3.2 Communication Protocol

The communication protocol is designed for flexibility, both in features but also in application. It is not designed specifically for the EEG system but can be reused for other systems, too. This allows for a general parser on the software side, along with more general and reusable code. The protocol itself therefore only describes how packets are structured and how commands can be used. The actual settings change per system (for example EEG versus ECG system).

This flexibility introduces a few extra requirements such as version numbering to distinguish which commands are supported in a system and the possibility for a system to identify itself and inform the user interface / PC which system settings it supports.

3.2.1 Packet Structure

The basic packet structure is displayed in Figure 26. The protocol shows the entire packet as transferred between the headset and the PC. This part of the protocol is fixed; any system which implements this protocol should use the packet structure below.

The first 3 bytes are the preamble which is used to indicate the start of a new packet. This preamble is fixed for the protocol: it is always 3 bytes in size and always contains the text “BAN” (ASCII).

Because packet length is dynamic a length half word follows the preamble. The length half word contains the length of the payload. The length field is 16 bits long and formatted in little-endian.

The payload always starts with a command byte which contains a command or the packet type. Depending on the command a payload can be added ranging from 1 to 65535 bytes.

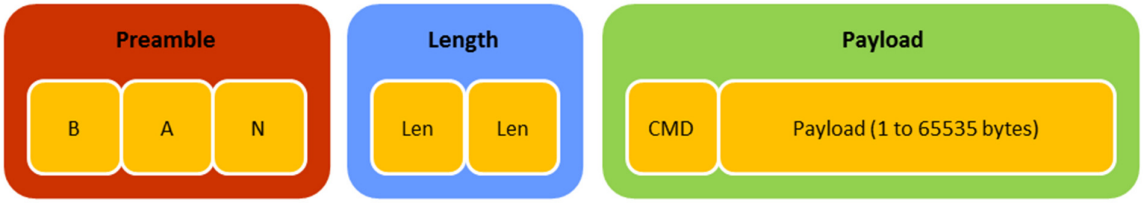

Figure 26 – Basic packet structure

Currently 8 commands/packet types are supported: request settings, change settings, data format, data packets, error (while attempting to change a setting), enumerate all settings, get information about the settings and optional remarks. The command bytes are listed in Table 8. The command bytes are listed as char, hex value and decimal respectively. Note that commands are case sensitive.

Table 8 – Command Listing

Command	**Command byte**		
	ASCII	HEX	DEC
Request setting	‘g’	0x67	103
Change setting	‘s’	0x73	115
Flash	‘f’	0x66	146
Error with change	‘x’	0x78	170
Data packet	‘d’	0x64	100
Enumerate Settings	‘e’	0x65	101
Settings Information	‘i’	0x69	105
Settings Remarks	‘r’	0x72	114

3.2.2 Settings structure

In this document the terminology around settings may be a bit confusing. Terms used are defined as following:

Setting – some characteristic feature that defines a part of the functionality about the system. For example “Firmware version”, “Sample frequency” or “Gain”.

Option – specific value of a setting, either fixed or variable where multiple options apply to one setting. For Example: “2.5” is the ‘option’ for the setting firmware version. “1024” and “512” are both settings for the sample frequency.

Key – same as setting usually used together with value (below).

Value – same as option usually used together with key. In example “Gain” is the key where “1200” is its current value out of the set of different options.

Settings and options are implemented as string and should therefore always be NULL-terminated.

Settings are implemented in a tree-like structure. Branches are defined with the *forward-slash* character “/”. On application level settings apply on all subgroups unless redefined at a deeper branch. For example, when the branch itself has on/off options, all lower branches and leaves are enabled / disabled by changing the setting of the branch. The values of the lower branches and settings keep their current value when the group is disabled and can even be changed or requested but have no effect until the group is switched on.

3.2.3 Bootloader / Startup

When the system is powered it will initially enter the bootloader mode to ensure the user is always able to overwrite the software even when the current software is corrupt. When connecting to a system which is still in bootloader mode it will send a command of 22 bytes with the command ‘b’ of bootloader. The packet is displayed in Figure 27.

