THE QCONTROL TOOLKIT

An Object-Oriented Alternative to XControls

Use QControls to receive the benefits of XControls without the headaches. Allow easy UI Logic Code reuse. Encapsulate and decouple the UI Logic away from the Business Logic of the main application and from the UI Skin.

Quentin "Q" Alldredge, CLA
support@qsoftwareinnovations.com
Q Software Innovations, LLC (QSI)
NI Alliance Partner

©2016 Q Software Innovations, LLC

Table of Contents

1 Quick Start

After installing the QControl Toolkit, create a new QControl by launching the QControl Creation Wizard. Launching the wizard can be done through the "New…" Dialog and selecting "QControl".
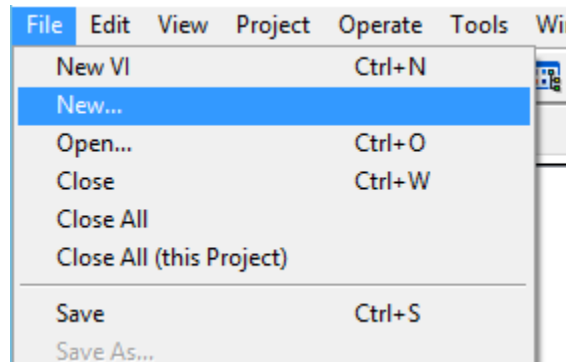
Figure 1 - Accessing the New Dialog from the File Menu
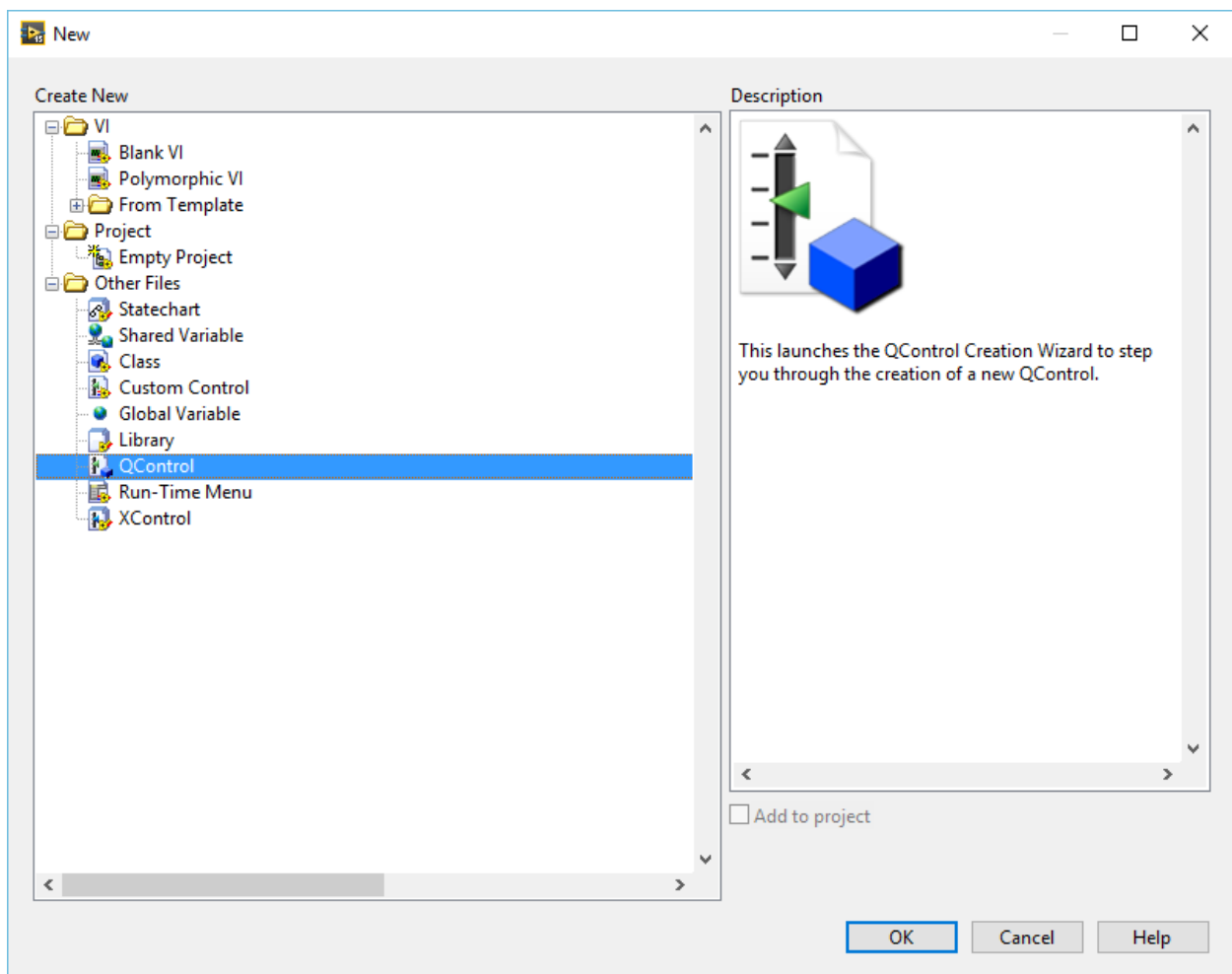
Figure 2 - QControl is listed under the "Other Files" folder in the New Dialog

After the wizard opens, follow the steps outlined in Section 2 of this document.

2   Tutorial

This tutorial will step you through the creation of a basic QControl. This particular QControl will be a string form field that you as the developer want to give the user of your software some indication that the information they entered is correct.
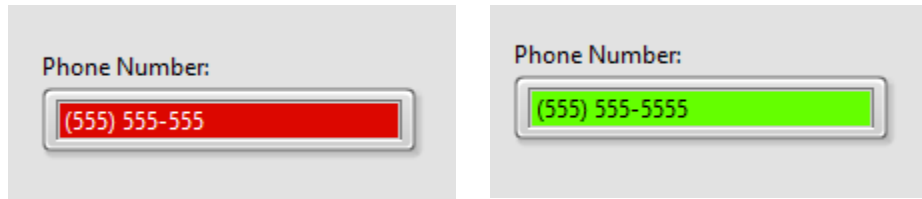
As Section 1 stated: start the QControl Creation Wizard from a project explorer window.

2.1   Step-by-Step Creation of a new QControl

1.   **START-UP SCREEN**

    The first step to creating a new QControl is to launch the QControl Creation Wizard. When it has started the start-up screen will be showing.
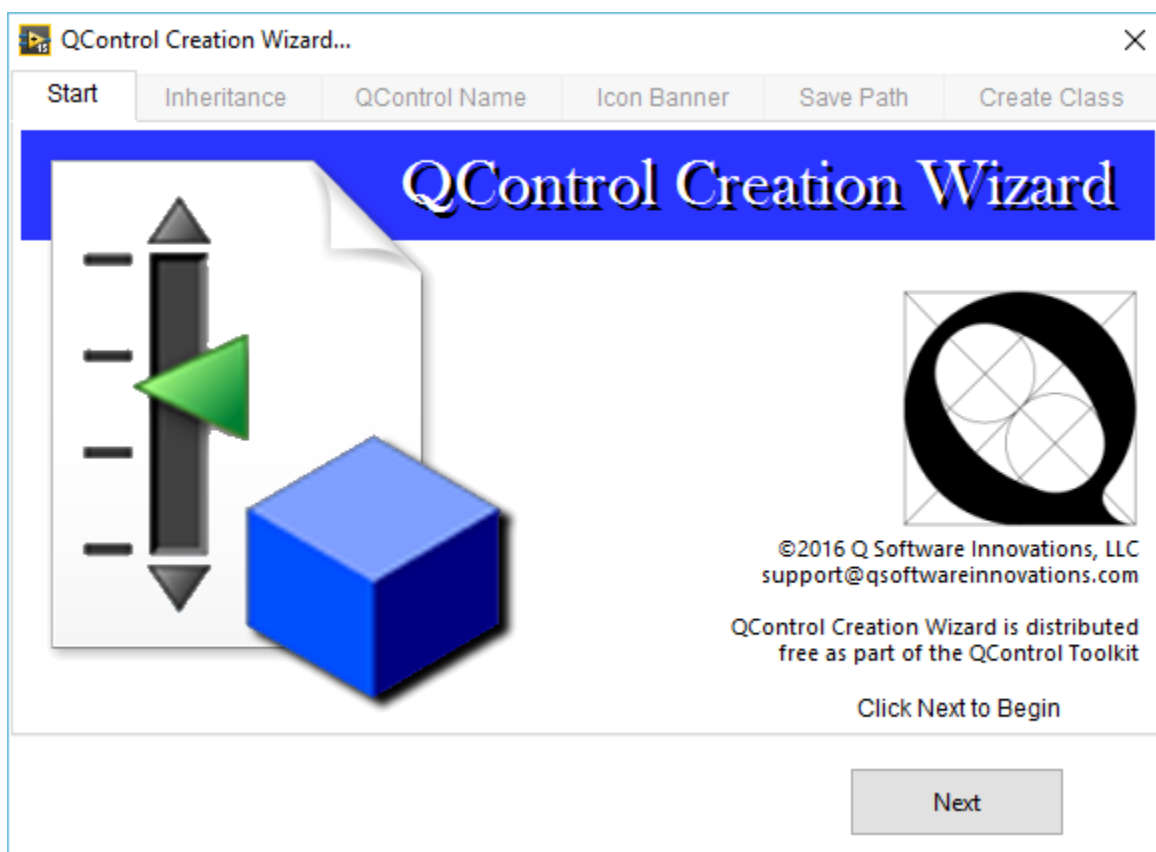
Figure 3 - Start-up Screen

2.   **SELECT CLASS INHERITANCE**

The next step is to select the class to inherit from. You must decide what type of control you want to extend the functionality of by ask yourself the following questions:

- *What do I want the new control to be based off of?*
- *What will the new control do?*
- *Do I want it to be based off of one control or have multiple controls in a cluster?*

Select the class that best give you the correct input reference to the Constructor Method that will be created and give you the inherited properties you need. For single controls selected the class for the correct reference. For multiple controls select the Cluster class.

If the class you desire to inherit from is not in the tree because it is one of your own Extended QControl Classes created earlier and not saved with the rest of the QControl Class Hierarchy, Click the radio button next to the path control. The path control will become enabled which will allow you to navigate to your QControl Class.

In the case of this tutorial select the **String** from the list.

Clicking **Back** at any time until the class is created will take you back to the previous step.
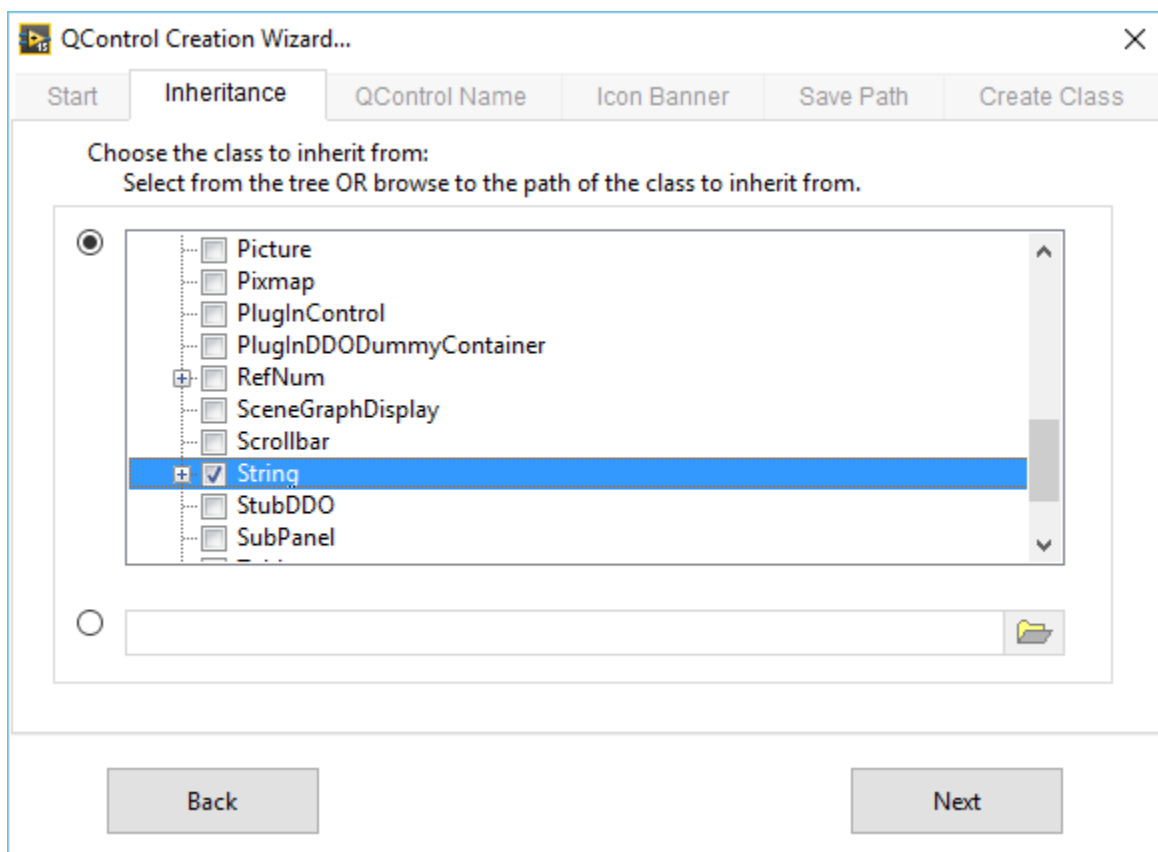

Figure 4 - Choose Class Inheritance

Clicking **Next**, will cause the next step to be available if no warnings have occurred. Possible warnings that could occur include:

- **Class Check**

  This check ensures that the selected path is a LabVIEW Class. If it is not the following message appears:

  The filename entered is not a LabVIEW Class. The new QControl must inherit from a class that is part of the QControl Class Hierarchy.

- **Inheritance Check**

  This check ensures the class is part of the QControl Class Hierarchy. If it is not the following message appears (where %s = the class name that was selected):

  The class selected, "%s", is not part of the QControl Class Hierarchy. The new QControl must inherit from a class that is part of the QControl Class Hierarchy.

3. ENTER CLASS NAME AND DESCRIPTION

Next, enter the name of the new QControl. Names must be unique from any other QControl Class so that it does not save over another QControl. The *Class Name* and *Localized Name* can be the same and are the same by default. Also the *Class Name* and *Localized Name* are required inputs. The **Next** button will be disabled until a unique, non-blank class name is entered. The *Description* is optional but recommended.

Let's name the new QControl "**PhoneFormField**".

