Preprint
UCRL-JC-135877

Using Python to Develop Graphical Interfaces to Scientific Data

L.H. MacFarland and G.J. Streletz

This article was submitted to
8th International Python Conference
Alexandria, VA
January 24-27, 2000

September 24, 1999

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

Approved for public release; further dissemination unlimited

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

This report has been reproduced
directly from the best available copy.

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information
P.O. Box 62, Oak Ridge, TN 37831
Prices available from (423) 576-8401
http://apollo.osti.gov/bridge/

Available to the public from the
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.,
Springfield, VA 22161
http://www.ntis.gov/

OR

Lawrence Livermore National Laboratory
Technical Information Department's Digital Library
http://www.llnl.gov/tid/Library.html

Using Python to Develop Graphical Interfaces to Scientific Data

Lynn H. MacFarland and Gregory J. Streletz

Lawrence Livermore National Laboratory, University of California,
P.O. Box 808, Livermore, CA 94551 USA
macfarland2@llnl.gov, streletz1@llnl.gov

Abstract

At LLNL, Python has proven to be a convenient language for the development of graphical user interfaces (GUIs) which allow scientists to view, plot, and analyze scientific data. Two such applications are described in this paper. The first, EOSView, is a browser application for an equation of state data library at LLNL. EOSView is used by scientists throughout the laboratory who use simulation codes that access the data library, or who need equation of state data for other purposes. EOSView provides graphical visualization capabilities, as well as the capability to analyze the data in many different ways. The second application, Zimp, is a GUI that allows interactive use of the Stark Line Shape Database. It is used to access and plot data. The quick construction of Zimp from elements of the EOSView code provides a useful lesson in code reuse, and illustrates how the object-oriented nature of Python facilitates this goal. In general, Python has proven to be an appropriate choice of language for applications of this type for several reasons, including the easy access to GUI functionality provided by Tkinter, the ease with which C functions can be called from Python, and the convenient handling of strings in Python. Moreover, the features of the Python language, combined with the fact that it is interpreted rather than compiled, have allowed for extremely quick prototyping.

1. Introduction

Two different divisions wished to provide tools that allow physicists and designers access to scientific data without worrying about computers. Both divisions agreed graphic user interfaces (GUI) would be the answer. Motif was chosen as it had an easy interface to C and resided on all machines. The person investigating alternative languages did not know that Python could be installed on all machines and that TkInter provided a GUI interface. This project stalled and was later abandoned as prototyping was difficult and development was extremely slow.

A similar GUI, EOSView, was developed in Python with TkInter. Prototyping was quick and easy. The users liked becoming part of the process and seeing a quick turn-around.

A basic Python Widget class was created. Tailored modules were built using this basic class and TkInter. Each window became its own module. In future designs, there may be a basic window module from which the specialized windows will be built. Subsequently, the Python Widget class was reused for another GUI, Zimp, and is available for future work.

Each GUI extends Python with C routines. One Python/C interface is created for each window module. In this way, the GUI accesses the various C functions or libraries. This ensures that the same functions are called from Python as from other C or C++ programs. Verification and Validation (V & V) are major concerns to both groups.

2. Why Python

The chief concerns in selecting a GUI language were portability, stability and the ability to embed C. LEOS resides on the Cray J90, DEC Alpha, Meiko CS2 and IBM SP2 machines. Originally, Python only had been installed on the DEC machines. Documentation and examples were also important.

TCL/TK interface to C was considered moderate difficulty and there were only two books. Java and Python were reportedly only on the DEC machines. Java was considered too unstable. This assessment said that Python had no GUI and a moderately difficult interface to C. The assessment did not mention TkInter. Graphics libraries considered were Gist, NCAR, Diglib, Java, IDL and X11. Java and IDL were not available on all machines. Motif was chosen as it resided on all the machines, has an easy interface to C and there was plenty of documentation. However, Motif is difficult to learn. It requires lots of coding before a prototype can be seen. GUI tools, such as UIMX, were not available.

