SANDIA REPORT

SAND2010-xxxx
Unlimited Release
Printed March 2006

Supersedes SAND1901-0001
Dated January 1901

Optika: A GUI Framework for Parametrized Applications (Report style, strict)

Kurtis L. Nusbaum

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2010-xxxx
Unlimited Release
Printed July 2010
Reprinted March 2006

Supersedes SAND1901-0001
dated January 1901

Optika: A GUI Framework for Parametrized Applications (Report style, strict)

Kurtis L. Nusbaum
Scalable Algorithms
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-9999
klnusba@sandia.gov

Abstract

In the field of scientific computing there are many specialized programs designed for specific applications in areas like biology, chemistry, and physics. These applications are often very powerful and extraodinarily useful in their respective domains. However, many suffer from a common problem: a poor user interface. Many of these programs are homegrown, and the concern of the designer was not ease of use but rather functionality. The purpose of Optika is to address this problem and provide a simple, viable solution. Using only a list of parameters passed to it, Optika can dynamically generate a GUI. This allows the user to specify parameters values in fashion that is much more intuitive than the traditional ”input decks” used by many parameterized scientific applications. By leverageing the power of Optika, these scientific applications will become more accessible and thus allow their designers to reach a much wider audience while requiring minimal extra development effort.

Acknowledgment

Thanks to Dr. Mike Heroux and Jim Willenbring. Their mentoring has been crucial to the development of Optika. Also, many thanks to the entire Trilinos Community in which Optika has found a welcoming home.

The format of this report is based on information found in [**?**].

Contents

List of Figures

List of Tables

Nomenclature

Dependency A relationship ship between to parameters in which the state or value of one parameter depends on the state or value of another.

Dependee The parameter upon which another parameter state or value dependes.

Dependent A parameter whose state or value is determined by another parameter.

Parameter An input needed for a program.

Parameter List A list of parameters and other parameter lists.

RCP Refernce counted pointer. RCPs refered to in this document reference the RCP class located in the Teuchos package of Trilinos.

Sublist A parameter list contained within another parameter list.

Widget A GUI element, usually used to obtain user input.

Chapter 1

Introduction

This report is comprised of two main sections. The first section discusses the development of Optika. Design choices and problems that arrised during development are discussed in this section. The second section is intended to be a manual on how to use Optika. If the reader desires simple to learn how to use Optika, it is suggested he/she skip to the second section.

This page intentionally left blank.

(Well, it's not blank anymore...)

Chapter 2

Development of the Optika package

This section of the report discusses the developement of the Optika package.

Initial Planning

In Fall of 2008. Dr. Mike Heroux identified a need for the Trilinos framework to include some sort of GUI package. Dr. Heroux wanted to give users of the framework the ability to easily generate GUIs for their programs, while still providing a good experience for the end user. Based on previous GUI work done for the Tramonto project, a few initial problems were identified:

- How would the GUI be laid out?
- Different types of parameters require different methods of input. How would we decide how we would obtain input for a particular parameter?
- What framework would we use to build the GUI?
- How would the application developer specify parameters for the GUI to obtain?
- How would the application developer specify dependencies between parameters. This was a crucial problem/needed feature that was identified in development of the Tramonto GUI.

After some deliberation, the following initial solutions were decided upon:

- The GUI would be laid out in a hierarchical fashion as shown in Figure **??**. Parameters organized into lists and sublists. This would allow for a clear organization of the parameters as well as intrinsically demonstrate the relationships between them.
- It be required that all parameters specify their type and the following types would be accepted:
  - int
  - short

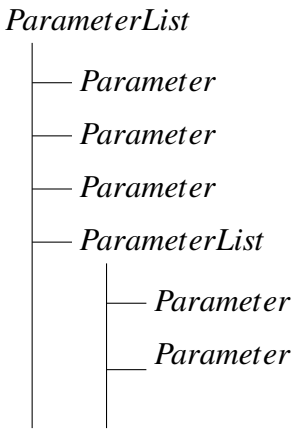
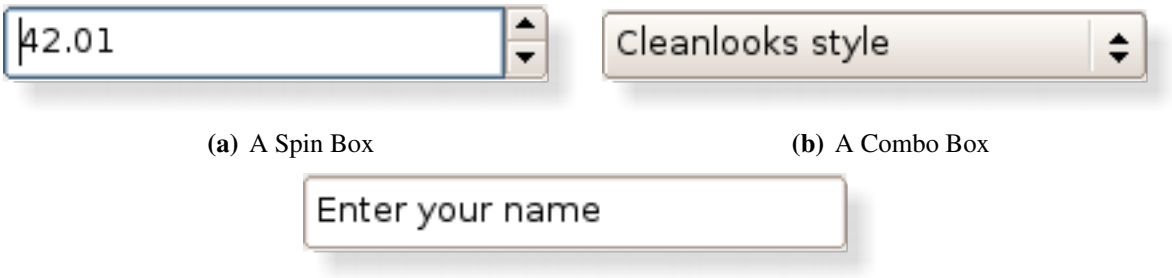


