Coding Challenge

The purpose of this exercise is to evaluate your ability to write good code in a real-world programming exercise with the development environment you’re already comfortable using. T**he scenario for this code challenge is based on an engineering challenge we had to solve early on as a team, and we think it’s a good way to get a sense of your abilities as an engineer working on our team.**

### We’ll assess your solution for the following (in priority order) :

1. Does the program work as specified (ie, given the input file, does it produce the output?)

2. Is the code written in a way that will perform well, scale well under load?

3. Is the code easy to read and maintain?

4. Did you include unit tests that validate your implementation?

### General Instructions:

This coding challenge is to be completed independently and returned by email (you can send as an attachment or just link your Git repo in the email) within a calendar week. If you have any questions about the instructions or want to take additional time please let us know - we want to make sure you have any opportunity to show us how awesome you are without adding additional burden to your schedule.

You may implement your solution in any language, we suggest C++, Python, or Java if possible.

### Next Steps after the code challenge:

Our team will immediately confirm receipt of your code challenge, review your submission and follow up within three business days.

## Let’s get started! See code challenge content on next page.

## 

## Code Challenge Scenario & Description

Your new project requires you to simulate ingesting data from an external Eye Tracking device ([look at this Tobii device if you want an example](https://www.tobiipro.com/product-listing/tobii-pro-x2-60/" \l "Features)) to be consumed by your custom application for use by internal and external services in near-to-real time. Your application is composed of two components running on a single or Linux machine or Android device. Component A should handle data acquisition from the Eye Tracking device and component B should handle data consumption, event handling and distribution. You can assume the Eye Tracking device has a wired connection, will publish data at 60Hz, and will provide all the fields required for you to create the data structure described below. The two components should communicate via a lightweight interprocess communication (IPC) mechanism. As the eye tracking data will be used to provide real-time gaze position to the UI, you will want to focus on a low latency implementation. You can use any message libraries, frameworks, encodings, languages, etc that best fit the following requirements.

**Your goal is to develop a system that:**

1. Launches a process that constructs and populates messages using the format specified below (simulate raw data provided from an eye tracker at 60Hz for this example). [Component A]
2. Send these messages to be consumed via another process on the same machine. [IPC]
3. Runs a process that receives these messages and logs the data to a file system or aggregate data store. [Component B]

Message Format

Timestamp : Seconds: Int64U, Nanoseconds: Int32U  
 Timestamp of message received  
ID : Boolean  
 0 – left eye, 1 – right eye  
Confidence : Float32  
 0 – no confidence, 1 – perfect confidence  
NormalizedPosX : Float32  
 Normalized x-coordinate of the pupil location, 0 – left, 1 – right  
NormalizedPosY : Float32  
 Normalized y-coordinate of the pupil location, 0 – bottom, 1 – top  
PupilDiameter : Int32U  
 Diameter of the pupil in image pixels

**Your solution’s non-functional requirements:**

1. You should design your code with modular interfaces that make it easy to move to different messaging and transport libraries if needed.
2. Your solution should focus on low latency whenever possible without sacrificing modular design constraints listed in 1.
3. Your solution should adhere to best practices in naming and documentation for the language you select, and should provide unit tests as needed.

**Things you should consider in constructing your solution:**

1. Design trade-offs such as why a particular message or transport library was selected, advantages/disadvantages, effects on latency and throughput, scalability, long term library support, etc.
   1. Include these and unit tests in jupyternotebooks
2. Changes needed to accommodate acknowledgement of the process’ identity and how to ensure appropriate handling of messages from unidentified processes.
3. How the data is logged and how you would query the information in post-hoc analyses.
4. How would your implementation differ given a wireless connection between eye tracker and Component A?
   1. Using an eyetracker, we no longer need to generate fake messages
   2. The message constructor would need to interpret messages from the eye tracker and then repackage them to be sent to the data logger.
   3. Alternatively, may be able to just use the data logger to listen directly for the wireless messages and cut out the middleman