# netClé Data Transfer Protocol

This document describes the protocol for transferring trigger and sensor information between the configuration utility and the netClé device.

### Update Information

As of version 1.2 of the hub and 1.1 of the config the protocol was changed slightly to conserve space. This is given the title of “Protocol 1.1”. The changes are referred to here by the use of the phrase “Protocol 1.1”.

## Goals

The goals of this protocol design are:

* Create a data stream that can be process one byte at a time, removing the need for memory-consuming buffers on the netClé.
* Allow for robust error detection.
* Allow for robust error recovery by using unique values for all major commands.
* Use ASCII printable characters only and avoid characters which have special meaning to JSON. Although any character (except null) can be transmitted it is not difficult to avoid characters that could cause problems. It is also useful to be able to read, print and copy/paste the characters.

## Encoding Elements

### Numbers

Numbers are transferred in MSB order with 4 bits of data transferred per byte. The high-order bits of each byte are set to place the whole byte in a printable part of the ASCII range.

Numbers are divided into two types: actual numeric values (trigger values and time values) and numeric identifiers (sensor and action ID numbers and state values).

For actual numeric values the high-order bits are set to 6. This means the number are encoded in the range 0x60 to 0x6F which is ‘`’ (back-quote) to ‘o’ (lower-case O).

Numeric identifiers have 4 as the high-order bits putting them in the range ‘@’ to ‘O’.

### Major Commands

This leaves the letters p to z and P to Z available for other things.

The upper case characters P to Z are reserved for commands and the start and end of transmission blocks.

|  |  |
| --- | --- |
| **Command or Block Marker** | **Byte Value** |
| Put Sensact into Report mode | Q |
| Start of trigger data block | T |
| Start of sensor data block | S |
| Put Sensact into Run mode | R |
| Request triggers from Sensact | U |
| Get version infomation | V |
| Start of Mouse Speed Block | Y |
| End of data block | Z |

### Condition

Triggering conditions (Trigger-on-low, Trigger-on-high and Trigger-on-equal) are sent as the ASCII digits ‘1’, ‘2’ and ‘3’.

### Boolean

It would have been nice to use ‘t’ for true and ‘f’ for false – but ‘f’ is being used to numeric data and ‘t’ also stands for trigger. So ‘p’ is used for true and ‘q’ is used for false.

### Summary of other values used

|  |  |
| --- | --- |
| **Byte Value** | **Meaning** |
| p | True |
| q | False |
| t | Start of trigger |
| z | End of trigger |
| 1 | Trigger on low |
| 2 | Trigger on high |
| 3 | Trigger on equal |

### Protocol 1.1

To conserve space Condition and Boolean are now encoded in the same byte.

The following table shows the values sent for all combinations of condition and Boolean values.

|  |  |  |
| --- | --- | --- |
|  | Boolean | |
| Condition | True | False |
| Trigger on Low | ‘1’ | ‘5’ |
| Trigger on High | ‘2’ | ‘6’ |
| Trigger on Equal | ‘3’ | ‘7’ |

In Protocol 1.1 ‘p’ and ‘q’ are no longer used.

In addition ‘t’ and ‘z’ are no longer used to mark the start and end of a single trigger.

## Error Detection and Recovery

Because unique data types have unique encodings detection of errors is easy. If the data does not match the expected type the stream is invalid.

Recovery is accomplished by ignoring everything in the stream until the next major command character. Because these characters are only used for major commands they can be confidently used as a reset point.

## Use of White Space

White space is not part of the protocol, but can be useful for human debugging activities. Because of this senders are allowed to add new-line characters to the stream to improve readability. Receivers filter out and discard white space. The only limitation is that there can be no white space before the command character at the start of a block, and none after to ‘Z’ which terminates the block.

