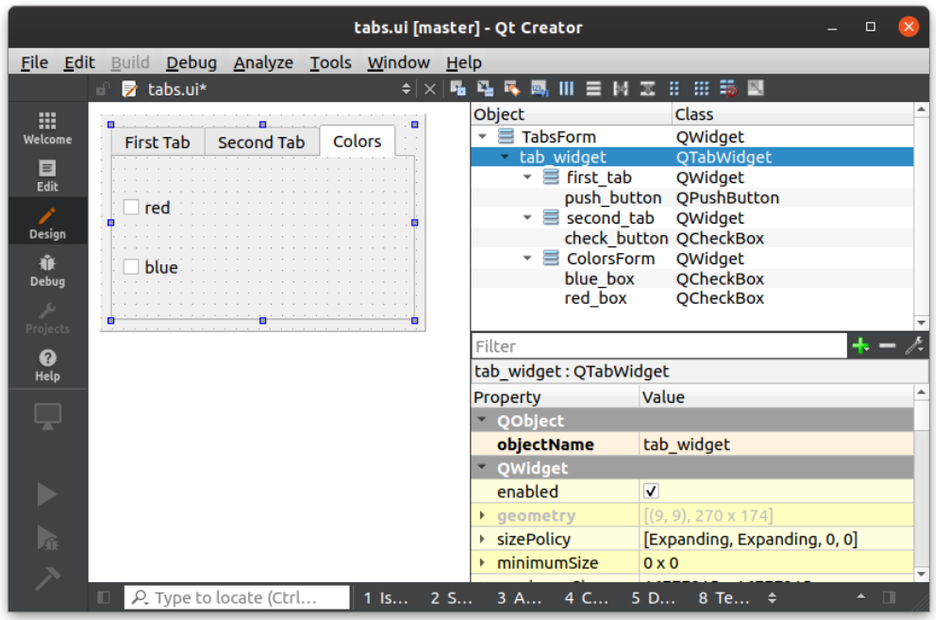
ABSTRACT

Visual tools for UI design are often not good at complex batch modifications of complex interfaces.

Here we show how a Julia Domain Specific Language (DSL) can be used as an aid in UI design.

This is bidirectional. Allowing us to make modifications of a GUI in both a Visual designer and the Julia UI DSL

