# Communication Framework For the Mionix Naos QG Mouse

For Online and Offline Usage

Andreas Wulff-Jensen
Dept. Architecture, Design and Media technology
Aalborg University Copenhagen
Copenhagen Denmark
awj@create.aau.dk

*Abstract*— **The Mionix Naos QG mouse has multiple sensors integrated. It can record all the metrics native to mice: being scroll, clicks and mouse movements. Moreover, this mouse has heart rate (HR) and Galvanic Skin Response (GSR) sensors embedded. Through Mionics API** [1] **WebSocket can be used to access all its data. This access has been demonstrated through Javascript in a local hosted webpage. In this page the measurements can be recorded to a .txt file and streamed to another .txt file. The latter file can be accessed in Unity to either send the data further through UPD, use the data within Unity or be recorded. Another Unity implementation have been developed as well. This was directly connected to the WebSocket, and has the same properties as the first Unity development. Since two nearly identical implementations were made, the quality of their recordings and data communication were tested. Based on the test the Unity implementation, which was directly connected to the Mionix HUB had higher GSR temporal accuracy, but showed more errors in its data acquisition process. GitHub page with the implementations can be found here: https://github.com/anwul4/Mionix-QG-Unity-and-webpage**

***Keywords*— *Mionix QG; biometrics; heart rate; gsr; framework; online; offline; data gathereing,***

## I. Introduction

The Mionix Naos QG mouse is a mouse with embedded Galvanic Skin Response (GSR) and Heart Rate (HR) sensors. This make the mouse convenient when you need to capture the HR and GSR of the user. For instance, in respect to computer applications, other screen based media or experiments without the interaction of one of the hands.

Out of the box the Mionix Noas QG mouse is ready to plug and play like a normal computer mouse. Moreover, to get access to the biometrics the Mionix HUB [2] needs to be installed. In that program, you can monitor all the biometrics and mouse metrics integrated in the mouse. With this HUB everything can be accessed through their WebSocket based API [1]. Though the API the data is accessible and monitorable, though implementations for saving the data and using it in different environments are needed to get the full out of this mouse in regards to affective computing and psychophysiological studies.

The next sections of this paper will describe the implemented solutions for these problems which is free to use both within and out of academia. To get the basic idea of what the API gives and how to communicate with it (II) will be about the API and its structure. (III) will continue describing a simple webpage, which can access, record and stream the data to the user's computer. (IV) describe a Unity implementation which can access the aforementioned stream, record, manipulate and send the data further through UDP connection to other applications. (V) describes another Unity implementation, which is similar to the first one, but accesses the API directly. (VI) compares the outcome and performance of the two Unity projects in respect to errors in the implementation and the temporal accuracy. (VII) will present a conclusion of it all and a GitHub link to the implementations.

## II. Structure And use Of API

Mionix wrote a tutorial of how to access the bio and mouse metrics of the mouse. They have done it through a networking protocol called WebSocket, which is a bi-directional server to browser networking paradigm using TCP communication protocol [3], [4].

This protocol sends a .json [5] to the following address "ws://localhost:7681", "mionix-beta". In any application or language which communicates with this server receives the .json file. In the API it is proposed to be done through Javascript in a HTML file.

The info given in the .json file can tell the user various different things: whether or not the device is connected, several device information such as its type and device capabilities, mouse metrics such as scroll, click and movement info, lastly the biometrics can be found, which include HR and GSR.

