Getting Started with MView

Noah Meltzer

What is MView?

At the highest level, MView is a framework that can be used to interface with virtually any external data source. In short, MView takes care of the overhead involved with running a GUI and device communication so that the end user can concentrate on their data.

Using MView, a variety of control and display elements can be easily configured. These include:

- Buttons
- Numerical/Textual Readouts
- Plots

In addition, some backend features include:

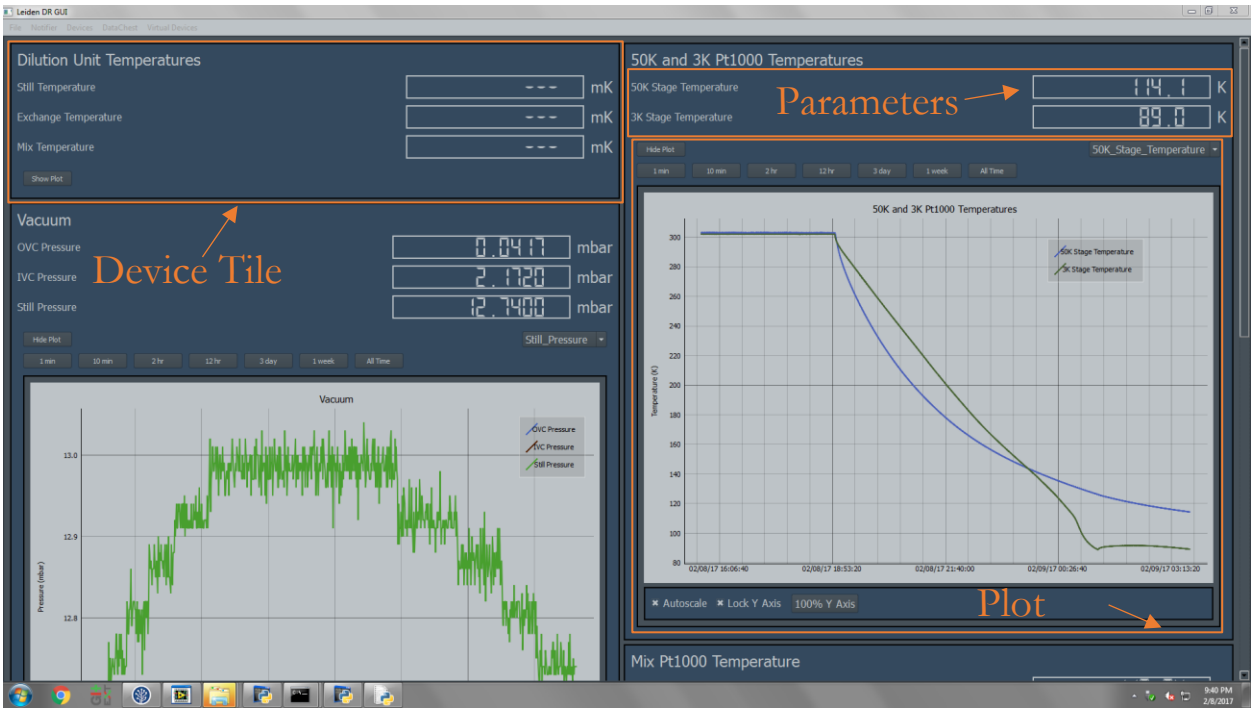
- Datalogging using Datachest
- Email/Text notifications
- Logic

MView accomplishes this by wrapping low-level communication and overhead into a **device driver** that abstracts all sources of data (**devices**) into an **MDevice** object. All MDevice objects implement a common software interface. This allows even the most complex devices to be handled in a simple manner.

With this system in place, MView implements a range of error-checking, graphing, and logging functionality which the end user can use with ease.

While not nearly as powerful as LabVIEW, MView is easy to set up and it works. This makes it ideal for simple monitoring tasks where the massive overhead of LabView is not needed. Additionally MView is also accessible by anyone who knows even the smallest amount of python without the need to learn G.

The MView Interface



MView for Our Purposes

Setting up a simple GUI for LabRad Devices

Setting up a new MView GUI is generally very easy. In this section, we will configure a simple MView project that monitors data from a LabRad device.

