LA-UR-15-26552
Approved for public release; distribution is unlimited.

Title: PyFLOTRAN-GUI: A Graphical User Interface for PFLOTRAN

Author(s): Islam, MD Raqibul

Intended for: Report

Issued: 2015-08-19

PyFLOTRAN-GUI: A Graphical User Interface for PFLOTRAN

MD Raqibul Islam

The City College of New York

Abstract

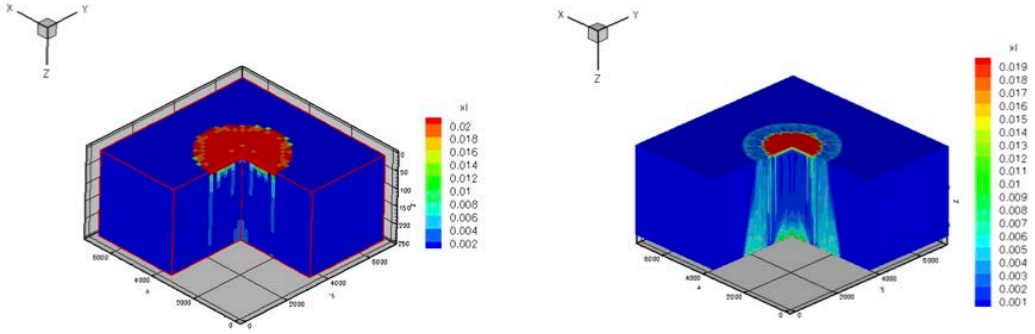
PyFLOTRAN is a Python front-end for improving workflow with PFLOTRAN, a massively parallel reactive flow and transport model for describing surface and subsurface processes. PFLOTRAN has been developed at the US Department of Energy national laboratories and is used for various applications related to energy and climate applications such as CO_2 sequestration, geothermal energy extraction, unconventional oil & gas extraction, nuclear waste repository science, Arctic hydrology, etc. PyFLOTRAN is capable of both pre- and post-processing of PFLOTRAN input files directly from Python code which allows users to be more productive as a result of not having to remember all the technical details of PFLOTRAN. The scope of this internship was to improve upon the code base of PyFLOTRAN, and also to create a graphical user interface (GUI) for PyFLOTRAN, for further improving the workflow. The GUI adds an abstraction layer, on top of PyFLOTRAN, which improves the time spent in setting up input files for PFLOTRAN. The GUI is built entirely in Python, with the help of the PySide module, and is capable of creating and running input files in addition to providing the benefits of a user interface, such as a visual representation of the creation of the input file.

I. Introduction

a. Background

PFLOTRAN [1] is an open source parallel code to model subsurface flow and reactive transport developed in the Department of Energy national laboratories. An application of the code is shown below of CO_2 sequestration of a dissolved CO_2 mole fraction over 300 years.

Figure 1. "Dissolved CO_2 mole fraction corresponding to an elapsed time of 300 years" [2]



PyFLOTRAN, written entirely in Python, provides an abstraction layer on top of the existing PFLOTRAN code, which is written in Fortran. PyFLOTRAN addresses some of the current bottlenecks in working with PFLOTRAN such as, the amount of time used in pre-processing, creating the input files (*Figure 2*), and post-processing the data.

Figure 2. Snippet of input file.

```
SIMULATION
  SIMULATION_TYPE SUBSURFACE
  PROCESS_MODELS
    SUBSURFACE_TRANSPORT transport
    /
  /
END

SUBSURFACE

CHEMISTRY
  PRIMARY_SPECIES
    H+
    HCO3-
    Ca++
  /
  SECONDARY_SPECIES
    OH-
    CO3--
    CO2(aq)
    CaCO3(aq)
    CaHCO3+
    CaOH+
  /
  GAS_SPECIES
    CO2(g)
  /
  MINERALS
    Calcite
  /
  ACTIVITY_COEFFICIENTS TIMESTEP
  MOLAL
  OUTPUT
    PH
    all
    FREE_ION
  /
END

…
```