With a background in Motif/X11 and Tcl/Tk background, we chose Python with TkInter. Python is designed to work with small programs at the prompt similar to Tcl/Tk. Python programming is much more flexible than Tcl and far superior in accessing C routines. In addition, Python has classes similar to C++. GUIs are inheritantly object-oriented. Future GUI work will extend Python using C++. Thus, the class structure of Python was attractive plus there was a knowledge base of Python developers.

Developing in an interpreted language is much faster and easier than Motif. From the requirements document we could show different storyboard scenarios. To show the users we created a widget or set of widgets and let them pick the look and feel. This technique certainly helps the end-user "buy into" the tool.

Python has direct manipulation, better than Java. The stability of the Java environment was strongly questioned.

Python was chosen for its ease of prototyping with the Tk by using TkInter. Python is object-oriented, which is inherent in the GUI schemes. Python can extend C++/C or be embedded into these languages. Python is an excellent way to stick things together. Plus Python is just plain fun!

3. EOSView

3.1. Overview

EOSView [1] is a Python-language browser application for a large equation of state data library at Lawrence Livermore National Laboratory. The data library, called LEOS ("Livermore Equation of State") provides equation of state (EOS) data for use by several large, mission-critical simulation codes at the laboratory. Due to the importance of obtaining accurate results with these codes, it is essential that the scientists who use them be able to verify the accuracy and appropriateness of the EOS data being passed to them. EOSView has been developed to serve this purpose, and has been quickly prototyped in Python. EOSView allows code users to view and analyze LEOS equation of state data through use of a convenient graphical user interface (GUI), and is expected to be a valuable verification and validation tool.

3.2. Background

The LEOS equation of state data library [2] provides tabular EOS data for almost 150 materials. For most materials, data is available for 18 different functions (see Table 1). In general, each function depends on the independent variables of density and temperature. There are a few functions that depend on density only.

TABLE 1. The Functions Currently Available in the LEOS Data Library

Pt	Total pressure	Kr	Rosseland mean opacity
Et	Total energy	Kp	Planck mean opacity
Pc	Cold pressure	Zeff	Effective charge
Pi	Ionic pressure	Tm	Melting temperature
Pe	Electronic pressure	Cs	Sound speed
Ec	Cold energy	P2p	Two-phase pressure
Ei	Ionic energy	E2p	Two-phase energy
Ee	Electronic energy	S2p	Two-phase entropy
St	Total entropy	Ecp	Chemical potential

The LEOS data library is located in a single binary file. The file format is defined by the PDBLib file management routines [3], and has an internal hierarchical structure. In general, the first level of the hierarchy consists of the set of materials available in the library. Once a material is selected, the next level of the hierarchy consists of the set of functions available for that material. Upon selection of a particular function, the next level consists of the various data for the table corresponding to the selected material and function. The most significant data are the list of density values, the

list of temperature values, and the list of function values. In addition to this general scheme, other data (material information, for example) are available at the various levels of the hierarchy.

There is a textual browser available for examining the contents of PDB files. This utility, called PDBView [4], allows the user to traverse the levels of the hierarchical data library by using UNIX style commands. Fig. 1 shows the use of PDBView with the LEOS data library.

While PDBView is extremely useful for the direct viewing of the contents of the LEOS data library, there are several reasons why a GUI based browser application was needed to replace it. First of all, a GUI is much easier to use, because it does not require the user to learn the set of commands accepted at a command prompt. Secondly, a general scheme for graphical visualization is lacking. Also, PDBView does not allow the user to compare the table data with experimental data points, and does not provide the functionality to analyze the data in various ways. Finally, PDBView does not access the interpolation algorithms that are encapsulated in the LEOS access routines used by an application code. This is a critical shortcoming from a verification and validation perspective, because it is desirable to assess the quality of the data that are actually being used by a code, not just the data that reside in the library itself.

