Utopia : Message Specification

Purpose

This document describes in simple terms the messages which are passed between the different components of the Utopia Noisetagging BCI system, and the structure used for the

messages. The message specification is *transport agnostic* in that the messages themselves may be sent over different ‘wire-protocols’, such as BTLE, UDP, TCP, etc.

Objectives

- Stateless: as much as possible messages are self-contained information updates such that the interpretation of a message depends minimally on the reception of other messages. In this way, we are both robust to loss of a single message and failure/rebooting of an individual component, and also simplify the later loading/replay of saved data as we can basically start playback at any point. The disadvantage of this is however the messages tend to be longer than needed due to the additional redundancy.
- Simple + Readable: as much as possible the message spec will be human readable, such that for example message types are encoded in plain strings. However, the spec will also be simple to read manipulate the messages, thus integers will be sent as integers, and arrays as packed binary arrays.
- Efficient: as much as possible (without violating the stateless and readable objectives) the message spec will be efficient in space usage in transmission.
- Compact : some of our transport layers have very small payload sizes (e.g. BLE has 20 bytes) thus the messages should be compact such that a useful stateless message can be sent in this payload size.
- Latency tolerant : we cannot guarantee timely transmission of messages between components. Thus, the spec will (where appropriate) include additional time-stamp information to allow a ‘true’ message time-line to be reconstructed.

Structure

The message specification is structured as follows:

- Message Name: a human readable informative name for the message
- Message UID : ascii character used to uniquely identify this message. This is required to be the first character of any message.
- Sender: the component which produces the message
- Receiver: the component which receives the message
- Purpose: the reason for sending this message between these two components
- Format: the detailed structure of the message in terms of the basic types (i.e. char, int8, int16, uint8, single, double etc.) it is made up from. Each slot of the format has:
    - Name:
    - Type Information:
    - Comment: human readable description of the purpose of this slot

Message Specifications

Endianness

To simplify things we *require* that all numbers be encoded in **LITTLE ENDIAN**.

Message Header

All messages start with a standard header consisting of:

Slot	**Type (= value)**	**Comment**
UID	1 of char	Message UID
version	1 of uint8 (0)	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes. This can be used to allow *forward compatibility* as clients can skip gracefully skip messages where they do not understand the payload, e.g. because of an unknown messageUID+version combination.
payload	[length] of byte	The message payload, i.e. the rest of the message

General Utility Messages

Name: **HEARTBEAT**	**UID: "H"**
Sender: **ANY component** **REQUIRED: STIMULUS** **REQUIRED: RECOGNISER**	**Receiver:** **ANY component** **REQUIRED: RECOGNISER** **REQUIRED: UI**
Purpose: These heartbeat messages have two main purposes: 1. As an **I'm alive** indication. If the heartbeat stop then we can conclude the recieve component has crashed. 2. As a '**clock alignment**' query. They payload of the HEARTBEAT signal is the current timestamp of the receiver component. Thus it can be used to track and align the sender's clock with the recievers. Notes: ● The *exact* interval between HEARTBEATS is dependent on the client. However the **maximum** interval is set to MAXHEARTBEATINTERVAL which is 4 seconds by default. ● It is **required** that **all** STIMULUS components send HEARTBEAT messages to RECOGNISER as soon as these two components establish an initial connection. ● It is **required** that RECOGNISER sends HEARTBEAT messages to all clients as	

soon as these two components establish an initial connection.

Format:

Slot	Type (= value)	Comment
UID	[1] of char = “H”	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time this signal quality measure was computed. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.

Name: **SUBSCRIBE**	**UID: “B”**
Sender: ANY component	**Receiver: RECOGNISER**

Purpose:
These subscribe messages main purpose is to reduce the network and computation load on both the Utopia-Hub message server and the client by allowing the client to inform the hub about which messages the client is interested in.
Notes:

- As subscribe message may be sent at any time, to update the set of messages forwarded to this client from that point on.
- If no subscribe message is sent to the utopia-hub then by default the client receives all messages.
- Even if a client attempts to subscribe to **no** messages, it will always receive HEARTBEAT messages.

Format:

Slot	Type (= value)	Comment
UID	[1] of char = “B”	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.

timestamp	[1] of uint32	Time this subscribe message was sent. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.
subscriptionList	[#messages] of char	A string array of the message UIDs this client would like forwarded to it.