Figure 5 - Class Name and Information

Clicking **Next**, will cause the next step to be available if no warnings have occurred. Possible warnings that could occur include:

- **Unique Name Check**
  This check ensures that the QControl Name is not the same as any other class that is distributed with or saved in the same folder as the rest of the classes in the QControl Class Hierarchy. It will not find other Extended QControl Classes saved in other places. If it is not unique the following message appears (where %s = the QControl Name):

  The QControl name, "%s", is the same as one distributed in the QControl Toolkit. QControl names must be unique.

- **In-Memory Check**
  This check ensures that a class with the same name is not in memory of the open application. If it is in memory the following message appears (where %s = the class name that was selected):

  A QControl, or other LabVIEW class with the same name, ("%s") is already in memory in this application instance.

- **In-Project Check**
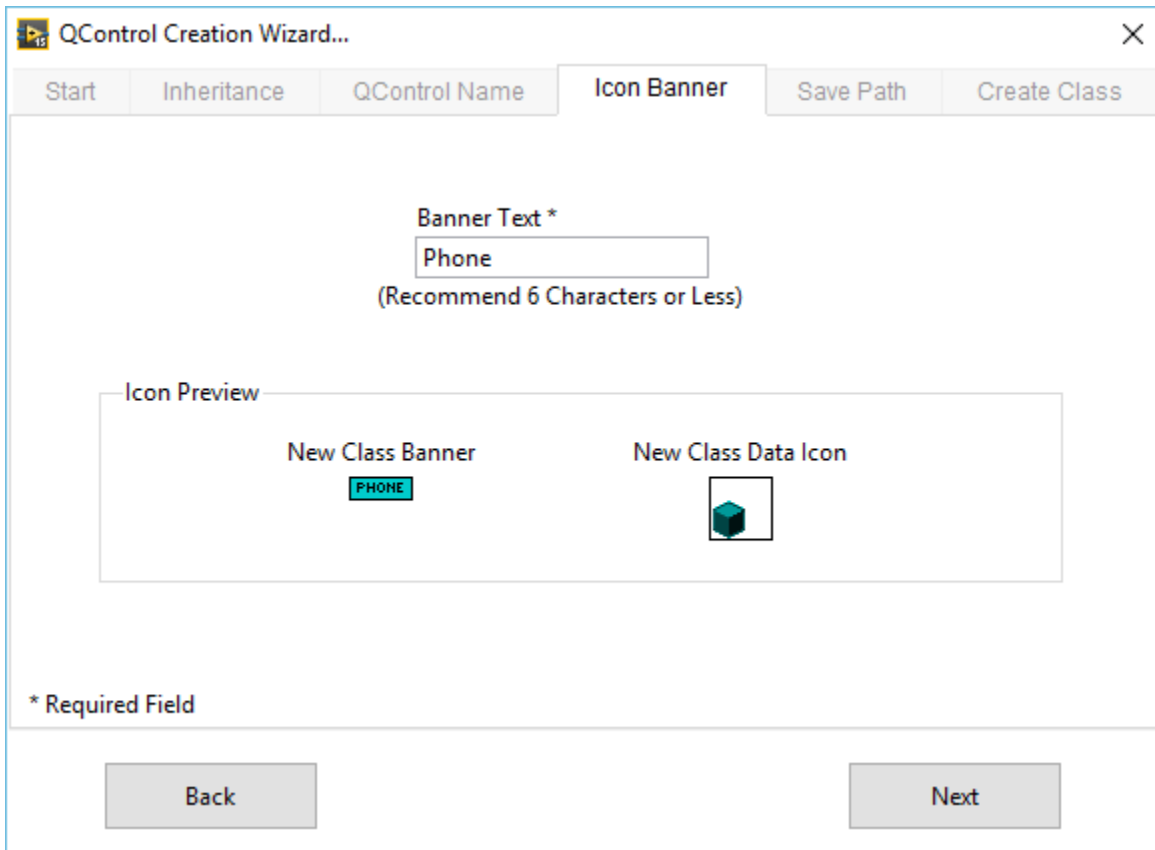
  This check ensures that the QControl by the same name is not already a member of the Project in which it will be added to upon creation. If there is a class already by the same name the following message appears (where the first %s = the QControl Name, and the second %s = the name of the Project):

  A QControl, or another LabVIEW class with the same name, ("%s") is already a member of the Project where the QControl was to be added ("%s").

4. **ENTER BANNER TEXT**

   On the next step the *Banner Text* will be filled in based on the *Class Name* entered in the previous step. View the *Icon Preview*; if the text is acceptable click **Next**. Otherwise, edit the Banner Text before clicking **Next**. If the *Banner Text* is erased, or is otherwise blank, the **Next** button will grey-out.

   In this case let's change the Banner Text to **"Phone"**.

Figure 6 - Enter Banner Text

5. **SELECT SAVE LOCATION**

On this step, the *Save Location* will be pre-filled to create a new path for the QControl. If launched from a Project the default path is based on the path to the project. If launched from the LabVIEW "Getting Started Window", the default path will be located in the user's documents folder, in the LabVIEW Data Sub-folder.

A different path can be chosen but keep-in-mind that a new folder is automatically appended to your chosen path. The *Final Path* is shown to help clarify where the QControl will end up after it is created.

The path for you could be different in this step depending on where your project is saved or if your project is saved. Accept the default, or pick a different path.
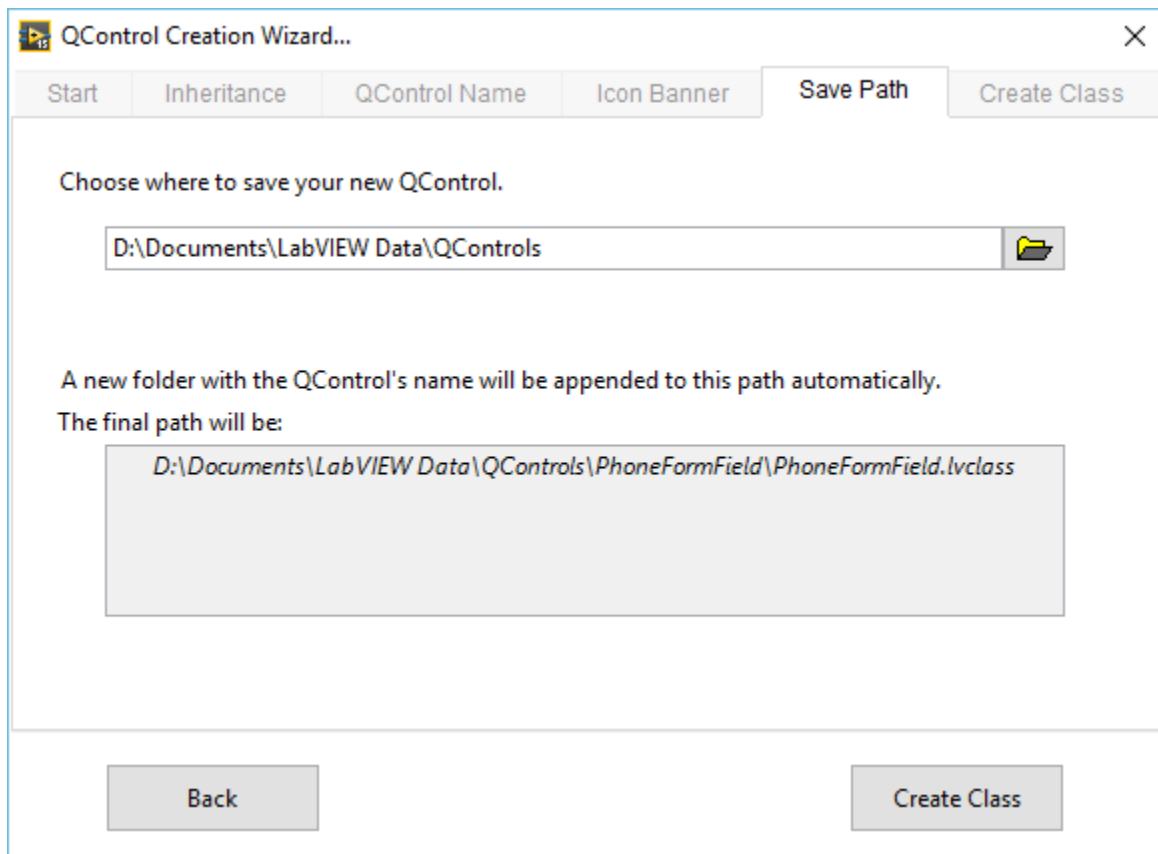
Figure 7 - Choosing the Save Location

Clicking on **Create Class** will start the actual creation of the new QControl Class if no warnings have occurred. Possible warnings that could occur include:

- **Class Already Exists Check**
  This check ensures that the QControl Class, or another class by the same name, does not already exist at the specified path. If a class does already exist the following message appears (where %s = the path to the existing class):

  A QControl, or other LabVIEW class with the same name, already exists at the specified path: (%s).

- **QControl Members Exist Check**
  This check ensures that method that are members of a default QControl Class does not already exist at the specified path. If any of the members do already exist the following message appears (where %s = the path to the folder where the members are found):

  One or more members of a QControl already exists at the specified path: (%s).

6. CREATE CLASS

The QControl Class is created by scripts through multiple steps. The status of each step is displayed ascending and fading out. The final status of *"Click "Finish" to begin editing the new QControl."* is displayed at the bottom when the creation of the new QControl Class is completed without any critical errors. If any errors do occur they will be displayed in an error dialog.

Errors in the range 5002-5021 are considered critical error which means they will cause the QControl Creation to be cancelled and the **Explore to QControl** button will not be visible. Then, the final status of *"Error Occurred. QControl not Created."* is displayed at the bottom.

Any other error will still be displayed in the error dialog but the QControl Creation still finish and the **Explore to QControl** button will be visible.

If the wizard was launched from a Project the new QControl should be added to the project. If launched from the LabVIEW "Getting Started Window" the new QControl will open in its own window.

Clicking on the **Explore to QControl** button will open the Windows Explorer to where the new QControl Class was saved (see Step 7). Clicking **Finish** will dismiss the wizard.
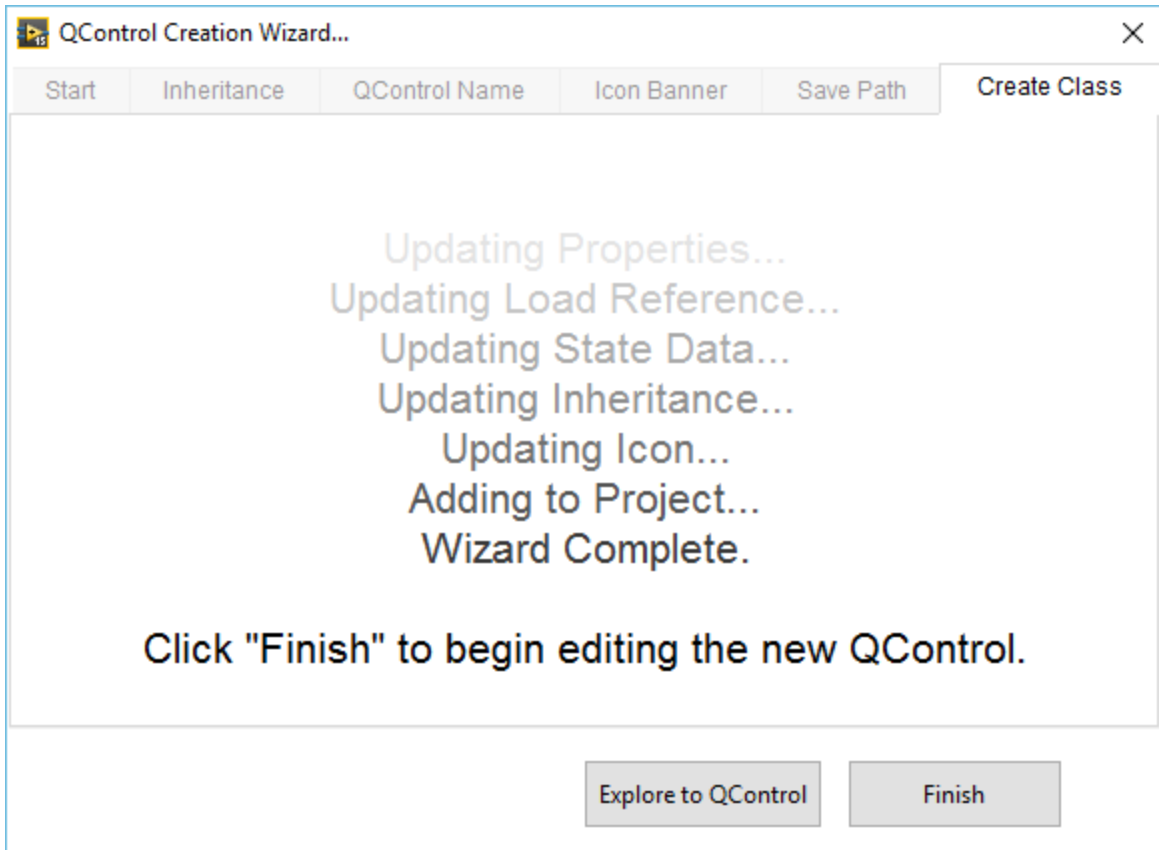
Figure 8 - Creating Class, Creation Complete

7. EXPLORE TO CLASS

Name	Date modified	Type	Size
Close Control.vi	9/14/2016 7:19 PM	LabVIEW Instrume...	22 KB
Close PhoneFormField.vi	9/14/2016 7:19 PM	LabVIEW Instrume...	22 KB
Close State Data.vi	9/14/2016 7:19 PM	LabVIEW Instrume...	22 KB
Event Handler.vi	9/14/2016 7:19 PM	LabVIEW Instrume...	22 KB
Initialize Control.vi	9/14/2016 7:19 PM	LabVIEW Instrume...	22 KB
Load Reference.vi	9/14/2016 7:19 PM	LabVIEW Instrume...	22 KB
Load State Data.vi	9/14/2016 7:19 PM	LabVIEW Instrume...	22 KB
New PhoneFormField.vi	9/14/2016 7:19 PM	LabVIEW Instrume...	22 KB
PhoneFormField.lvclass	9/14/2016 7:19 PM	LabVIEW Class	49 KB
State Data.ctl	9/14/2016 7:19 PM	LabVIEW Control	14 KB

8. OPEN AND CUSTOMIZE

The new QControl Class is now ready to be customized. The VIs that should be modified are:

- *Event Handler.vi*
- *Initialize Control.vi*
- *Close Control.vi*
- *State Data.ctl*

Also as many VIs as needed can be created for the Event Handler as SubVIs; Private, Protected, and Public Methods; and Properties. Read previous section of this document to learn more details of the function of each of these methods.

2.2 Modifying the Event Handler

After dismissing the wizard, you should see the new QControl Class, **PhoneFormField.lvclass**, has been added to your project.

