TIPI PROTOCOL - V1.03

1. Overview
   1.1. Tipi is a simple but powerful ASCII text based protocol for controlling and monitoring a range of Linea Research products from any number of third party control panels or software applications over Ethernet (TCP/IP) or Serial (RS232/RS485)
2. Physical layer
   2.1. Tipi would typically be transported using one of the following:
      2.1.1. An EIA-RS232 interface running NRZ, LSB first, 1 start, 8 data, 1 stop, no parity, no flow control, 38,400 baud.
      2.1.2. An RS485 interface interface running NRZ, LSB first, 1 start, 8 data, 1 stop, no parity, no flow control, 38,400 baud. Since RS485 is a shared bus, there has to be mechanisms to control access to the bus to avoid contention. It is the responsibility of the master to coordinate access to the bus. This would be done for example by allowing time for a device to respond when the Controller sends a Get. It also means that often-requested parameters such as metering should not use 'Subscriptions'. These would instead be gathered via periodically sent **`Get`** commands. It also means that a wildcard **`Get`** could not be issued since this will almost certainly cause bus contention.
      2.1.3. An Ethernet Interface (10/100/1G) using the TCP/IP protocol, port 51456
3. General frame format
   3.1. 8-bit ASCII text strings are used throughout
   3.2. The start of a message is delimited by a $ character
   3.3. Fields are separated by one or more space characters (although no space is required after the Start delimiter (**`$`**) or before the End delimiter (**\r**), (sometimes denoted as **`<CR>`**)
   3.4. Spaces may *not* be used in name fields. We suggest the use of underscore instead (_).
   3.5. Numbers are expressed in decimal units
   3.6. The case of alpha letters is not important (although Upper Case Commands (e.g. SET) and CamelCase Method names (e.g. Out1/Gain) will be used in all responses from a device). Similarly, Boolean values will be returned in lower case (e.g. 'yes', 'no')
   3.7. The general format is SD C M VE - where:
      3.7.1. S is the Start character = $

3.7.2. C is the Command type (GET, SET, NOTIFY etc). *A minimum of three characters is required; others may be omitted (e.g. NOT rather than NOTIFY). The full command is returned by devices in any responses.*

3.7.3. M is the Method name (such as Out2/Eq3Freq), describing which parameter is to be acted upon. The MethodName is sometimes made up of concatenated sub-fields separated by a forward slash ("/"). These fields are usually:

3.7.3.1. The Path (the input/output name/number) e.g. "Out1". Valid path names are or the form: "In1", "InA", "Out1"

3.7.3.2. The Parameter name - e.g. "Eq1Freq".

3.7.3.3. The full MethodName in this example would be "Out1/EqFreq".

3.7.3.4. Methods which are not associated with particular input/output channels do not have a Path name or the forward slash(/), so are simply of the form "Snapshot". A typical command string then might be: **$SET Snapshot 8<CR>**

3.7.4. V is the value (such as "-3.8dB"). The units of measure are merely for convenience of reading, so may be omitted ("-3.8" in this example).

3.7.5. E is the end delimiter: Carriage Return (<**CR**>) ('\r', ASCII 13, 0x0D Hex).

3.8. Any characters outside of a S…E frame are ignored

4. Commands

4.1. A command tells the connected device what general action is to be performed

4.2. Defined commands are:

4.2.1. **SET** This tells a receiving device that it should set the value of the specified parameter to a particular value.

4.2.1.1. Example: **$SET Out2/Gain 3.5dB<CR>**

4.2.2. **GET** This requests the value of the specified parameter. The receiving device will return a "NOTIFY" command, appended with the value of that parameter

4.2.2.1. Example: **$GET Out8/Eq2Freq<CR>**

4.2.2.2. Response: **$NOTIFY Out8Eq2Freq 330Hz<CR>**

4.2.2.3. Note that since parameter values are quantised in the device, the value returned in any subsequent GET response may not be exactly the same as the value in a SET command. For example, **$SET Out1/Gain -22.415dB<CR>** might set the parameter value to -22.42dB. Similarly, if a SET command attempts to set a parameter to a value outside of its permitted range, a subsequent GET command will return the nearest permitted value.

