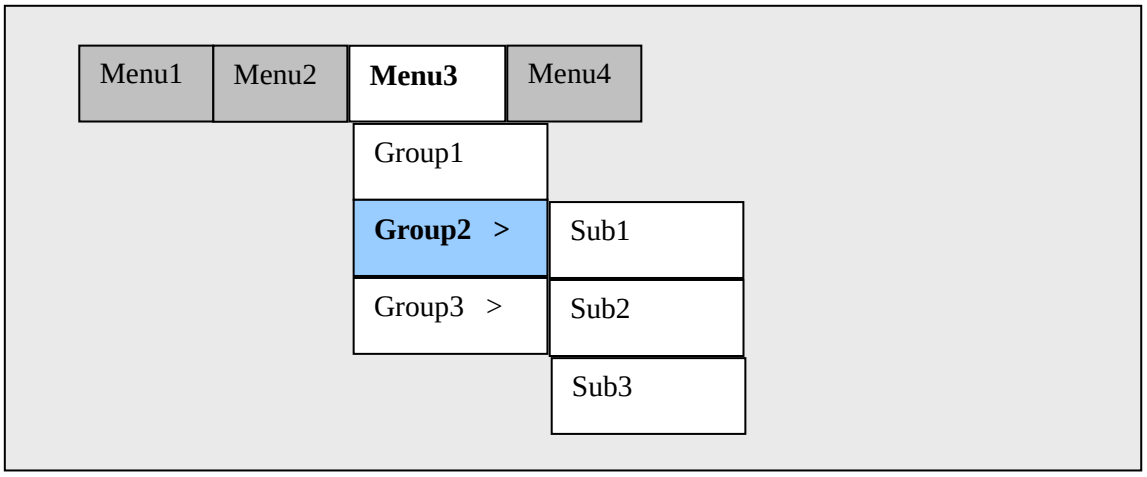

Figure 3-2: Suggested menu structure

3.2 lstMIBDescription, Overview

```
COM1>lmd, Overview
$R; lstMIBDescription, Overview
$-- BLOCK 1 / 0
-- ----------------------------------------
RXFFExt DEFINITION ::= BEGIN

ssn                 OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 27073 }
rxff                OBJECT IDENTIFIER ::= { ssn 5 }

-- I/O groups
I/O                 OBJECT IDENTIFIER ::= { rxff 1 }
ioSelection         OBJECT IDENTIFIER ::= { I/O 1 }
--    G <Selection> "Selection of type of input/output "
    DataInOut                   OBJECT IDENTIFIER ::= { ioSelection 10 }
--    - <Input/Output Selection> "Select typs of input/output on the different connections"
ioOutput            OBJECT IDENTIFIER ::= { I/O 20 }
--    G <Output Settings> "Output settings for SBF, NMEA, ..."
    SBFGroups                   OBJECT IDENTIFIER ::= { ioOutput 20 }
--    - <SBF Output Selection> "Select SBF message types and update intervals."
    SBFOutputOnce               OBJECT IDENTIFIER ::= { ioOutput 21 }
--    - <SBF Output Once> "Send SBF output once"
    NMEAGroups                  OBJECT IDENTIFIER ::= { ioOutput 30 }
--    - <NMEA Output Selection> "Select NMEA message types and update intervals."
    NMEAOutputOnce              OBJECT IDENTIFIER ::= { ioOutput 31 }
--    - <NMEA Output Once> "Send NMEA output once"
    NMEAPrecision               OBJECT IDENTIFIER ::= { ioOutput 32 }
--    - <NMEA Precision> "Set NMEA precision"

---->
$-- BLOCK 2 / 0
    RTCMv2DataForm              OBJECT IDENTIFIER ::= { ioOutput 40 }
--    - <RTCM2 Data Formatting> "Set Reference ID, ..."
    RTCMv2DataInterv            OBJECT IDENTIFIER ::= { ioOutput 41 }
--    - <RTCM2 Data Interval> "Set RTCM2 data interval per message type"
    RTCMv2DataOutput            OBJECT IDENTIFIER ::= { ioOutput 42 }
--    - <RTCM2 Output Selection> "Select RTCM2 message types"
    RTCMv2Message16             OBJECT IDENTIFIER ::= { ioOutput 43 }
--    - <RTCMv2 message 16> "Set text for message 16"
    RTCMv3DataForm              OBJECT IDENTIFIER ::= { ioOutput 50 }
--    - <RTCM3 Data Formatting> "Set Reference ID, ..."
    RTCMv3DataInterv            OBJECT IDENTIFIER ::= { ioOutput 51 }
--    - <RTCM3 Data Interval> "Set RTCM3 data interval per message type"
    RTCMv3DataOutput            OBJECT IDENTIFIER ::= { ioOutput 52 }
--    - <RTCMv3 Output Selection> "Select RTCMv3 message types"
    CMRv2DataForm               OBJECT IDENTIFIER ::= { ioOutput 60 }
--    - <CMRv2 Data Formatting> "Set Reference ID, ..."