Whenever this command is received one should respond with a “Startup” command to exit the bootloader and enter the main program. The layout of the startup command is displayed in Figure 28 - Startup command packet. This is a special command which should only be used to exit the bootloader. Note this command does not use a 1 byte command but instead directly sends the null-terminated string “Startup’.

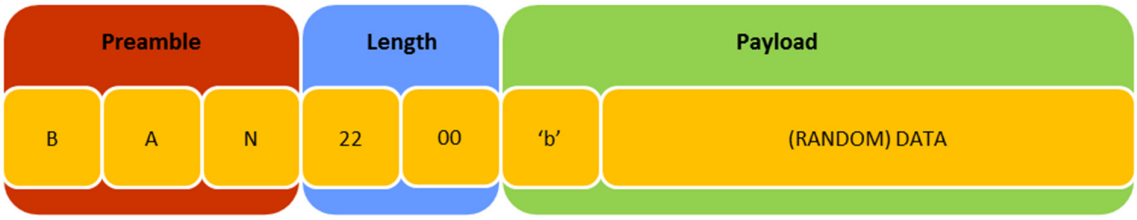

Figure 27 – Bootloader packet

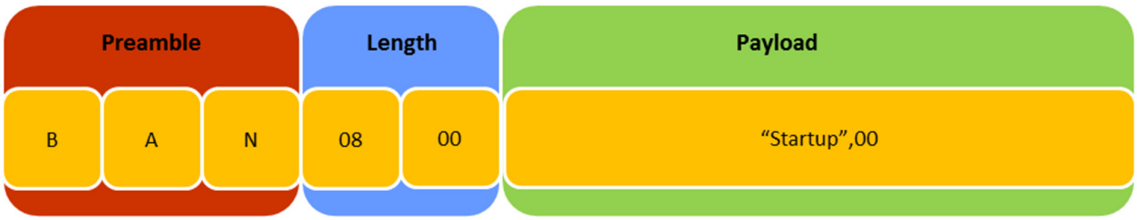

Figure 28 - Startup command packet

3.2.4 Request setting (get)

A get request will always contain one string: the name of the setting which is requested. The “get” packet from the PC to the node is always a get request. A “get” packet from the headset to the PC is always a response to an information request from the PC. An example of a setting request packet is shown in Figure 29.

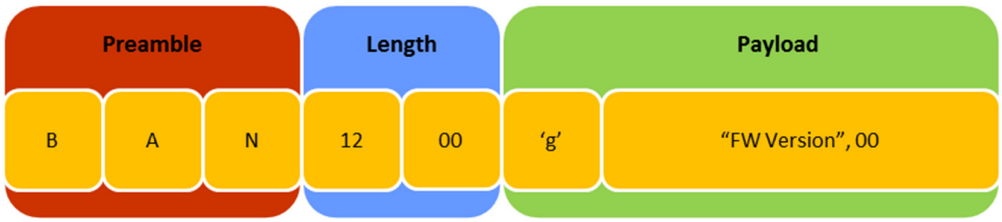

Figure 29 – Firmware Version request example

The preamble is fixed as “BAN”. The length in this example is set to 12 this is the length of command (1) + length of the key “FW Version” (10) + length of null-byte (1). The command is ‘g’ or get/request setting. The key in the example is set to “FW Version” which is the command to request the firmware version, followed by the mandatory null-byte.

An example response by the system is shown in Figure 30. The length in this case is 18. The command is repeated so the user will know this packet is the response to a “get setting” request. The data in a get response always consists of 2 strings: the first is the original requested setting (key) and the second one is the actual/current setting (value). Both strings are terminated by a null-byte and both null-bytes are included in the length byte.