PyFLOTRAN allows users to write code in Python, and generate the input files within Python, in addition to doing post-processing work. The PyFLOTRAN-GUI adds a further layer of abstraction to what PyFLOTRAN does by presenting a graphical representation of the workflow. Instead of having to write Python code, the user can interact with the GUI elements to obtain the same results as with PyFLOTRAN; this allows more meaningful work to be done rather than learning how to use the code, and spending large amounts of time in setting up input files.

b. Sections

Section two addresses the dependencies of the GUI, the features, the design pattern of the code, issues, and comparisons of the two workflows (with and without PyFLOTRAN-GUI). Section three addresses the future work regarding the concerns with the PyFLOTRAN-GUI. Section four contains acknowledgements.

II. PyFLOTRAN-GUI Description

This section describes the Python modules used for developing PyFLOTRAN-GUI, the functionality and design of the GUI, along with the issues that were faced during the development of the GUI.

a. Python Modules

1. PySide

PySide [3] was the primary Python module used to create the graphical user interface, and provides bindings to the Qt framework [4] since Qt is built with C++; most of the advantages of Qt are provided through the module such as, cross-platform support, and many GUI components.

2. IPython

The IPython [5] provides a scientific environment as an added layer to the Python interpreter, and is incorporated into the PyFLOTRAN-GUI as an optional feature.

3. Unittest

Unittest [6] provides a unit testing framework which allows automated testing, and saves time when testing the GUI or changing the code and making sure the code still functions correctly.

b. Functionality of PyFLOTRAN-GUI

Figure 3. Main window of the graphical user interface.

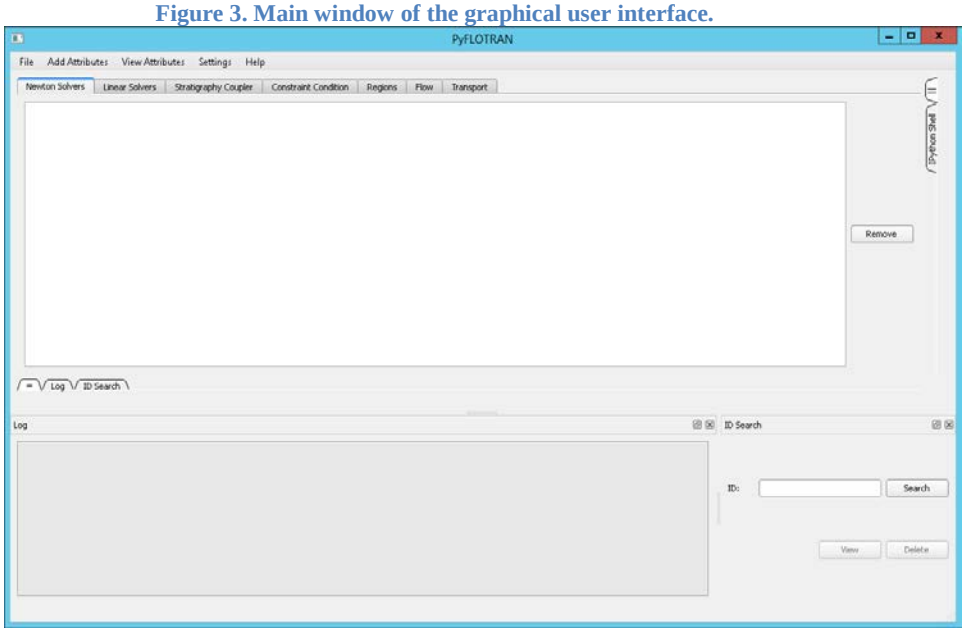


Figure 3 represents how the graphical user interface appears in Windows 7, although it is cross-platform as a result of Qt supporting Windows, Linux, and Mac OS. The main window consists of a logger, which records actions taken in the GUI; an important function of this is to provide an easy way to re-open widgets, which allow previous input to be overwritten or corrected.

Figure 4. The simulation settings widget in Ubuntu 14.10.