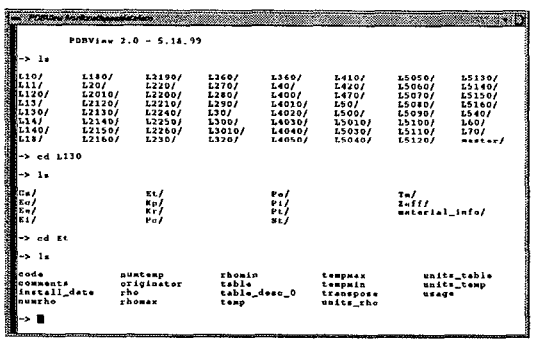


FIGURE 1. Using PDBView to navigate the hierarchical structure of the LEOS data library.

3.3. The EOSView GUI

The EOSView browser provides a graphical user interface that allows scientists to access the functionality of the browser quickly and easily. The GUI widgets are provided by the Tk widget set, and are accessed through Python by using the TkInter package [5]. Fig. 2 shows the EOSView main window.

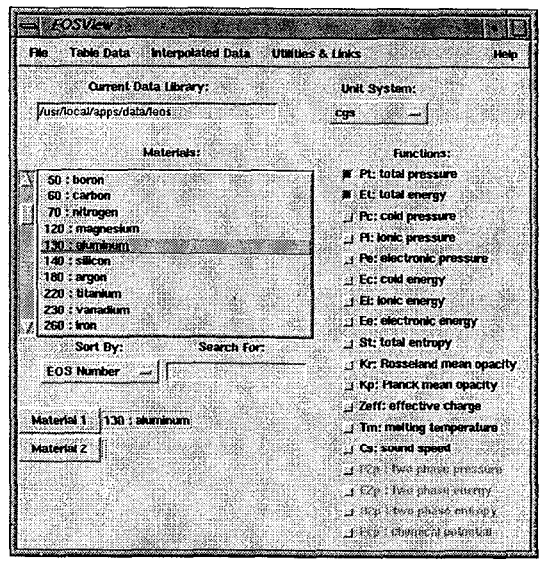


FIGURE 2. The EOSView main window. The Motif look and feel are provided by the Tk widget set, which is accessed by Python through the TkInter interface.

The structure of the EOSView GUI is based on a parent-child paradigm, as illustrated in Fig. 3. In general, after selecting the materials and functions of interest in the EOSView main window, a user can select one or more analysis options from the menu bar. Usually, this results in the creation of an appropriate analysis window. The analysis window accepts inputs for the various parameters necessary for the particular analysis to be performed. All analysis windows are children of the main window, in the sense that when the main window is closed, all the analysis windows disappear as well. From a given analysis window, a user can select the desired parameters for the analysis, and then initiate the computation at the touch of a button.

The resulting data are displayed in a data window. A data window can be either a text window or a plot window, as appropriate. Again, the parent-child relationship is preserved. If an analysis window is closed, either directly or because its parent (the main window) was closed, all of the data windows that have been created using that particular analysis window will be closed as well. However, any data windows created using other analysis windows will remain displayed until those windows are closed. Data windows also can be closed individually.

There is one more level to the parent-child hierarchy: a text window can be created from any given plot window in order to display the data of that plot in tabular form. If the plot

window is closed for any reason, the text window is closed as well. It should be noted that it is possible for data windows to be children of the main window. This occurs when there are no parameters to be selected, making an analysis window unnecessary. An example is the displaying of table data for a single material and function.