    CMRv2DataInterv             OBJECT IDENTIFIER ::= { ioOutput 61 }
--    - <CMRv2 Data Interval> "Set CMRv2 data interval per message type"
    CMRv2DataOutput             OBJECT IDENTIFIER ::= { ioOutput 62 }
--    - <CMRv2 Output Selection> "Select CMRv2 message types"

---->
$-- BLOCK 3 / 0
    CMRv2Message2               OBJECT IDENTIFIER ::= { ioOutput 63 }
--    - <CMRv2 message 2> "Set data (Id's, COGO) for message 2"
ioInput             OBJECT IDENTIFIER ::= { I/O 30 }
--    G <Input Settings> "Input settings for Diff. Corr, ... "
    RTCMv2DataUsage             OBJECT IDENTIFIER ::= { ioInput 10 }
--    - <RTCMv2 Data Usage> "Set Reference ID to be used"
    RTCMv3DataUsage             OBJECT IDENTIFIER ::= { ioInput 20 }
--    - <RTCMv3 Data Usage> "Set Reference ID to be used"
    CMRv2DataUsage              OBJECT IDENTIFIER ::= { ioInput 30 }
--    - <CMRv2 Data Usage> "Set Reference ID to be used"
ioCOMPorts          OBJECT IDENTIFIER ::= { I/O 40 }
--    G <COM Port settings> "Setting of baudrate, data-bits, ..."
    COMSettings                 OBJECT IDENTIFIER ::= { ioCOMPorts 10 }
--    - <COM Port Settings> "Set COM port settings (Baudrate, ...)"

subSBFOut OBJECT-GROUP
    OBJECTS {
      SBFGroups,
      SBFOutputOnce }
    STATUS  current
    DESCRIPTION
      "<SBF> Settings for SBF output."
  ::= { ioOutput 255 1 }

subNMEAOut OBJECT-GROUP
    OBJECTS {
      NMEAGroups,
      NMEAOutputOnce,
      NMEAPrecision }
    STATUS  current
    DESCRIPTION
      "<NMEA> Settings for NMEA output."
  ::= { ioOutput 255 5 }

subDifCorOut OBJECT-GROUP
    OBJECTS {
```

```
      RTCMv2DataForm,
      RTCMv2DataInterv,
      RTCMv2DataOutput,
      RTCMv2Message16,
      RTCMv3DataForm,
      RTCMv3DataInterv,
      RTCMv3DataOutput,
      CMRv2DataForm,
      CMRv2DataInterv,
      CMRv2DataOutput,
      CMRv2Message2 }
    STATUS  current
    DESCRIPTION
      "<Dif. Output Selection> Settings for differential Output."
  ::= { ioOutput 255 9 }

---->
$-- BLOCK 4 / 0

tabSBFGroup OBJECT-GROUP
    OBJECTS {
      SBFGroups }
    STATUS  current
    DESCRIPTION
      "<SBF Groups Intervals> Set SBF Update Intervals per group."
  ::= { ioOutput 254 1 }

tabSBFOnce OBJECT-GROUP
    OBJECTS {
      SBFOutputOnce }
    STATUS  current
    DESCRIPTION
      "<SBF once> Get SBF messages Once."
  ::= { ioOutput 254 6 }

tabNMEAGroup OBJECT-GROUP
    OBJECTS {
      NMEAGroups }
    STATUS  current
    DESCRIPTION
      "<NMEA Groups/Intervals> Set NMEA Update Intervals per group."
  ::= { ioOutput 254 11 }

tabNMEAOnce OBJECT-GROUP
    OBJECTS {
      NMEAOutputOnce }
    STATUS  current
    DESCRIPTION