## Trigger Encoding

Trigger encoding is used to:

* send trigger data from the configuration tool to the Sensact
* send trigger data from the Sensact to the configuration tool
* send trigger data to and from EEPROM storage
* send trigger data to the Save box in the configuration tool
* read trigger data from the Restore box in the configuration tool

A list of triggers is encoded as follows:

Start-Of-Triggers (‘T’)

The number of triggers in the list (4 bytes of number-encoded data)

List of trigger data

End-Of-Block (‘Z’)

### Protocol 1.1

For Protocol 1.1 this was changed. 4 bytes is not needed for the number of triggers since this – so far – has never been a number more than 40. In addition the software needed to be able to tell whether a stream was using protocol 1.0 or 1.1. If it is 1.0 the second byte will be a back quote (representing a high-order zero in the trigger count). If it is 1.1 the second byte will be the digit ‘1’. After this 2 bytes are used to hold the trigger count.

Start-Of-Triggers (‘T’)

Protocol ID Marker (‘1’)

The number of triggers in the list (2 bytes of number-encoded data)

List of trigger data

End-Of-Block (‘Z’)

Each trigger is encoded as follows:

|  |  |  |
| --- | --- | --- |
| Parameter | Possible Values | Encoding |
| Start-of-trigger | ‘t’ | (not in protocol 1.1) |
| Sensor ID | 1-127 | AA |
| Required State | 0-15 | A |
| Trigger Value | 0-1023 | aaaa |
| Condition | HIGH, LOW or EQUAL | 1 or 2 or 3  (and 5 or 6 or 7 in protocol 1.1) |
| Action ID | 1-127 | AA |
| Action State | 1-15 | A |
| Action Param | 4-bytes | aaaaaaaa |
| Delay | 0-30000 | aaaa |
| Repeat | Boolean | p or q (not in protocol 1.1) |
| End-of-trigger | ‘z’ | (not in protocol 1.1) |

In the table above ‘A’ stands for one byte of numeric ID data and ‘a’ stands for one byte of numeric data.

### Protocol 1.1

To conserve space the Start-of-trigger and End-of-trigger characters were dropped. They contain no data and add little value to the data stream. Also Repeat and Condition were folded into a single byte.

## Sensor Data

Sensor data is sent from the Sensact to the configuration utility when in report mode.

A list of sensor data is sent as:

Start-Of-Sensor-Data (‘S’)

The number of sensors reporting

List of sensor data

End-Of-Block (‘Z’)

Data for a single sensor is sent as:

Sensor ID (AA)

Sensor value (aaaa)

## Mouse Speed Data

Mouse speed data is sent after the trigger data is sent. Mouse speed data is an optional extension to the trigger encoding. It is not an error if it is missing (allowing for backwards compatibility). Also it can be skipped. There is some provision in this design, for future extensions which may be optional.

The configuration interface allows the user to select three mouse speeds and two time intervals. The mouse begins to move at speed 1. After the first time interval the speed in changed to speed 2 etc. This allows for an accelerating mouse.

The configuration utility converts the user’s selection of a mouse speed into two values: the time between the sending of mouse move HID commands (called the *delay*), and the number of pixels the mouse will move per command (called the *jump*). For example, a slow mouse speed would be a jump of 3 pixels every 60 milliseconds, while a fast speed would be a jump of 10 pixels every 24 milliseconds. There is an attempt here to make the mouse movement as smooth as possible without overwhelming HID (especially Blue Tooth HID), and Sensact capabilities.

Delay and jump values are sent using ID encoding. Time values are sent using numeric encoding.