Open the Event Handler.vi an open the Block Diagram. It should look like Figure 10. This is where the bulk of the code needed for this control should go. Follow these steps to add the customization:
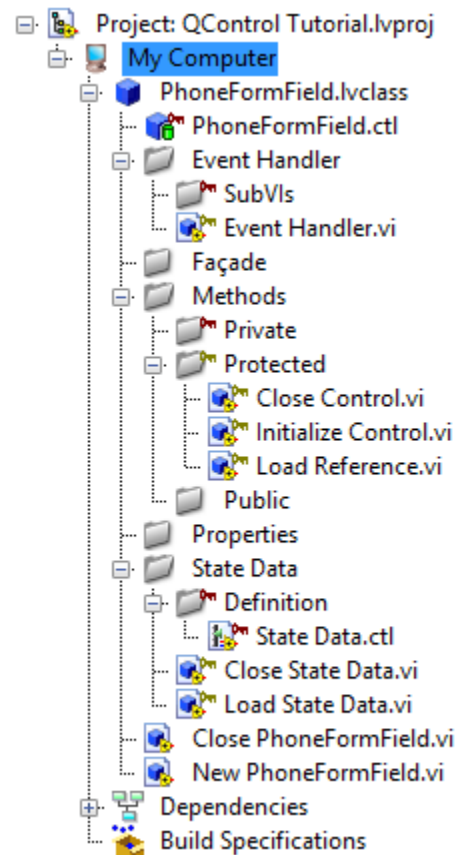
Figure 9 - Basic Starting Structure created by the QControl Wizard
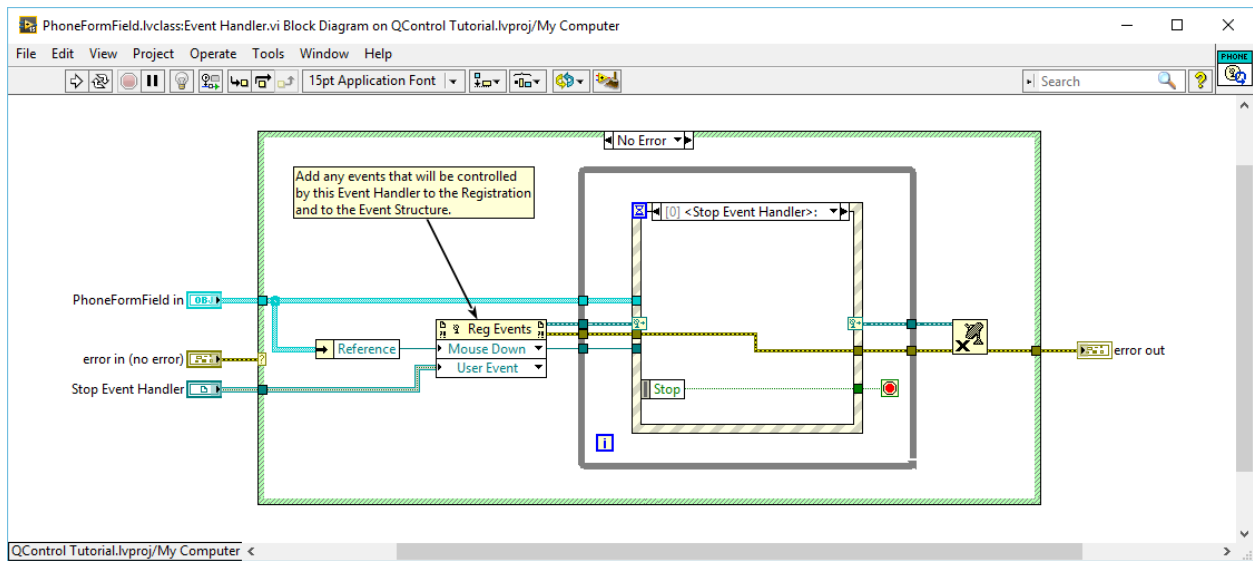
Figure 10 - PhoneFormField Event Handler

1. Change the "Mouse Down" to "Value Change" in the *Register for Events Node*.
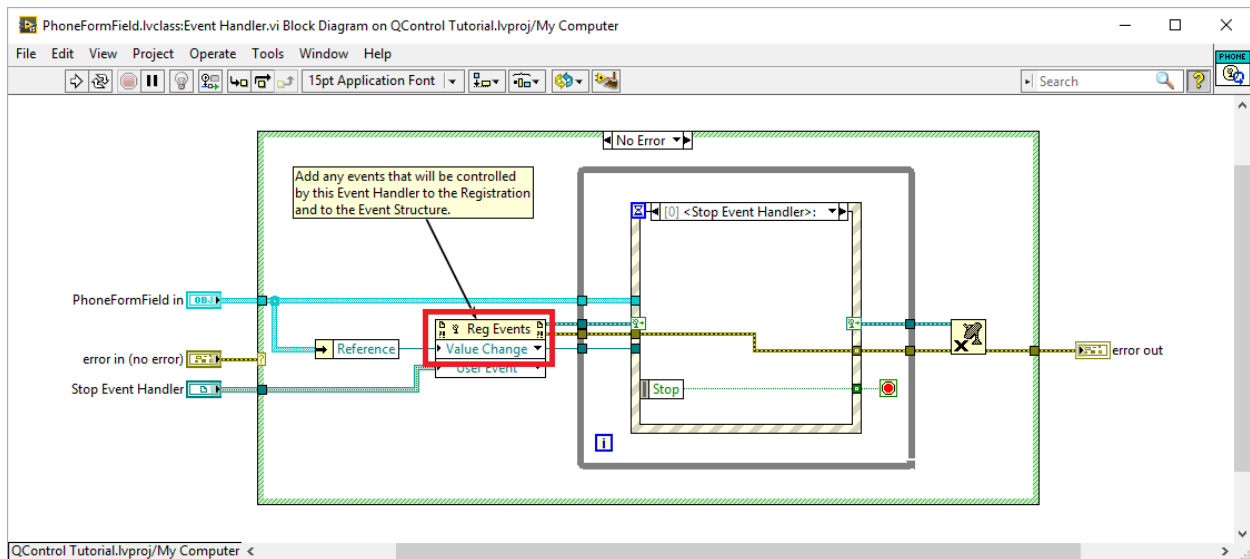
2. Add the Event Case for *<Reference>: Value Change*.
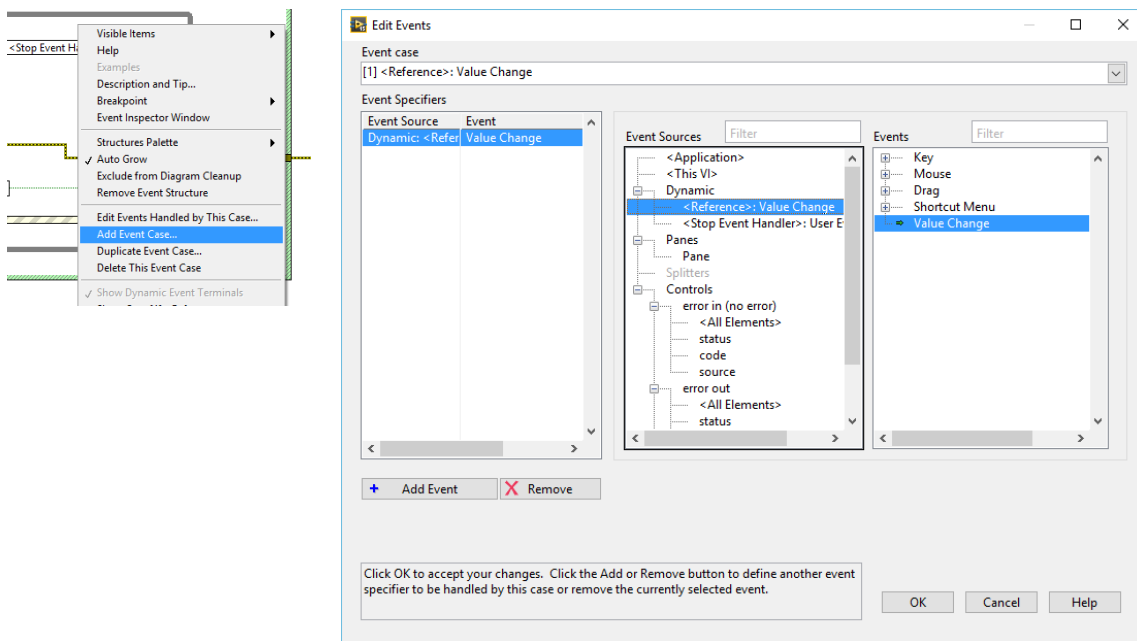
3. Expand the Event Structure and set the input Node to *NewVal*.
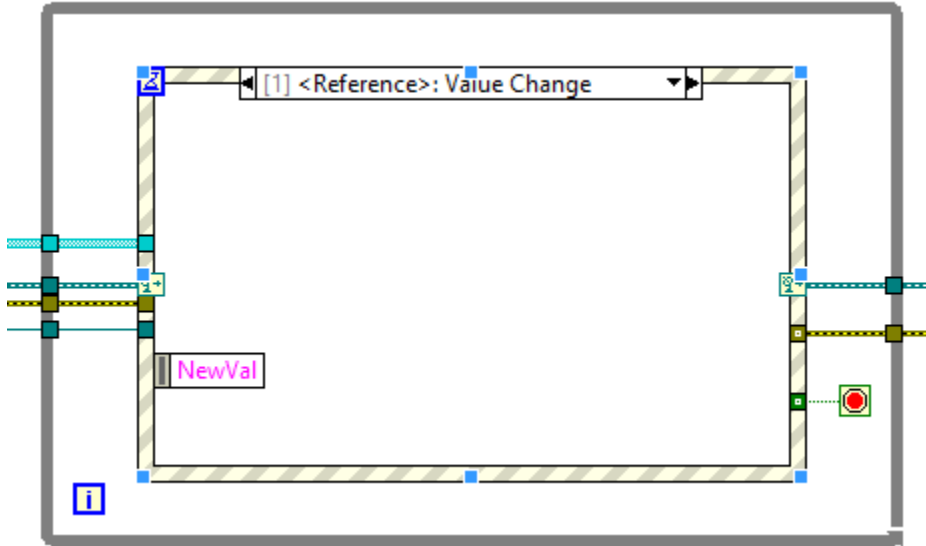
4. Add the check code as follows:
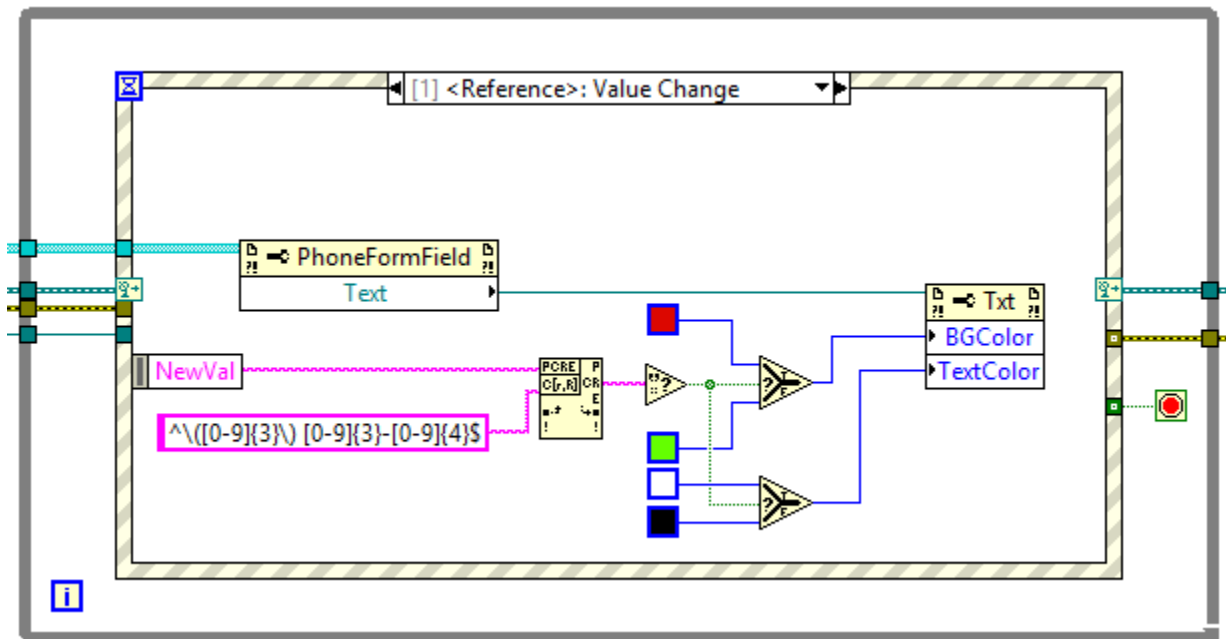
What this code does is to match the text to a regular expression which would give the phone number the form (###) ###-####. If the entry matches the output from **whole match** would be the number. If it is not a match the output from **whole match** is an empty string. The code checks for an empty string and sets colors accordingly.

