

## Sabine Navigator Communication Protocol (October 19, 2007)

This document applies to the following Sabine Navigator Models:  
NAV240, NAV360, NAV480, NAV4802, NAV8802.

### Serial Port Settings

115200 baud is the default baud rate. This is changeable via the front panel if desired.

No start bit, 8 data bits, **2 stop bits**.

IE **115200,N,8,2**

### General structure of communication packets:

<0x01><R/W><SRC><COMMAND HEADER><VALUE>...<0x1F><CHKSUM><0x02>

1 1 1 4 1 1 1 1<= sizes in readable ascii bytes

<R/W> - Read = 4 (ascii 0x24), Write = 5 (ascii 0x25)

<SRC> - Device number which sent packet: <0x7f> => PC

The <COMMAND HEADER> is used to identify the types of value following the header. The value of the header indicates the number of readable ascii bytes in the value that follows.

NAME	CODE	Comment
Command Header	<0x03 to 0x07>; 0x03 = 4 bytes ... 0x07 = 8 bytes	How many bytes follow for the command name (usually 4)
Device Header	<0x08> 1 byte	
I/O Header	<0x09> 1 byte	
Channel Header	<0x0A> 1 byte	
Aux Header	<0x0B> 1 byte	
Column Header	<0x0C> 1 byte	Always set to 0
Data Header	<0x10 to 0x18>; 0x10 = 1 byte ... 0x18 = 8 bytes	# of bytes that follow

<VALUE> must always follow the <HEADER>.

Command Value	Command ID in ASCII character (see table below)
Device Value	Device # where this command is sent to
I/O Value	0x20 for Input, 0x21 for Output
Channel Value	Channel #, zero-based
Aux Value	For extra information such as EQ Num, FBX Num ,etc
Column Value	Reserved for later use

Data Value	The actual value to change for the command targeted
------------	---

There is no limit in the number of <HEADER><VALUE> combo as long as the whole block (from 0x01 to 0x02) is less than 256 readable ASCII bytes. When a <HEADER><VALUE> is received, the corresponding new parameters are stored in the memory. However, it is not processed until a <0x1F> is received.

Format used for data values is readable ASCII (96 values for each character) unless specified otherwise.

Data Value	ASCII representation
0	0x20
:	:
95	0x7F

In other words, to send a data value of 0, send 0x20. To send a data value of 1, send 0x21, etc.

**Checksum calculation:**

1. Add up all of the bytes from the first (0x01) to the last byte before the check sum.
2. AND this result with 0xFF
3. Take the modulus this result with a base of 0x60 (hex 60 or decimal 96)
4. Add 0x20 to this value to get the ASCII representation
5. Use the hex value of this result for the check sum

Checksum Example (in Hexadecimal)

The string below mutes output 3. 0x53 is the checksum.

<01><25><7F><03><4D><55><54><30><08><20><09><21><0A><22><0B><20><0C><20><10><21> <1F><**53**><02>

1. Adding up all the bytes from 0x01 to 0x1F, the raw hex checksum is 0x2F3.
2. AND 0x2F3 with 0xFF: 0x2F3 AND 0xFF = 0xF3
3. Now take the modulus base 0x60: 0xF3 MOD 0x60 = 0x33.
4. Add 0x20 to 0x33 for the ASCII representation: 0x20+0x33=0x53. So the checksum is **0x53**.

Another Example:

To set the gain to 0dB on input 1, device 1, the string would be (in HEX):

<1><25><7F><3><4C><56><4C><30><8><20><9><20><A><20><B><20><C><20><11><21><60><1F><69><2>

- 1 is the start byte
- 25 means a write command follows
- 7F means the command came from the PC
- 3 means the command that follows is 4 bytes long
- 4C 56 4C 30 is ASCII for "LVL0", the signal level (gain) command code

**8 20** means the command is addressed to device 0

**9 20** means the command is addressed to the inputs

**A 20** means the command is addressed to channel 1

**B 20** means the aux value is 0 (unused for signal gain, but still required in the command string)

**C 20** means the column value is 0 (unused for signal gain, but still required in the command string)

**11** is the data header and means that 2 data bytes follow

**21 60** is the value: 160.

**1F** is the execute byte

**69** is the checksum

**2** is the end byte

#### **For the data value:**

160 is the 0db value for gain. The range of values is 0 to 220, corresponding to gain values of -40db to +15 db in 0.25 dB steps.

The data values are encoded as ASCII values and have a maximum value of 96, so the gain value of 160 requires 2 bytes to represent it. It is calculated like this:

Take 160 and divide by 96: the remainder is 64, so the MS byte is 1 and the LS byte is 64. Convert the values to ASCII representations by adding 0x20 and the result is (in hex) **21 60**

#### **For the checksum:**

1. Adding up all the bytes from 0x01 to 0x1F, the raw hex checksum is 0x349.

2. AND 0x349 with 0xFF: 0x349 AND 0xFF = 0x49

3. Now take the modulus base 0x60: 0xF3 MOD 0x49 = 0x49.

4. Add 0x20 to 0x33 for the ASCII representation: 0x20+0x49=0x69. So the checksum is **0x69**.

#### Another Example (in Hexadecimal):

This string will mute input 2 of Device 1 (zero-based):

```
<01><25><7F><03><4D><55><54><30><08><20><09><21><0A><22><0B><20><0C><20><08><20>  
<09><20><0A><21><0B><20><0C><20><10><21><1F><CS><02>
```

The mute will be turned on as specified by <10><21>. Notice that the mute of Output 3 of Device 1 is not affected because there is no <1F> right after it, so the target of the MUT0 is overridden by the second target

#### Another Example (in Hexadecimal):

This mutes output 3 of Device 1 and also input 2 of Device 1

```
<01><25><7F><03><4D><55><54><30><10><21><08><20><09><21><0A><22><0B><20><0C><20>  
<1F><08><20><09><20><0A><21><0B><20><0C><20><1F><CS><02>
```

This example actually puts the DATA before the TARGET. When the first <1F> is received, Output 3 of Device 1 will be

muted, when the second <1F> is received, Input 2 of Device 1 will be muted as well.