      "<NMEA once> Get NMEA messages Once."
  ::= { ioOutput 254 16 }

tabNMEAPrecision OBJECT-GROUP
    OBJECTS {
      NMEAPrecision }
    STATUS  current
    DESCRIPTION
      "<NMEA Precision> Set NMEA Precision."
  ::= { ioOutput 254 21 }

tabDifCorOutRTCM2 OBJECT-GROUP
    OBJECTS {
      RTCMv2DataForm,
      RTCMv2DataInterv,
      RTCMv2DataOutput,
      RTCMv2Message16 }
    STATUS  current
    DESCRIPTION
      "<RTCM2> Settings for RTCM2 differential Output."
  ::= { ioOutput 254 51 }

tabDifCorOutRTCM3 OBJECT-GROUP
    OBJECTS {
      RTCMv3DataForm,
      RTCMv3DataInterv,
      RTCMv3DataOutput }
    STATUS  current
    DESCRIPTION
      "<RTCM3> Settings for RTCM3 differential Output."
  ::= { ioOutput 254 56 }

tabDifCorOutCMR2 OBJECT-GROUP
    OBJECTS {
      CMRv2DataForm,
      CMRv2DataInterv,
      CMRv2DataOutput,
      CMRv2Message2 }
    STATUS  current
    DESCRIPTION
      "<CMR2> Settings for CMRv2 differential Output."
  ::= { ioOutput 254 61 }

subDifCorIn OBJECT-GROUP
    OBJECTS {
      RTCMv2DataUsage,
      RTCMv3DataUsage,
      CMRv2DataUsage }
    STATUS  current
```

```
    DESCRIPTION
      "<Dif. Input Selection> Settings for differential input."
  ::= { ioInput 255 1 }

tabDifCorInRTCM2 OBJECT-GROUP
    OBJECTS {
      RTCMv2DataUsage }
    STATUS  current
    DESCRIPTION
      "<RTCM2> Settings for RTCMv2 differential Input."
  ::= { ioInput 254 1 }

---->
$-- BLOCK 5 / 0

tabDifCorInRTCM3 OBJECT-GROUP
    OBJECTS {
      RTCMv3DataUsage }
    STATUS  current
    DESCRIPTION
      "<RTCM3> Settings for RTCMv3 differential Input."
  ::= { ioInput 254 5 }

tabDifCorInCMR2 OBJECT-GROUP
    OBJECTS {
      CMRv2DataUsage }
    STATUS  current
    DESCRIPTION
      "<CMR2> Settings for CMRv2 differential Input."
  ::= { ioInput 254 9 }

-- Navigation groups
Navigation           OBJECT IDENTIFIER ::= { rxff 2 }
navConf              OBJECT IDENTIFIER ::= { Navigation 10 }
--    G <Configuration> "Manage configuration files"
navPosMode           OBJECT IDENTIFIER ::= { Navigation 20 }
--    G <Positioning Mode> "Positioning Mode related settings"
    PVTMode                OBJECT IDENTIFIER ::= { navPosMode 10 }
--    - <PVT Mode> "Set PVT Mode"
    StaticPosGeodetic      OBJECT IDENTIFIER ::= { navPosMode 20 }
--    - <Static Pos Geodetic> "Set static geodetic position "
    StaticPosCartesian     OBJECT IDENTIFIER ::= { navPosMode 30 }
--    - <Static Pos Cartesian> "Set static cartesian position "
    SBASCorrections        OBJECT IDENTIFIER ::= { navPosMode 40 }
--    - <SBAS Corrections> "Set SBAS corrections."


         …
```