2.3 Creating the Main VI

1. Now let's start the VI the PhoneFormField will be used in. Start a new VI that is part of your project but not part of the QControl Class. Add a String Control to the Front Panel. (Note: do not use the System String Control because colors are not changeable in them.)
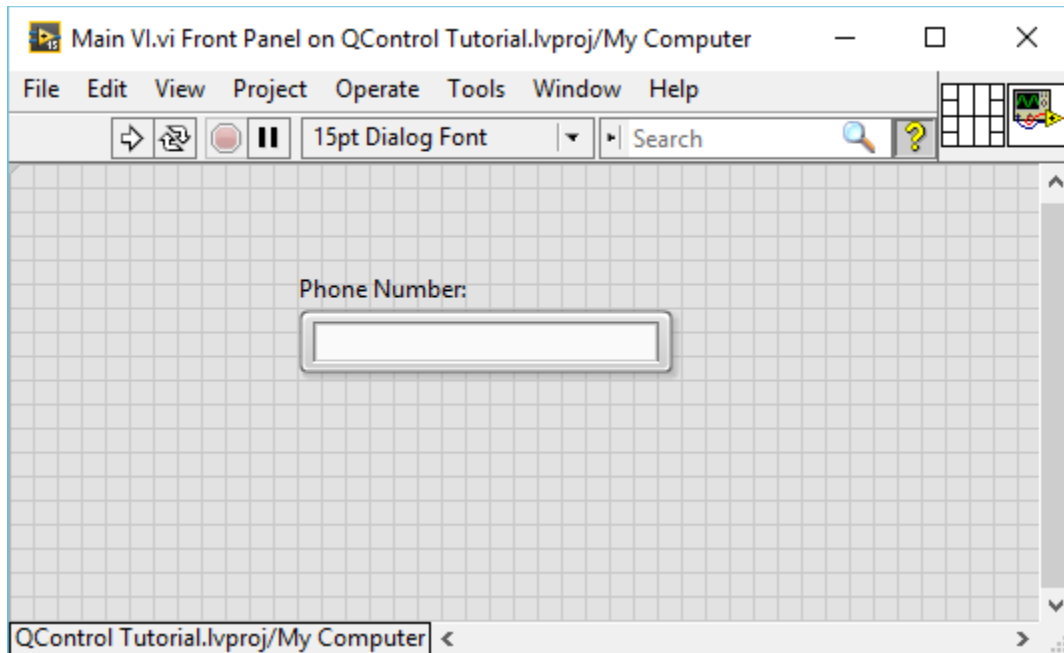
2. Change to Block Diagram and add code for a basic User Event Loop. Edit the event to discard the Panel Close and end the loop.
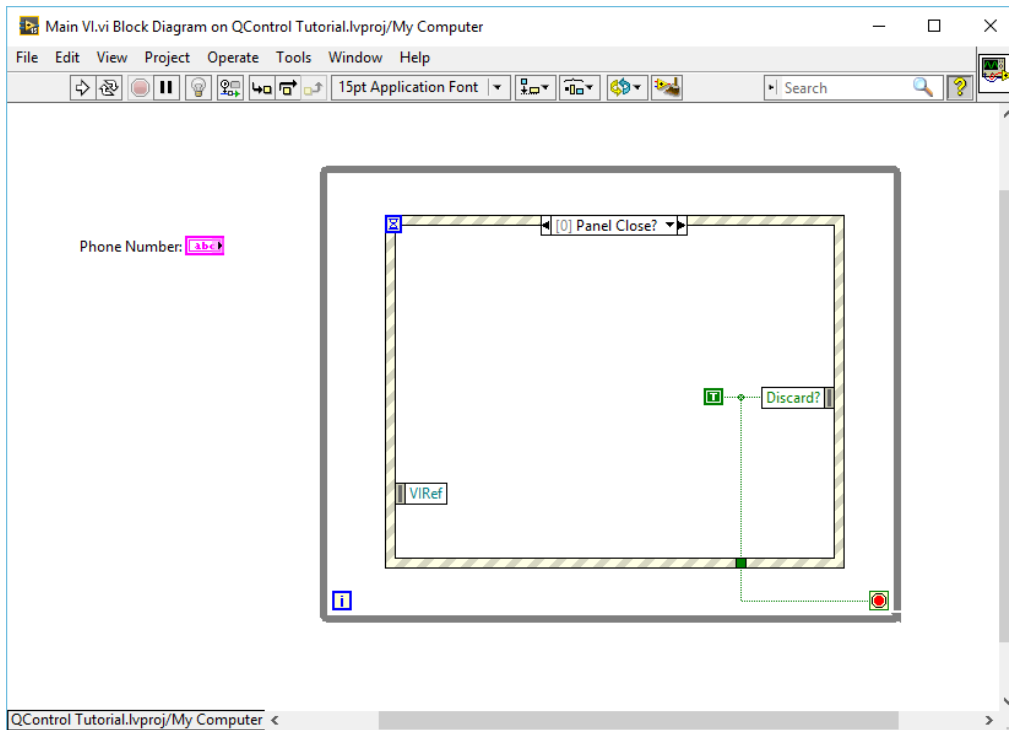
3. Next drag the method, *New PhoneFormField.vi*, to the block diagram and wire a reference to the string as input into it. Wire the output to shift registers on the loop.
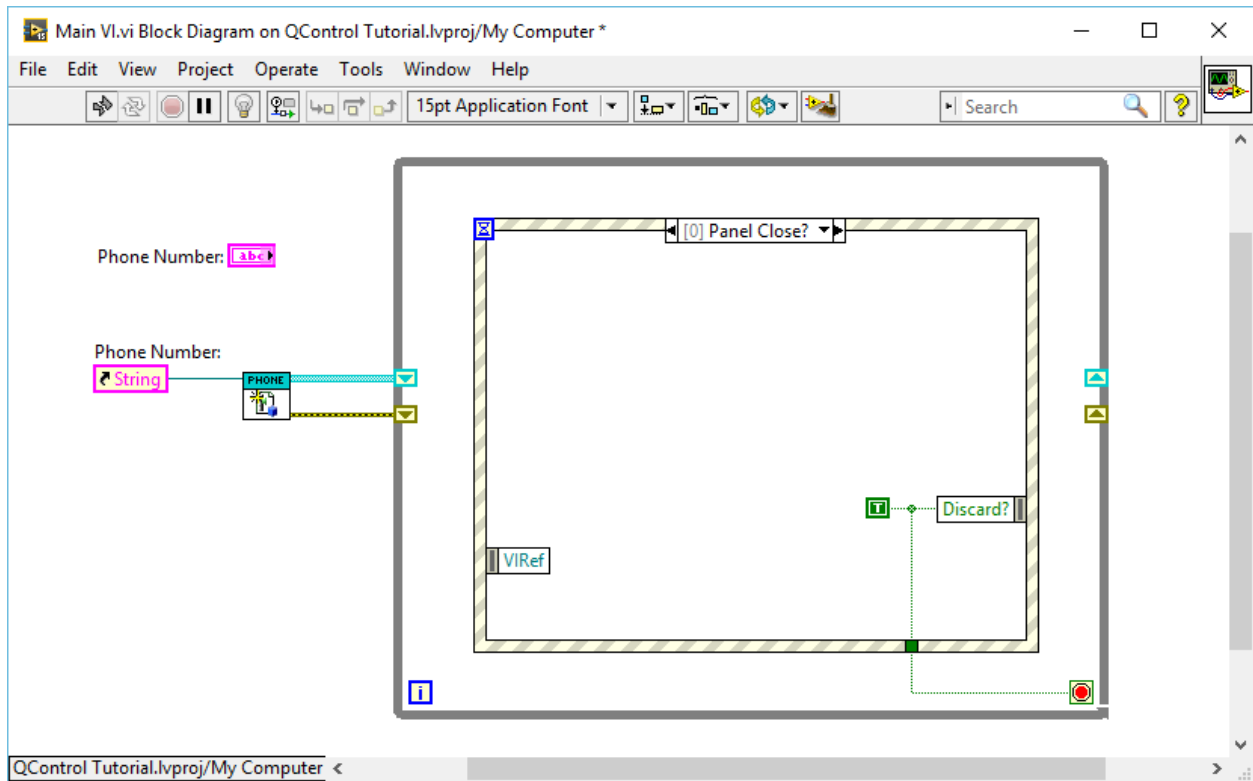
4. Next drag the method, *Close PhoneFormField.vi*, to the block diagram and wire the output shift registers to it.
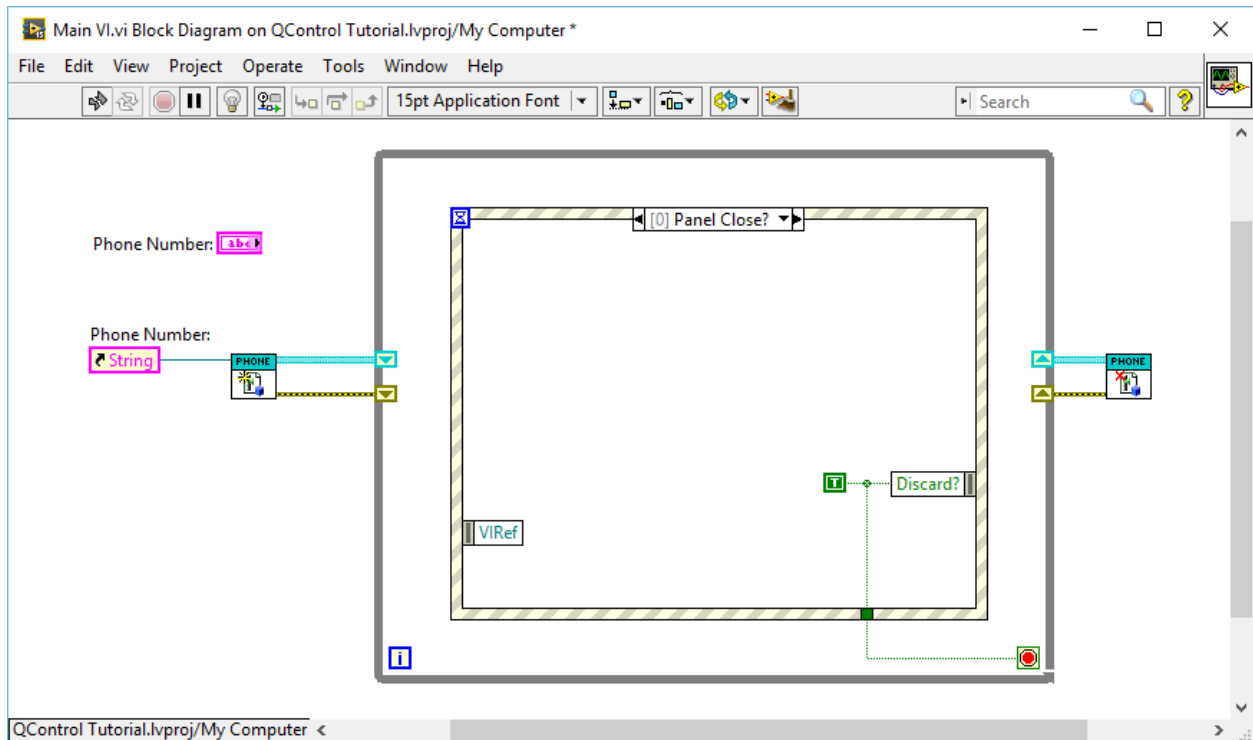
5. Because all of the logic for controlling the string is in the Event Handler. All that is needed is to pass the QControl Class wire and error wire through.
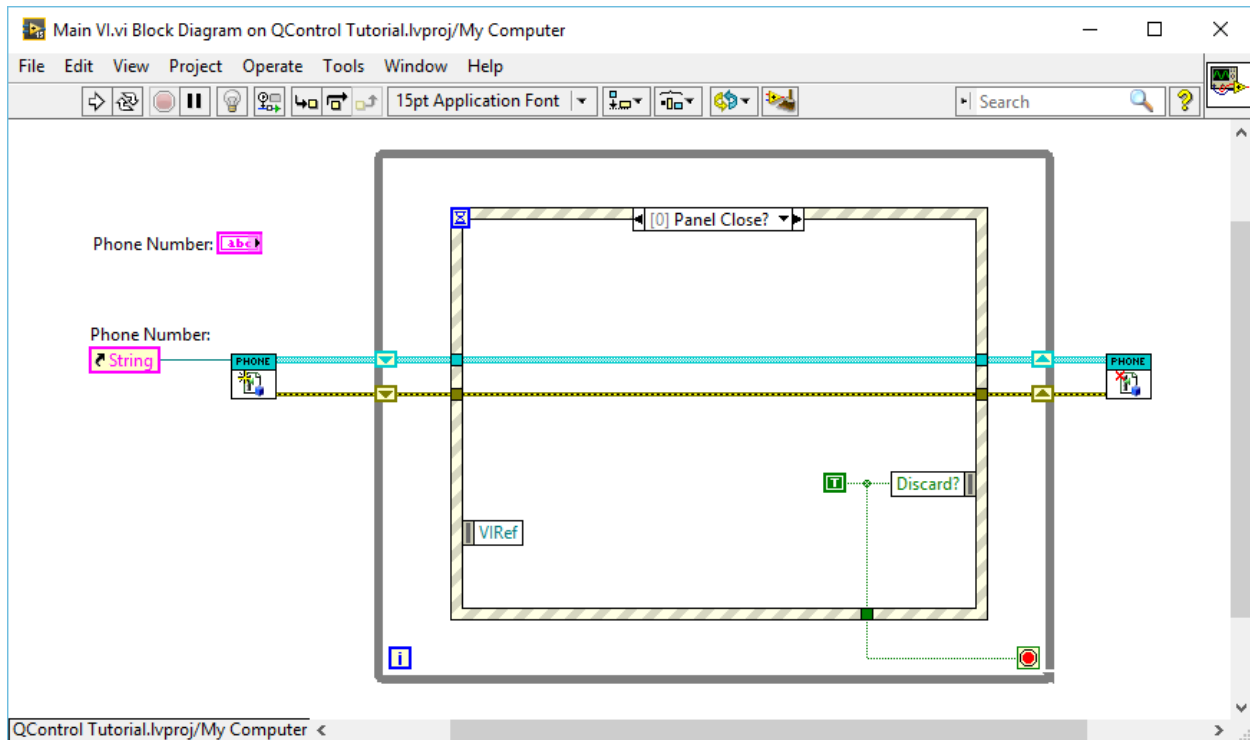
6. Switch to the front panel. Right click on the string and enable *Update Value while Typing*. Run the VI and try it out. The field will be red until the correct format is entered.
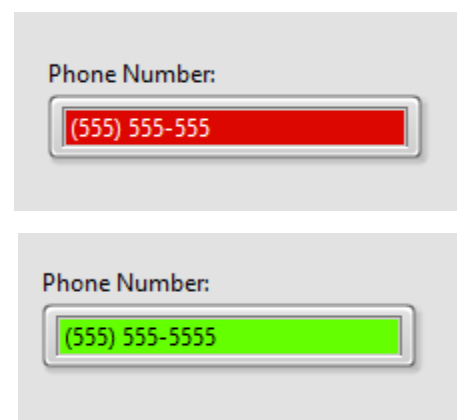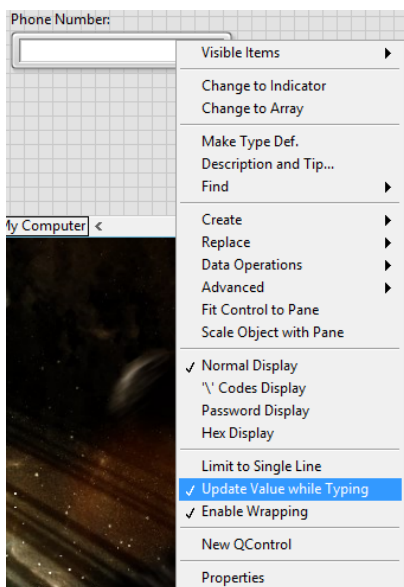
When finished playing, click the "X" in the corner of the Main VI.vi.

Lots of other possibilities could be created with this.

- The colors could be made to be settable through property nodes.
- The field could be limited to number entry only and put in the parenthesis and dashes automatically.
- By editing the regular expression other data entry could be allowed.
- The regular expression itself could be settable through a property node making the form field flexible.
- Instead of setting the color, the string could be combined with a ring control that displays a green check for good or red "x" for bad.

What you get is UI control logic that can be reused though multiple projects and with difference skins.

2.4   Modifying the State Data and Creating Properties

Now let's take one of the suggestions above and make the colors settable through property nodes.

1. First the State Data Definition given in the *State Data.ctl*. By default, QControls are created with only a Boolean in the State Data Cluster. This can be replaced with whatever is needed. The State Data Cluster is where you should put anything that is changeable that needs to be seen in your properties (that we haven't created quite yet) and the Event Handler.
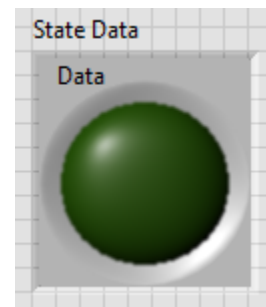
   In this case let's delete the Boolean and replace it with four color boxes. One for correct background color, one for correct text color, one for error background color and one for error text color. If you want to set default colors, set the color boxes, then right-click on the cluster and select *Data Operations -> Make Current Values Default*.