Figure 2.1. The hierarchical layout of the GUI

- float
- double
- string
- boolean
- arrays of int, short, double, and string

For number types, a spin box would be used as input. If the valid values for a string type were specified, a combo box would be used. Otherwise a line edit would be used. For booleans, a combo box would also be used. For arrays, a pop-up box containing numerous input widgets would be used. The widget type would be determined by the array type.



(a) A Spin Box

(b) A Combo Box

(c) A Line Edit

Figure 2.2. Some of the various widgets used for editing data

- QT was chosen as the GUI framework for several reasons:
  - It is cross-platform.
  - It is mature and has a well developed set of development tools.

  - It has a rich feature-set.
  - It has been used by Sandia in the past.
  - The Optika developer was familiar with it.

- Initially it was decided that the application developer would specify parameters via an XML file. A DTD would be created specifying the legal tags and name spaces.
- Dependencies would be handled through special tags in the DTD.

Early Development

The first several months of development were spent on creating and implementing the XML specification. The name of the XML specification went through several revisions but was eventually called Dependent Parameter Markup Language (DPML).

After several months of development it was realized that creating an entirely new way of specifying parameters might hinder it's adoption. It pointed out that Trilinos actually had a ParameterList [**?**] class in the Teuchos package. The ParameterList semmed to be better than DPML for several reasons:

- It was already heavily adopted.
- It had the necessary hierarchical nature.
- It was serializable to and from XML.

For these reasons, DPML was scrapted in favor of using Teucho's ParameterLists. Development moved forward with the goal of creating a GUI framework that, in addiiton to meeting all the challenges outlined above, would also be compatiable with any existing program using Teuchos's ParameterLists.

Heavy development

Starting in May 2009 a more heavy focus was put on developement of the Trilinos GUI package. With the backend data-structure of the Teuchos's ParameterList already in place, attention was turned to the developing the actually GUI itself. A key technology provided by Qt was it's Model/View framework [**?**]. Using the Model/View paradigm, a wrapper class named TreeModel [**?**] was created around the ParameterList class by subclassing the QAbstractItemModel [**?**].

However, in subclassing the QAbstractItemModel it was realized that the ParameterList class fell short in certain areas. At this point the main issue was that a given ParameterEntry [**?**] located

Figure 2.3.

graphics/treeview

within a ParameterList or a given sublist located within a ParameterList was not aware of it's parent. This was an issue because Qt's Model/Veiw framework requires items within a model to be aware of their parents. In order to circumvent this issue the TreeItem [**?**] class was created. Now instead of simply wrapping around a ParameterList class, the TreeModel created by giving it a ParameterList. It would then read in the ParameterList and creat a structure of TreeItems. Each TreeItem then contained a pointer to it's corresponding ParameterEntry.

Once the TreeModel and TreeItem class were complete an appropriate delegate to go between and View and the TreeModel was needed. A new class called Delegate [**?**] was created to fill this role by subclassing QItemDelegate [**?**]. As specified above, the delegate would return the apporiate editing widget based on it's datatype.

With the model and delegate classes inplace, an appropriate view could be applied. At first a simple QTreeView [**?**] was applied to the model. The results was something like that in **??**.

DISTRIBUTION:

1 An Address
99 99^{th} street NW
City, State

3 Some Address
and street
City, State

12 Another Address
On a street
City, State
U.S.A.

1	MS 1319	Rolf Riesen, 1423
1	MS 1110	Another One, 01400
1	M9999	Someone, 01234
1	MS 0899	Technical Library, 9536 (electronic)

Second Printing, (January 2006):

1 An Address
99 99^{th} street NW
City, State
3 Some Address
and street
City, State
12 Another Address
On a street
City, State
U.S.A.

1 MS 1319 Rolf Riesen, 1423
1 M9999 Someone, 01234

Third Printing, (February 2006):

1 MS 1319 Rolf Riesen, 01423

Fourth Printing, (March 2006):

1 MS 1319 Rolf Riesen, 1423

Sandia National Laboratories