Name: **LOG**	**UID: "L"**
Sender: Any	**Receiver: Recogniser**

Purpose:
General logging of the state of any individual component. Currently this is used in the Recogniser to log things like it's code version, where the data is being saved, and where the internal log-files are being saved.
Note:

- This message is **not** a general debug logging system, in particular it is not designed for high-performance logging, but a more general logging ability to send general system information between components, and in particular to any UI component. For debug logging we suggest you save to local storage on the client device and only log information on where this debug-log can be found for later retrevial.

Format:

Slot	**Type (= value)**	**Comment**
UID	[1] of char = "L"	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time the mode change message was sent. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.
message	[1] of string	String with the logging information.

Presentation -> Recogniser

Name: **STIMULUSEVENT**	**UID: “E”**
Sender: Stimulus	**Receiver: Recogniser**

Purpose:
Provide the recogniser with updated information about the current stimulus state.
Note this message format is designed to allow for *stateless* updates with very small message sizes. Thus, each message is a self-contained report of (part of) the stimulus state at a given time, allowing to cut simulus-state updates over multiple messages without having to worry about message sequence numbers etc.

Note: If we are **really** pressed for message sizes then we can define a less general but more compact STIMULUSEVENT message based on compressing the STIMULUSSTATE structure into a single ‘byte’ with 7-bits for objectUID and 1-bit for state. However, I think the implementation simplicity of the current message format makes this prefered even with the message size overhead.

Format:

Slot	**Type (= value)**	**Size (total) in bytes**	**Comment**
UID	[1] of char = “E”	1 (1)	Message UID
version	1 of uint8 = 0	1 (2)	Message version number (0)
length	[1] of uint16 (short)	2 (4)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	4 (8)	Time of the *first* sample of this data packet. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.
nobjects	[1] of uint8	1 (9)	Number of objects with stimulus information ***contained in this message***

stimulusdict	[nobjects] of **STIMULUSDICT**	nobjects*2 (nobjects*2 +9)	A dictionary of stimulus state information for nobjects. The format of a single stimulus state entry is given below

STIMULUSDICT structure

Slot	**Type (=value)**	**Size (total)**	**Comment**
objectUID	[1] of uint8	1 (1)	A unique identifier for object for which the stimulus state is being reported. N.B. objectUID=0 is **reserved** for the **true-target** object when in supervised training mode.
stimulusstate	[1] of uint8	1 (2)	The updated stimulus state for the object with objectUID. The stimulus state indicates a relevant characteristic of the stimulus used by the recogniser. For example if this object is in a high or low brightness state.

NOTES:

- **objectID** - an object UID is (as the name implies) a ***unique*** identifier for a particular stimulus object. This is used ****both**** for indication of the stimulus state of an object **and** for indication of the identified target object when predictions are generated.
  Object IDs for which *no* stimulus state information has been provided since the last recogniser reset command are assumed to not be stimulated. Object IDs do *not* have to be consecutive, and can be allocated arbitarly by the STIMULUS component
- **Stimulusstate** - The **encoding** used by the stimulus state is flexible in this version of the spec. By convention stimulus state is treated as a **single** continuous level of the stimulus intensity, e.g. a grey-scale value. However other encoding formats as possible without changing the message spec. Example encodings could be; bit 0=long, bit(1)=short, or bits 0-4 for intensity and bits5-8 for color. **It is the responsibility of CONFIG to ensure that STIMULUS and RECOGNISER agree on the interperation of the stimulus state object.**
- Example Bandwidth Requirements: for a 36 output display with messages packed into 20bytes (as in BLE). We have 6 bytes header overhead, leaving 7 objects in the message packet. Thus we require 6 packets for a update on all objects in the display, i.e. 6*20 = 120 bytes / display update. Thus @ 60 display rate we require: 120*60 = 7200 bytes/sec. The spec for BLE gives, an *application* data rate of .27Mbit/sec = 270000 bit/sec = 33750 byte/sec. Thus we need about 25% of the total available BLE bandwidth for this common use-case.

Recogniser -> Selection or Output

Name: **PREDICTEDTARGETPROB**	**UID: “P”**
Sender: Recogniser	**Receiver: Output**

Purpose:
Inform the output component of the current predicted target and it’s probability of being correct so the output decide: a) if the prediction is confident enough, b) if so to generate the appropriate output.

