* 1. **MODAC - MorphOptic Data Acquisition and Control**
  2. Proof of Concept (PoC) System - Summer 2020
  3. *Design Documentation*
  4. (DRAFT)
  5. June 2020
  6. Developed under contract by Jerry Isdale, Maui Institute of Art and Technology ([jerry@mioat.com](mailto:jerry@mioat.com))
  7. Directed by Joe Ritter of Morphoptic

1. MODAC Design documentation is intended for support engineers. It gives an overview of the architecture, design constraints, and details of the hardware and software included in MODAC PoC, circa June/July 2020.
2. *Table of Contents*
3. [1 Intro MODAC](#_5uzgc8z9vxiy)
4. [2 Design Constraints & Goals](#_ktli1nualaa)
5. [3 MODAC Hardware](#_1l3umf182lul)
6. [3.1 Hardware Components](#_bfzu02h4x8hu)
7. [3.2 SD Card Setup](#_85lo8aggnysi)
8. [3.3 Hardware Connections](#_mbmlh420gbei)
9. [3.4 PoC Case Design](#_37bu1576khw5)
10. [3.4.1 Kiln Adjacent Components](#_yv3g8lp10oqy)
11. [3.4.1.1 Heater Relay Switch Box](#_to4hxup2nna9)
12. [3.4.1.2 I2C Sensor Array](#_tjkjl13vi26f)
13. [3.4.2 Main Case](#_vlropaoinwag)
14. [4 Software Architecture](#_s4cwdy3zscb8)
15. [4.1 Overview of MODAC Software Design](#_miczh3w8c09x)
16. [4.2 External Packages Used](#_v4nza4wi85w6)
17. [4.2.1 Trio – a friendly Python library for async concurrency and I/O](#_9glsd2qgnim3)
18. [4.2.2 pyNNG (python Nanomsg Next Gen)](#_hdtzkiqk0xa1)
19. [4.2.3 pyGTK/PyGobject - Gnome (UI) Tool Kit](#_2vukki5ng7m5)
20. [4.2.3.1 Glade GUI Layout Tool](#_bozce8h9ufzt)
21. [4.2.4 MatPlotLib via PyPlot and GTK](#_4o6xqco4shrt)
22. [4.2.5 Thermocouples Reference](#_dmusxgr36n96)
23. [4.2.6 Numpy](#_dmusxgr36n96)
24. [4.2.7 Other Software Modules](#_5sytz8xy7gw9)
25. [4.3 Software Modules](#_wnewfbq05zpj)
26. [4.3.1 Top level MODAC folder](#_o2ztuqqj6chq)
27. [4.3.2 Holds main routines for server, guis and various other test programs](#_3ubseg2bue60)
28. [4.3.2.1 MODAC\_Server:](#_espyo5ivp18i)
29. [4.3.2.2 Client Applications](#_ke9hdcvx0134)
30. [4.3.3 Unit and Other Test Folders](#_o5ip80b5ibcp)
31. [4.3.4 module modac](#_ch0b0jwqj02m)
32. [4.3.4.1 Hardware Support Files](#_e5103ra5icjo)
33. [4.3.5 package kilnControl](#_4omle2x4bfci)
34. [4.3.5.1 Kiln.py](#_nudh22jw6w1x)
35. [4.3.5.1.1 Kiln.KilnControlProcess()](#_7edhqugixw2u)
36. [4.3.5.1.2 Kiln.KilnStep()](#_feadtrnurmdp)
37. [4.3.5.1.3 Kiln.DoSinglePID()](#_b0y0cra3izt0)
38. [4.3.5.1.4 Kiln.DoMultiplePID()](#_owwgso9t9ar1)
39. [4.3.5.1.5 Kiln.NextScriptSegment()](#_h9b1ahx7kaxf)
40. [4.3.5.1.6 Kiln.LoadScriptStep()](#_7otq69b7s4pm)
41. [4.3.6 package modac\_GUI](#_84xbf1di3447)
42. [4.3.6.1 Table/Graph Data Panels](#_2nwqqdbwp02d)
43. [4.4 Software Artifacts (Files)](#_ahqt4ulyfwdi)
44. [4.4.1 Kiln Script](#_h3tnuctl2vk)
45. [4.4.2 Log Files](#_qhqryxxh6nx)
46. [4.4.3 CSV Data](#_v8m9mddw2z28)
47. [4.4.4 JSON Message Log](#_42hqcwsdudqe)
48. [4.4.5 Configuration Files](#_9lwncwvsyqij)