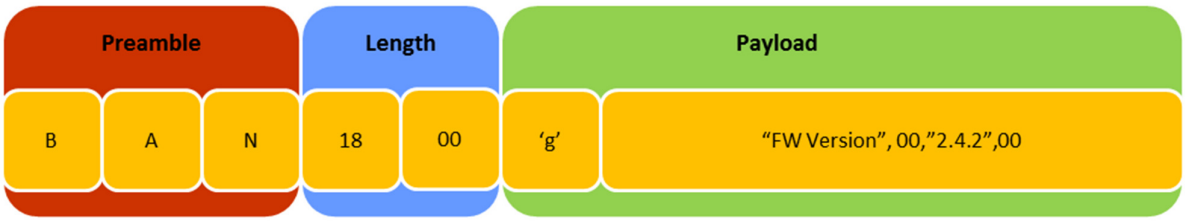

Figure 30 – Possible response to firmware version request

3.2.5 Change setting (set)

A change setting packet has a similar structure as the settings request response, however it will always contain 2 strings: the name of the setting that should be changed and the value to what it should be changed. Set packets from the PC to the system always indicate a request to change a setting. Set packets from the system to the PC can either be a response to a set command or as an indication that a setting has changed (for example by external input).

An example of a “set” packet is shown in Figure 31. In this example the gain of the sensor node is set to 1200. The length is set to 11 (1 + 4 + 1 + 4 + 1). The payload consists of 2 strings, the setting that has to be changed (key) and the new setting (value). In this example those two strings are “Gain” and “1200”, both terminated by a null-byte.

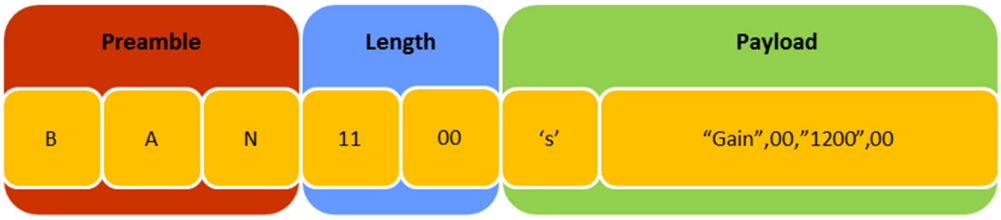

Figure 31 – Change Gain example

Assuming this is a valid command executed by the system a response from the system will be identical to original request (except it travels from system to PC instead of the other way around). The command in the response packet is ‘s’ to indicate that an attempt has been made to change a setting. The key remains the key that was in the original packet and the value is the current active value which is 1200 after the change.

3.2.6 Flash setting

The flash command is completely similar to the set command except the command ‘f’ is used instead of ‘s’. The result is quite different however. Settings changed with the flash command will be stored in the flash memory and will retain their value even when the system is power cycled. Future versions of the firmware will contain settings which support a combination of the two commands, flash to set the default/startup value, set to set the current value.

3.2.7 Error while attempting to change a setting

The previous paragraph described the changing of settings. When an attempt is made to change a setting to an invalid value an error is generated. The system will respond with 2 messages to the request. For this example we assume the user tries to change the gain from 1200 to 4000 (Which is not a valid value for gain).

First the system will return the request with the command changed to ‘x’ indicating an error. This packet is shown in Figure 32. Error packets (when attempting to change a setting) are only sent from the system to the PC.

After this first reply a “set” packet is sent with the actual value of the setting on which the change failed. In this example this packet is a set gain to 1200 which is shown above in Figure 31. Please note that even though the response contains a “set” command, no action is performed by the system (except sending the reply) since no setting has actually changed.

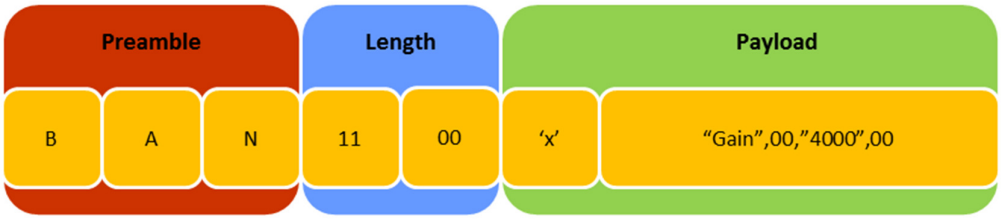

Figure 32 – Error response containing faulty request

3.2.8 Data Packets

The generic data packet structure is displayed in Figure 33 – Data packet structure. The data packets start with the command identifier 'd', followed by a timestamp corresponding to the first sample within the data packet. The ID field indicates the kind of data in the packet (EEG, DC, Accelerometer etc.). The DATA field and packet length depends on the kind of data (identified by ID).

