Mouse Dashboard Requirement

Table of Contents

[Introduction 3](#_Toc369468350)

[Desired Features 3](#_Toc369468351)

[Data sources 3](#_Toc369468352)

[Live 4](#_Toc369468353)

[Previously Saved Data 4](#_Toc369468354)

[Flash Control 4](#_Toc369468355)

[Display 5](#_Toc369468356)

[Complete Maze 5](#_Toc369468357)

[Zoomed View 6](#_Toc369468358)

[Sensors 6](#_Toc369468359)

[Cell 6](#_Toc369468360)

# Introduction

Invariably, during a mouse run, it does something that doesn’t make sense to the mouse designer. So, as the mouse designer, unless you instrument, record and then analyze the recording of what the mouse actually did, you are usually left speculating what it must have done. Sometimes you guess correctly and sometimes you don’t. If you did guess correctly, you have to then figure out why it did it, then fix it and then you can try to get it to do it again and if you are lucky, you find a good repro. case and are able to verify the fix/change. If you didn’t guess correctly or you can’t get a repro. you are out of luck.

To minimize the guess work, we propose to capture state information while the mouse is running through the maze and then if something goes wrong, we have the state information to work with to determine why the mouse did what it did. If we are lucky or were smart enough to instrument the right things, then a fix and verification of the fix should be much easier than if we didn’t have the info.

Also, over time, we can build up captures of data from various runs during tests and competitions and we can use this to not only find and fix obscure issues but compare behaviors over time, venues i.e. sensor and performance response to different mazes, etc.

This document contains the requirements for a Dashboard which will be used to satisfy the above need i.e. control and analyze the captured data.

# Desired Features

This section documents the features, what they are, what they do, how they are used, the inputs and outputs.

When using the dashboard we want to be able to have it show data in real time, aka Live, or previously saved data.

## Data sources

The mouse has the ability to store data locally on a flash and to stream data over a wireless Bluetooth serial link. The reason for two methods is that during development, we want to quickly and continuously see what the mouse is doing. So, if we only had the method where we store data on the flash, then during development, we would have to do a run, then connect the mouse to the host and transfer the data and then do the analysis. The step of connecting the mouse to the host is a productivity buster.

However, we cannot always use the wireless link. In particular, during a contest, the rules prohibit the use of a wireless link. So, we need the local storage capability.

To simplify things, we have one message format for storing data to the flash:

* Message ID: One byte.
* Message Size: One byte – this is the size of the Message Data. It does not include the Message ID or the Message Size.
* Message Data.

Since a wireless link can be lossy and we don’t have an easy way to detect the start of a message, when data is transmitted over the wireless link, we have to add some additional structure. A wireless message consists of:

* Packet Header: “Zt” - Two bytes.
* Packet Checksum: One byte checksum of the Packet Header and the Packet payload.
* Packet Payload: Message ID, Message Size, Message Data.

The overall length of a packet is:

* + + 2 (Packet Header)
  + + 1 (Packet Checksum)
  + + 1 (Message ID)
  + + 1 (Message Size)
  + + ? (Message Data)

### Live

For the Live case, we want to receive and display data from the mouse, in real time. In addition, this data shall be stored for future analysis. If the user hasn’t specified a filename, a default /intelligent /reasonable filename shall be used. This way, the user screwing up and not saying, save this session doesn’t result in data loss. The filename intelligence should include date and time such that sorting in the file explorer is trivial.

While data is coming in, in addition to saving it, the viewer must be able to use the data i.e. pause, go backwards, catch up to real time, etc.

A model could be that incoming data is always stored off and interaction is always from the saved file. An advantage of doing it this way is that the dashboard essentially has two functions – save data and display saved data.

### Previously Saved Data

For Previously Saved Data case, we want to read the saved data from a file on the local file system and look at the contents in a mouse specific manner.

### Flash Control

In addition to viewing data, the dashboard has the capability to control the flash. The control features are: Read Flash, Erase Flash, Abort Flash, Read Parameters, Set Parameters.

#### Read Flash

Read the contents of the flash and save it to a file. The format of the data in flash was described in section Data sources.

To perform this operation, the Dashboard will send a command to the Mouse and then the mouse will send back the data. The Dashboard will display progress of the operation including percentage and time left for the complete transfer.

**TODO:** define protocol – Host sends the command to the mouse. Mouse sends back total # of bytes and then starts sending the bytes. The host can use the total # of bytes to display percentage and based on the transfer rate, time left.

During this, the host can send an *Abort Flash* command.

#### Erase Flash

Erase the data stored in the flash. This doesn’t erase the parameters stored in flash.

To perform this operation, the Dashboard will send a command to the Mouse and then the mouse will send back progress. The Dashboard will display progress of the operation including percentage and time left.

**TODO:** define protocol – Host sends the command to the mouse. Mouse sends back total # of bytes to erase. The mouse sends back how many bytes it has erased.

During this, the host can send an *Abort Flash* command.

#### Abort Flash

If there is a Read Flash or Erase Flashoperation in progress, abort it.

To perform this operation, the Dashboard will send a command to the Mouse and then the mouse will abort the command in progress and send back an ACK.

**TODO:** define protocol

#### *Read Parameters*

Read the mouse operational parameters from the flash and save it to a file. The format of the data is **TODO: TBD**.

To perform this operation, the Dashboard will send a command to the Mouse and then the mouse will send back the requested information. The Dashboard will display progress of the operation including percentage and time left.

**TODO:** define protocol

#### *Set Parameters*

Read the mouse operational parameters from a file and send it to the mouse. The format of the data is **TODO: TBD**.

To perform this operation, the Dashboard will send a command to the Mouse and along with it, the parameters. The Dashboard will display progress of the operation including percentage and time left.

**TODO:** define protocol

## Display

The Dashboard shall have the following display elements:

Complete Maze – show mapped/unmapped walls.

Every time a wall is updated, except for the first time, display an incrementing number.

If a wall was marked as wall and then marked as no wall or vice versa, then display a wall mismatch. Properly mapped walls are displayed using a red line. A mismatch wall is displayed using a green line.

This view will show the current path.

Zoomed View – show the mouse moving through the maze. The mouse stays fixed in the middle but it rotates to show the current position i.e. the maze orientation doesn’t change and stays aligned with

show mapped/unmapped walls – follow style from Fat Cat – see appendix.

Show the sensors – follow style from Fat Cat. The Sensor distance shows

Sensors – on the mouse and bar graph.

Cell – visited VS not visited.

Path – received from the mouse.

Ability to create/generate/draw a path and send it to the mouse.

File format?

Over the air  
 files stored on the host

Visualization  
 Sensor graphs  
 Mouse moving through the maze  
 Showing sensor readings

The Serial port should support a ping command with a checkbox. If it is enabled, and there has been no traffic for 500ms, the host will send a ping command with a timeout of 250ms. If there is no response, the host shall close and reopen the serial port. If the port closure/reopen is successful, the host shall continue normal traffic. If it doesn’t succeed, the host shall delay for 500ms and retry the serial port, ad nauseam.