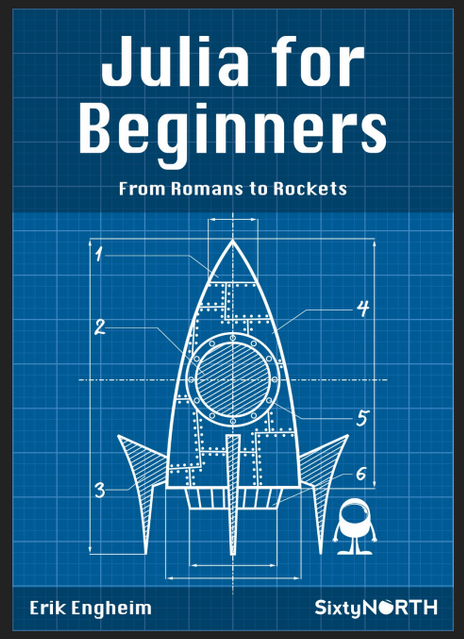


Qt Creator GUI Designer

ERIK ENGHEIM

SixtyNORTH

Principal Consultant



Author of Julia for Beginners

USING JULIA FOR UI DESIGN

NATIVE XML FORMAT

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>ButtonForm</class>
  <widget class="QWidget" name="ButtonForm">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>180</width>
        <height>150</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>Choose Colors</string>
    </property>
    <layout class="QVBoxLayout" name="top_vlayout">
      <item>
        <widget class="QCheckBox" name="red_checkbox">
          <property name="text">
            <string>red</string>
          </property>
        </widget>
      </item>
      <item>
        <widget class="QCheckBox" name="green_checkbox">
          <property name="text">
            <string>green</string>
          </property>
        </widget>
      </item>
      <item>
        <spacer name="vertical_spacer">
          <property name="orientation">
            <enum>Qt::Vertical</enum>
          </property>
          <property name="sizeHint" stdset="0">
            <size>
              <width>20</width>
              <height>40</height>
            </size>
          </property>
        </spacer>
      </item>
      <item>
        <layout class="QHBoxLayout" name="bottom_hlayout">
          <item>
            <widget class="QPushButton" name="ok_button">
              <property name="text">
                <string>Ok</string>
              </property>
            </widget>
          </item>
          <item>
            <widget class="QPushButton" name="cancel_button">