<b>NORMAL COMMAND</b>	<b>ID</b>
Meter	MTR0
Comp. Gain Reduct. Meter	CME0
Mute	MUT0
Signal Level	LVL0
Signal Polarity	POLO
Signal Delay (100ms@48k)	DLY0
Signal Delay (90ms@48k)	DLY1
Signal Delay (200ms@48k)	DLY2
Signal Delay (80ms@48k)	DLY3
EQ Type	EQT0
EQ Frequency	EQF0
EQ Bandwidth	EQB0
EQ Level	EQL0
Mic Trim Gain	MIC0
Crossover Type	XRT0
Crossover Frequency	XRF0
Crossover Slope	XRS0
Compressor Threshold	CMT0
Compressor Attack	CMA0
Compressor Release	CMR0
Compressor Ratio	CMX0
Bypass Delay	BPDY
Bypass EQ	BPEQ
Bypass Compressor	BPCP
Bypass Crossover	BPXR
Mixer	MIXA
Channel Name	CHN0

<b>SYSTEM COMMAND</b>	<b>ID</b>
Down Sync	%SD0
Up Sync	%SU0
Lock Password	%LP0
Lock Key	%LK0
Channel In Number	%nI0
Channel Out Number	%nO0
Company	%NC0
Sampling Frequency	%nS0
Product Name	%NP0
Version	%NV0
Device Name	%ND0
Device Number	%nD0
Program Number	%Pn0
Program Name	%PN0
Program Recall	%PR0
Program Store	%PS0
Program Download	%PD0
Program Upload	%PU0
RESET	%RS0
Mic Preamp On/Off	%MP0
Digital I/O	%DI0

<b>FBX COMMAND</b>	<b>ID</b>
FBX Type	FTYP
FBX Level	FLVL
FBX Frequency	FFRQ
FBX Bandwidth	FBNW
FBX Global Lock	FLCK
FBX Global Bypass	FBYP
FBX Global Mdepth	FMDP
FBX Global Width	FWID
FBX Global Sensitivity	FSEN
FBX Global Persistence	FPER
FBX Global Threshold	FTHS

FBX Setup	FSET
-----------	------

FBX Reset Dynamic	FRST
-------------------	------

**Program Download**

1. PC sends Download ON command (%PDO with data = TRUE)
2. PC starts sending each parameter (just like downsync) for program #30
3. PC sends Program Store command (%PS0 with data = 29)
4. PC receives Program Store command (%PS0 with data = 29) as acknowledgement
5. Repeats step 2 to 4 for the rest of the program in descending order
6. PC sends Download OFF command (%PDO with data = FALSE)

**Program Upload**

1. PC sends Upload ON command (%PU0 with data = TRUE)
2. PC sends Program Recall command (%PR0 with data = 29)
3. PC receives Program Recall command (%PR0 with data = 29) as acknowledgement
4. PC sends Upsync command (%SU0) to receive data for program #30  
 \*\* Upsync will always send system parameter such as Channel #, Password, etc. Simply ignore them and take the main data part \*\*
5. Repeats step 2 to 4 for the rest of the program in descending order
6. PC sends Upload OFF command (%PU0 with data = FALSE), the firmware will automatically reload the data in memory (program #31) upon completion

**Downsync (Reset All)**

1. PC sends Downsync ON command (%SD0 with data = TRUE)
2. PC starts sending a parameter to the firmware
3. PC receives Meter (%MTR0), repeat step 2 for another parameter. When all parameters are sent, goto next step.
4. PC sends Downsync OFF command (%SD0 with data = FALSE)
5. PC receives Downsync OFF command as acknowledgement (%SD0 with data = FALSE)

**Password, Program Name, Device Name**

Each character of the string is sent along with the Aux header/value. The value of the character is based on the character code map. For example, a password for "ABCD" would be packed like this:

```
<01><57><7F><03><25><4C><50><30><08><20><0B><20><10><21><1F><0B><21><10><22><0B><22><10><23><0B><23><10><24><CS><02>
```

**CHARACTER CODE MAP**

\_ABCDEFGHIJKLMN0PQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyz-<>? , / ~ ! @ # \$ % ^ & \* ( ) = + ' ; :

## **Security Lock**

Similar to Password, the menu that is to be locked is identified by the Aux header/value. The data will specify LOCK (true) or UNLOCK (false).

0 - INPUT SIG

1 - INPUT EQ

2 - INPUT FBX

3 - INPUT FBX-GLOBAL

4 - INPUT FBX-SETUP

5 - INPUT LIMIT

6 - INPUT NAME

7 - INPUT BYPASS

8 - OUTPUT SIG

9 - OUTPUT EQ

10 - OUTPUT XOVER

11 - OUTPUT LIMIT

12 - OUTPUT SOURCE

13 - OUTPUT NAME

14 - OUTPUT BYPASS

15 - PROGRAM RECALL

16 - SYSTEM

17 - MUTE