# 1 Intro MODAC

MorpOptic Data Acquisition and Control (MODAC) Proof Of Concept (PoC) is a small linux based computer, with additional Windows 10 PC python UI software, to read and control a glass slumping kiln. This PoC system will be integrated with an existing kiln and additional sensors provided by MorphOptic (and their client). Refer to [MODAC Introduction Document](https://docs.google.com/document/d/1HRw2WPV9y1TPR2d8Q3VKzfa-KQfhsHj5BoFcWUjV3Pk/edit?usp=sharing) for more basic information.

[MODAC User Documentation](https://docs.google.com/document/d/131gk2v6jvdlfa9OxDu13b9-0aTxPXDuq0zz-3A7oEMg/edit?usp=sharing) is in a sibling folder. It contains instructions on how to setup and use the MODAC PoC system for kiln control.

# 2 Design Constraints & Goals

* Modular design for adaptive configuration and unit testing,
* high resolution AD,
* lots of binary control,
* real time interaction,
* some level of scripting behavior,
* PID heater/thermocouple control,
* reasonable UI for monitoring & control.

(copy pasted from Intro) The MODAC system was conceived as a modular system for collecting and controlling laboratory equipment. The Proof of Concept (PoC) project was undertaken to build a basic framework around a small Linux computer with a Graphical User Interface running on either the same machine or a local network attached system.

*Note: Web connection is not desired at this time. System should be air gapped from the Internet, as overall system security of the PoC was not addressed. A wifi connection is desirable for updates, but otherwise should be turned off. The PoC design could be extended to provide web interface in the future.*

One half of the software (MODAC\_Server) controls the hardware directly on a Raspberry Pi, while a second (Client) software package provides the Human Interface. These communicate over an IP Message Passing Protocol (i.e. NNG). That User Interface (Graphical UI Client) application could run on the same rPi, or it could run on a different machine. The message protocol is (very simply) encrypted and uses uncommon ports, providing a modicum of security with hooks to implement more robust protections if later required.

Modules supporting a variety of input/output devices are implemented, as are simple user interfaces for monitoring and controlling those devices. All devices post their data to a common ‘blackboard’ (in memory data) which is shared with the UI Client. Specific application (i.e. Kiln Control) can add their own data, processing and UI.

# 3 MODAC Hardware

MODAC hardware is documented in the hardware [spreadsheet](https://drive.google.com/file/d/14kRG222QJ8bDXm9dK72yMRpR9fs0lM_9/view?usp=sharing). Basically it consists of a Raspberry Pi, power supplies, UI components, a Relay Board for output controls and a variety of input sensors (AD, environment)

## 3.1 Hardware Components

See the spreadsheet for details, links, etc.