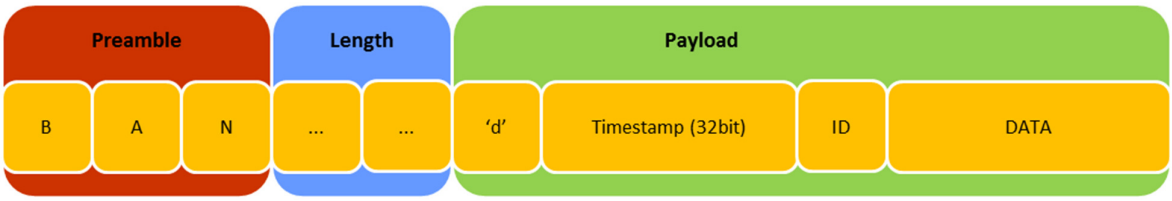

Figure 33 – Data packet structure

The current protocol supports a number of fixed data formats.

3.2.8.1 EEG Packets

The contents of the EEG packets is displayed in *Figure 34 - EEG Packet format*.

EEG packets are identified by ID 0, or 0x00. Each packet contains 4 sample blocks, each consisting of 4 EEG samples and 1 Impedance sample. Each sample (both EEG and Impedance) contains all 8 channels, of which each EEG channel is represented by 2 bytes in little endian format, and the Impedance data is a combined value of the four most significant bits of both the IMPI and IMPQ data. The first EEG + Impedance samples belong to the timestamp provided in the header of the data packet.