Figure 3-3: **lstMIBDescription, Overview**

OBJECT IDENTIFIER: gives the full SNMPv2 OID definition for the item which name precedes 'OBJECT IDENTIFIER' e.g:

```
ssn                  OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 27073 }
rxff                 OBJECT IDENTIFIER ::= { ssn 5 }
I/O                  OBJECT IDENTIFIER ::= { rxff 1 }
ioOutput             OBJECT IDENTIFIER ::= { I/O 20 }
--    G <Output Settings> "Output settings for SBF, NMEA, ..."
RTCMv2DataForm            OBJECT IDENTIFIER ::= { ioOutput 40 }
--    - <RTCM2 Data Formatting> "Set Reference ID, ..."
```

Figure 3-4: OID definition

Results in a full SNMPv2 OID for the command RTCMv2DataForm (which is part of the group ioOutput under the I/O menu of a rxff type of receiver) of:

- 1.3.6.1.4.1.27073.5.1.20.40

The resulting SNMP' OID only starts from the menu level resulting in OID:

- .1.20.40

-- G : is a comment about the Group OID that's defined above it. It suggests:

- <Label> : to be used as text within menu for the group on the GUI.
- "Tooltip" to be used as help text/tooltip in the GUI.

-- X :
-- - :

-- ~ : is a comment about the command OID that's defined above it. It suggests:

- X : Default ON checkbox to be used with command in the GUI
- - : Default OFF checkbox to be used with command in the GUI
- ~ : No checkbox t be used with command in the GUI
- <Label> : to be used as text for the command within the GUI window.
- "Tooltip" : to be used as help text/tooltip in the GUI.

Commands with no checkbox are always sent when <OK> or <APPLY> is selected in a window. For commands with a checkbox: these are only sent to the receiver when the checkbox is on when selecting <OK> or <APPLY>.

```
subSBFOut OBJECT-GROUP
    OBJECTS {
      SBFGroups,
      SBFOutputOnce }
    STATUS  current
    DESCRIPTION
      "<SBF> Settings for SBF output."
  ::= { ioOutput 255 1 }
```

Figure 3-5: OBJECT-GROUP : sub

```
tabDifCorOutRTCM3 OBJECT-GROUP
    OBJECTS {
      RTCMv3DataForm,
      RTCMv3DataInterv,
      RTCMv3DataOutput }
    STATUS  current
    DESCRIPTION
      "<RTCM3> Settings for RTCM3 differential Output."
  ::= { ioOutput 254 56 }
```

Figure 3-6: OBJECT-GROUP : tab

OBJECT-GROUP: object groups are used to group object ID's that somehow belong together. In this case they contain suggestions for grouping commands into either sub's or tab's.
At Septentrio we reserve :

- 254 for tab's (See **Error! Reference source not found.**)
- 255 for sub's (See Figure 3-2: Suggested menu structure.)

Please note that although the 'DESCRIPTION' field is a free format field in SNMPv2, Septentrio uses a proprietary format for SNMP'.
The text between <> is a suggestion to be used as name in the GUI. The following text is a suggestion for the lmd text in the GUI.