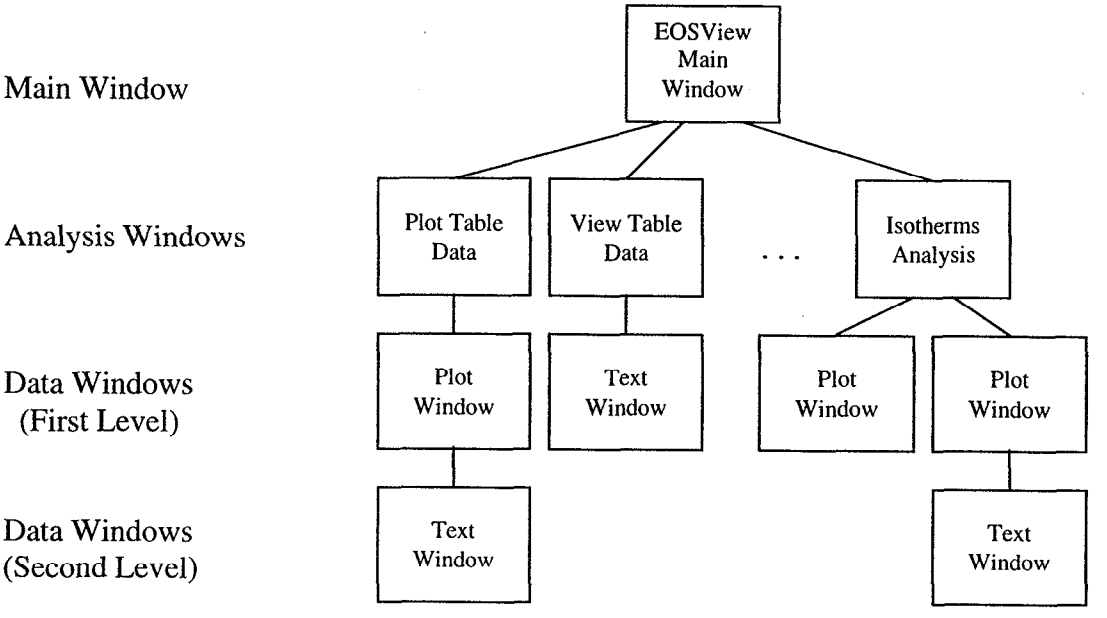


FIGURE 3. The parent-child structure of the EOSView GUI.

Python makes it easy to implement the parent-child model of a multiple-window GUI. First of all, Python's object-oriented nature can provide, almost automatically, the correct behavior upon the closing of a given window. By implementing the analysis windows and data windows as Python *classes* (as opposed to functions), each open toplevel window in EOSView is an instance of some class. Moreover, because the class instance for each toplevel window was instantiated from within the code of its parent (itself the instance of some class), the destruction of the parent results in the destruction of the child. The parent-child model described above is the result.

As mentioned previously, there are some cases in which the parent-child model of EOSView requires that a data window be the child of the main window, rather than of an analysis window. This occurs when the only parameters necessary for the execution of the desired task can be selected from the main EOSView window. For example, if only one material and one function have been selected at the main window and the user wants to view the table data, no other input is required, so the requested data should be displayed immediately, without the intervention of an analysis window.

In the Python implementation, this effect is achieved by some minor cheating. Data windows are always instantiated from analysis windows. However, if no additional information is required in order to obtain or calculate the desired data, the analysis window "hides" itself using code such as the following:

```
# If only one material and function,
# do not show the GUI window.
if (noGUI == TRUE):
    self.withdraw()
```

Moreover, after performing the necessary initialization code, the analysis window bypasses waiting for user input by immediately running the Apply() method, which obtains and displays the desired data:

```
# If only one material and function,
# get the data immediately.
if (noGUI == TRUE):
    self.Apply()
```

Therefore, as far as the Python code is concerned, the data window technically is the child of an analysis window (albeit an invisible one). However, we have given the *appearance* that the main EOSView window is the data window's parent.