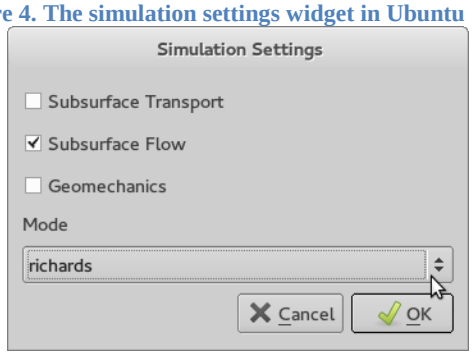


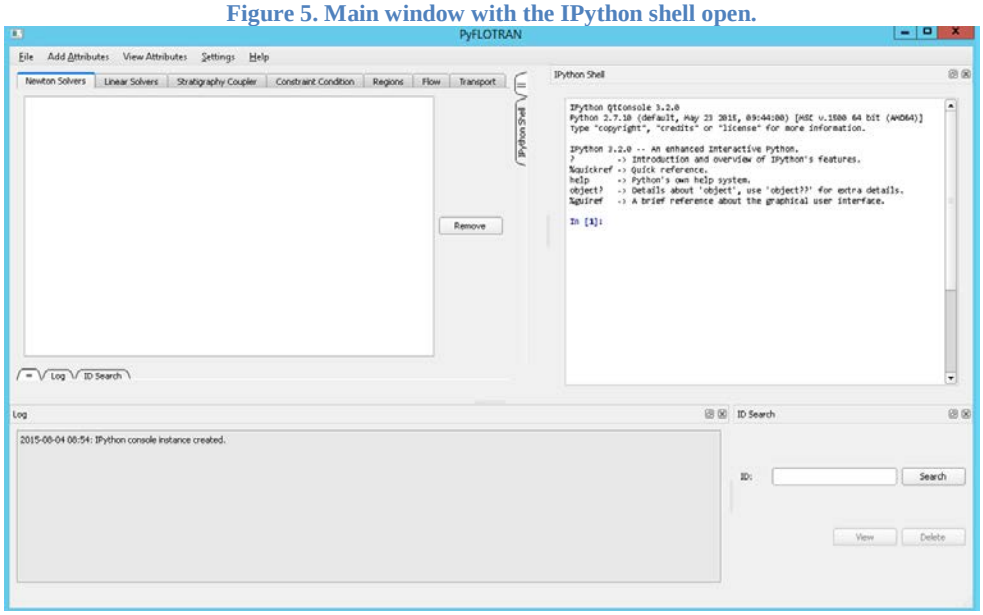
Figure 4 shows a widget where simulation settings can be inputted and, later on, if a user wants to change the data, the ID of the widget can be extracted from the logger and used to re-open the widget. The ID Search toolbox, bottom-right on the main window (*Figure 3*), allows widgets to be instantly modified, or deleted, if necessary. It is important to note that instances of these GUI elements are stored through persistent labels in Python, and the instances are called back when they need to be modified or deleted. As such, the instances are lost when the main window is closed since Python will use the garbage collector to remove them.

Besides using the ID Search toolbox, the ‘View Attributes’ menu option and the tabbed panes are available to alter existing data inputted by the widgets. The tabbed panes represent the

attributes which there can be multiples of and these attributes are presented in a list format. The menu option is for attributes where only one instance can exist, as such, when the data is modified for a particular attribute in that category, the old data is dumped and overwritten.

The main window also allows the usage of the IPython module directly from the graphical user interface, which is natively supported by the module.

Figure 5. Main window with the IPython shell open.

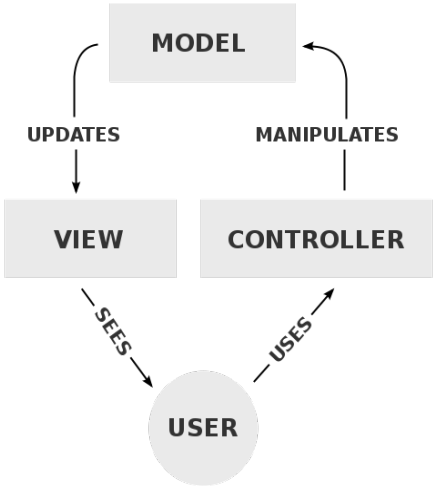


Any layout changes in the GUI is persistent, as long as the application runs on the same operating system, and the registry keys are not wiped. The PySide module provides a method of saving the state of the user interface locally which means the GUI can be made more personal or tailored to one's tastes. Reverting the layout is also an option under the 'Settings' menu.