3.3 lstMIBDescription, CommandName

```
COM1>lmd, gtm
$R; lstMIBDescription, getTroposphereModel
$-- BLOCK 1 / 0
-- ----------------------------------------
-- TroposphereModel ARGUMENTS
--

---->
$-- BLOCK 2 / 0
tmZenith   OBJECT-TYPE
    SYNTAX        Integer32 {
        off(0),
        Saastamoinen(1),
        Hopfield(2) }
    UnitParts     " , , , "
    MAX-ACCESS    read-write
    STATUS        deprecated
    DESCRIPTION
      "Zenith model"

    DEFVAL        { 1 }
  ::= { TroposphereModel 1 }


---->
$-- BLOCK 3 / 0
tmMapping   OBJECT-TYPE
    SYNTAX        Integer32 {
        Neil(1) }
    UnitParts     " , , , "
    MAX-ACCESS    read-write
    STATUS        deprecated
    DESCRIPTION
      "Mapping model"

    DEFVAL        { 1 }
  ::= { TroposphereModel 2 }


---->
$-- BLOCK 4 / 0
tmTemperature  OBJECT-TYPE
    SYNTAX        Integer32 (-1000..1000)
    UnitParts     "0.1, , , "
    MAX-ACCESS    read-write
    STATUS        deprecated
    DESCRIPTION
      "Temperature"

    DEFVAL        { 150 }
  ::= { TroposphereModel 3 }


---->
$-- BLOCK 5 / 0
tmPressure  OBJECT-TYPE
    SYNTAX        Integer32 (-80000..150000)
    UnitParts     "0.01, , , "
    MAX-ACCESS    read-write
    STATUS        deprecated
    DESCRIPTION
      "Pressure"

    DEFVAL        { 101325 }
  ::= { TroposphereModel 4 }


---->
$-- BLOCK 6 / 0
tmHumidity  OBJECT-TYPE
    SYNTAX        Integer32 (0..100)
    UnitParts     "1, , , "
    MAX-ACCESS    read-write
    STATUS        deprecated
    DESCRIPTION
      "Humidity"

    DEFVAL        { 50 }
  ::= { TroposphereModel 5 }

-- ----------------------------------------
COM1>
```

Figure 3-7: **lstMIBDescription, commandname**

The example above gives an example of a simple command with 5 arguments.

Following types of arguments are supported:

- FInt32:32 bit integer in a predefined range from minimal value to a maximum value - a precision is used to represent floats/doubles as fixed point integers;
- FInt64:64 bit integer in a predefined range from minimal value to a maximum value - a precision is used to represent floats/doubles as fixed point integers;
- Enum32: Limited number of 32 bit integers each represent-able by a small text;
- BitSets: Each bit of the bit set can be represented by a small text. The bits can be concatenated to a set of bits. (The closest multiple of bytes that can represent all bits is actually used.);
- String: Text string with predefined maximum length.

Please note that although the 'UnitParts' field is a free format field in SNMPv2, Septentrio uses a proprietary format for SNMP'.
It uses a CSV (**C**omma **S**eparated **V**alues) format with 4 fields:

- Field1:
  - Precision is only used for FInt32 and FInt64. For the other types this one is empty. It contains different data depending:
    - FInt32 : <precision> 1 or 0.1 or 0.001 or 0.0001 or …
    - FInt64: <precision> (<from>..<till>)
  - Bytes_Used gives the number of bytes used for a BITS bitset.
- Field2: <Suffix> optional field containing the units e.g. meter or deg.
- Field3: <Prefix> optional field containing a hint for a label in the GUI for the argument
- Field4: OR-able concatenation of GUI hints through '|'. See Table 3-1: Extensible table of GUI hints.