In order to create a GUI using MView, we must set up a program that configures our devices and tells MView how to behave. This is done below.

Before You start:

- ***Please refer to the DataChest manual for DataChest setup.***
- ***Please refer to the comments in telecomm.py for telecomm server setup.***

Step 1: Imports

The first thing that must be done is to import the necessary MView libraries.

```
import MGui # Handles all GUI operations. Independent of LabRAD.
from MDevices.Device import Device # This is the device driver that represents a LabRad
server
```

MGui: Handles the overhead of initializing MView
MDevices.Device: All device drivers are stored in the MDevices folder. Device is the device driver that talks to LabRad servers.

Step 2: Initialization

Next, we need to write the class that initializes MView as well as all Devices.

```
class MyGuiClass:
  my_gui = None
  my_devices = []
```

my_gui: Will hold a reference to the MVeiw Gui.
my_devices: Will hold the list of devices.

Step 3: Create a LabRad and Telecomm Server Connection

A connection to LabRad is created so that it can be passed to devices.

```
try:
  # Attempt to establish a labrad connection.
  cxn = labrad.connect()
except:
  # If no connection can be made, abort with an error message.
  print("Please start the LabRAD manager")
  time.sleep(2)
  sys.exit(0)
try:
  # As of writing, there is one class in MView itself that is dependent
  # on LabRad, and it requires the telecomm server to be running.
  # This is subject to change.
  tele = cxn.telecomm server
except:
  # If no connection can be made, abort with an error message.
  print("Please start the telecomm server")
  time.sleep(2)
  sys.exit(1)
```

NOTE: The code in steps 1-3 is the same for all MView GUIs.

Step 4: Initializing Devices

A) Instantiating a new device

We must now initialize the LabRad Devices. Let's create a device that represents a CP2800 compressor.

First, we instantiate a new Device:

```
Compressor = Device("Compressor")
```

This creates a device called "Compressor." Not much else happens until we tell it how to communicate.

Second, we pass it a reference to our connection:

```
Compressor.connection(cxn)
```

Third, we tell it what the name of the server is. The name of the server for the CP2800 is "cp2800_compressor."

```
Compressor.setServerName("cp2800 compressor")
```

B) Adding Buttons

Next, let's add a button that turns off the compressor when we click it. This is done using the MDevice.addButton(label, message, setting, setting arguments) method.

```
MDevice.addButton(label, message, setting, setting arguments)
```

Here is how it should look in the context of this example:

```
Compressor.addButton("Turn Off",
   "You are about to turn the compressor off.",
   "stop", None)
Compressor.addButton("Turn On",
   "You are about to turn the compressor on.",
   "start", None)
```

The first argument sets the button's text. The second argument is the text to be displayed in a warning popup, if no warning is to be displayed, then the second argument should be None. The third argument is the LabRad server setting that is triggered when the button is pushed, and the fourth argument is an array of arguments for the LabRad setting, None if no arguments.

C) Adding Parameters

It is now time to add parameters to the device's tile. This is done with the MDevice.addParameter() method.

```
MDevice.addParameter(Label, LabRad Setting, Arguments, Array Index,
       Preferred Units, Readout Precision)
```

Here is how it should look in the context of this example:

```
Compressor.addParameter("Input Water Temperature",
   "current_temperatures_only", None, 0, 'degC', 1)
Compressor.addParameter("Output Water Temperature",
   "current_temperatures_only", None, 1, 'degC', 1)
Compressor.addParameter("Helium Temperature",
   "current_temperatures_only", None, 2, 'degC', 1)
Compressor.addParameter("Oil Temperature",
   "current_temperatures_only", None, 3, 'degC', 1)
```

The first argument tells MView what to call the new parameter. The following argument is specific to the LabRad device driver, and it tells MView which LabRad setting to call in order to get the value. Next, are the arguments to be passed to the LabRad setting, they are None in this case, because the setting does not take arguments. This LabRad setting returns a list, and the next numbers are the index in the list where the parameter value can be found. Next are the preferred units. MView will try to use the preferred units. Lastly is the precision (number of decimal places) to be displayed.