The 'File' option in the menu allows input files to be written, and also to be run; any errors that may occur will be recorded in the logger.

c. Design Pattern

Figure 6. MVC design diagram. [7]



The graphical user interface uses the MVC [8], model-view-controller, (*Figure 6*) design pattern. Using this design pattern allows a separation of the business logic, and user interface (UI) components which makes it possible for these individual components to be modified without having to change every part of the code structure when a minor, or major, change is implemented. Figure 7 shows the directory structure of the PyFLOTRAN-GUI project using the MVC design pattern.

Figure 7. Directory structure using MVC.

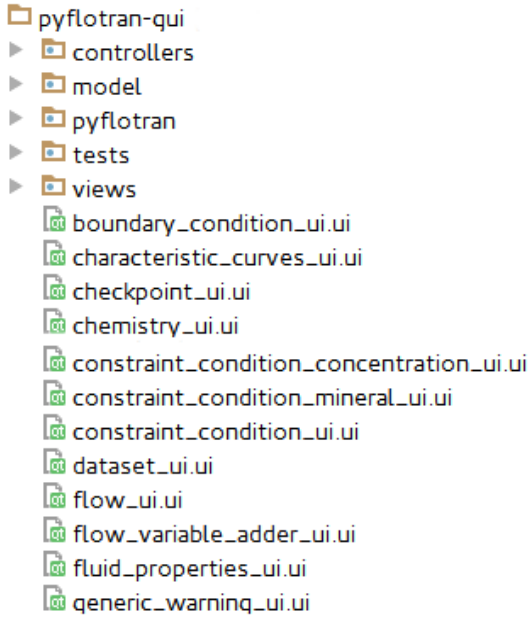


Figure 8. The process to implement a GUI component from design to code.

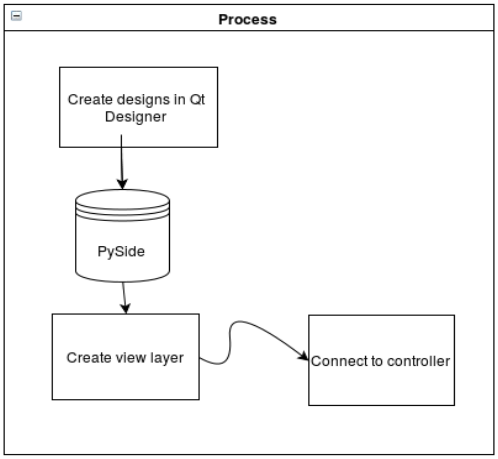
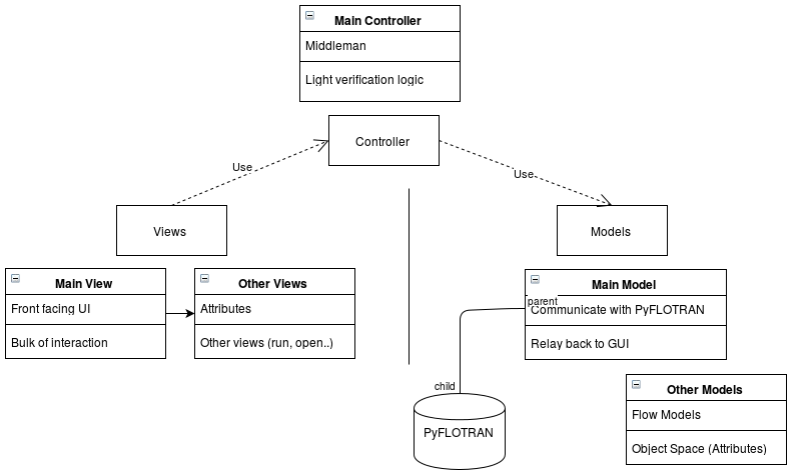


Figure 8 shows the steps it takes to create a piece of the graphical user interface; this process allows changes made at the design level to stay at that level and not affect the Python code layer. UI components are first designed in Qt Designer, which is provided by Qt, then put through the PySide converter that converts the XML code generated by Qt Designer to Python code. Since the converted code is overwritten every time the converter is run, an additional view layer needs to be added to give functionality to the GUI, such as making an action happen when a button is clicked. An important aspect of the view layer is that it also connects the view to the controller.