4.2.3. **NOTIFY** Used to respond to a GET

Example: **$NOTIFY Out8Eq2Freq 330Hz<CR>**

4.2.3.1. Note that since parameter values are quantised in the device, the value returned in any subsequent GET response may not be exactly the same as the value in a SET command. For example,

$SET Out1/Gain -22.415dB<CR> might set the parameter value to -22.42dB. Similarly, if a SET command attempts to set a parameter to a value outside of its permitted range, a subsequent GET command will return the nearest permitted value.

4.2.4. **ERROR** This is returned when the message was erroneous. Error responses will return the original message, prepended with ERROR and appended with a brief error description and an error number. Examples of error responses are:

4.2.4.1. A message is badly formed: **$ERROR <original message> BadCommand 06<CR>**

4.2.4.2. The Method was not supported in the device: **$ERROR <original message> UnsupportedMethod 09<CR>**

4.2.5. **NOP** This is optionally used for an Ethernet connection to keep the connection open. It instructs the receiving device to perform No Operation - the command is ignored but the Ethernet message restarts the connection timeout. For TIPI/IP the timeout is 120s so NOPs should be sent more frequently than that.

4.2.5.1. Example: **$NOP<CR>**

5. Responses
   5.1. There are several types of response:
      5.1.1. **ERROR** (when the message is erroneous)
      5.1.2. **NOTIFY** (when a device sends back a parameter value in response to a Get: **$NOTIFY Out2/Eq3Gain 2.6dB<CR>**
6. MethodNames
   6.1. For each type of device, a list of compatible MethodNames is published. Please refer to the documentation for the specific device.
7. Security
   7.1. Not all the published MethodNames will be accessible; some of these might have been ‘locked’ by the vendor.
8. Parameter Values
   8.1. Parameter Values are stated in ‘natural’ units of measure (e.g. Hz, dB etc) in decimal format with any desired precision (although this will be truncated to the precision the device is capable of interpreting).
   8.2. Although not necessary, for convenience, a suffix may be added after the value to indicate units of measure (without a space). Examples are:
      8.2.1. Hz //Freq type (e.g. 650.3Hz)
      8.2.2. Oct //Bandwidth type (e.g. 0.32Oct)
      8.2.3. dB //dB type (e.g. -3.4dB)
      8.2.4. ms //Time type (e.g. 120.3ms)
      8.2.5. X //multiplier type (e.g. 3.2X)
      8.2.6. :1 //Ratio type (e.g. 4:1)
      8.2.7. W //Power type (e.g. 452.3W)

8.2.8. R //resistance type (Ohms) (e.g. 4.2R)
8.2.9. Min //minutes type (e.g. 122.5Min)
8.2.10. C //temperature type (e.g. 34.5C)
8.2.11. V //voltage type (e.g. 233.6V)
8.2.12. A //current type (e.g. 72.3A)
8.2.13. % //percent type (e.g. 52.5%)

8.3. We suggest no suffix for Integer, Select or Boolean values.

8.4. Boolean values are represented using one of the two values: **`yes`** or **`no`** (e.g. **`Out1/Mute yes`**)

8.5. The values of *Select* types (such as “24dB Bessel”) are conveyed using index numbers which we publish.

8.6. Multipliers etc (such as k for 1000) are NOT supported. 15kHz must be scripted as 15000Hz.

9. Multiple Substrings

9.1. It is sometimes useful, particularly with slower communications media, to convey several commands in one string, e.g:

`$SET Out1/Mute yes $SET Out2/Mute yes $SET Out3/Mute yes $SET Out4/Mute yes<CR>`

9.2. Most devices should support this, although there will sometimes be a limit to the maximum number of characters in such a string. A limit of 255 characters is not unusual.

10. Script Examples

10.1. **`$SET Snapshot 5<CR>`** – *Recall Snapshot 5*

10.2. **`$SET Out1/Mute yes<CR>`** – *Mute channel 1*

10.3. **`$SET InA/Gain -3.2dB<CR>`** – *Set the gain of input A to -3.2dB*