<GUI>	Comment
PX	X-position
PY	Y-position
PZ	Z-position
PLAT	Latitude-position
PLON	Longitude-position
PALT	Altitude-position
NL	Show data non-localised
DATE_TIME	Parse/Show as data time string
TABLE_SWAP	The first argument forms the row headers i.s.o the column headers.
UNIT_SWAP	Swap the units for stringlists (eg "ms1000" becomes "1000 ms")
…	

Table 3-1: Extensible table of GUI hints

3.3.1 FInt32

ASCII Command Line Interface:

- Floats must be limited between a minimum and a maximum number;
- There is to be defined a precision;
- A default value that is within the valid range is to be defined;
- There is no support for scientific notation;
- In the MIB the value will be stored as a fixed point. On the ASCII command line the floating value is used though;
- (The floating point representation may need to be a double as float has a precision of less than 7 decimal digits).

MIB language translation:

```
<NameOfArgument>  OBJECT-TYPE
    SYNTAX            Integer32 (<MinVal>..<MaxVal>)
    UnitParts         "<precision>, <suffix>, <prefix>, <GUI|GUI|…>"
    MAX-ACCESS        <'read-write' 'read-create' or 'read-only'>
    STATUS            <'current'/'deprecated' for 'mandatory'/'optional'>
    DESCRIPTION
        "Textual description of the argument"
```

```
    DEFVAL              { <Default Value> }
  ::= { <NameOfCommand> <Argument number> }
```

Figure 3-8: FInt32

Please note that 'read-create' for Septentrio receivers is used for arguments that have a callback function attached.

E.g. the pressure for the troposphere model in Figure 3-7: **`lstMIBDescription, commandname`** gives 101325 (precision 0.01) as default value. This means that SNMP' will use the integer 101325. In the ASCII interface one types in 1013.25 though.

See Table 2-4: Payload.

3.3.2 FInt64

ASCII Command Line Interface:

- Doubles must be limited between a minimum and a maximum number;
- There is to be defined a precision;
- A default value that is within the valid range is to be defined;
- There is no support for scientific notation;
- In the MIB the value will be stored as a fixed point. On the ASCII command line the floating value is used though;

MIB language translation:

```
Integer64 ::= TEXTUAL-CONVENTION
    STATUS              current
    DESCRIPTION
        "Litle endian representation of a 2's complement 64 bit integer
         The serialisation of the value 15 as a result would give:
         '0F00000000000000'H "
    SYNTAX              Opaque  (SIZE(8))



<NameOfArgument>  OBJECT-TYPE
    SYNTAX              Integer64
    UnitParts  "<precision> (<MinVal>..<MaxVal>), <suffix>, <prefix>, <GUI|GUI|…>"
    MAX-ACCESS         <'read-write' 'read-create' or 'read-only'>
    STATUS             <'current'/'deprecated' for 'mandatory'/'optional'>
    DESCRIPTION
        "Textual description of the argument"
    DEFVAL              { { '<LSByte>'H, '<>'H, '<>'H, '<>'H,
                            '<>'H, '<>'H, '<>'H, '<MSByte>'H } }
  ::= { <NameOfCommand> <Argument number> }
```

Figure 3-9: FInt64

Please note that 'read-create' for Septentrio receivers is used for arguments that have a callback function attached.

Please note that Integer64 doesn't exist within SNMPv2. The Integer64 used is a Septentrio customization through the TEXTUAL-CONVENTION. On receivers with the need for 64 bit values the textual convention will be added in the "lstMIBDescription, Overview" and will not be repeated for each separate argument using a 64 bit value.

See Table 2-4: Payload.

3.3.3 Enum32

ASCII Command Line Interface:

- Every name has a maximum of 31 characters (=32 byte \0-terminated);
- The value associated with a name is in the following range (–2147483648 .. +2147483647);
- There's no limitation in the number of names that can be defined other than the addressable/available amount of memory;
- Following characters are allowed within names:
  ABCDEFGHIJKLMNOPQRSTUVWXYZ
  abcdefghijklmnopqrstuvwxyz
  0123456789|
- The first character within a name may come only out of the following characters though:
  ABCDEFGHIJKLMNOPQRSTUVWXYZ
  abcdefghijklmnopqrstuvwxyz
  This to allow for discontinuous ranges like for instance baudrate "B300" "B600" "B1200" "B2400" …;
- The decimal value that the different names represent can also be typed in.

