**StreamLabs Test Assignment README**

**Requirements**:

*NamedPipe* is the only **hard requirement** for the IPC

1. The client should be able to connect to the server through a NamedPipe
2. The client should be able to make both sync and async calls to the server
3. The client should be able to send trivial data (strings, numbers) to the server
4. The client should be able to create objects on the server (based on req-7 below), retrieve them, their attributes and call methods on them
5. The server should be able to receive both sync/async connection requests from clients
6. The server should be able to store data provided by the client via NamedPipe in a reasonable data structure
7. The server should be able to register a custom class (w/ related functions, attributes) which can be used by the client (see req-4)
8. The server should be able to store the custom objects created by the client for the custom class created in req-7

**Design Choices**:

1. I developed all the functions of the applications using one single \*.cpp file for each (one for the Client and one for the Server).
2. I Used Visual Studio 2017 Community as IDE.
3. In this test assignment, the storage will consist of text files (for the data) and folders (for organization of data types and to allocate the custom class files), the best choice is to use a Database (maybe MySQL) but there isn’t a requirement asking for a Database feature and the time asked for the execution of this test doesn’t help to design one with some many data types and possibilities.
4. The Client was designed with 5 main functions:
   1. Main – Responsible to instantiate all the services and UI (the UI will be developed for testing purposes only)
   2. Read – Read data from the server
      1. It has two types of read: for common data and for custom class/objects
   3. Write – Write data from the server
      1. There are two types of write too, on for the common data and another for the custom class/objects
   4. There are other functions to do secondary tasks, as:
      1. Convert data types;
      2. Compile and execute the Custom Class
5. The Server has 3 main Functions
   1. Main – Responsible only for the initialization and to call the selected connection type.
   2. Message Interpreter – Responsible for interpret the messages received by the server, it execute the requested operation, I didn’t create an external function for it, so all this function is allocated internally on the Sync/Async Operation Function
   3. Connection Verification – Responsible verify if the pipe isn’t broken, if it’s broken then this function starts the disconnectAndConnect function. This function is allocated inside the read function.
   4. Read – Read data from the Client
      1. There’s a loop to make the server ask for requests from the Client all the time
      2. It also has two types of read, one to register Custom Classes/Objects and another to make requests/receive common data
   5. Write – Write data to the Client
      1. The same as the read function: two types, one for custom class/object and another to send common data (text and numbers)
6. All the operations that involve Custom Objects manipulation as responsibility of the custom class, the server has support to store the custom objects and to send to the client when asked, but the client doesn’t have any function to manipulate them. The simple reason is because the custom class can create any kind of object and it’s impossible to predict which data type it is.

**Assumptions**:

1. The req-7 function of registering a custom Class will be made like this:
   1. The server receive from the client the \*.cpp and the \*.h files and save them into a custom folder.
   2. When the Client ask for the custom Class the server will send the \*.cpp and \*.h files and the client will compile it using MinGw compiler (the compiler mus be installed under c:\Mingw folder to the custom class functionality work). The the client will make an instance of the custom class using LoadLibrary() function (I know the best practice is to use the CreateProcess() Function but for some reason it didn’t work during the tests and the LoadLibrary() works well) and execute a process (a constructor method maybe) using the GetProcAddress function.
   3. In order to store custom objects the developer of the custom class need to call the read and write functions from the main Client application using a similar method as used to execute the custom class functions.
2. The Custom objects created by the Custom Class (req-8) will be stored in a folder allocated within the custom Class files (\*.cpp and \*.h)
3. The System only allows the connection of one client at a time to the server and, it accept both sync and async connections but only one kind of connection each time the server is initialized.
4. The requests are made by the request ID (always mData[0]), which are:
   1. “w” – request a welcome message from the server.
   2. “as” – request all the texts (strings) saved within the server
   3. “af” – request all the numbers (floats) saved within the server
   4. “s” – Ask to save a text (string)
   5. “f” – ask to save a number (float)
   6. “ccr” – Custom Class/Object Register/Save
      1. If the request data mData have a third argument (mData[2]) which is the custom object name the server will understand this request as an object save one, if the mData doesn’t have a third argument so the server will understand that the User wants to register a custom Class named mData[1].
   7. “ccc” – Custom Class/Object Call
      1. If the request data mData have a third argument (mData[2]) which is the custom object name the server will understand this request as a custom object call, if the mData doesn’t have a third argument so the server will understand that the User wants to instantiate a registered custom Class named mData[1].