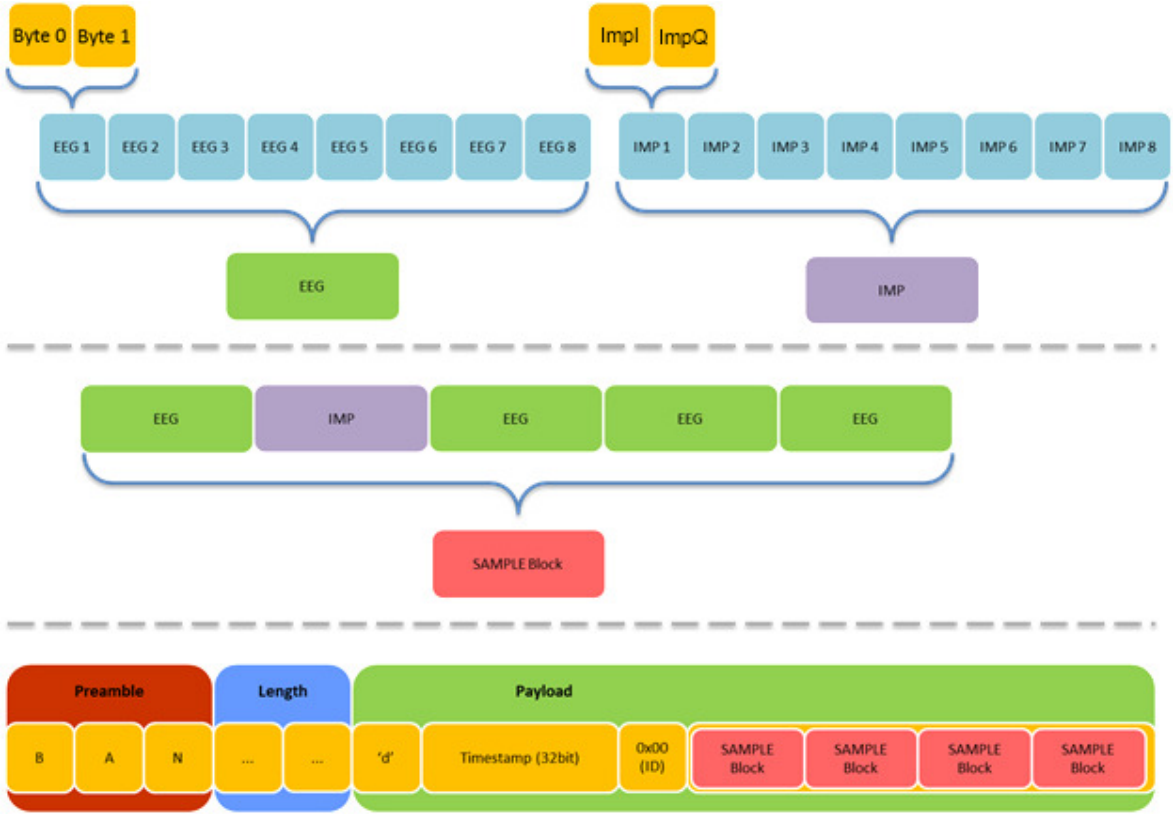

Figure 34 - EEG Packet format

3.2.8.2 DC Offset packets

The format of the DC offset packets is displayed in Figure 35. The packet contains 18 samples each consisting of the DC offset of channels 1 through 8 + reference. Each channel is represented with 2 bytes in little endian format. The identifier for a DC Offset packet is 0x20 (or 32 decimal). The timestamp in the header corresponds with the first set of samples in the packet.

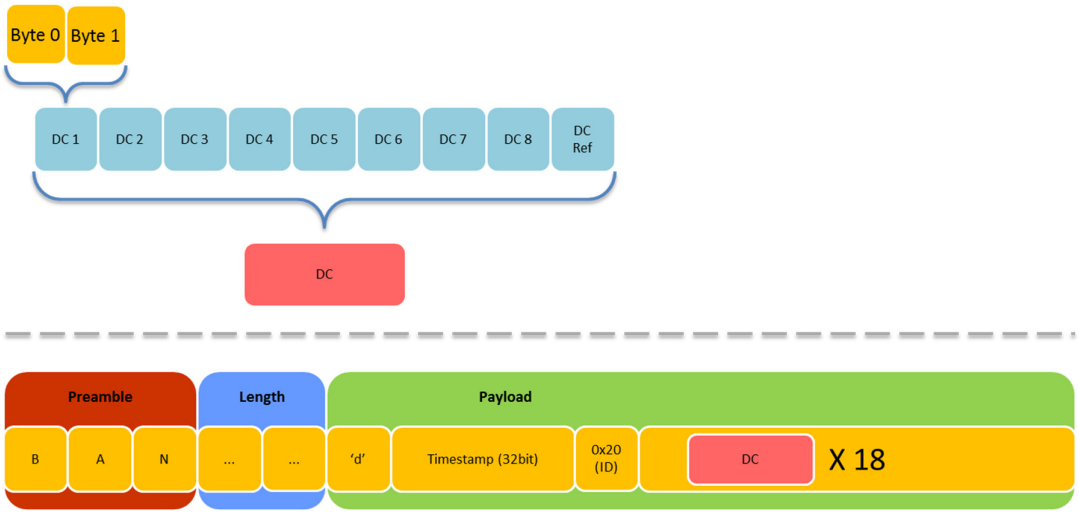

Figure 35 – DC Offset packet format

3.2.8.3 Accelerometer packet

The format of the accelerometer packet is similar to the DC Offset packet, except it contains 32 samples of only 3 channels (XYZ). The identifier for the accelerometer data is 0x10 (or 16 decimal). The format is displayed in Figure 36.