   Save and close the control.
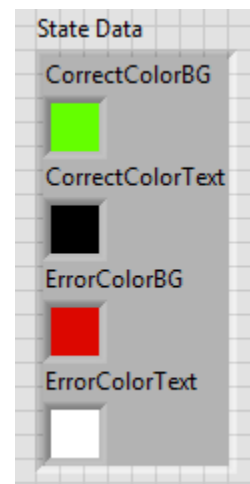
2. Back in the Project Explorer window, right-click of the Properties folder and select New -> Property Definition Folder. Name the Property Definition Folder **CorrectColorBG**.
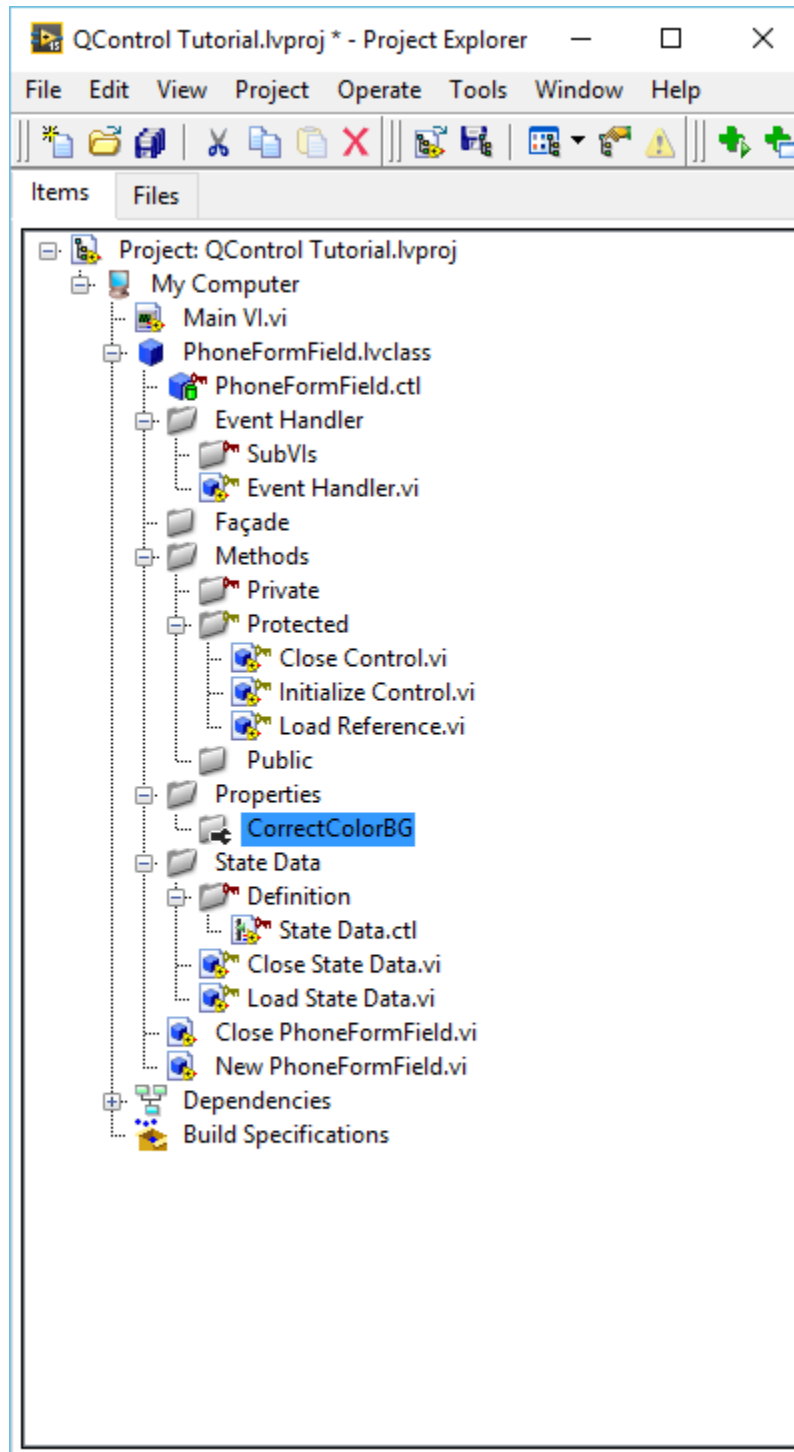
3. Next right-click on the CorrectColorBG Property Definition Folder and select *New -> VI from Static Dispatch Template*. A new VI should open. Save it Write **CorrectColorBG.vi**.

4. Add a Color Box to the front panel and name it **CorrectColorBG**. Connect it to the connector pane.
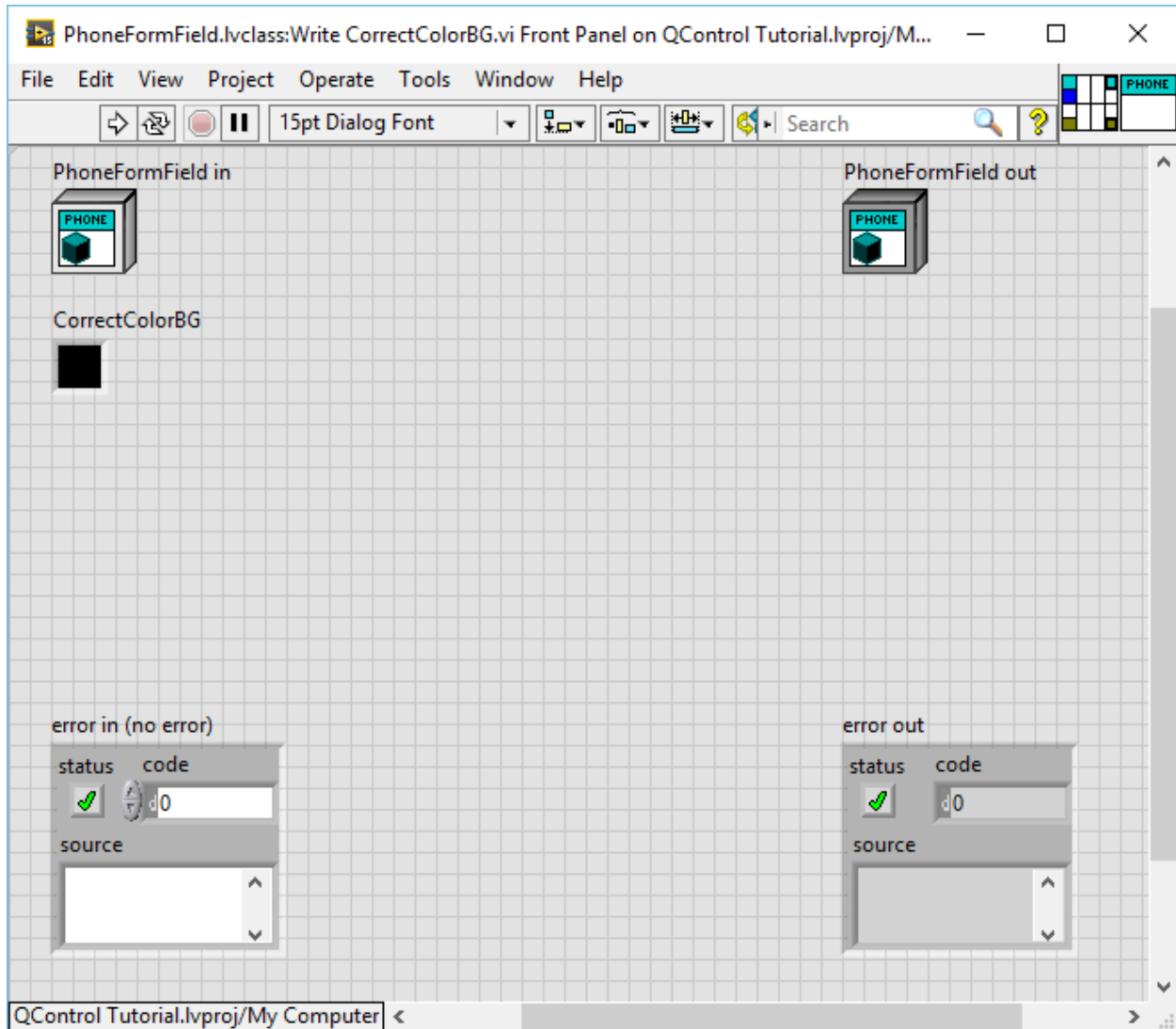
5. Open the block diagram.

6. Delete the wires inside the *Case Structure No Error Case* and add an *In Place Element Structure*.
7. Right-click on the *In Place Element Structure* and select *Add Unbundle / Bundle Elements*.
8. Change the Unbundle from Reference to State Data.
9. Add another *In Place Element Structure* inside of the first.
10. Right-click on that *In Place Element Structure* and select *Add Data Value Reference Read / Write Elements*.
11. Add a *Bundle by Name* inside of that and wire as shown:
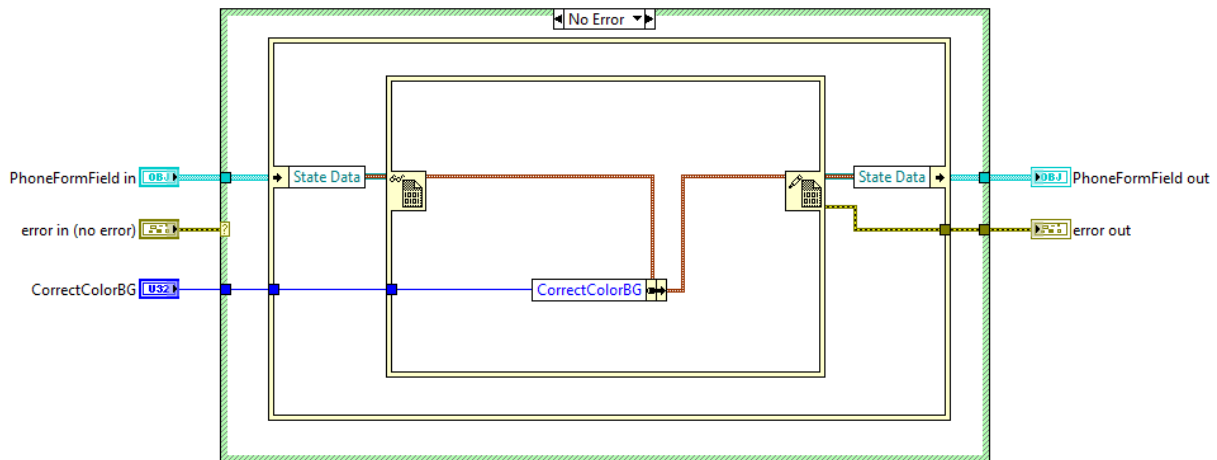
12. Save this VI with the changes.

13. Repeat steps 2-12 for CorrectColorText, ErrorColorBG, and ErrorColorText.

14. Do the steps 3-12 again for each but this time make the Read versions of the properties by

    a. making the color boxes indicators,
    b. connecting to the other side of the connector pane
    c. using an unbundle inside the *In Place Element Structures*

    The code for each should look similar to this:
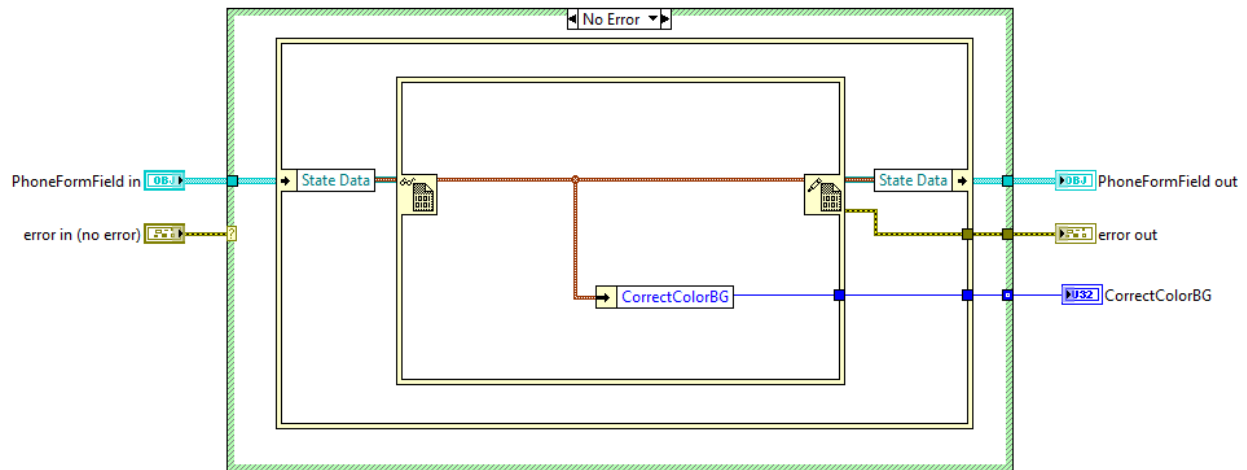
15. Once complete your Properties folder should look this this:
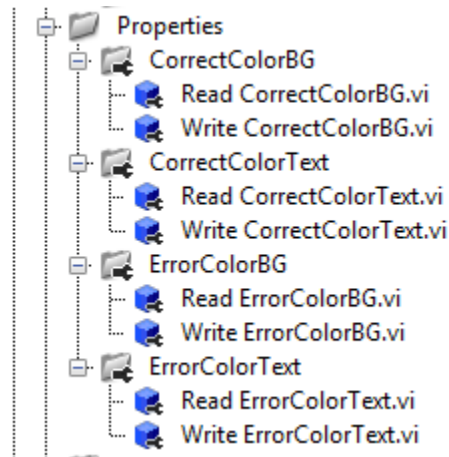
16. Now open the Event Handler code again. You might need to move things around to make room in the event structure.