Step 5: Selecting the Device

Just as with any LabRad device, we must call the 'select_device' command. This is done in the following way:

```
Compressor.selectDeviceCommand("select_device", 0)
```

This selects device 0.'

Step 6: The Plot

To add a plot to our graph, the MDevice.addPlot() can be called.

```
Compressor.addPlot()
```

To set the y-axis label, the following command is used:

```
MDevice.setYLabel(y-axis label, custom units = None)
```

The first argument is the label displayed on the y-axis, and the second is an optional override of the default units. For example, it is a good idea to use this override when the server does give units. In our case, this takes the form of

```
Compressor.setYLabel("Temperature")
```

Step 7: Begin()

The next thing we must do is to tell the device to start. This is done with the MDevice.begin() method.

```
Compressor.begin()
```

Step 8: The Device List

Note that in step 2, we created a class variable called 'my_devices.' This is the list of devices that needs to be passed to MView. This means that we must add our new device to this list of devices.

```
    self.my_devices.append(Compressor)
```

Step 8: Starting MView

We created a my_gui variable. This will hold a reference to the MView GUI so that it does not get garbage-collected.

This is done using the MGui.startGui() method.

```
MGui.startGui(self, devices, title, tele, autostart = True):
```

- **devices:** The device list.
- **title:** The title on the gui.
- **tele:** Reference to the telecom server.
- **autostart:** Allows gui to run when startGui() is called. The purpose of this option will be discussed in a later section.

```
    self.gui = MGui.MGui()
    self.gui.startGui(self.devices, 'Leiden DR GUI',
        tele)
```

Step 9: Calling __init__()

As with any python class, we must call our init method **outside** of the main class.

```
class myGuiClass
...
viewer = myGuiClass()
viewer.__init__()
```

Step 10: Putting it All Together

Here is how our new piece of code should look.

```
import sys
import time
from tendo import singleton

import labrad

from dataChestWrapper import *
import MGui # Handles all GUI operations. Independent of LabRAD.
from MDevices.Device import Device


class nViewer:
  gui = None
  devices = []
  def __init__(self, parent = None):
    # Establish a connection to LabRAD.
    try:
      # Thiss will sys.exit(-1) if other instance is running.
      me = singleton.SingleInstance()
    except:
      print("Multiple instances cannot be running")
      time.sleep(2)
      sys.exit(1)
    try:
      cxn = labrad.connect() # Attempt to establish a labrad connection.
    except:
      # If no connection can be made, abort with an error message.
      print("Please start the LabRAD manager")
      time.sleep(2)
```

```
      sys.exit(0)
    try:
      # As of writing, there is one class in MView itself that is dependent
      # on LabRad, and it requires the telecomm server to be running.
      # This is subject to change.
      tele = cxn.telecomm_server
    except:
      # If no connection can be made, abort with an error message.
      print("Please start the telecomm server")
      time.sleep(2)
      sys.exit(1)

    Compressor = Device("Compressor")
    Compressor.connection(cxn)
    Compressor.setServerName("cp2800 compressor")
    Compressor.addButton("Turn Off",
       "You are about to turn the compressor off.",
       "stop", None)
    Compressor.addButton("Turn On",
       "You are about to turn the compressor on.",
       "start", None)
    Compressor.addParameter("Input Water Temperature",
       "current_temperatures_only", None, 0, 'degC', 1)
    Compressor.addParameter("Output Water Temperature",
       "current_temperatures_only", None, 1, 'degC', 1)
    Compressor.addParameter("Helium Temperature",
       "current_temperatures_only", None, 2, 'degC', 1)
    Compressor.addParameter("Oil Temperature",
       "current_temperatures_only", None, 3, 'degC', 1)
    Compressor.selectDeviceCommand("select_device", 0)
    Compressor.setYLabel("Temperature")
    Compressor.addPlot()
    Compressor.begin()
    self.devices.append(Compressor)

    # Create the gui.
    self.gui = MGui.MGui()
    self.gui.startGui(self.devices, 'Leiden DR GUI',
       tele)


# In Python, the main class's __init__() IS NOT automatically called.
viewer = nViewer()
viewer.__init__()
```

Table of Contents