**FUNCTIONAL SPECS V1.1**

**QT GUI Parameter Capture**

**Kevin Wang, David Somocurcio, Navaneet Galigali**

**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 have the pipeline class that can save/load parameters in the gui.

The derived (controller) class provide 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. User should be able to read and write to the parameters XML file. Therefore, validation during loading phase is a must.
2. Each field that needs to be captured will be denoted by special characters preceding its QT GUI XML tag name. Tag attributes will also potentially be explored.
3. User can activate the program via command line to generate model class for the QT GUI. The derived class that inherits from the QT GUI should be able to import the model class and use it as data store for the GUI.
4. The program should be cross platform as QT itself is cross platform. But, there may be corner cases needing to be addressed
5. XML from a different study could be loaded. However, checking is required to ensure the consistency with the GUI model.
6. User can get the option to overwrite to XMLs on save.
7. User should be able to specify the name of the XML file they wish to load/save to at the start of the program. However, changing the file en running should not pose any major development issues.
8. OPTIONAL: Automatically generated class that can call the appropriate setters/getter in the GUI model. The derived class can then inherit from this class to gain such functionalities. This is done so that the users do not have to manually code getters and setters into derived class.
9. OPTIONAL: User will be able to combine different parameter XML files into one GUI model class. This will enable the user to mix and match different parameter XML files.

**CONSTRAINTS**

1. XML file from a different study can only be loaded under the condition that all fields in the XML file corresponds to a field in the GUI. Else, the program should fail gracefully (maybe output a warning message). The user’s program should not be interrupted.
2. The program should fail gracefully under common errors, such as:
   1. Loading a conflicting XML.
   2. Saving to an XML that does not exist (Create a new on potentially?)
   3. Saving to an XML that does not match the GUI.
   4. OPTIONAL: saving to a model that does not correspond to the GUI. Only if the automatic saving class generation is complete.

**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.

**Revision History:**

**9/28/2014 V1.1**

**-**Redone Requirements

**9/10/2014 V1.0**