For the purpose of closing windows in a manner consistent with the parent-child model, taking advantage of the object-oriented aspects of Python makes it unnecessary to keep track of all of the class instances that correspond to open toplevel windows. However, it may still be desirable to do so, for other reasons. For example, EOSView allows users to close all of the data windows that were created using a given analysis window, *without* closing the analysis window itself, and also to close all open analysis and data windows from the main EOSView window without exiting the application. Fortunately, Python makes this easy as well. Using mutable, dynamically allocated Python lists, keeping track of window IDs becomes almost trivial:

```
self.window = PlotWindow(self,
    title="Isotherm Plot Window",
    datafile=datafile,
    imagefile=self.tempDir+
              "/isotherms.ppm")
self.windowList.append(self.window)
```

In this case, pressing the "Close Plot Windows" button on the Isotherms analysis window would result in the execution of the following code, which would result in the closing of all plot windows that had been created using that particular Isotherms analysis window:

```
def CloseWindows(self):
    # Close Plot Windows
    for i in range(len
              (self.windowList)):
        self.windowList[i].destroy()
    self.windowList = []
```

In summary, several convenient features of the Python language, such as the availability of mutable lists and an OO framework, have made it fairly easy to deal with the need for a GUI that can spawn an arbitrary number of windows. The solution to managing multiple windows, a parent-child model, has been implemented easily in Python. Fig. 4 shows a typical multiple-window situation encountered in the use of EOSView.