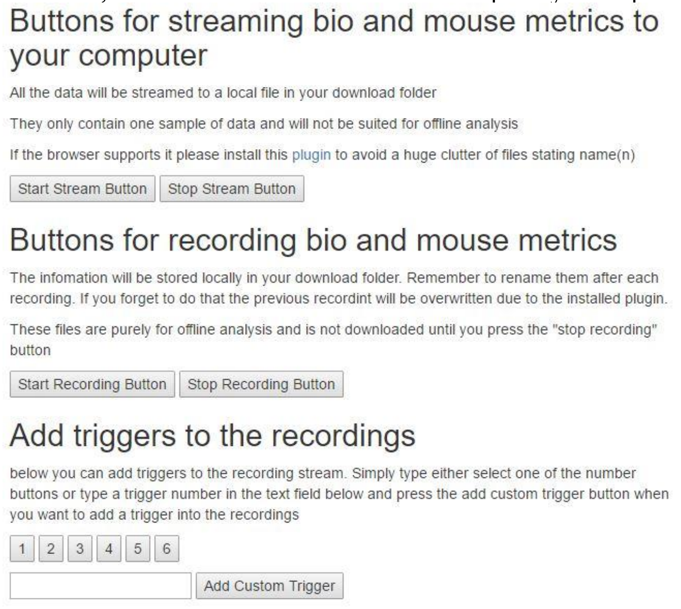
These three categories are furthermore separated in the .json file through a keyword, which can be accessed by asking for what the type of the data is. In case the type is "device" the device info will be given to the user, if it is "mouseMetrics" all the mouse data will be delivered to the user, last of all, if the type is equal to "bioMetrics" HR and GSR data will be shown.

Through the brief description the user can get access to all the data he wants, to get more details go to the API [1] for more info of which keyword to write in order to get access to the info you want.

## III. Access, Record and Stream through WebBrowser

The API demo was shown through a webpage, by further developing that page it is possible to make it act as a data recorder and streamer. On the webpage, a simple UI button interface has been developed to control how the data stream is manipulated (see figure 1). First, a start and a stop recording button are created. When the start recording button is pressed all the mouse and bio data are gathered every time a new websocket message has been received. When the stop recording button is pressed the gathering of data is stopped and all of it is saved to a Blob [6]. This Blob serves as an input to a file saving script, which saves a .txt file (the file saving script can be found here [7]). The .txt file can then be read by the analysis software you desire. While the webpage "records" the data the user of the webpage can add triggers to the stream by pressing one of the buttons labeled with a number fx "2". You are also capable of adding your own trigger number by writing it in the text field available and press the "add trigger" button. The purpose of adding triggers is to segment the data stream in relation to different events occurring through the stimuli. On the webpage, the process of adding triggers is manual, and is best fitting non-interactive stimuli. Furthermore, the triggers will be places in their own container and have their own row of data in the created .txt file.

The webpage is also capable of "streaming" the data to a file. This process can be toggled on and off by pressing the "start stream button" or the "stop stream button". When the "start stream button" is pressed the webpage takes the current message from the WebSocket and saves it to a file through the same process as recording. The program then waits 100ms before the webpage gathers a new sample and saves it by the same file name, but only if the following plugin is installed [8], without it, the new file will get a file index number after the name, thus will the amount of data files quickly end up as


Buttons for streaming bio and mouse metrics to your computer

All the data will be streamed to a local file in your download folder

They only contain one sample of data and will not be suited for offline analysis

If the browser supports it please install this plugin to avoid a huge clutter of files stating name(n)

Buttons for recording bio and mouse metrics

The infomation will be stored locally in your download folder. Remember to rename them after each recording. If you forget to do that the previous recordint will be overwritten due to the installed plugin.

These files are purely for offline analysis and is not downloaded until you press the "stop recording" button

Add triggers to the recordings

below you can add triggers to the recording stream. Simply type either select one of the number buttons or type a trigger number in the text field below and press the add custom trigger button when you want to add a trigger into the recordings

Figure 1 shows the UI for the webpage application.

a huge clutter, and the purpose of streaming will be lost.

## IV. Unity: Data Through File Stream and Beyond

The "stream" from the webpage application can be accessed through the C# class called filestreamer [9]. The code is implemented in Unity to conveniently access the file. The file is structured in a rather simple manner, where each line contains the keyword of what it contains, a omma, and a value. By asking the program to recognize the different keywords, the data can be separated and added to the Unity system. To avoid too many instances of the same sample the streamer is only activated every 200ms. This delay also solves some of the access file conflicts, as Unity cannot access the file while it is being written to by the webpage.