Format:

Slot	**Type (= value)**	**Comment**
UID	[1] of char = “P”	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time the prediction was generated. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.
objectID	[1] of uint8	The objectID (as in the STIMULUSSTATE message) of the predicted target
errrorprobability	[1] of single	The probability that objectID is ****not**** the true target object.

Controller or Output -> All

Name: **MODECHANGE**	**UID: “M”**
Sender: Controller	**Receiver: Recogniser**

Purpose:
Tell the recogniser to switch to a new operating mode, e.g. switch from calibration to testing.
Currently, we have the following operating modes for the recogniser:

- Calibration.supervised - calibration with user target instruction.
- Calibration.unsupervised - calibration without user target instruction, a.k.a. zerotrain
- Prediction.static - generate predictions with a fixed model
- Prediction.adaptive - generate predictions with adaptive regularisation
- NonStopLearning - run in non-stop learning mode

Format:

Slot	**Type (= value)**	**Comment**
UID	[1] of char = “M”	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time the mode change message was sent. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.
newmode	[1] of string	String with the new recogniser mode to enter. One-of: Calibration.supervised, Calibration.unsupervised, Prediction.static, Prediction.adaptive, ElectrodeQuality

Name: **NEWTARGET**	**UID: “N”**
Sender: Controller	**Receiver: Recogniser**

Purpose:
Tell the recogniser that the user has switched to attempt selection of a new target. For example, in the speller because the OUTPUT has made a character selection and the user is moving on to the next letter. The RECOGNISER is expected to use this message to clear it’s prediction history and start fresh on a new output.

Format:

Slot	Type (= value)	Comment
UID	[1] of char = “N”	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time the the new target change happened. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.

Name: **SELECTION**	**UID: “S”**
Sender: Controller	**Receiver: Recogniser**

Purpose:
Tell other clients that a selection has been made (and possibly trigger output to be generated). This further implies that the user has switched to attempt selection of a new target (thus an explicit newtarget message is not required). For example, in the speller because the OUTPUT/SELECTION has made a character selection and the user is moving on to the next letter.
The RECOGNISER is expected to use this message to clear it’s prediction history and start fresh on a new output.
An OUTPUT module may use this message to decide if it should perform it’s tasked output if the selection matches it’s trigger criteria, e.g. changing the TV channel in a TV remote.

Format:

Slot	Type (= value)	Comment
UID	[1] of char = “S”	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time the the new target change happened. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.

objID	[1] of uint8	Selected objectID. This objID should match that used in stimulusEvents

Name: **RESET**	**UID: “R”**
Sender: Controller	**Receiver: Recogniser**

Purpose:
Reset the RECOGNISER to a ‘fresh-start’ state, i.e. as if it has just been started with no saved information about training data or predictions.

Format:

Slot	**Type (= value)**	**Comment**
UID	[1] of char = “R”	Message UID
version	1 of uint8	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time the the reset happened. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.

Recogniser -> User Interface

Name: SIGNALQUALITY	**UID: “Q”**
Sender: Recogniser	**Receiver: UI**

Purpose:
Inform user about the quality of the electrode fit, so they can adjust the electrode positioning or contact to improve the connection to the scalp as much as possible.

Format:

Slot	Type (= value)	Comment
UID	[1] of char = “Q”	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time this signal quality measure was computed. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.
signalquality	[nChan] of single	A numeric measure in the range 0-1 of the channel noise to signal quality, where: 0 -> a perfect electrode connection 1 -> a completely bad electrode connection.

Acquisition -> Recogniser

Name: **DATAPACKET**	UID: “D”
Sender: Acquisation Device (e.g. EEG)	**Receiver:** Recogniser

Purpose:
Send raw data as measured by the acquisition device to the recogniser.

Format:
Basically this is not something we can specify as it depends on the exact hardware device. Minimum spec for us:

Slot	Type (= value)	Comment
UID	1 of char = “D”	Message UID
version	1 of uint8	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.

timestamp	[1] of int32	Time of the *first* sample of this data packet. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.
nsamples	[1] of int 32	The number of samples (i.e. time-points) in this datapacket (Note: the nchannels is infered to be (length-8)/nsamples/4)
data	[nchannels x nSamp] of single	The raw packed data

Notes:
32bit timestamps @1ms accuracy means the timestamps will wrap-around in 4294967296/1000/60/60/24 = 50 days.. Which is way more than we really need….
With 24 bits this would be 4hr.. For implementation simplicity standard 32bit ints are prefered.