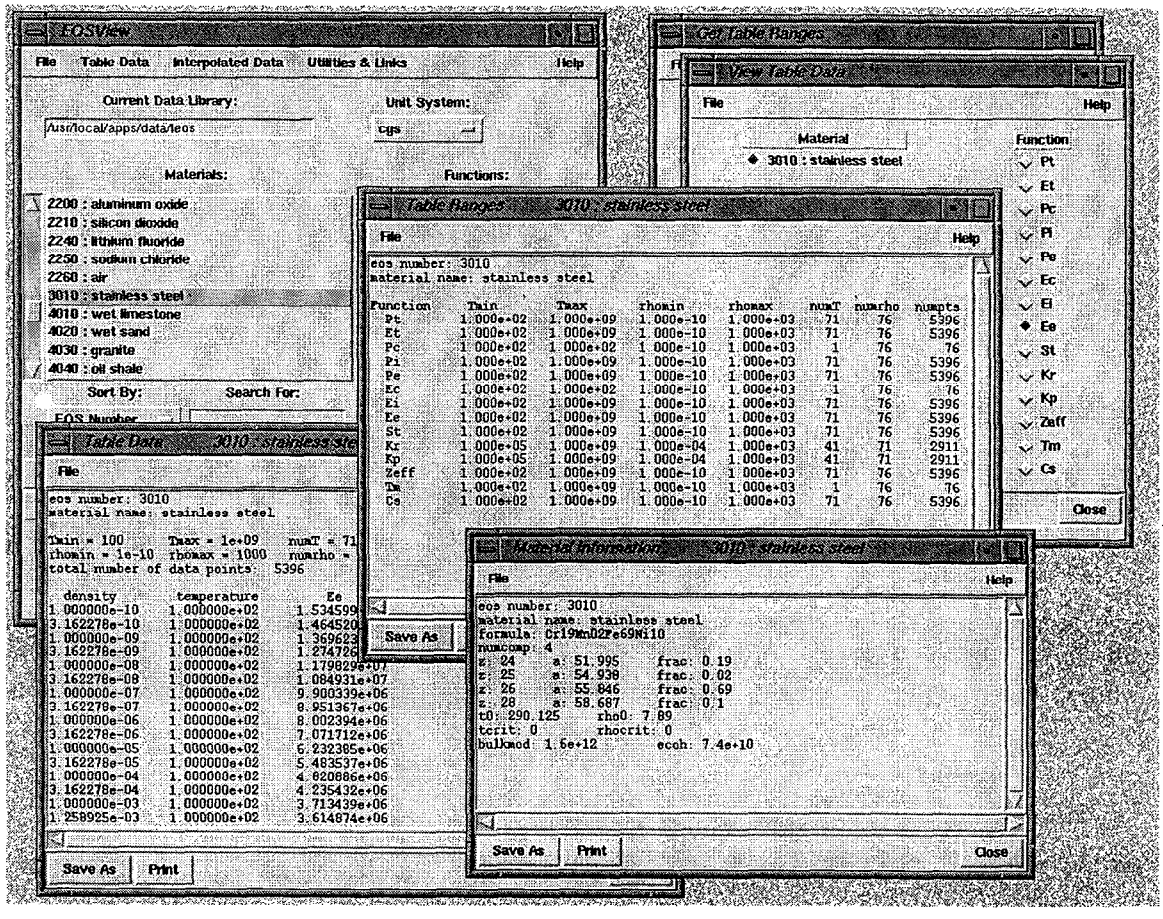


FIGURE 4. The fact that EOSView is a multiple-window GUI makes it convenient to establish parent-child relationships between GUI windows. In this screenshot, the "Get Table Ranges" and "View Table Data" windows are children of the main EOSView window. Both have a data window as a child. In addition, there is an open "Material Information" window. This data window is a child of the main window because getting material information for a single material was a task that did not require the input of additional parameters, and thus did not require an analysis window. Python makes it easy to enforce these parent-child relationships.

3.4. Calling C from Python: a V&V necessity

The entire GUI could have been written in Python. However, the Shock and High Pressure Physics group wished to test and validate the database and access function results before making public to the general user community. In order to run the same access library, the GUI must call the same functions. Extending Python with C makes this possible.

Before calling the C functions, Python ensures that the incoming data is valid. EOSView does not allow data that is out of range to be tested or interpolated. It verifies that the length of arrays is consistent and correct. Thus, EOSView is not a true black box test. It does not test the error handling of the C routines in the cases of bad data. EOSView is a great tool to quickly verify that the access routines give the expected data.

3.5. Handling Multiple Data Files with Python

Explicitly keeping track of open windows could be useful for managing the temporary data files EOSView writes to the file system. Currently, EOSView creates a temporary directory in which to write all of the data files computed during the given session.

When the application exits, this entire directory is removed. However, by keeping track of the window IDs of the open windows and using an appropriate scheme to name the temporary data files, all data files linked to a given analysis.

4. "Glue Language"

Python is far superior to many compiled languages, such as C/C++, in its ability to parse and manipulate strings. Python handles all file manipulation. The GUI allows wild cards or exact name matches.

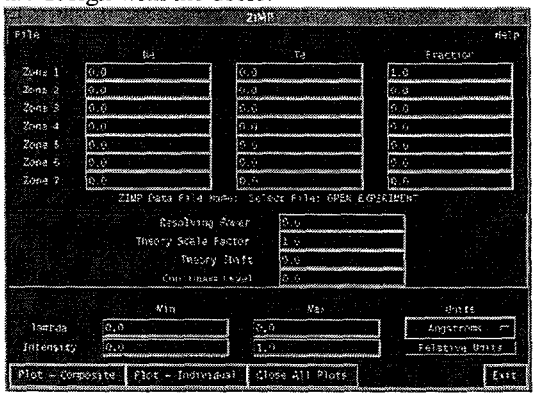
EOSView uses Python to alphabetize the material names or to order them numerically. In addition, the user may search for a material using either its name or equation-of-state identification number.

Python verifies the validity of user-entered data. If an error occurs, the user is notified immediately by a pop-up message box so the data may be corrected. Python can accumulate data and then send the arguments to the correct routines. It sets up the C queries ensuring that the C routines receive valid data.

5. Code Reuse

A basic widget class was created for EOSView. This class allows a standard set-up for composite widgets. There are standard windows to select or save a file. This base class organizes a composite of like in a given format (horizontal or vertical). Some composites include restrictions. For instance, this class ensures that if there is a File pull-down that it is the left-most pull-down menu and will be attached to the left. Likewise if there is a help menu, it will be the right-most pull-down menu and be attached to the right. The bottom bar with push buttons will have an Exit button in the right-most corner. The formats comply with the OSF/Motif Style Guide.

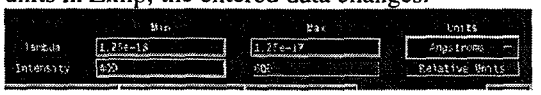
The basic widget class allows quick construction of examples in an interactive mode. This proved to be very useful when discussing the design with the users.