In the Unity realm, we are capable of recording them into the system, send them further to other applications, analyze them online, use them in the system and separate them by adding triggers.

The following structure of the code showcases how the framework could be worked with. Furthermore, this showcase also applies to the next Unity application, which will be described in the next section.

When the application starts Unity checks to see if the following folders are created "E:/HrTestData/" + participantID", where participantID is the number of participants there has been recorded. If that folder exist participantID will be incremented until the folder does not exist on your computer. This is a suitable approach to let the application keep track of how many participants there has been through your experiment or experience.

After the initial folder setup the program starts getting access to the stream file. It saves the current values, which is send further through UDP connection to another application such as Simulink or wherever you want the data to go. Furthermore, the data starts being collected and a timer is activated. The timer will be important to elicit different kinds of trial events. Firstly, after 2 seconds the system asks if the feature of placing a trigger is enabled. If so, a trigger is added to the recording and send through UDP networking. Furthermore, the ability to add a trigger is temporarily disabled for 100ms.

When the recording has lasted 10 seconds, the baseline for all the biometrics are calculated. By that moment baseline correction (change score values [10]) and percentage difference from the baseline is calculated utilizing the newest data update. All the new values are gathered and saved. From the baseline calculation and 20 seconds forward gathering of the raw and the manipulated data is active. Afterwards three different data files containing raw biometrics, baseline corrected biometrics and percentage different from baseline biometrics are created, ready to be analyzed offline.

During the recording process two other triggers are integrated in the same manner as the first one.

In the program the following functions are public and can be used in other contexts than the showcased. In the script which has access to the file stream (txtStreamReader.cs) there are public Get functions for all the bio and mouse metric data.

In the script “HeartRateManager.cs”, which is responsible for gathering and all the baseline calculations the following functions are public.

“MeasureBaseline(arrayList)”this function takes an unknown arrayList based on its length the function calculates the average, which can be used as baseline.

“LiveChangeScore(double biomeasure, double baseline)” this function calculates the numerical difference between the baseline and the bio metric measure you want.

“LivePercentageDiffFromBase(double bioMeasure, double baseline)” calculates the percentage difference from the baseline to the bio measure you want.

Lastly, “SetTrigger(int triggerNo )”is public and can be used wherever it fits your experience. It sends a trigger through UDP and place the same trigger in the biometric recordings both raw and baseline manipulated recordings.

The functionality of this Unity application has been described in the next section the other Unity application will be presented. It has the same functionality as the one just described, but gets the data in a different manner.

## V. Unity: Data Through Mionix HUB

This Unity application are quite similar to the other one, the mayor difference is how the program gets the data from the mouse. In this Unity application the data is send directly from the HUB to Unity. C# in Unity is natively not capable of using communicating through WebSockets, however a WebSocket-Sharp script [11]. has been developed and can be used as an external library. By this library the communication to the Mionix HUB can be established the same way as in the webpage service. When the communication to the server is established and receives the .json objects C# interprets it in a different manner. It divides the objects into two chunks. One for the mouse metrics and one for the bio metrics. These chunks are strings, in which words can be searched for. If they are found, part of the string up to and with the word is deleted, and the characters from the next word to the end is erased as well. By this operation the only available characters in the string is the number corresponding to the data metric of interest for instance “

```
if (data.Contains ("heartRateMax"))

{

string str;

str = data.Remove (0, data.IndexOf ("heartRateMax")+ 14);

str = str.Remove (str.IndexOf ("gsr") - 2, str.Length +2 - str.IndexOf ("gsr"));

heartRateMax = Convert.ToDouble(str);

}”.
```

This chunk of code has the purpose of finding the value corresponding to heartRateMax, as it first looks for that part of the string continued by looking for “gsr” which is the next text in the string.

The values gathered through this method is stored in the corresponding containers and can be addressed by Get functions.