17. Drag the property node open and add the new properties.
18. Delete the color box constants and wire the appropriate property to the Select Nodes.
19. Your code should now look similar to this:
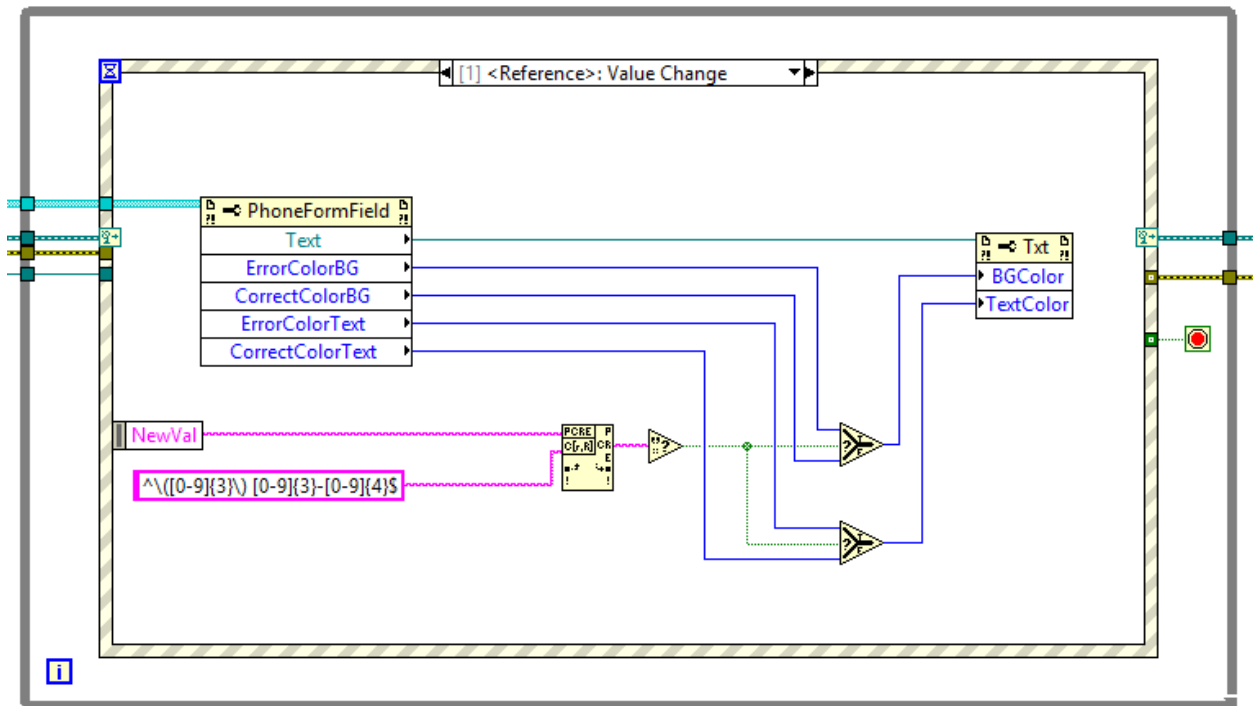
20. Save and close the Event Handler.
21. Open the Main VI.vi. Because of the default the VI should run and appear as it did before.
22. Switch to the block diagram.
23. Make room after the *New PhoneFormField.vi* and add a property node on the QControl Class wire.
24. Drag the property node open and set to the new properties (Change All to Write if necessary).
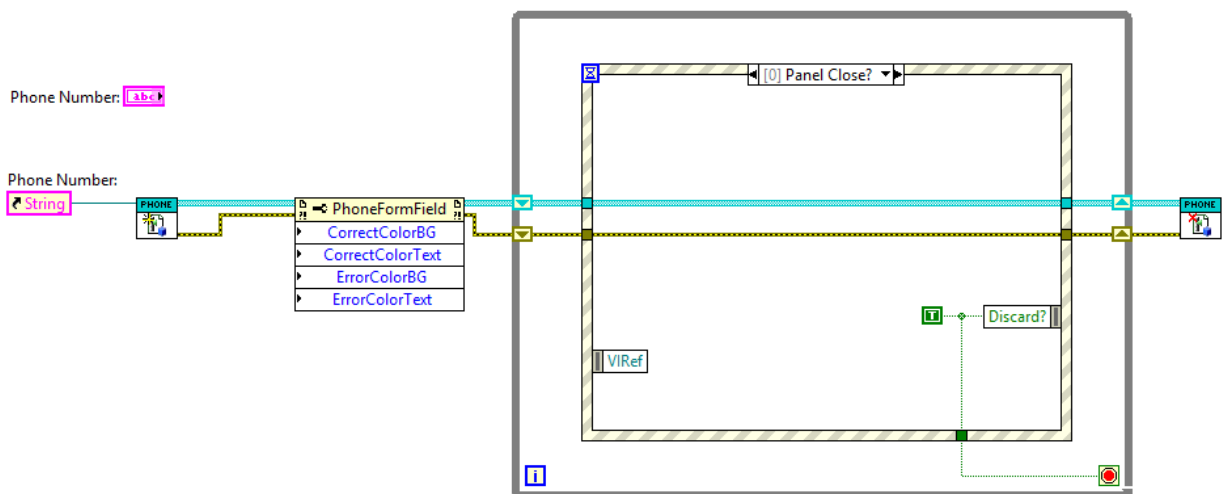
25. Create four color box constants and set them to any color you want to try. Wire them to the appropriate property.
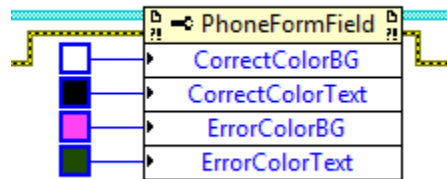
26. Run the VI again and observe the behavior.

2.5   Tutorial Summary

Now have fun creating other options for your new QControl. Maybe try some other suggestions listed.

For more examples search for QControls in the NI Example Finder. There are four examples distributed with the QControl Toolkit. From most simple to most complex, they are the:

- TreeDirectory Example
- StatusHistory Example
- TreeSelection Example
- LargeScrollbar Example

The TreeSelection Example shows how you can start the inheritance from one of your own QControls to extend its behavior in different ways.

The LargeScrollbar Example shows how you can combine multiple controls for use in one more complex control.

3   Definitions

Business Logic	Code that controls aspects of the program not associated with the User Interface (i.e. data acquisition, data analysis, file system, etc.)
UI Logic	Code that controls elements of the User Interface (i.e. checkboxes, color changes, enable/disable, etc.)
Skin	Appearance of the User Interface (i.e. use of Modern, System, or Silver Controls or creation of custom skinned controls)
Class Hierarchy	The hierarchy which defines the inheritance chain of the class.
VI Server	The LabVIEW VI Server is used to programmatically control objects in LabVIEW defined by the VI Server Class Hierarchy.
VI Server Class Hierarchy	"All properties, methods, and events belong to a class. Classes are arranged in a hierarchy with each class inheriting the properties, methods, and events associated with the class in the preceding level. For example, a button is a member of the Boolean class, which has a set of properties unique to it, such as the button height and width. In addition, all Booleans are members of the Control class, which includes the properties found in most other front panel controls and indicators, such as the Visible, Label, and Default Value properties." (LabVIEW Help)
QControl Toolkit	Collection of classes to aid in creating an object-oriented alternative to XControls. Also includes a QControl Creation Wizard to aid in developing Extended QControl Classes.
QControl Class Hierarchy	All classes in the QControl Toolkit.
QControl Class	Any class in the QControl Class Hierarchy which includes Interface Classes and Extended QControl Classes.
Interface Classes	Classes in the QControl Class Hierarchy which are made to mimic the class hierarchy found in the VI Server Class Hierarchy. They should be used for inheritance purposes only and are utilized only by creating an Extended QControl Class that inherits from one of them or from another Extended QControl Class.
Control.lvclass	The main interface class in the QControl Class Hierarchy. Inheriting from Control.lvclass or from a class that is descended from it gives the class the ability to have an asynchronously launched Event Handler Method.
Extended QControl Classes	A QControl Class that inherits from an Interface Class in the QControl Class Hierarchy or from another Extended QControl Class.
By-Reference Class	An Object-Oriented design structure where a class holds only references pointing to data (i.e. control references, DVRs, Queues, etc.).

4   What is a QControl?

A QControl is an object-oriented alternative to using an XControl. It is:

- A LabVIEW Object-Oriented Class with a Control Reference as part of its Private Data where all manipulation of the Control should be done through Properties and Methods of its Class
- A class that can be reused to recreate the UI Logic wherever required
- Could have an asynchronously called Event Hander that handles UI Logic
- Part of the QControl Class Hierarchy which mimics the VI Server Class Hierarchy

4.1   Tradeoffs of a QControl vs an XControl

There are tradeoffs to using a QControl versus using a regular XControl.

A QControl has the same benefits of an XControl by:

- Encapsulating UI Logic Code
- Maintaining the same functionality when used in multiple instances
- Allowing for complicated UI code to be abstracted from other developers
- Allowing for custom properties accessible through Property Nodes

It is better than an XControl by:

- Not requiring separate Edit Time behavior to be developed
- Decoupling the UI Look (Skin) from the UI Logic
- Being easier to use with Object Oriented Programing
- Being easier to use with LabVIEW Libraries (LVLibs) and Packed Project Libraries (PPLs)
- Inheriting from current LabVIEW controls to give access to their properties
- Being more stable

However, it does have the drawbacks of:

- Not having customizable Edit Time behavior
- Not being able to define Data Type
- Not being able to enforce the use of the defined façade
- Slightly more complicated programming in usage (does not exist as just a terminal on the block diagram)
- Slightly more complicated when programming a custom control with more than one control in it

4.2   Why use a QControl instead of an XControl?

If the Tradeoffs above didn't help here is a more detailed explanation. XControls have two main problems with their implementation that boil down to when an XControl is running.

First, the nature of an XControl is that it is always running when it's owning VI is in memory but only reacts during the user's interaction with it. An XControl starts and its Init Ability executes when the XControl is placed on the Front Panel of a VI that is not in a Library or Class or when said VI opens and is loaded into memory. The XControl then closes and its Uninit Ability (if it has one) executes when that VI leaves memory or when the XControl is deleted from the VI.

This becomes more complicated if the owning VI is in a Library or Class. All VIs in a Libraries or Classes are loaded into memory when the Library or Class is loaded into memory. This means that the Init and Uninit Abilities run at the load and unload of the Library or Class and not during the load and close of the owning VI. Loading of Libraries and Classes, unless called dynamically, are when the Project loads and closes. XControls become even more unstable if it is compiled and used from Packed Project Libraries (PPLs) and/or is used when a LabVIEW Class Object is its data type. As such XControls are also not recommended for use with the Actor Framework.

Second, XControls are difficult to handle during the edit time of the owning VI. Because they are always running, interaction during edit time has to be programmed into the XControl (an XControl has a flag that tells it if the owning VI is in Run Mode or Edit Mode). This causes a lot more coding necessary just for the ease of the developer that will never be seen by the end user. Also any properties and method to the control(s) on the Façade have to be recreated as Property and Method Abilities of the XControl.

QControls on the other hand do not execute until the owning VI executes. All edit time behavior is conserved. There is no limitations for use in VIs that are part of other Libraries or Classes including PPLs and the Actor Framework.

Also, while using the QControl Toolkit, all properties of the control it is based on, when consisting of single controls, are passed through by nature of inheritance in the QControl Class Hierarchy. (QControls that have more than one control in a cluster can still have the properties and methods pass through but they would have to be created just as they would have to be created in an XControl.)

5   VI Server Class Hierarchy

The VI Server Class Hierarchy is the internal LabVIEW organization of classes which manipulates objects via properties and methods of those classes. The LabVIEW Help defines it as:

> "All properties, methods, and events belong to a class. Classes are arranged in a hierarchy with each class inheriting the properties, methods, and events associated with the class in the preceding level. For example, a button is a member of the Boolean class, which has a set of properties unique to it, such as the button height and width. In addition, all Booleans are members of the Control class, which includes the properties found in most other front panel controls and indicators, such as the Visible, Label, and Default Value properties." (LabVIEW Help)

The organization of these classes can be seen in the submenus when selecting the class in Class Specifier Constants, (see Figure 11); and in the segregation of properties when selecting a specific property in a Property Node, (see Figure 12).
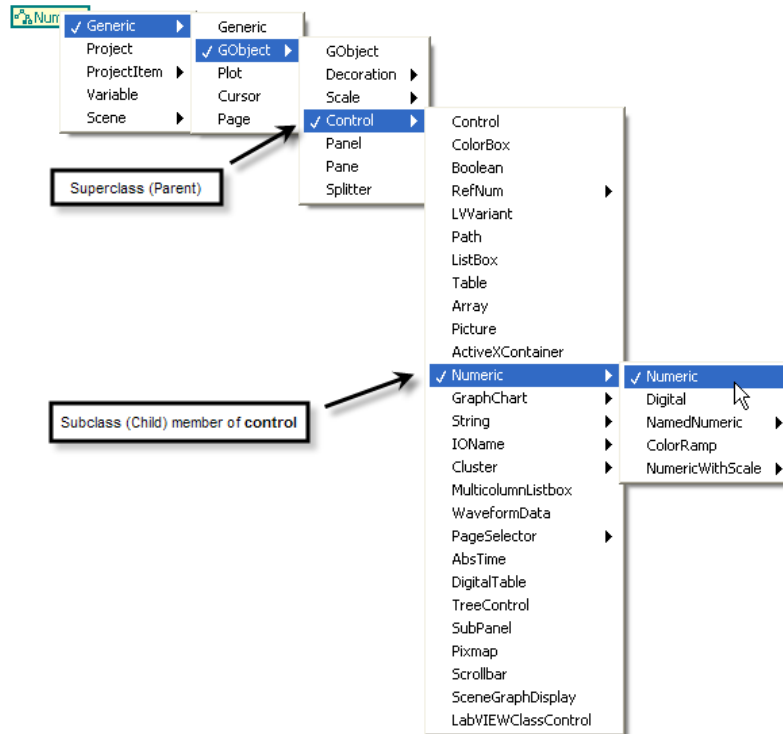
Figure 11 - VI Server Class Hierarchy
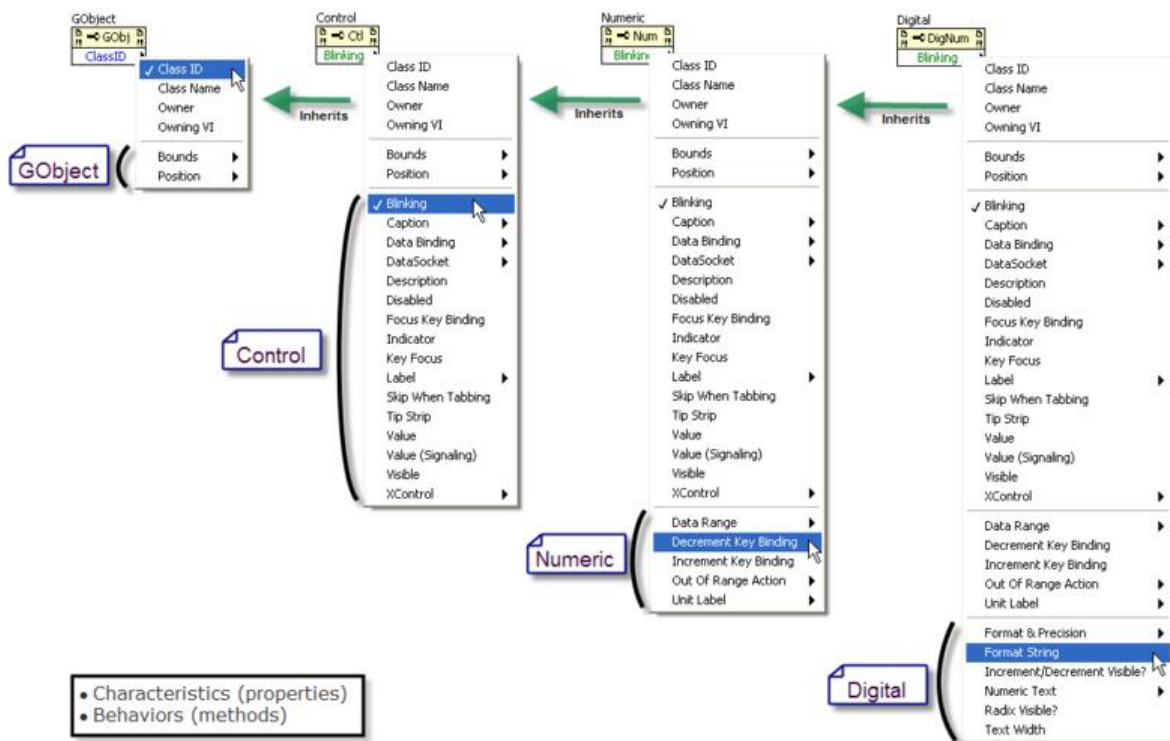
Figure 12 - Organization of Properties by the Classes of the VI Server Class Hierarchy