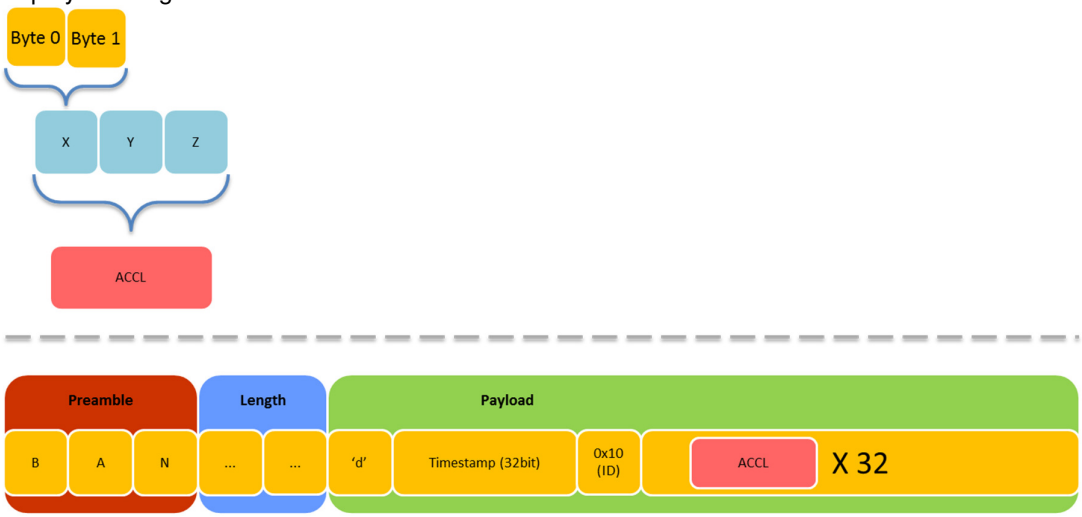

Figure 36 – Accelerometer packet format

3.2.9 Enumerate settings

As mentioned in the beginning of this chapter the protocol should provide some way for the systems to identify itself to the user interface. The protocol supports 3 commands for this: enum, information and remarks.

The enumerate command will return the full list of system settings including all options. For settings with multiple options it will also return a get command with the current active option. It is also possible to apply this command to a specific setting instead of the whole system.

The enum request packet (from Interface to Headset) is in structure similar to the get packet. Aside from the command it contains one string indicating the setting of which the enumeration is request. In Figure 37 an example is given of the enumeration request of the *"Gain"* setting. There is one significant difference between the enumeration and the get request: a get has to be performed on a valid setting or it is ignored, an enumeration request can be sent with an empty string to request a complete listing of all settings with their options. This packet consists of a 2 byte payload with only the NULL-byte following the command.

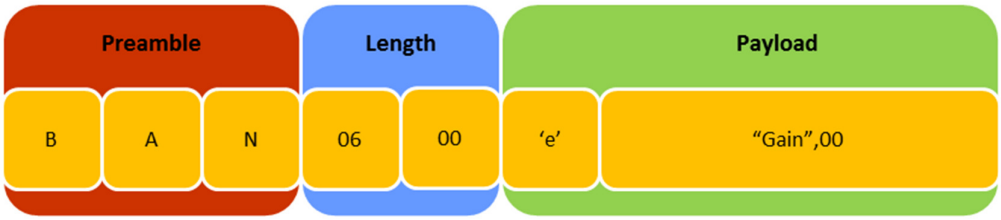

Figure 37 – Enumeration Request

An enum packet from the headset to the interface is always a response to an enumeration request. The structure is similar but not equal to the get response packet. An example response to the "*Gain*" request above is shown in

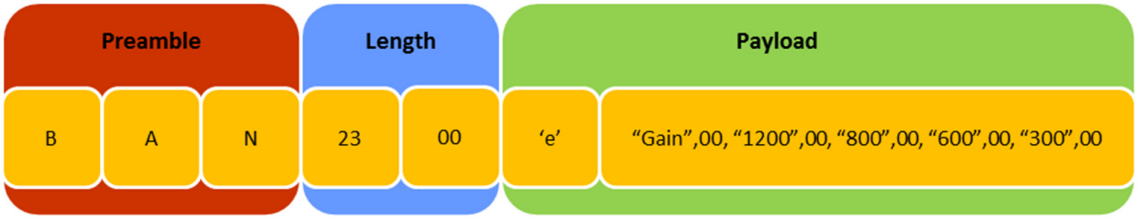

Figure 38. The response contains 2 or more strings, the first one is the setting which is requested. All strings that follow are options of the setting. The number of options is at least one (for a fixed setting) but it can also contain multiple options (variable setting).

When multiple options are available the enum response is always followed by a get response (even though not requested explicitly) indicating the active setting. When there is only one option (fixed setting) the get packet is omitted.