Mouse speed data is sent after all the trigger data and should be followed by the end-of-transmission mark (‘Z’). The data count at the start of the data block makes it possible for an implementation to skip this data block if the size is not what was expected. This may make future extensions easier.

|  |  |  |
| --- | --- | --- |
| Parameter | Possible Values | Encoding |
| Start of Mouse Speed Data | ‘Y’ |  |
| Data block size | 0-30000 | aaaa |
| Delay 1 | 0-1023 | AA |
| Jump 1 | 0-1023 | AA |
| Delay 2 | 0-1023 | AA |
| Jump 2 | 0-1023 | AA |
| Delay 3 | 0-1023 | AA |
| Jump 3 | 0-1023 | AA |
| Timer 1 | 0-30000 | aaaa |
| Timer 2 | 0-30000 | aaaa |

## Summary of Letter Usage:

|  |  |
| --- | --- |
| 1 – 3 (0x31 to 0x33) | Condition (HIGH, LOW, EQUAL) |
| ‘@’ to ‘O’ (0x40 to 0x4F) | One nibble of a numeric ID. Used for Sensor and Action Ids and states. |
| ‘P’ to ‘Z’ (0x50 to 0x5A) | Reserved for commands  ‘Q’ – Report command to Sensact ‘  ‘R’ – Run command to Sensact  ‘S’ – Start of sensor data from Sensact  ‘T’ – Start of trigger data to or from Sensact  ‘U’ – Get Triggers command to Sensact  ‘V’ – Version command to Sensact  ‘Y’ – Start of Mouse Speed Data  ‘Z’ – end of a transmission block. |
| ‘`’ to ‘o’ (0x60 to 0x6F) | One nibble of numeric data. Used for counts, trigger values, action parameters and delay. |
| ‘p’ and ‘q’ (0x70 and 0x71) | Boolean True and False (not in Protocol 1.1) |
| ‘r’ to ‘z’ (0x72 to 0x7A) | Reserved to mark data blocks:  ‘t’ – start of trigger data (not in Protocol 1.1)  ‘z’ – end of trigger data (not in Protocol 1.1) |
| ‘1’, ‘2’, ‘3’ (0x31 to 0x33) | Condition values: Trigger-on-low, Trigger-on-high, Trigger-on-equal (and Boolean False in Protocol 1.1) |
| ‘5’, ‘6’, ‘7’ (0x35 to 0x37) | Condition values: Trigger-on-low, Trigger-on-high, Trigger-on-equal and Boolean True in Protocol 1.1 only |

## BNF Notation (maybe clearer – probably not?) – for Protocol 1.0

<List-of-Triggers> ::= <Start-of-Triggers> <Trigger-Count> <Trigger-List> [<Mouse-Speed-Data>] <End-of-Block>

<Start-of-Triggers> ::= ‘T’

<End-of-Block> ::= ‘Z’

; Trigger-Count encodes the number of triggers in the list.

<Trigger-Count> ::= <num-byte> \* 4

<Trigger-List ::= <Trigger> | <Trigger> <Trigger-List>

<Trigger> ::= <Start-of-Trigger><SensorID> <ReqState> <Trigger-Value> <Condition> <ActionID> <ActionState> <Action-Parameters> <Delay> <Repeat><End-of-Trigger>

<Start-of-Trigger> ::= ‘t’

<SensorID> ::= <ID-byte> \* 2

<ReqState> ::= <ID-byte>

<Trigger-Value> ::= <num-byte> \* 4

<Condition> ::= ‘1’ | ‘2’ | ‘3’

<ActionID> ::= <ID-byte> \* 2

<ActionState> ::= <ID-byte>

<Action-Parameters> ::= <num-byte> \* 8

<Delay> ::= <num-byte> \* 4

<Repeat> ::= <Boolean>

<End-of-Trigger> ::= ‘z’

<Mouse-Speed-Block> ::= <Start-of-Mouse-Speed> <Data-Count> <MS-Delay> <MS-Jump> <MS-Delay> <MS-Jump> <MS-Delay> <MS-Jump> <MS-Time> <MS-Time>

<Start-of-Mouse-Speed> ::= ‘Y’

<Data-Count> ::= <num-byte> \* 4

<MS-Delay> ::= <ID-byte> \* 2

<MS-Jump> ::= <ID-Byte> \* 2

<MS-Time> ::= <num-byte> \* 4

<num-byte> ::= ‘`’ | ‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘g’ | ‘h’ | ‘i’ | ‘j’ | ‘k’ | ‘l’ | ‘m’ | ‘n’ | ‘o’