From this point the application does the same as the previous Unity projects.

## VI. Unity Application Comparisson Test

Since the Unity projects are nearly similar, and only differs by how they get the data the coming sections will investigate the temporal and reliable differences between the implementations. Each of them will be tested 10 times through the demo procedure of accessing, recording and printing data. The temporal accuracy will be compared, the deviation of it, the amount of errors in the communication to the HUB or the .txt file will as well be taken into account when the two applications are compared.

## VII. Test Results And Discussion

The ten runs of each Unity application showed the following averages and standard deviations (STD).

For the Mionix Hub to the Unity application the average amount of communication error during 10 runs of 30 seconds is 26.1 with a standard deviation of 10.69. the update rate for the HR is 1 per second with a STD of 0, the GSR is 8 per second with an STD of 0.

For the Webpage’s file to Unity the following results appeared. The average amount of accessing the txt file at the same time conflicts were 2,5 with a STD of 1.62. The update rate of HR was 1 per second with STD of 0 and for the GSR it was 4 per second with STD of 0.44.

These results are rather interesting. There are more communication errors between the HUB and Unity than the communication between the common .txt file and Unity, but the update rates seem not to be affected by the errors. In fact, the update rates are stable. Opposed to the other implementation the sharing conflicts of the file are rather low, but the update rate for the GSR seems to fluctuate a little, and only updates half as many times as the other.

These observations impose that the direct communication to the HUB produces temporal more accurate results, than the communication with the webpage file. Despite higher amount of communication errors the temporal accuracy does not suffer in anyway.

## VIII. Conclusion

Through this paper a communication framework from Mionix HUB to either a Webpage or Unity has been described. Both of which capable of accessing, sending and recording the data. Furthermore, the Unity applications are able to do online analysis of the data in respect to a calculated baseline, such as baseline correction and percentage difference from the baseline.
Since there were two ways the data could get into Unity, through the Mionix HUB and through a .txt file created by the webpage, a small technical test was established. The test showed higher temporal accuracy for the GSR for the Mionix HUB to Unity communication and higher amount of communication errors as well. Furthermore, with this framework users with the Mionix Naos QG mouse will be able to carry out online and offline analysis of the data in respect to different kind of stimuli.

To get access to the implementations please go to this GitHub page. https://github.com/anwul4/Mionix-QG-Unity-and-webpage

## References

[1] Mionix, ‘Mionix Naos QG API’. Mionix, Stockholm, p. 1, 2017.

[2] Mionix, ‘Mionix HUB’. Mionix, Stockholm, p. 1, 2017.

[3] C. Kale, ‘TCP/IP tutorial’, *Spider Systems Limited*, 1991. [Online]. Available: https://www.rfc-editor.org/info/rfc1180%5Cnhttp://www.rfc-editor.org/rfc/rfc1180.txt%5Cnhttps://www.rfc-editor.org/info/rfc1180%5Cnhttp://www.rfc-editor.org/rfc/rfc1180.txt.

[4] I. Hickson, ‘The websocket api’, *W3C, Editor’s Draft*, vol. 15. 2010.

[5] JSON.org, ‘Introducing JSON’, *json.org*, 2014. [Online]. Available: http://www.json.org/.

[6] J. Reid and T. Valentine, *JavaScript Programmer’s Reference*, no. 1. 2013.

[7] J. K. R. Wärting, ‘FileSaver.js’. GitHub, Stockholm, p. 10, 2016.

[8] Benshayden, ‘Downloads Overwrites Existing Files’. Chrome Web Store, p. 1, 2016.

[9] J. Ferguson, B. Patterson, J. Beres, P. Boutquin, and M. Gupta, *C # Bible*. 2002.

[10] R. F. Potter and P. D. Bolls, *Psychophysiological Measurement and Meaning*, vol. 1, no. 1. New York: Routledge, 2012.

[11] STA, ‘Websocket-sharp’. GitHub, p. 5, 2017.