6   QControl Class Hierarchy

The QControl Class Hierarchy mimics the organization of classes as defined by the VI server Class Hierarchy.
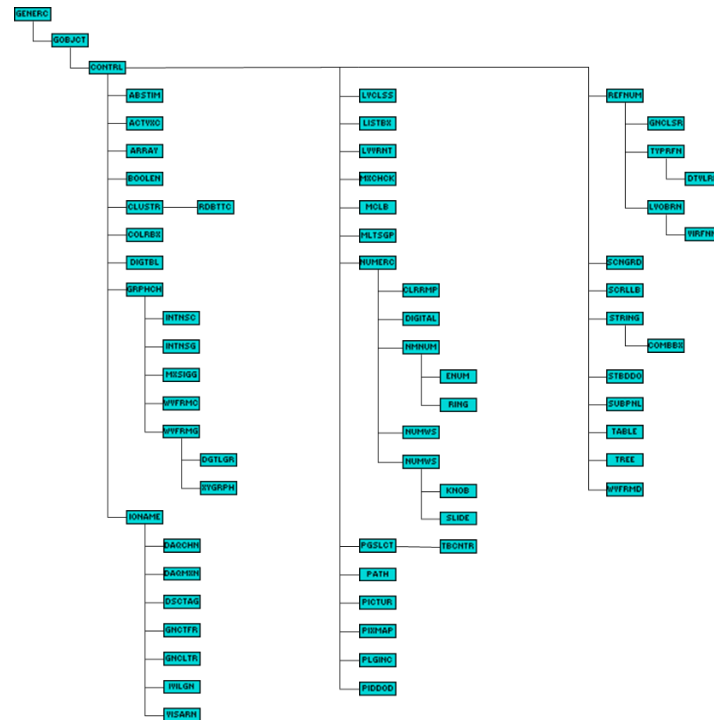
Figure 13 - QControl Class Hierarchy
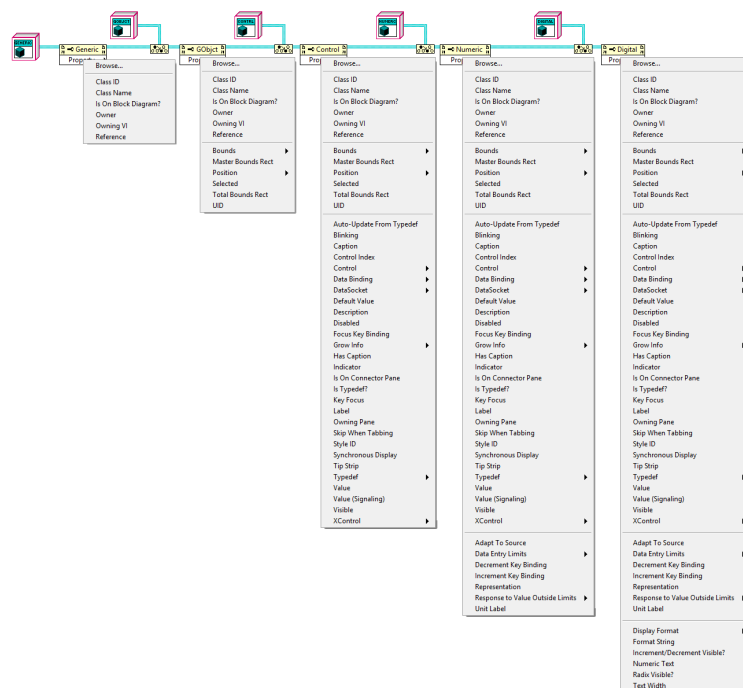
Figure 14 - Organization of Properties by the Classes of the QControl Class Hierarchy

7   Parts of a QControl Class

A QControl Class is first defined as a class that inherits from the *Control.lvclass*. This class must be a By-Reference Class therefore, the private data of this class must only hold:

- The control reference of the type the class it is based on (i.e. Boolean, Numeric, etc.), see Section 7.1.1
- The DVR reference to the QControl Class' State Data, see Section 7.1.2
- Any other User Event, Queues, or Notifier references used by Properties, Methods and/or the Event Handler

No Accessors to these reference should be given outside of the class.

By inheritance the new Class inherits:

- Properties and Methods of Classes in its Class Hierarchy (see Sections 7.2 and 7.3)
- State Data of Classes in its Class Hierarchy (see Section 7.1)
- Automatic Launching of Event Handlers, if the Event Handler Method is overridden (see Section 7.1.4)
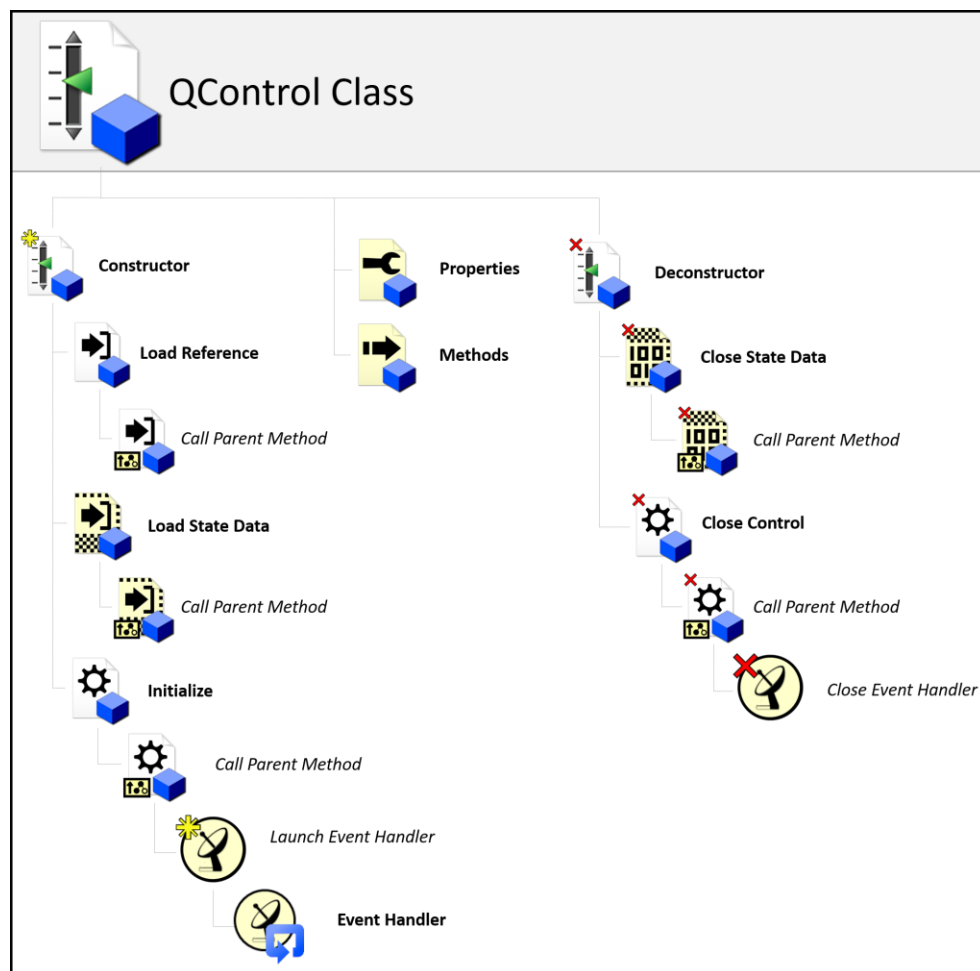
Figure 15 - Parts of a QControl Class

7.1 Constructor Method

The Constructor Method is the method that instantiates the new QControl Class. It is named "New" and then the name of the QControl Class (i.e. *New StatusHistory.vi*). Its only input is the control reference that the new QControl Class will control.
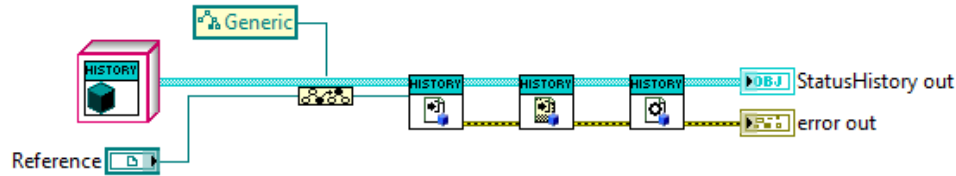
Figure 16 - Example Constructor Method code with State Data and Event Handler

The Constructor will then:

1. Cast the reference to "Generic" and call the Load Reference Method to load the control reference to the class private data
2. Call the Load State Data Method which loads the State Data DVR to the class private data
3. Call the Initialize Method to perform any initialization and start any other references used by the QControl Class and launch the Event Handler

The output of the Constructor Method is the QControl Class Wire. This wire should be handled and passed just like the reference of a basic control/indicator. **Any manipulation of the control after the Constructor Method should use Properties or Methods on this wire or handle UI Logic in the Event Handler.**

7.1.1   Load Reference Method

The Load Reference Method is a dynamic dispatch VI that must be overridden in every QControl Class and must call parent method. It is only used in the Constructor Method.
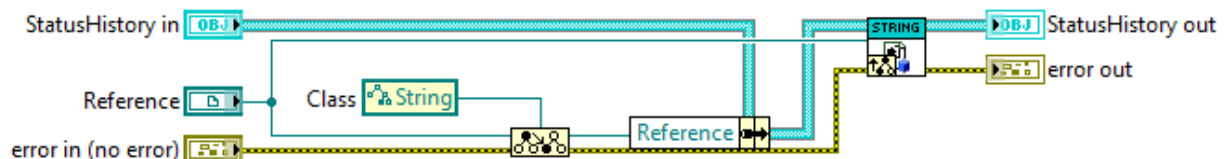
Figure 17 - Example Load Reference Method

Its purpose is to:

- Cast the reference back into the correct type for the QControl Class
- Bundle the reference into the QControl Class' private data
- Uses Call Parent Method to pass the reference up the class hierarchy

It must use Call Parent Method because this will cause the reference to be recursively loaded to the data space for the QControl Class up the class hierarchy. These references must be loaded in this way to use all of the built-in properties.

7.1.2　　Load State Data Method

The Load State Data Method is a dynamic dispatch VI that is optionally overridden if needed. It is only used in the Constructor Method. The Constructor Method will call the parent method if it is not overridden.
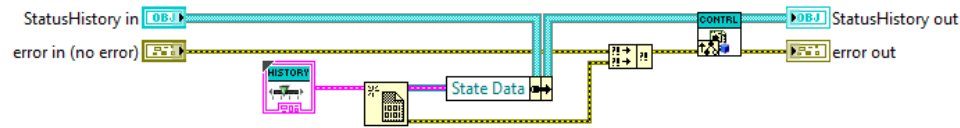
Figure 18 - Example Load State Data Method

Its purpose is to:

- Create the DVR based on the State Data Definition (see Section 7.1.2.1)
- Bundle the DVR into the QControl Class' private data
- Use Call Parent Method to Call the Load State Data Method up the class hierarchy

7.1.2.1　State Data Definition

The State Data is defined as a type-definition cluster in the QControl Class. It is then wrapped in a DVR and the DVR reference saved in the class private data.

The State Data holds any data that could change via a Property or an event in the Event Handler. This is data necessary for the extended behavior of the control that is not held in the control itself. For example:

- Properties like: Colors, Fonts, Size of History, Flags etc. (changed through user interaction)
- State Data like: Flags, Selections, Counters etc. (changed programmatically to store state)

7.1.3　　Initialize Method

The Initialize Method is a dynamic dispatch VI that is optionally overridden if needed. It is only used in the Constructor Method. The Constructor Method will call the parent method if it is not overridden.

This method should contain any code need to start the QControl Class. For example:

- Initial state UI
- Creation of other references (i.e. Notifiers, Queues, User Events, etc.) that are used in the Properties, Methods, and Event Handler

The Initialize Method must call the parent method because it is the Initialize Method of the *Control.lvclass* that calls the Launch Event Handler which ultimately calls the overridden Event Handler if there is one.

7.1.4　　Event Handler Method

The Event Handler Method is an optional override. This is where any UI Logic code must be programmed. It is a dynamic dispatch VI that should be overridden if events are needed to create the enhanced QControl Class.

By inheriting from the main *Control.lvclass* the overridden Event Handler will be launched asynchronously in the Initialize Control.vi of the QControl Class.

7.2 Properties

Properties are any aspect of the UI Logic that the developer using the Control Class can manipulate through only one input (Write Property) or one output (Read Property). These are setup like accessors of a class that are available through a Property Node but can have more complicated logic than just a bundle/unbundle of private data. Properties are defined by the Property Definition Folder in a class. Properties created should be logical manipulation of the QControl Class like:

- Setting appearance (color, size, etc.)
- Setting flags (visibility, state, etc.)

These can be part of the control being controlled or be further accessors through the DVR to change State Data. After written to, in the case of a Write Property, the control appearance could be updated and the value saved to the State Data.

7.3 Methods

Methods like Properties are any aspect of the UI Logic that the developer using the QControl Class can manipulate. The difference being that they could have multiple inputs and/or outputs or even none at all. On a regular control or indicator a method would be invoked using an Invoke Node. Unfortunately, classes do not have the option of accessing methods through the Invoke Node. Therefore, Methods must be public and are only accessible by using the VI on the block diagram of the QControl Class' owning VI.

7.4 Deconstructor Method

The Deconstructor Method closes the QControl Class and ensures all references are closed. If this method is not used when closing the owning program, the Event Handlers will still be closed but the State Data DVRs and any other references started might not be and cause memory leaks. Best practice would be to use the Deconstructor Method. It is named "Close" and then the name of the QControl Class (i.e. *Close StatusHistory.vi*).