<ID-byte> ::= ‘@’ | ‘A’ | ‘B’ | ‘C’ | ‘D’ | ‘E’ | ‘F’ | ‘G’ | ‘H’ | ‘I’ | ‘J’ | ‘K’ | ‘L’ | ‘M’ | ‘N’ | ‘O’

<Boolean> ::= ‘p’ | ‘q’

<List-of-Sensor-Data> ::= <Start-of-Sensor-Data> <Sensor-Count> <Sensor-Data-List> <End-of-Block>

<Start-of-Sensor-Data> ::= ‘S’

; Sensor-Count encodes the number of sensors reporting

<Sensor-Count> := <num-byte> \* 4

<Sensor-Data-List> ::= <Sensor-Datum> | <Sensor-Datum> <Sensor-Data-List>

<Sensor-Datum> ::= <SensorID> <Sensor-Value>

<SensorID> ::= <ID-byte> \* 2

<Sensor-Value> ::= <num-byte> \* 4

## BNF Notation for Protocol 1.1 – changes highlighted

<List-of-Triggers> ::= <Start-of-Triggers> <Trigger-Count> <Trigger-List> [<Mouse-Speed-Data>] <End-of-Block>

<Start-of-Triggers> ::= ‘T1’

<End-of-Block> ::= ‘Z’

; Trigger-Count encodes the number of triggers in the list.

<Trigger-Count> ::= <num-byte> \* 2

<Trigger-List ::= <Trigger> | <Trigger> <Trigger-List>

<Trigger> ::= <SensorID> <ReqState> <Trigger-Value> <Condition&Repeat> <ActionID> <ActionState> <Action-Parameters> <Delay>

<SensorID> ::= <ID-byte> \* 2

<ReqState> ::= <ID-byte>

<Trigger-Value> ::= <num-byte> \* 4

<Condition&Repeat> ::= ‘1’ | ‘2’ | ‘3’ | ‘5’ | ‘6’ | ‘7’

<ActionID> ::= <ID-byte> \* 2

<ActionState> ::= <ID-byte>

<Action-Parameters> ::= <num-byte> \* 8

<Delay> ::= <num-byte> \* 4

<Mouse-Speed-Block> ::= <Start-of-Mouse-Speed> <Data-Count> <MS-Delay> <MS-Jump> <MS-Delay> <MS-Jump> <MS-Delay> <MS-Jump> <MS-Time> <MS-Time>

<Start-of-Mouse-Speed> ::= ‘Y’

<Data-Count> ::= <num-byte> \* 4

<MS-Delay> ::= <ID-byte> \* 2

<MS-Jump> ::= <ID-Byte> \* 2

<MS-Time> ::= <num-byte> \* 4

<num-byte> ::= ‘`’ | ‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘g’ | ‘h’ | ‘i’ | ‘j’ | ‘k’ | ‘l’ | ‘m’ | ‘n’ | ‘o’

<ID-byte> ::= ‘@’ | ‘A’ | ‘B’ | ‘C’ | ‘D’ | ‘E’ | ‘F’ | ‘G’ | ‘H’ | ‘I’ | ‘J’ | ‘K’ | ‘L’ | ‘M’ | ‘N’ | ‘O’

<Boolean> ::= ‘p’ | ‘q’

<List-of-Sensor-Data> ::= <Start-of-Sensor-Data> <Sensor-Count> <Sensor-Data-List> <End-of-Block>

<Start-of-Sensor-Data> ::= ‘S’

; Sensor-Count encodes the number of sensors reporting

<Sensor-Count> := <num-byte> \* 4

<Sensor-Data-List> ::= <Sensor-Datum> | <Sensor-Datum> <Sensor-Data-List>

<Sensor-Datum> ::= <SensorID> <Sensor-Value>

<SensorID> ::= <ID-byte> \* 2

<Sensor-Value> ::= <num-byte> \* 4