              <property name="text">
                <string>Cancel</string>
              </property>
            </widget>
          </item>
        </layout>
      </item>
    </layout>
  </widget>
  <resources></resources>
  <connections></connections>
</ui>
```

JULIA DSL

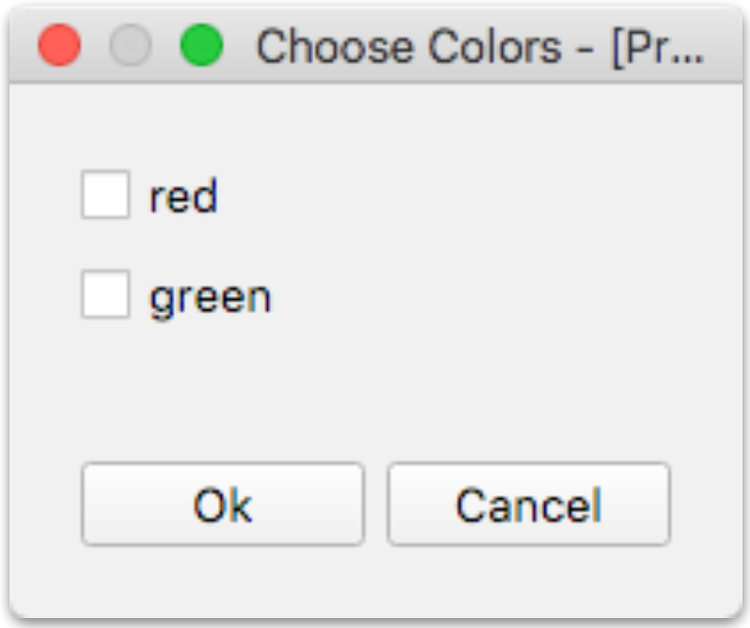
```julia
Ui(
    class   = "ButtonForm",
    version = "4.0",
    root = QWidget(
        name        = "ButtonForm",
        class       = :QWidget,
        geometry    = Rect(0, 0, 180, 150),
        windowTitle = "Choose Colors",
        layout = VBoxLayout(
            name = "top_vlayout",
            items = [
                QCheckBox("red_checkbox", "red"),
                QCheckBox("green_checkbox", "green"),
                Spacer("vertical_spacer", VERTICAL, Size(20, 40)),
                HBoxLayout(
                    name = "bottom_hlayout",
                    items = [
                        QPushButton("ok_button", "Ok"),
                        QPushButton("cancel_button", "Cancel")
                    ]
                )
            ]
        )
    )
)
```

NOTE WE CAN CONVERT TO AND FROM BOTH FORMATS

THE PROBLEM

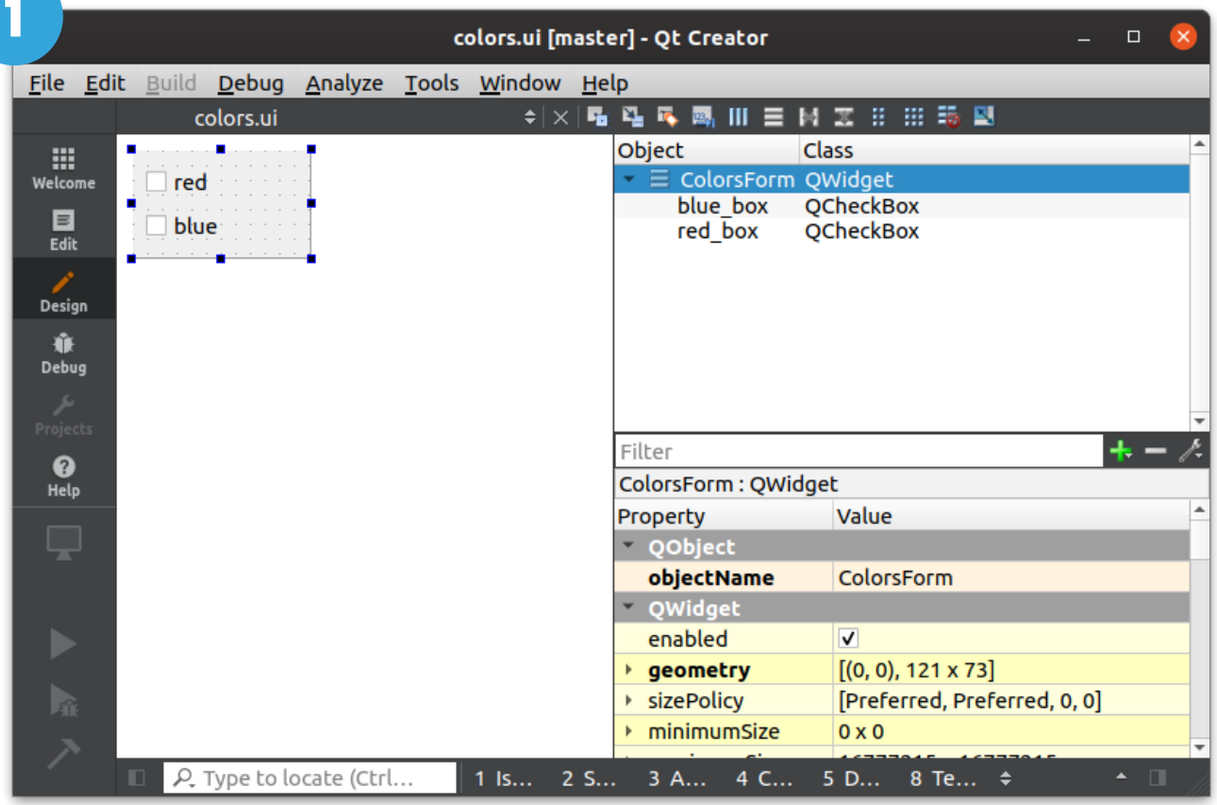
Sometimes the visual tools are **inadequate** and we want to edit UI definition file directly.

However this format is not always **human readable** or friendly towards manual edits.



Look on the right at the profound difference in complexity of representing this GUI in its **native XML** format and using a **Julia DSL**

1



Initial GUI Design we want to change

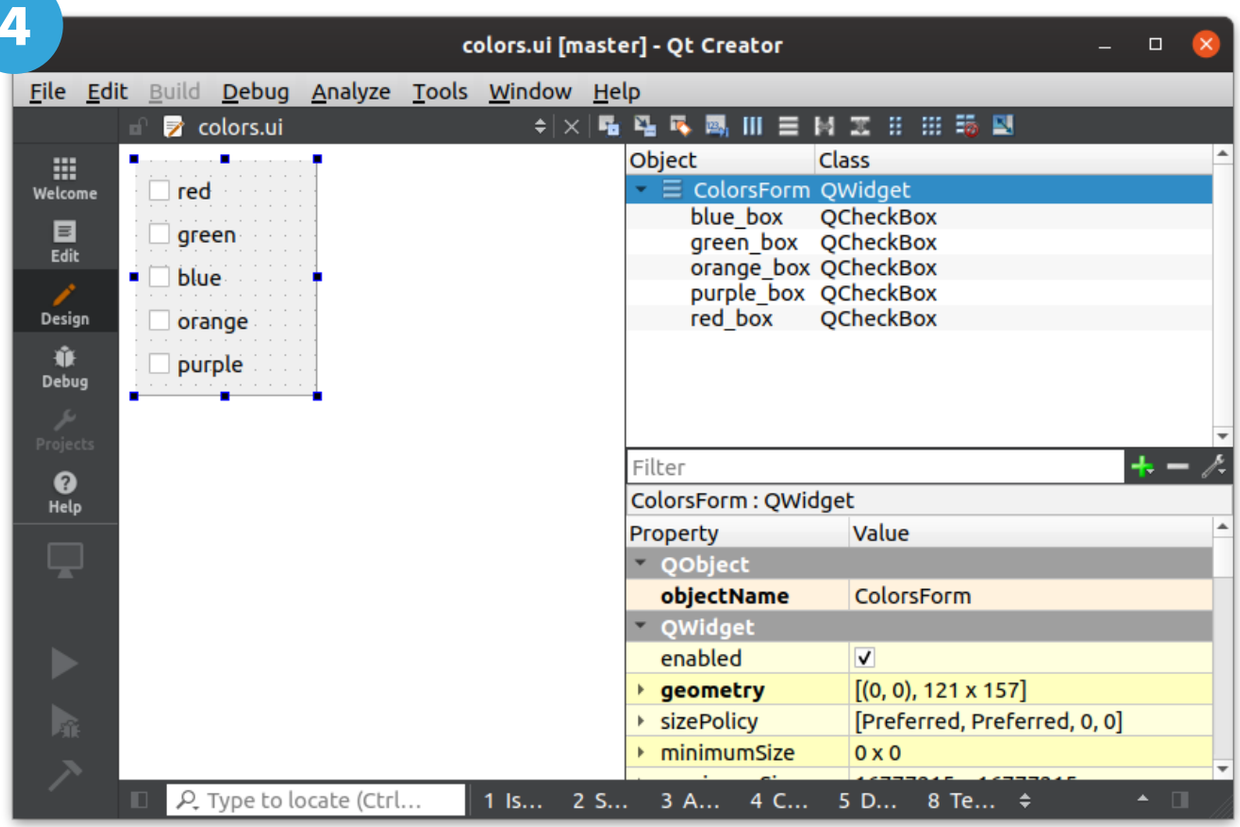
2