The Deconstructor Method will then:

1. Call the Close State Data Method
2. Call the Close Control Method

7.4.1 Close State Data Method

The Close State Data Method is a dynamic dispatch VI that is optionally overridden. It is only used in the Deconstructor Method. The Deconstructor Method will call the parent method if it is not overridden.

It is required if the Load State Data Method was overridden to delete the State Data given by the DVR in the class private data. It then calls the parent method to delete the State Data up the class hierarchy.
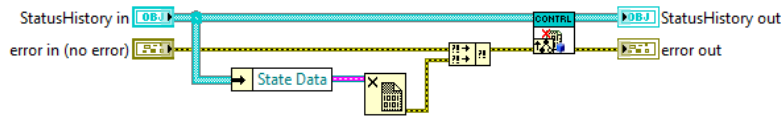
Figure 19 - Example Close State Data Method

7.4.2   Close Control Method

The Close Control Method is a dynamic dispatch VI that is optionally overridden if needed. It is only used in the Deconstructor Method. The Deconstructor Method will call the parent method if it is not overridden.

In the Close Control Method close any references that were started in the Initialize Method. Ultimately, the Call Parent Method will cause the Close Control Method owned by the *Control.lvclass* to run which will run the Close Event Handler Method as a failsafe to ensure the Event Handler is left running.

7.5   Façade Control (Optional)

A Façade Control is optional because the use of the defined façade is not enforceable programmatically as it is with an XControl. However, a Façade Control can be included with the QControl Class and is recommended if the control is more than just a built in LabVIEW control.

> For Example, included with the toolkit is a LargeScrollbar QControl Class that is built as a cluster that has two buttons and a slider in it. Included with this QControl Class is two Façade Controls: a Horizontal Scrollbar, and a Vertical Scrollbar. Given the nature of the cluster as the control and how the methods of the QControl Class are built expecting this exact configuration, it is easier to use one of these to start rather building something different. However, the developer can build something different if they wish as long as it conforms to the expected structure.

When using a Façade Control it is recommended that it remain a Control and not a Type Definition or Strict-Type Definition. This is due to the fact that some properties and methods because unavailable when a control becomes a Type Definition or Strict-Type Definition.

8   Interface Classes

The Interface Classes have three types: (See Figure 20)

1. Classes that are in the class hierarchy of *Control.lvclass*, (*GObject.lvclass* and *Generic.lvclass*)
2. The *Control.lvclass*
3. Classes that have *Control.lvclass* somewhere in its' class hierarchy
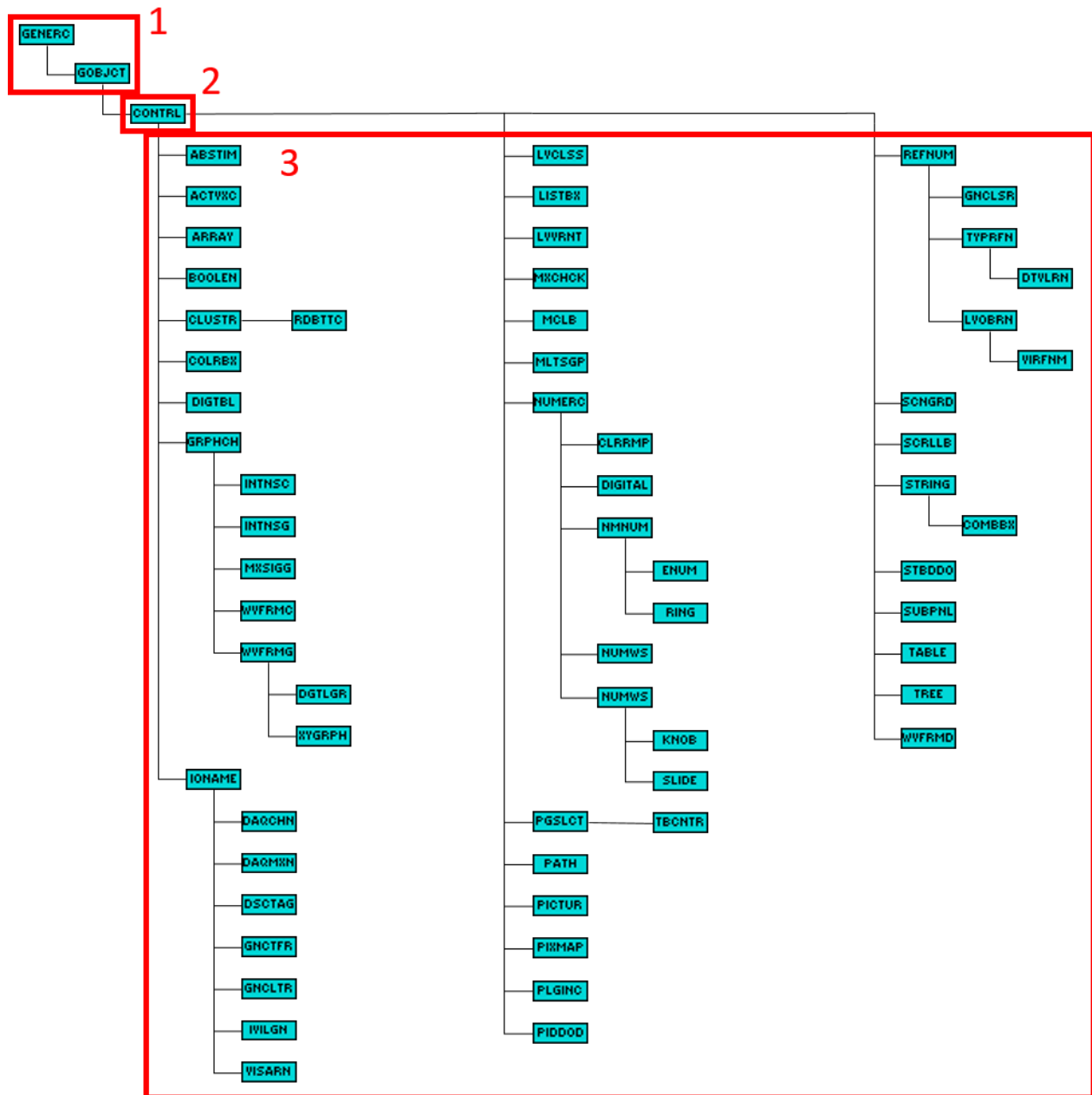
Figure 20 - QControl Class Hierarchy, Interface Classes Only

8.1   GObject.lvclass and Generic.lvclass

The *Control.lvclass* inherits from the *GObject.lvclass* and the *GObject.lvclass* inherits from the *Generic.lvclass* to mimic the class hierarchy in the VI Server Class Hierarchy. This is setup in this way to recreate the organization of and access to properties and methods.

The *GObject.lvclass* and *Generic.lvclass* should not be inherited from directly to create extended QControl Classes. They do not allow the launching of event handlers or the defining of State Data DVR's (See Section 8.2 Control.lvclass).

8.2   Control.lvclass

The *Control.lvclass* is the main class which by inheriting from it makes a QControl Class functional. The code that controls launching and closing of the Event Handlers, References, and State Data DVRs are controlled by this class.

The *Control.lvclass* should be used only as an interface class. To create an Extended QControl Class from this class use the QControl Creation Wizard and set to inherit from this class (see Section 0).

This class inherits from the *GObject.lvclass* and *Generic.lvclass* but this is so the properties of those classes mimic the look of properties when accessing them through the VI Server.

8.3   Other Interface Classes

All VI Server Classes for Controls have been recreated as QControl Classes for use in inheritance to recreate the organization of properties and methods. These classes can be used to extend the capability of any of the built-in controls.

9   Extended QControl Classes

Extended QControl Classes are any classes that inherit from an Interface Class or from another Extended Control Class. There are some distributed with the QControl Toolkit and three used in the QControl Creation Wizard; namely: the StatusHistory Class, the Steps Class, and the TreeSelectionSingle Class. The code for the wizard is open source to show an example. However, do take care not to modify the code to prevent the risk of making it operate incorrectly.

Extended QControl Classes distributed with the QControl Toolkit are as follows:

9.1   LargeScollbar Class

This class is an **Extended QControl Class** that inherits from the **Cluster Class**. The purpose of this class is to provide a Large/Wide Scrollbar for use with resistive touchscreen displays. It expects two buttons and a slide control to be inside of the cluster in the correct order of button->slider->button. The Façade folder contains custom controls with the necessary components to be used with this QControl and contains a vertical and a horizontal option.

9.2   MulticolumnListboxSelection Class

This class is an **Extended QControl Class** that inherits from the **MulticolumnListbox Class**. The purpose of this class is to provide checkbox functionality that shows which items are selected.

9.3   SliderBackgroupGradient Class

This class is an **Extended QControl Class** that inherits from the **Slide Class**. The purpose of this class is to provide the ability to set two colors. The slider at on side will be one of these colors and then transition to the second color as the slider is moved to the other end.

9.4   StatusHistroy Class

This class is an **Extended QControl Class** that inherits from the **String Class**. The purpose of this class is to be a string control that will remember a history of the last values sent to it and will display them in a list. There is properties to set up the size of the history and a starting and ending gradient color. The most recent value will have the text color set to be the starting color with the history blending to the

last color with the last item in the history. This **Extended QControl Class** is used in the **QControl Creation Wizard**.

9.5 Steps Class

This class is an **Extended QControl Class** that inherits from the **TabControl Class**. The purpose of this class is to provide wizard-like functionality where only one tab at a time is enabled. The Next and Previous Methods can be used to programmatically switch the active tab to provide the steps through the wizard. This **Extended QControl Class** is used in the **QControl Creation Wizard**.

9.6 TreeDirectory Class

This class is an **Extended QControl Class** that inherits from the **TreeControl Class**. The purpose of this class is to display the contents of a folder in a tree. When double-clicked the item will open (Windows OS only).

9.7 TreeSelection Class

This class is an **Extended QControl Class** that inherits from the **TreeControl Class**. The purpose of this class is to provide checkbox functionality to show selection of items in the tree.

9.8 TreeSelectionHierarchal Class

This class is an **Extended QControl Class** that inherits from the **TreeSelection Class**. The purpose of this class is to provide checkbox functionality to show selection of items in the tree. When Items are selected the selection is propagated to its descendants. Ancestors could also be changed to the Mixed Checkbox symbol if all descendants are not all TRUE or FALSE.

9.9 TreeSelectionSingle Class

This class is an **Extended QControl Class** that inherits from the **TreeSelection Class**. The purpose of this class is to provide checkbox functionality to show selection of items in the tree. When a selection is made, only one item at a time in the entire list is allow to be TRUE. All other will be forced to FALSE. This **Extended QControl Class** is used in the **QControl Creation Wizard**.

10 QControl Creation Wizard

When the QControl Creation Wizard is opened it will lead the user through a series of steps in order to create the skeleton of the new Extended QControl Class.

10.1 Parts of the QControl Creation Wizard

The wizard distributed with the QControl Toolkit contains two major components:

- The QControl Creation Wizard Class – Wizard User Interface and example of QControl usage
- The QControl Creation Class – script to create new QControl Classes

10.1.1 The QControl Creation Wizard Class

The QControl Creation Wizard Class is open source and the wizard, itself, uses the some Extended QControl Classes to show an example of usage. It inherits from the Dialog Class and uses sibling classes Warning Dialog and Error Dialog. It also uses the QControl Creation Class to actually create the QControl Class and to carry out Pre-check on the New QControl Information.

10.1.2 The QControl Creation Class

The QControl Creation Class is the scripting engine behind the wizard. If the necessary information is passed to it, it can be ran independently of the wizard or a new wizard could be created. The creations is started by giving the correct inputs to and launching the QControl Creator Main Method.

10.1.2.1 The QControl Creator Method

The QControl Creator Main Method starts the creation of the new QControl Class asynchronously. It requires the inputs of:

- Application Reference
- New QControl Information Cluster

The inputs of the StatusUpdate and CreatorComplete User Events are up to the wizard. If the wizard is going to display status of the individual steps during creation, it will need to create a String Typed User Event and pass it to the method. If the wizard wants the errors, if any, as well as notice the creation script is complete, it will need to create a Boolean Typed User Event and pass it to the method.

The output of the method is the CreatorRefnum which is used by the QControl Creation Complete Method.
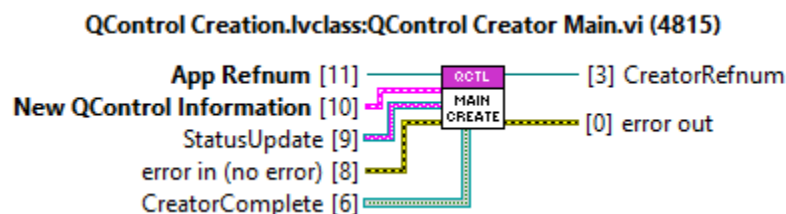

Figure 21 – Connector Pane of the QControl Creator Main Method

10.1.2.2 The QControl Creator Complete Method

In the case that the CreatorComplete User Event is used, the QControl Creator Complete Method must be used in the Event Structure of that user event case. This is the method that retrieves any error the

occurred during the script. The input is the same CreatorRefnum that was the output of the QControl Creator Main Method. The output of this method is the error that occurred, if any.
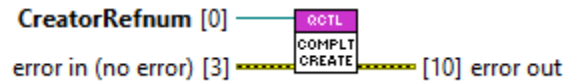

Figure 22 - Connector Pane of the QControl Creator Complete Method

10.1.2.3 Other Public Methods

The Check Methods can be used as a Pre-Check to perform first to sanitize the New QControl Information before running the QControl Creator Main Method.

The Create Class Banner Icon and Create Private Data Control Icon Methods can be reused for any classes but the QControl Creation Wizard uses them to preview the look of the Banner Icon and Private Data Control Icon. The QControl Creation Class uses them to create the icons that are applied to the new Extended QControl Class.

10.1.2.4 Template Class

The Template Class inherits from the Control Class in the QControl Class Hierarchy and is therefore a QControl Class. It contains the methods, controls, and folders necessary for the basic structure that is recreated by the script code. If this class or any of its members (methods or controls) are edited in any way, it might cause the scripts to fail. Any changes are not recommended but if done, it could necessitate changes to the script code. **NOTE: DO THIS AT YOUR OWN RISK.**

11 Support

The QControl Toolkit is distributed with only limited support. Questions can be emailed to:

support@qsoftwareinnovations.com

Questions will attempt to be answered at the convenience of Q Software Innovations personnel. A discussion on the NI Community UI Interest Group will be available for questions or sharing of developed QControls.

 For further help or development Q Software Innovations can be contacted and contracted for work specific to your application.

11.1 License and Disclaimer

©2016 Q Software Innovations, LLC (QSI)
Written by Quentin Alldredge

support@qsoftwareinnovations.com
www.qsoftwareinnovations.com

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Q Software Innovations, LLC. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.