**FUNCTIONAL SPECS V1.0 (Kevin)**

**QT GUI Parameter Capture**

**Kevin Wang, David Somocurcio, Navaneet Galagali**

**September 10, 2014**

**NOTE: FUNCTIONAL SPECS MAY CHANGE AS PROJECT DEVELOPES.**

**USE CASES**

This project will likely consist of several C++ header files. Each will have a notable class in the architecture diagram of the project. The entry header file will be Pipeline.h which is a C++ header file that has the pipeline class that can save/load parameters in the GUI.

The derived (controller) class provided by the user should handle all instantiation of the pipeline class. As such, it will also need to handle method calls to the pipeline instance and add in the necessary parameters for the methods to start the save/load workflow.

Computer Science/Psychology graduate student can use the header file packs to instantiate the pipeline class, which serves as the entry point to the whole program package to saving the parameters in the QT GUI into another XML file called parameters.xml.

They will use the library package to save the parameter in the GUI. This will most likely be a method call on the instance of the pipeline class in the pipeline header file. The method signature will include parameter for an XML file representation of the QT GUI XML file and another that represent the parameter needed to be saved from the QUI GUI to the output parameter XML file.

Computer Science/Psychology graduate student can use the header file packs to instantiate the pipeline class, which serves as the entry point to the whole program package to loading the parameters from an parameters XML file.

Similar to saving, they will use the library package to load the parameter into the GUI. This will most likely be a method call on the instance of the pipeline class in the pipeline header file. The method signature will include path for a parameter XML file of the QT GUI. The method will return the GUI parameter data instance which the derived controller class will then use to fill out the relevant fields in the QT GUI.

Additional functionalities may be added upon request of client given enough deadline clearance.

The most likely error in the save/load workflow is a wrong GUI parameter data instance created by the client’s derived class that does not contain the adequate amount of fields. This will most likely result in the ParameterIO class crashing during compile when getters are called on the GUI parameter data instance. This can be mitigated by emitting an error when the number of field data in the GUI parameter data instance does not match that of the QT GUI XML.

**Requirements**

1 Pipeline Class (AKA User input/output)

The pipeline file handles all interactions with the client’s codes. Moreover, the pipeline is the one to handle the save/load operations. The client needs to obey the save precondition that the correct data from the GUI are inserted as parameter and the correct QT GUI XML object are parameter of the save function call. In return, the save method call will return a boolean result; false for save fail and true for save success. The client will also need to obey the load precondition that the correct file name/path of the already created parameter XML exists. Moreover, the parameter XML file is the correct parameter for the QT GUI XML or the loader will encounter runtime error putting the data from the XML file into the GUI data parameter instance. The load method will return the GUI data parameter instance filled with data from the parameter XML. As a guarantee, the pipeline will call the parser (for save) or loader (for load).

1 Parser Class

The parser class is another extremely significant class. Based on whether the operation is load or save, the parser will engage in the appropriate set of operations. For save, the parser will first create an empty XML file from the QT GUI XML to store the parameter capture. Next, the parser will create dynamical parameter IO class that will another, most likely built-in, XML parser class to save the fields from the GUI parameter data model instance into the parameter XML file. For load, the parser will create a parameterIO class that can parse XML with the aforementioned built-in XML parser class and put the data in the parameter XML into the GUI parameter data model instance.

2 XML Parser

Not to be confused with the parser class. The XML parser is mostly likely going to be QT GUI’s built-in parser or a class that employs delegation to add additional functionality to the vanilla built-in parser. This class will be used to write to or read from various XML files. Therefore, it requires 3 methods at a minimum: save, insert, and load.

3 Error Handling

The header library should be able to handle most notable errors from the client side code. The most important of which is the client side providing mismatched QT GUI XML and the GUI field data. The program should be able to safely swallow such errors and just emit a warning.

3 General Requirement

Save and load to an XML files that contain tags for each GUI elements and its corresponding data.

**Interfaces**

There is only one input and output interface into the client application: the pipeline class. The pipeline will handle all save info call by the derived (Controller) class. Ideally, the class should consist of a save method called by the derived class and the parameter will be the read in QTGUI XML and the GUI data instance (or file). The GUI data instance should be some form of description of the data already present in the GUI. The pipeline class can also be called on to load data from a parameter XML file. A GUI data instance should be returned to the client’s derived class where the data is put into the GUI.

Additionally, the program will save the input GUI data file into XML. The parser will create ParameterIO class that will take data from the GUI data instance and insert the separate data into the parameter XML. The users can then easily access the parameter XML file and look at the saved parameters for each GUI element in the GUI.