The MODAC server consists of:

* [Raspberry Pi 3B+](https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/)
* HAT (Daughter) boards
  + [Pi-EZ Connect Termina](https://www.adafruit.com/product/2711)l Breakout HAT (screw terminal daughter board)
  + [Waveshare 24 bit AD/DA Converter](https://www.amazon.com/Waveshare-Raspberry-AD-Board-High-Precision/dp/B00ZZGDL32) HAT (8 AD, 2 DA) w screw terminals
    - note: the 24 bit Digital/Analog output is not supported in software
    - note: 24bit A/D is not used in latest version as a) its overkill and b) it was noisy, given long wires between kiln and MODAC
    - a previous 24Bit A/D converter board failed and had to be replaced
* I2C Connected:
  + i2c-RJ45 driver pair (run i2c signals over Cat5 cable)
  + 16 bit AD Converter (4 AD, i2c connect)
  + 4 channel k-type thermocouple amplifier
  + i2c Environment Sensor (Temp, Humidity, Pressure)
* Binary Outputs
  + 8 Channel Relay Module
  + 3 Power strips w/relay control (always on, normally on, normally off)
* Leica Disto BlueTooth laser distance sensor
* Power Supplies:
  + 5v 10A dc power supply + screw connector bus
  + 5v 3A dc -usb for rPi
  + 12vdc power supply (output side of Relay Module)
* UI components
  + powered USB hub (low quality, missing power supply)
  + USB keyboard
  + USB mouse
  + Desktop HD0MI Monitor (ok DVI with dvi-hdmi cable)
* Case: upcycled rack mount from old UPS system
* Unused Components
  + 7” touch screen
  + 1.3” OLED screen (i2c)
  + USB-Serial for Pi
  + 4 port ethernet switch)

## 3.2 SD Card Setup

There are two ways to setup an SD card for Modac:

* Rebuild from Linux image and install script: see doc [SD card setup](https://docs.google.com/document/d/1vcEaLtuQ_AqeQSqTlI7zB-rP2MYEZJO3FoMQ5v-v96M/edit?usp=sharing)
* Duplicating existing SD Card (links to generic articles)

Note that the script was created early in the development process. There may be several modules that are not used in the final PoC and there may also be incompatible newer versions in the network archives. The version of Linux used in MODAC is NOT the latest from the Raspberry Pi Foundation. Newer versions of all software modules have NOT been tested

(15Aug2020: rebuilding under latest rPi OS and python3.7)

## 3.3 Hardware Connections

The Raspberry Pi GPIO 40 pin usage is documented in the [pin connection spreadshee](https://docs.google.com/spreadsheets/d/1vba0zdkXcd7s9NBltFB2hNXa_KTWDYvlo2O8q8lNjRY/edit?usp=sharing)t. MODAC uses two HAT (daughter) boards to break these out for other uses. The EZ Connect HAT provides screw terminals and as well as thru-hole solder prototyping area. These are used by MODAC for i2c, relays, and a few other things. Pin connections on the board are documented in the annotated image file:[PDF of EZConnect](https://drive.google.com/file/d/1lXTT9aOx74zH3_xg5xNT9f3d5zS1G0l-/view?usp=sharing) ([draw doc](https://drive.google.com/file/d/1d4x_RBnt6WHVrEK8HJ113lnnc6iiejgY/view?usp=sharing))

We have not (yet?) generated a circuit diagram of the PoC MODAC system (need to learn tool to draw)

Connections for relays, thermocouples, etc.

## 3.4 PoC Case Design

The PoC system divides the majority of hardware into two parts - Kiln Adjacent and Main Case. Additionally there are three relay controlled power strips.

### 3.4.1 Kiln Adjacent Components

There are two sets of components that are adjacent to the kiln: the Relay Switch, and the Sensor Array. This area changed a lot in the last months of the PoC Development and case needs to be revisited.

#### 3.4.1.1 Heater Relay Switch Box

First, and simplest, is a switch box that connects the Kiln Heater Relays to either the original controller or a cable that attaches to the MODAC relay connectors. The box was provided by Morphoptic. The cable to MODAC terminates with a connector on the back of the Main Case.

(document connection to Main Case - picture?)

The Heater Relay Switch Box is adequate, but does need better attachment to kiln structure.