MIB language translation:

```
<NameOfArgument>  OBJECT-TYPE
    SYNTAX            Integer32 {
                    <Name1>(<value1>), <Name2>(<value2>), …
                    <NameX>(<valueX>) }
    UnitParts         " , <suffix>, <prefix>, <GUI|GUI|…>"
    MAX-ACCESS        <'read-write', 'read-create' or 'read-only'>
    STATUS            <'current'/'deprecated' for 'mandatory'/'optional'>
    DESCRIPTION
        "Textual description of the argument"
    DEFVAL            { <value> }
  ::= { <NameOfCommand> <Argument number> }
```

Figure 3-10: Enum32

Please note that 'read-create' for Septentrio receivers is used for arguments that have a callback function attached.

See Table 2-4: Payload.

3.3.4 BITS

ASCII Command Line Interface:

- Every name has a maximum of 31 characters (=32 byte \0-terminated);
- The BITS actually uses an OCTET STRING to put the bits in and as such any number of bytes can be used. (For instance for a 10 bit BITS a char[2] will be used);
- There's no limitation in the number of names that can be defined other than the addressable/available amount of memory;

- Following characters are allowed within names:
  ABCDEFGHIJKLMNOPQRSTUVWXYZ
  abcdefghijklmnopqrstuvwxyz
  0123456789|
- The first character within a name may come only out of the following characters though:
  ABCDEFGHIJKLMNOPQRSTUVWXYZ
  abcdefghijklmnopqrstuvwxyz
- The names can be concatenated with a '+' character in between them;

MIB language translation:

```
<NameOfArgument>  OBJECT-TYPE
    SYNTAX            BITS {
                    <Name0>(<bitnr0>), <Name1>(<bitnr1>), …
                    <NameX>(<bitnrX>) }
    UnitParts          "<bytes_used> , <suffix>, <prefix>, <GUI|GUI|…>"
    MAX-ACCESS         <'read-write' 'read-create' or 'read-only'>
    STATUS            <'current'/'deprecated' for 'mandatory''optional'>
    DESCRIPTION
        "Textual description of the argument"
    DEFVAL            { {  <Name1>, <Name2> } }
  ::= { <NameOfCommand> <Argument number> }

-- Note that in this case the default value = name1 + name2
```

Figure 3-11: BITS

Please note that 'read-create' for Septentrio receivers is used for arguments that have a callback function attached.

See Table 2-4: Payload.

3.3.5 String

ASCII Command Line Interface

- Strings can have any arbitrary length;
- Strings can contain the following characters:
  ABCDEFGHIJKLMNOPQRSTUVWXYZ
  abcdefghijklmnopqrstuvwxyz
  0123456789 %'()+-./:;<=>?[]_{|}
- The space character is allowed only when the entire string is enclosed in quotes (").

MIB language translation:

```
<NameOfArgument>  OBJECT-TYPE
    SYNTAX            OCTED STRING (SIZE(<MIN>..<MAX>))
    UnitParts         " , <suffix>, <prefix>, <GUI|GUI|…>"
    MAX-ACCESS        <'read-write' 'read-create' or 'read-only'>
    STATUS            <'current'/'deprecated' for 'mandatory''optional'>
    DESCRIPTION
        "Textual description of the argument"
    DEFVAL            { "<text>" }
  ::= { <NameOfCommand> <Argument number> }
```

Figure 3-12: String

Please note that ‘read-create’ for Septentrio receivers is used for arguments that have a callback function attached.

See Table 2-4: Payload.

3.3.6 Tabular Commands

Septentrio only supports 2 dimensional tables. The first argument is always used as the table index.
Commands that represent a table need a predefinition of the number of rows and columns in the table.

The argument 1 is always of the type Enum32.