Figure 9. PyFLOTRAN-GUI code structure diagram.

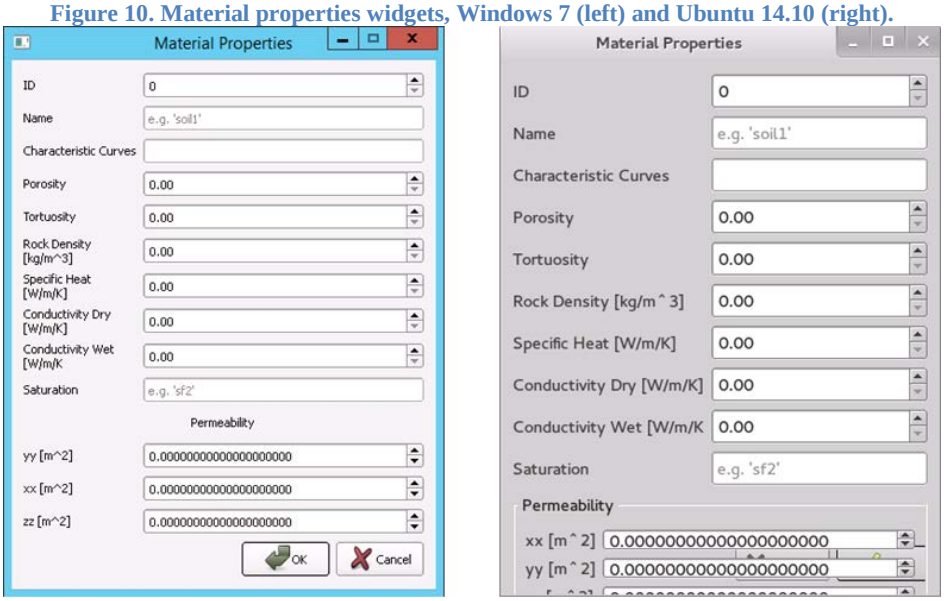


The controller connects the components together making it the bridge which talks to both the model, and view. The view presents the graphical user interface to the user, this is what is shown on the display, while the controller takes the user input from the view and translates it to the model. PyFLOTRAN is responsible for the bulk of the business logic; it can run and create input files provided by the model from the MVC tier. Using MVC allows the user interface component to be separated from the PyFLOTRAN code which makes it so any optimization made in the code will not be hindered by the addition of the GUI. As such, testing can be done for each layer of code individually without having concern with one layer directly affecting the other. Figure 9 illustrates the full PyFLOTRAN-GUI code structure diagram.

d. Issues

While Qt provides cross-platform support, GUI elements do not always come out as intended. For example, consider the following two components, left is Windows 7 and right is Ubuntu 14.10.

Figure 10. Material properties widgets, Windows 7 (left) and Ubuntu 14.10 (right).



While a view can look perfectly fine on one operating system, in this case Windows 7, the GUI can be misrepresented, such as the toolbox buttons being pushed behind the Permeability box (right), in another operating system. Even though these designs were constructed in Qt's native form editor, Qt Designer, the GUI elements still have issues. This issue came up with quite a few graphical components, and specifically occurred with the Group Box elements in Qt. Fortunately, since the project structure uses MVC, only the Qt UI files needed to be altered without any additional modification to other major parts of the code in the project.

GUIs aim to tell the user precisely what the program requires, such as having number input fields for data that is required as numbers. However, a limitation with Qt is that scientific notation is not supported in such number fields. Furthermore, to obtain a certain precision, a line of zeros must be added which makes the user interface suffer, as shown under Permeability (*Figure 10*). An alternative is to use text fields, which some widgets compromise on; however, this dilutes the purpose of a graphical representation of the PyFLOTRAN workflow since the user needs to be told directly what kind of input the field requires instead of naturally recognizing it if it were a number field.