Name: **DATAHEADER**	**UID:** A
Sender: Acquisation Device (e.g. EEG)	**Receiver:** Recogniser

Purpose:
Provide meta-information about the data provided by the acquisation device. As a minimum this should be the number of channels sent and their sample rate. Optionally should include channel location information.

Format:
Basically this is not something we can specify as it depends on the exact hardware device. But suggestion is:

Slot	Type (= value)	Comment
UID	1 of char = “”	Message UID
version	1 of uint8	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
sample_rate	1 of single	Sampling rate of measurements
nChan	1 of int32	Number of channels

labels	string	Comma separated list of the textual names of the channels. E.g. “C3,C4,C5”

Extension Messages

In this section are listed messages which may be useful in future, but will not be implemented in the V1.0 version of the system.

Config -> Recogniser

Name: **CONFIGURECOGNISER**	**UID: “C”**
Sender: **Config**	**Receiver:** **Recogniser**

Purpose:
Update the general configuration of the recognizer, e.g. the response length.

Note: The side-effects of changing the configuration on the RECOGNISER is **UNDEFINED.** In the worst case this may result in a complete restart of the RECOGNISER clearing all previous state, i.e. removing the trained classifier etc.

Format:
This message payload is a dictionary of name-value pairs encoded in JSON format of the configuration parameters that the RECOGNISER needs.

Note: This message could potentially be *very large*, and need to extend over multiple packets. It is assumed the underlying transport will deal with this effectively.

Slot	**Type (= value)**	**Comment**
UID	[1] of char = “C”	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time the the new target change happened. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.

configJSONdict	[1] <string>	This is the configuration encoded as a JSON dictionary string, e.g. “{ responseLength : 100 }”

Other

Name: **TICKTOCK**	**UID: “T”**
Sender: **ANY component**	**Receiver:** **ANY component**

Purpose:
This message tells the receiver what the current time-stamp from the sender is. It also optionally represents a *query* to the receiver to send back it’s time-stamp as rapidly as possible. These messages have two main purposes:

1. To ‘clock alignment’ distribution method. Where the sender can inform different receivers of it’s current clock state.
2. As a ‘**clock alignment**’ query. (Optionally) the receiver may response to this message by replying with the original query + it’s local time-stamp information. This can then be used to estimate the message latency to more accurately align the different components clocks.

Format:

Slot	**Type (= value)**	**Comment**
UID	[1] of char = “T”	Message UID
version	1 of uint8	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time this signal quality measure was computed. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.
yourClock	[1] of uint32	(Optional) for a ticktock response message, the timestamp of the orginal TICKTOCK message we are responding to.

Name: **PREDICTEDTARGETDIST**	**UID: “D”**
Sender: **Recogniser**	**Receiver:** **Output**

Purpose:
Inform the output component of the current distribution over all possible predicted targets --to allow more fine grained decision making w.r.t. when and what output to generate.

Format:

Slot	**Type (= value)**	**Comment**
UID	[1] of char = “D”	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time prediction was generated. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.
targeterrordist	[nObjects] of TARGETERRORDIST	Dictionary of the error probabilities for all of the output objects known to the Recogniser

TARGETERRORDIST

Slot	Type (=value)	Comment
objectUID	[1] of uint8	The objectID (as in the STIMULUSSTATE message) of the predicted target
errorprobabililty	[1] of single	Error probably for this object

Name:	**UID: TBD**

CURRENTMODEL	
Sender: **Recogniser**	**Receiver:** **UI**

Purpose:
Inform user about the parameters of the current model.

Format:

Slot	**Type (= value)**	**Comment**
UID	[1] of char = “”	Message UID
version	[1] of uint8 = 0	Message version number (0)
length	[1] of uint16 (short)	Total length of the remaining message in bytes.
timestamp	[1] of uint32	Time this model was computed. Time is measured **in milliseconds** relative to an arbitrary device dependent real-time clock.
sizeSpace	[2] of uint32	[nChan,#E] dimensions of the following spatial filter matrix
spatialfilter	[nChan x #E] of single	The current model’s spatial filter matrix
sizeTime	[2] of int32	[Tau, #E] dimensions of the following impulse response matrix
impulseresponse	[Tau x #E] of single	The current model’s estimated impulse response