The description for this predefinition is put on the commandnumber OID. The description of which argument to use as index follows the command OID hence the '.1.' in the OID (See Table 2-3: Variable Binding):

.mibnumber.groupnumber.commandnumber.1.argument.index

'SEQUENCE' and 'SEQUENCE OF' are used to create the predefinition of the table.

```
COM1>lmd, gsi
$R; lstMIBDescription, getSmoothingInterval
$-- BLOCK 1 / 0
-- ----------------------------------------
-- SmoothingInterval TABLE (predef, command, index)
--
smoothingIntervalEntry::= SEQUENCE {
    siSignal              Integer32,
    siInterval            Integer32,
    siAlignment           Integer32 }


---->
$-- BLOCK 2 / 0
SmoothingInterval  OBJECT-TYPE
    SYNTAX       SEQUENCE OF smoothingIntervalEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
      "Command Table Entry"
  ::= { navRecOp 10 }


---->
$-- BLOCK 3 / 0
SmoothingIntervalEntry  OBJECT-TYPE
    SYNTAX       smoothingIntervalEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
      "Index = argument used as Entry"
    INDEX        { siSignal }
   ::= { SmoothingInterval 1 }


---->
$-- BLOCK 4 / 0
-- ----------------------------------------
-- SmoothingInterval ARGUMENTS
--

---->
$-- BLOCK 5 / 0
siSignal   OBJECT-TYPE
    SYNTAX       Integer32 {
        GALL1A(1),
        GALL1BC(2),
        GALE5A(3),
        GALE5B(4),
        GALE5(5),
        GALE6A(6),
        GALE6BC(7),
        GPSL1CA(8),
        GPSL1_L2P(9),
        GPSL2C(10),
        GEOL1(11),
        GLOL1CA(12),
        GLOL1_L2P(13),
        GLOL2C(14) }
    UnitParts    " , , , "
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
      "GNSS signal type"
```

```
    DEFVAL       { 0 }
  ::= { SmoothingIntervalEntry 1 }


---->
$-- BLOCK 6 / 0
siInterval  OBJECT-TYPE
    SYNTAX       Integer32 (0..1000)
    UnitParts    "1, , , "
    MAX-ACCESS   read-write
    STATUS       deprecated
    DESCRIPTION
      "Value for smoothing interval"

    DEFVAL       { 0 }
  ::= { SmoothingIntervalEntry 2 }


---->
$-- BLOCK 7 / 0
siAlignment  OBJECT-TYPE
    SYNTAX       Integer32 (0..1000)
    UnitParts    "1, , , "
    MAX-ACCESS   read-write
    STATUS       deprecated
    DESCRIPTION
      "Alignment value for smoothing interval"

    DEFVAL       { 0 }
  ::= { SmoothingIntervalEntry 3 }

-- ----------------------------------------
COM1>
```

Figure 3-13: Example Tabular Command

‘SEQUENCE’: As the table entry is defined before the definition of the actual arguments a predefinition of the arguments is needed before the actual table entry can be defined. The SEQUENCE is merely a list of arguments that are going to be defined later on.

‘SEQUENC OF’: Defines the command as a table. This is actually done at the OID:
.mibnumber.groupnumber.commandnumber

‘INDEX’: Table Entry: Defines that the first argument is going to be used as an index in the table. This is done at OID:
.mibnumber.groupnumber.commandnumber.1

Lets say for SmoothingInterval entry above:

- Mibnumber = 2 = Navigation
- Groupnumber = 30 = navRecOp
- CommandNumber = 10 = SmoothingInterval

Then the OID’s for the following ASCII command is:

- **setSmoothingInterval, GALE5, , 100**
- Sets Alignment argument to 100 for 5th element of the array of Alignment values.

- OID = **.** Navigation **.** navRecOp **.** SmoothingInterval **.** SmoothingIntervalEntry **.** siAlignment **.** 5
- OID = **.** 2 **.** 30 **.** 10 **.** 1 **.** 3 **.** 5