Technical Documentation – VMol

**Technologies/Software used for building the application:**

1. Eclipse IDE, Mars .2 Release (4.5.2)
2. JDK 8 (Mine is Java version - 1.8.0\_101)
3. Java FX – Install this as a new software within eclipse – named “e(fx)clipse – IDE”. Link: <https://www.eclipse.org/efxclipse/install.html#for-the-lazy>
4. SceneBuilder – for visualizing *fxml* files (UI files of the application). Download it from here – (http://gluonhq.com/products/scene-builder/). We can configure to open and edit the fxml files directly from Eclipse. After downloading the SceneBuilder set the path to its exe file at Window -> Preferences -> JavaFX -> Scenebuilder Executable in Eclipse.
5. JMol Jar file (used for visualizing the molecules within the application).

Refer to this javafx and SceneBuilder tutorial - http://code.makery.ch/library/javafx-8-tutorial

**WorkFlow and relevant files/pages in the application:**

*Main.java* – This is the main file which launches the application by loading *MainView.fxml* file. Every fxml file has an associated controller file. MainView.fxml is linked with MainViewController.java.

*MainViewController.java* – This class is responsible for handling calls to open/visualize files. The method *openFile* has the appropriate logic for either visualizing the xyz or pdb files/ or parsing the file (in case of libefp input/output file). Xyz and pdb files are directly passed to *JmolVisualization*.*java* for visualization. If not, the file is opened in a new window (for parsing) by loading *FileParser.fxml.* The other method is *openLibEFPWindow (or openQChemWindow –* The name is changed to libEFP to indicate that appropriate changes need to be done to implement LibEFP instead of QChem, QChem acts as a base for its implementation*).* This method’s name is responsible for generating LibEFP/Qchem Input files necessary for submitting jobs to LibEFP server. The other method is *openServersListWindow* which is responsible for loading the *ServersList.fxml* used for editing or configuring servers.

*JmolVisualization.java –* This class contains code for integration with JMol for visualizing the xyz or pdb files. JMol uses swing’s framework for displaying the molecules. However, as we’re using Javafx framework it was tricky to open this JMol window through javafx. Also, there were some bugs which couldn’t be fixed easily when we tried to integrate Jmol with javaFX. Hence we’ve adopted to open the files separately in a new jFrame window for visualization. The current file has two main methods - *show* and *showMultipleFiles.* Show method is called for displaying a single xyz file at a time. showMultipleFiles is used to visualize multiple files simultaneously within a single window. Both the methods logics are similar however, showMultipleFiles method has an additional logic of displaying prev and next buttons used for navigating across the xyz files. The showMultipleFiles method was implemented recently, hence some refactoring is needed for this file.

*QchemInputController.java* – This class is responsible for generating QChem Input file. All the parameters in the file are written in such a way that a Qchem input file can be generated. However, we need to modify this so that LibEFP input can be generated appropriately. This should hopefully be straightforward. Just change the names of the parameters in *QChemInput.fxml* using the SceneBuilder. All the fields however are initialized in the initialize method of the controller, hence this method should be changed appropriately. Slight changes might need to be done to the *generateQChemInputFile* method (like changing the extensions). Submitting the current input file to a server is handled by the *handleSubmit* method. This method is currently implemented for a local server scenario (testing has been done too). In case of SSH scenario, first a connection need to be established before executing any command. *Jsch* jar can be used for this purpose.

*ServerConfigController.java –* This class handles the adding/editing the servers list in the application by loading *ServersList.fxml*. As mentioned earlier, jobs can be submitted to any of the configured server. Users can add and edit servers using this module. Each server is implemented as a serializable class named *ServerDetails.* For testing purposes, a dummy server is created and added by default and is shown to the user when the user first launches the application (This can be deleted or can be replaced with appropriate default LibEFP server). However, the user can add/remove/edit servers. These calls are handled by methods – *handleAddServer*, *handleEditServer* and *handleDeleteServer.* Once any of these methods is successfully executed (i.e., either user clicks ok in the next module or deletes a server), the *updateServerDetailsListInPreferences* method is called which updates the preferences with appropriate servers list. Java Utils Preferences class is used for storing this user level information.

When the user clicks add/edit server, *ServerEditView.fxml* is loaded which enables the user to edit the server accordingly. A default set of queue options (used for PBS) are loaded when server hits the add server. This method can be accordingly for a LIbEFP server.

*ServerEditViewController.java-* This class handles adding/editing a server by loading *ServerEditView.fxml*. *setServerDetails* method needs to be called before loading this module. There are three main methods in this class – *handleOk*, *handleCancel* and *handleConfigure.* Once the input is validated, handleOK sets the appropriate values in the serverDetails and returns. The *handleCancel* method closes the window directly. The *handleConfigure* methodloads the *ServerEditConfigView.fxml* file for editing the queue options (necessary for PBS and in other cases).

*ServerEditConfigViewController.java* – This class sets the values of the queue options for that server by loading *ServerEditConfigView.fxml* file. It does a basic validation of the user inputs and set the queue options appropriately. Queue options is implemented as a new serializable *QueueOptions* class inside the ServerDetails class. This class might need to be updated for LibEFP. Once input validation is done, queueoptions are set appropriately which are then used in the *ServerEditViewController* class.

These are the high-level technical details of the important classes in the application.

**Things to note for future work:**

As already said, appropriate libEFP input needs to be generated and can be submitted as a job to any of the configured local servers. Testing has been done for local server job submission scenario for QChem case. The testing has also been done for libEFP job submission scenario (just by copying the libEFP input file contents into the *qChemInputTextArea* of the QChemInput.fxml file and submitting the job to a configured LibEFP local server). This needs to be extended for SSH sever scenario.

Visualizing all the geometries contained in the output file (obtained after submitting the job to the server) is currently being implemented. However, once the job is submitted, the output file can directly be visualized without going through File->open workflow. But this can only be done for small jobs. For time consuming jobs or jobs submitted via PBS, work has yet to be done to check the status of the submitted jobs and show them to the users.

**Instructions:**

Running the project from Eclipse – Import the ‘*Vmol*’ project to eclipse’s workspace and run the Main.java file. This will launch the application.

Exporting the Project as jar file – From eclipse, click File -> Export -> Runnable Jar file. For the launch configuration, you need to run the Main.java for the first time and mention it in the configuration.