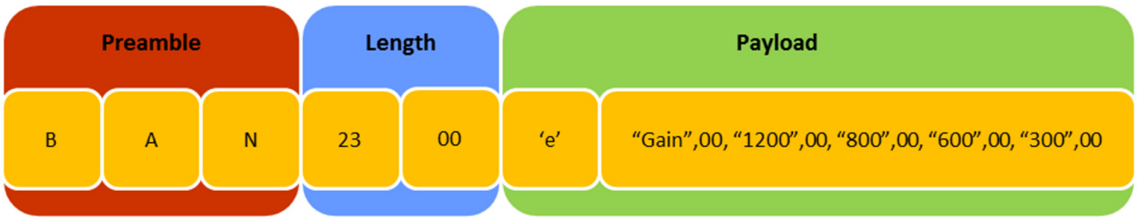

Figure 38 – Example enumeration response

If a complete enumeration is requested (NULL-Byte as setting) the headset will send an empty enum packet (enum command followed by 2 NULL-Bytes) to indicate the end of the enumeration. This way the interface will know when it has received the complete enumeration. When the enumeration of a specific command is requested this end-packet is omitted.

3.2.10 Settings Information

The enumeration command lists all possible settings and their options or value. This is enough to request and change settings of the system. The information command is introduced to provide additional information about the commands which can be used in the GUI.

The information request is similar to the enum request. It contains a string of the setting which information is requested or the string may be left empty to request the information of all settings in the system. In Figure 39 an example is given of the response to an information request of the "*Current Mag*" setting: magnitude of current generated by the system for impedance measurement.

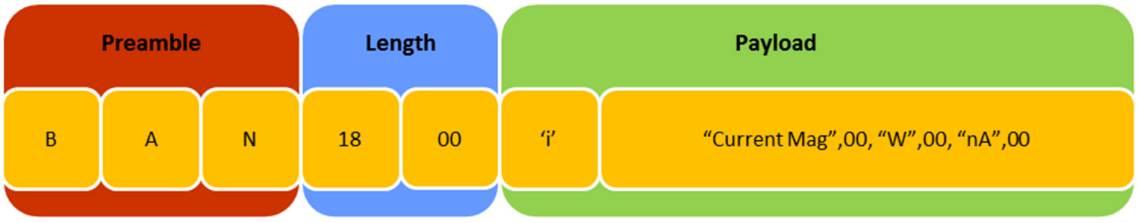

Figure 39 – Example response to information request

The information response contains 3 strings. The first is the name setting of which information is requested. The second string identifies the type of the setting. In this version of the protocol the type can be either read, write or flash: "R", "W" and "F" respectively indicating whether a setting is fixed (only accessible with get command) if it can be changed (using the set command) or if the changes will be stored in the flash memory of the ARM Cortex M4.

The third string contains the unit of the options, if there is any. When no unit applies the unit string is left empty. When a unit applies the string consists of two parts: the first byte contains the prefix and the rest contains the unit. Prefixes can be used in the interface for the presentation of the option values. Prefixes are based on the SI system and possible options are listed in

Table 9. Note that when the prefix is 10^0 (* 1) a space character (0x20 hex) should be placed before the Unit, else the first letter of the unit would be parsed as a prefix.

3.2.11 Setting remarks

Remarks are also used to provide extra information about settings to the user through the interface. The command must always be implemented by a system however the field is not mandatory and may be left empty. It can be used to provide a description or additional information about a command to the user. There are no restrictions or guidelines on the contents of the remark field and thus it should be treated as such by the interface.

The remark command works similar to the enum and information command: it can be requested for a specific command or for all command. The request packet from the interface to the node is always a remark request and a remark packet from the node to the interface is always a response packet. The request contains one string, indicating the setting (or empty for all remarks) and the response will always contain 2 strings: the setting name followed by the actual remark.

Table 9 – Command Listing

Prefix Byte (as ASCII character)	**Meaning**
Y	Yotta (10^{24})
Z	Zetta (10^{21})
E	Exa (10^{18})
P	Peta (10^{15})
T	Tera (10^{12})
G	Giga (10^{9})
M	Mega (10^{6})
k	Kilo (10^{3})
‘ ‘ (Space)	*No prefix* (10^{0})
m	Milli (10^{-3})
u	Micro (10^{-6})
n	Nano (10^{-9})
p	Pico (10^{-12})
f	Femto (10^{-15})
a	Atto (10^{-18})
z	Zepto (10^{-21})
y	Yocto (10^{-24})