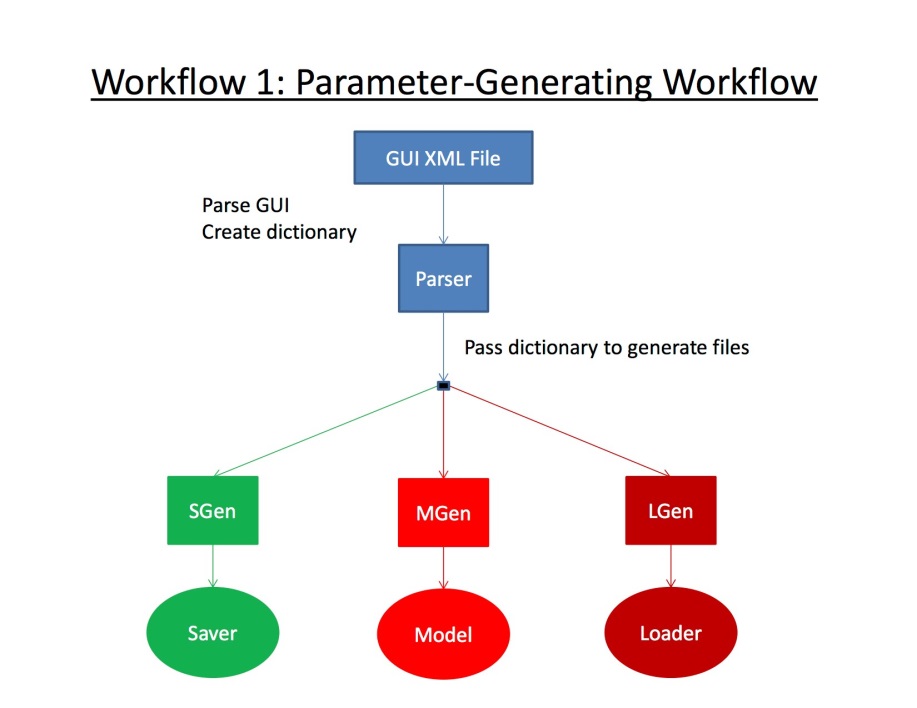
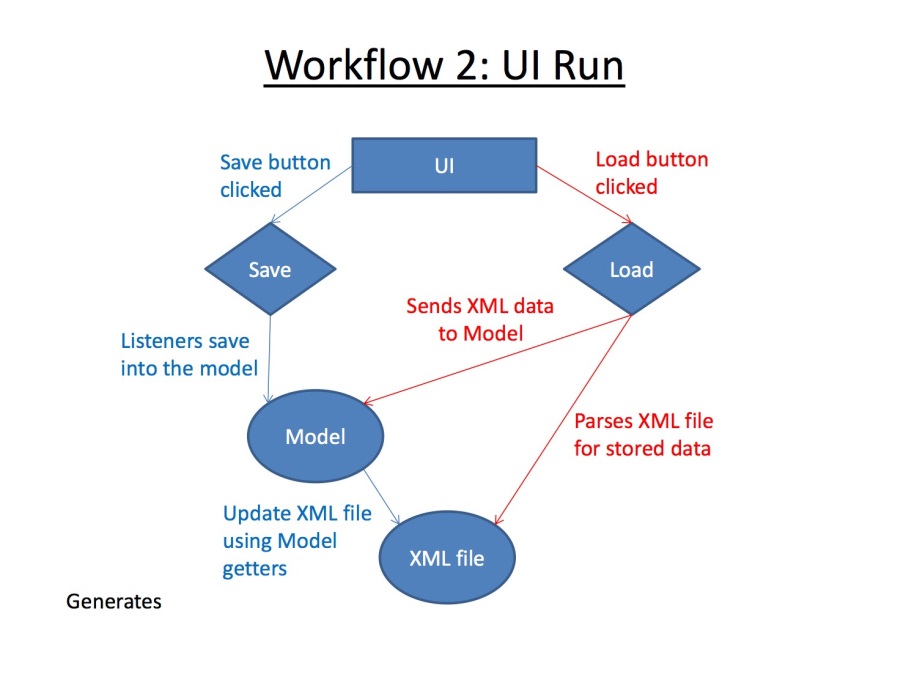
[](https://sites.google.com/site/qtguiproject2014/final-design-document/Workflow1.jpg?attredirects=0)[](https://sites.google.com/site/qtguiproject2014/final-design-document/Workflow2.jpg?attredirects=0)

**Modules**

The software runs on two different workflows: one handles parameter generating and the other handles updating from and to the model. The Parameter Generating Workflow uses our designed Parser class (parser.h and parser.cpp) to analyze the user designed UI file from the Qt project to create a dictionary of widget classes (also known as types), names, and internal data values. This dictionary is then passed to three different classes: SGen (sgen.h and sgen. cpp), MGen (mgen.h and mgen.cpp) and LGen (lgen.h and lgen.cpp). These generators were designed to create Saver, Model and Loader classes based on the various forms of input the user designed in their project.

The Model class contains properties that hold every possible data input the user designed on their GUI. Based on the widget name, the Model's method name for each property's getters and setters are built. The Saver and the Loader classes take in the instantiated Model class but their uses will be explained thoroughly in the next workflow.

The other workflow (UI Run) handles model updating through the use of model/UI interaction. For example, if the user's project had "Save" and "Load" listeners (i.e. buttons) then the Model/Saver and Model/Loader interactions would be called, respectively.

The Model/Saver interaction starts when the UI addresses "Save". The inputted info that is currently in the UI is updated into the Model (using its setters). Afterwards, the now updated Model is fed into the Saver, which uses the Model's getters to extract its data and update the data.xml file. If a data.xml is not currently in the directory (i.e. first time using this software's save feature), one will be created. Else, the current data.xml file will be overwritten with this updated data.xml file.

The Model/Loader interaction starts when the UI addresses "Load". The Loader class takes in the Model class and begins to parse through the data.xml file. For each property within the Model, the Loader pulls its value from the data.xml file and updates the Model's corresponding property using its setter methods. Once the Loader is done updating the Model, the Model updates the UI file using the Model's getter methods. Since the UI file has been updated, Qt Creator will notice the change and asked to consider these new changes. If "Yes to All" is clicked, the currently running GUI will be updated with these values.

**Data**

As of December 6, 2014, the software handles the following Qt widgets: QTextEdit, QCheckBox, QRadioButton, QComboBox, QSpinBox, QDoubleSpinBox, and QListWidget. These widgets store the user data as QStrings, Booleans, integers, and doubles. The data types of Model's properties are determined by the analysis of the Parser. This handling of assigning data types based on specific QWidgets is pre-programmed. As explained in the Modules section, the Model's getters and setters are heavily used within the Model/Saver, Model/Loader, and Model/UI interactions. The end components of the architecture, UI file and the data.xml file, are connected by these interactions. While the UI file will store the value for displaying in the GUI, the data.xml is the actual storage of the parameters.

**Design Decisions**

The primary dependency for our application is the Qt XML file, which needs to be fed in for parsing to take place. We have chosen to separate the saving and loading aspects of our application into different modules, named Qt XML Parser Class and Qt XML Loader Class respectively. In this way, each component can be modified and used independently of the others.

The functional spec goes into further detail on each module and details of implementation.

Link to the code repository: <https://github.com/QTGUIFolks/QTGUI>