```
julia> ui = load("colors.ui")
Ui(
    class   = "ColorsForm",
    version = "4.0",
    root = QWidget(
        name        = "ColorsForm",
        class       = :QWidget,
        geometry    = Rect(0, 0, 121, 73),
        windowTitle = "Form",
        layout = VBoxLayout(
            name = "top_layout",
            items = [
                QCheckBox("red_box", "red"),
                QCheckBox("blue_box", "blue")
            ]
        )
    )
)

julia> ui.root.layout
VBoxLayout(
    name = "top_layout",
    items = [
        QCheckBox("red_box", "red"),
        QCheckBox("blue_box", "blue")
    ]
)

julia> colors = ["red", "green", "blue", "orange", "purple"]

julia> ui.root.layout.items =
   [QCheckBox("$(color)_box", color) for color in colors]
5-element Array{QWidget,1}:
 QCheckBox("red_box", "red")
 QCheckBox("green_box", "green")
 QCheckBox("blue_box", "blue")
 QCheckBox("orange_box", "orange")
 QCheckBox("purple_box", "purple")
```

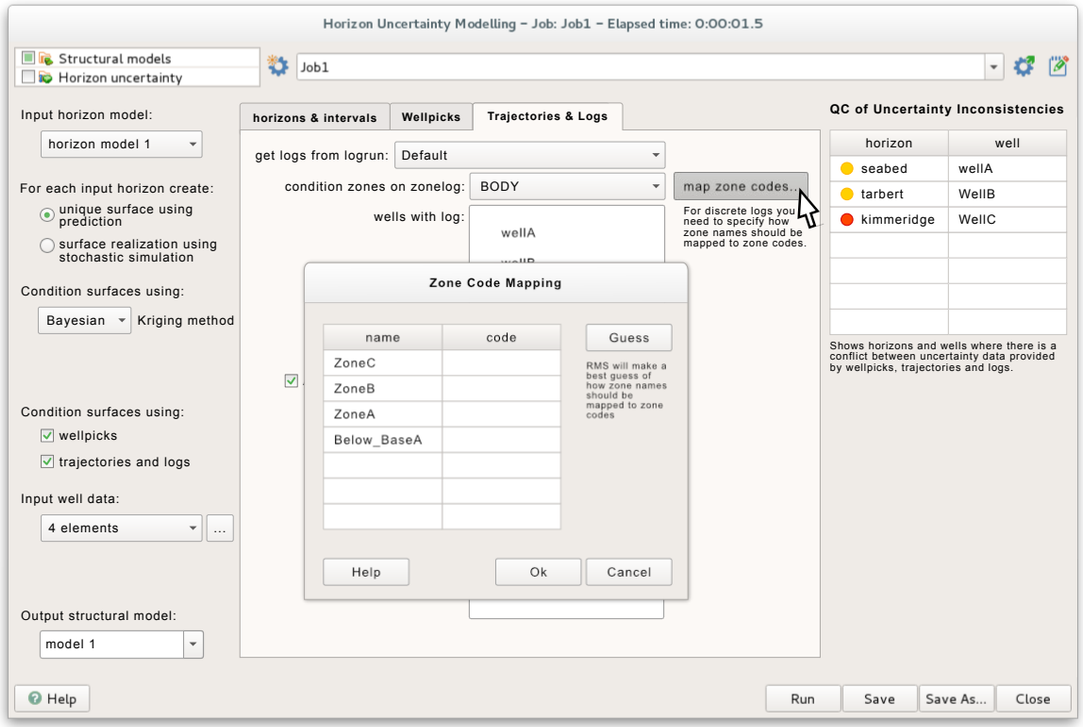
3

```
julia> save_ui()
```

4



GUI after modification in the Julia REPL and calling save_ui()



Graphical user interfaces can become very complex.

With the Julia UI DSL we can easily composite multiple separate UI files. Here the UI for each tab is defined in a separate file.

To learn more, try out the **QtUIParser** Package

https://github.com/ordovician/QtUIParser.jl

CUSTOM WIDGETS

Adding Custom and Composite Widgets

```
Ui(
    class   = "HUMForm",
    version = "4.0",
    root = QWidget(
        name     = "humpanel",
        class    = :QWidget,
        layout = HBoxLayout(
            name = "top_column_layout_",
            items = [
                load_root_widget("data-selection-column.jl"),
                QWidget(
                    name         = "main_tab_widget_",
                    class        = :QTabWidget,
                    currentIndex = 1,
                    children = [
                        Tab("Horizons and Intervals", "horizonstab.jl"),
                        Tab("Wellpicks", "wellpickstab.jl"),
                        Tab("Trajectories and logs", "trajlogstab.jl")
                    ]
                ),
                load_root_widget("info-column.jl")
            ]
        )
    ),
    customwidgets = [
        CustomWidget(:WObjectsSelector, :QWidget, "panel/ui/wobjectsselector.h"),
        CustomWidget(:QtCompComboBox, :QComboBox, "uiqt/qtcompcombobox.h")
    ]
)
```

Inserts UIs defined in external files