Obtaining PySide can also be an issue on Linux distributions since the package must be compiled from source, and does not always properly compile or install. During the development process, PySide would often throw a Segmentation Fault error; this persisted until the code was recompiled and reinstalled which took a considerable amount of time especially under the lab's security constraints.

e. Comparing Workflows

Currently, to create an input file, which PFLOTRAN can understand, the input file is written by the user and can get over a thousand lines long. The input file must also adhere to the format guidelines provided by PFLOTRAN.

With PyFLOTRAN, the user can now work natively in Python to generate these input files while utilizing all the functionality of Python. On top of that, the GUI can be used to see a visual representation of creating the input files if the user does not want to work with Python code directly. This also allows the user to ignore the syntax of the input file and concern themselves more with the data they are inputting. In addition, error-checking can also be done with PyFLOTRAN-GUI.

III. FUTURE WORK

Since the graphical user interface can only run and write input files, the next iteration should be able to read, which means that the GUI should be able take a pre-existing input file and present views with the information in the files. Although PyFLOTRAN can already do this, it is a lengthy process for the GUI to do the same since there needs to be code written for each individual component of the GUI. There is a massive overhead in writing this code for each component, and the time allocated under SULI does not allow for this work to be done in time. Integrating visualization tools to illustrate the data, such as a domain box for the regions, is also a necessary step to further enhance the functionality and feel of the GUI.

Having used PySide to emulate Qt, it is apparent now that using a web interface instead would have been a superior choice since using a web framework would allow for a consistent layout across all operating systems. Not only is consistency a benefit from using a web framework, but so is creating attributes/models since most web frameworks allow models to be easily retrieved, modified, created, and updated; this would make it possible to present the user interface in a more refined manner without having to write much code to deal with the backend. Additionally, there are many open source web interfaces which would allow customization of the UI to be done more efficiently.

IV. ACKNOWLEDGMENTS

I would like to thank Satish Karra (Earth & Environmental Division (EES-16), Los Alamos National Laboratory (LANL)) for the opportunity to work at LANL, and for his mentorship during this internship, which has helped me learn the previous codebase quicker, and I was able to learn a lot more about Python by working on this project.

This work was supported in part by the U.S. Department of Energy, Office of Science, and Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships Program (SULI).

V. References

[1] P. C. Lichtner, G. E. Hammond, C. Lu, S. Karra, G. Bisht, B. Andre, R. T. Mills and J. Kumar, "PFLOTRAN User Manual," 2013. [Online]. Available: http://www.pflotran.org/docs/user_manual.pdf. [Accessed 12 August 2015].

[2] P. C. Lichtner, G. E. Hammond, C. Lu, S. Karra, G. Bisht, B. Andre, R. T. Mills and J. Kumar, "Example applications," 2013. [Online]. Available: http://www.pflotran.org/applications.html. [Accessed 12 August 2015].

[3] PySide, "PySide 1.2.1 documentation," 2013. [Online]. Available: http://pyside.github.io/docs/pyside/index.html. [Accessed 12 August 2015].

[4] The Qt Company, "Qt Documentation," 2015. [Online]. Available: http://doc.qt.io/. [Accessed 12 August 2015].

[5] IPython development team, "Documentation," August 2015. [Online]. Available: http://ipython.org/documentation.html. [Accessed 12 August 2015].

[6] Python Software Foundation, "unittest — Unit testing framework," 30 July 2015. [Online]. Available: https://docs.python.org/2/library/unittest.html. [Accessed 12 August 2015].

[7] RegisFrey, "The model, view, and controller (MVC) pattern relative to the user.," Wikimedia Common, May 2010. [Online]. Available: https://commons.wikimedia.org/wiki/File:MVC-Process.svg. [Accessed 12 August 2015].

[8] T. M. H. Reenskaug, "MVC XEROX PARC 1978-79," [Online]. Available: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html. [Accessed 12 August 2015].