Zimp, a GUI that accesses the Stark Line Shape Database, was designed and developed in a few days. Zimp compares experimental data with the database. The focus is the electron density (Ne) by the electron temperature (Te). Like EOSView Zimp needs to open files to read data. Both allow users to enter data. Both have ranges which need to be checked. Both have units. Both use an option menu to make choices.

Zimp's base set of widgets is the same as EOSView. Code reuse makes development quicker, and now there is a more consistent interface among various tools.

Both Zimp and EOSView use the TVariable module written by Nils Fischbeck. This module extends Tkinter.Variable, Tkinter.StringVar, etc. by methods trace_read, trace_write to trace read and write variables. The option list depends on TVariable to indicate when the user changes an option. A callback is associated with the TVariable so the proper action occurs when the option label changes. Thus, when the user selects units in Zimp, the entered data changes.



The user changes units of lambda from Angstrom to eV. Not only does the range change accordingly, but also the label changes from lambda to Energy as both eV and keV are units of energy.

6. Extending Python

Following the Extending and Embedding the Python Interpreter guide was straightforward. This allowed the GUI to access existing C functions and libraries. This is similar to code reuse.

EOSView became a V & V tool by calling the access routines that extract the information from the database. The data still can be viewed in text format. In addition, the data can be plotted using Gnuplot. In this way known data can be verified and visualized.

Sometimes extending Python with C helps speed. EOSView called interpolation functions written in C. Performance improved. Also these functions could later be used by straight C code.

Both EOSView and Zimp use Gnuplot to statically plot the data points. Other plotting tools are being investigated. EOSView is considering BLT. Zimp will continue to use Gnuplot.

Future GUIs need to integrate a plotting package. These plots are in two or three dimensions. Both line and contour plots are

required. The ability to zoom in a particular section of the plot and re-plot it with the same number of interpolation points would be desirable. In addition, the ability to overlay two graphs is desirable. For instance, one plot contains the library data and the other contains the results of a simulation. Pgplot or one of its variants and Gist are being considered.

7. Conclusions

Python with TkInter is an excellent way to develop GUIs. Building a basic widget class for composites helped in code reuse and a consistent interface among tools.

EOSView and Zimp allow the user easy access to the C libraries which access the databases. However, in both cases the user does not need to know how to program in order to use the tool. EOSView is able to also function as a verification and validation tool as it uses the same functions and libraries as other C code.

EOSView does need a better plotting mechanism. Future development requires it. Recently, Python can access more plotting packages instead of executing a package as if it were a standalone program. Progress needs to continue in this area.

8. Acknowledgements

We thank David Young, Ellen Corey, and Jeff Nash for many useful discussions. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-ENG-48.

9. References

1. Streletz, G.J. and MacFarland, L.H., "A New Browser for the Visualization of Equation of State Data," in *Shock Compression of Condensed Matter - 1999*, edited by M. D. Furnish. Not yet published.
2. Corey, E. M. and Young, D. A., "A New Prototype Equation of State Data Library," in *Shock Compression of Condensed Matter - 1997*, edited by S. C. Schmidt, *et al.*, AIP Conference Proceedings 429, New York, 1998, pp. 43-46.
3. Brown, S. A., *et al.*, PDBLib User's Manual, LLNL Document M-270 Rev. 3, 1995.
4. Brown, S. A., *et al.*, PDBView User's Manual, LLNL Document UCRL-MA-108968 Rev. 1, 1994.
5. Lundh, Fredrik. An Introduction to TkInter. http://www.pythonware.com/library/tkinter/introduction/index.htm
6. Open Software Foundation. OSF/Motif Style Guide Revision 1.1, PTR Prentice Hall, 1991.
7. Fischbeck, Nils. Communique to the comp.lang.python.newsgroup.
8. van Rossum, Guido. Extending and Embedding the Python Interpreter. http://www.python.org/doc/current/ext