#### 3.4.1.2 I2C Sensor Array

The Sensor Array is a collection of i2c sensors and other hardware. Earlier configuration of MODAC had the 3 thermocouples attached to an amplifier board, 5vdc power and a 4 pin connector to carry the signals back to MODAC 24bit AD HAT. A small metal prototype case was provided by Morphoptic. The components barely fit inside the prototype box. When the 24bit AD use as changed to using the 16bit AD on i2c, the number of components went up dramatically.

A new case is required for to properly house these devices and connections:

* 5vdc power connect
* I2c Differential interface (RJ45 connector, Cat5 cable)
* 4 channel 16 bit AD board (i2c connect)
* 4 channel thermocouple preamp board
* Environment Sensor (ambient temperature, pressure, humidity; i2c)

Two cables connect the Sensor Array to the MODAC Main Case.

* Twisted pair for 5vdc power
* Cat5 cable with RJ45 connectors

### 3.4.2 Main Case

The main case of PoC system is a salvaged rack mount case from an old UPS power system. Its size, cutouts and other features make it about the perfect case for the PoC. It has 6 AC power outlets on the back that could easily distribute power from the breakered AC input, however that was not implemented for PoC. One cutout on the front is nearly perfect size for the 7” touch screen display. Others on the back are near perfect size for the cable connections

A set of cardboard panels were designed and laser cut to fit over the cutouts to hold these items.

physical layout

case panel laser files

# 4 Software Architecture

MODAC is built on Python v3.7+. This and some of the other dependent libraries (pyNNG, Trio) require this version and particular OS support. Windows 10 and current Linux for Raspberry Pi should be adequate. Of the default software editors available on rPi, [Thomny](https://thonny.org/) seems to be the best for simple edit/test cycle. For a more professional development environment, [JetBrains PyCharm](https://www.jetbrains.com/pycharm/) Community Edition works quite well, providing a debugger, help, and many other features. It does run slow on the rPi, often stopping for several seconds to do something. However, the benefits of a professional IDE are worth it. Also a reminder to take a deep breath and relax a bit.

## 4.1 Overview of MODAC Software Design

* Roughly Server/Client
* Multiple threads share a common Data Dictionary
  + Server threads for … Client GUI threads
* Server process interfaces with hardware io, runs kiln scripts
* Client GUI - communication thread in addition to GUI loop
* Other Clients built for network and architecture testing
  + Network logger - subscribes to Server, saving all messages to file. Note: does not log client->server (control) messages
* Common Data Dictionary (moData) with posts from sensors, control loops and messaging. ServerMessage system periodically collects data from moData and sends as messages to clients.

## 4.2 External Packages Used

MODAC relies on a number of publicly available software libraries. The [getModac.sh](https://github.com/morphoptic/modac1/blob/master/setupPi/getModac.sh) script is used to install these from Internet libraries. Examine that file for a full list of dependencies, and note that this script was created early in the development process. There may be several modules that are not used in the final PoC and there may also be incompatible newer versions in the network archives.

### 4.2.1 Trio – a friendly Python library for async concurrency and I/O

* + - * Used in Server and non-gui clients
      * <https://github.com/python-trio/trio>
      * radically simpler than older competitors like [asyncio](https://docs.python.org/3/library/asyncio.html) and [Twisted](https://twistedmatrix.com/), yet just as capable.
      * “nursery” spawns asynchronous tasks, manages scheduling

### 4.2.2 pyNNG (python Nanomsg Next Gen)

MODAC utilizes the Nanomsg Next Gen (NNG) protocols, and specifically the pyNNG implementation, to communicate between the Server and Client applications. NNG is a lightweight, brokerless suite of protocols providing a variety of connection styles. Documentation on PyNNG can be found at <https://pynng.readthedocs.io/en/latest/>. Background on NNG and other “Scalability Protocols” is in the presentation <https://staysail.github.io/nng_presentation/nng_presentation.html> . MODAC uses two types of protocols:

* Publish/Subscribe: server publishes variety of Topics, clients subscribes
  + modac: contents of moData repository; sent periodically (see config)
  + kilnStatus: just the kiln controller status (also in modac)
* Pair 1:1 : server listens for commands, (multiple) client sends
  + ‘Polyamorous’ mode used so single server port can talk to multiple clients.
  + First message is “Hello”
  + Others should be commands (toggle relay, run script)

### 4.2.3 pyGTK/PyGobject - Gnome (UI) Tool Kit

MODAC uses the PyGobject library to provide the Graphical User Interface. [PyGObject](https://pygobject.readthedocs.io/en/latest/) is a Python package which provides bindings for [GObject](https://developer.gnome.org/gobject/stable/) based libraries such as [GTK](https://www.gtk.org/), [GStreamer](https://gstreamer.freedesktop.org/), [WebKitGTK](https://webkitgtk.org/), [GLib](https://developer.gnome.org/glib/stable/), [GIO](https://developer.gnome.org/gio/stable/) and many more. It was chosen for basic simplicity and open license tools.

#### 4.2.3.1 Glade GUI Layout Tool

[Glade](https://glade.gnome.org/) is a RAD tool to enable quick & easy development of user interfaces for the [GTK](http://www.gtk.org/) toolkit and the [GNOME](http://www.gnome.org/) desktop environment. It was chosen for laying out the basics of MODAC client GUI.

### 4.2.4 MatPlotLib via PyPlot and GTK

[MatPlotLib](https://matplotlib.org/3.1.0/index.html) is a widely used python plotting toolkit. MODAC uses the [pyplot implementation under GTK](https://matplotlib.org/3.1.0/gallery/user_interfaces/pylab_with_gtk_sgskip.html)

### 4.2.5 Thermocouples Reference

The [Thermocouples\_Reference](https://pypi.org/project/thermocouples_reference/) module provides calibration data and lookup functions for standard [thermocouples](https://en.wikipedia.org/wiki/Thermocouple) of types **B**, **C**, **D**, **E**, **G**, **J**, **K**, **M**, **N**, **P**, **R**, **S**, **T**, and some less standard types too. MODAC uses the K-Type functions for handling the Kiln’s thermocouples

### 4.2.6 Numpy

[Numpy](http://www.numpy.org/) is the fundamental package for scientific computing with Python. MODAC use it for various array handling functions

### 4.2.7 Other Software Modules

GPIO: adafruit, sparkfun, gpiozero, rpi.gpio

I2C module: i2c-tools

Sensor device specific modules

## 4.3 Software Modules

### 4.3.1 Top level MODAC folder

### 4.3.2 Holds main routines for server, guis and various other test programs

#### 4.3.2.1 MODAC\_Server:

MODAC\_Server.py is the primary application. It handles the data collection, posting and kiln control. It records data in a CSV file by default.

* + - Initialize Data, Network, Hardware
      * multiple asynchronous tasks
        + read hardware

BLE Leica requires special processing

fork pexpect process and get updates back

* + - * + publish data
        + read/dispatch commands

#### 4.3.2.2 Client Applications

Three Client applications are present in the final PoC.

* moGui\_all.py : shows all tab panels
* moGui\_Kiln.py: shows only kilnControl and Binary Output tabs
* modac\_netLogger.py: headless app to log JSON files (not recently tested

ModacGUI

* + - runs on moClient framework (network, Data, message dispatch)
    - Tabbed Notebook Style
    - Tab pages designed in GLADE tool, plus python code
    - easy to add new pages, mix existing into custom GUI
    - tabs get/put data into moData dictionary
    - trigger sending

Basic form of a Client with and without GUI

### 4.3.3 Unit and Other Test Folders

There are several folders of ‘scaffolding’ code created during development. These include a number of “Unit” or module tests. These may or may not function if run from their sub folders, as they may need to include the various modac packages. They may also be out of date with the current packages

* MODAC\_UnitTests
* MODAC\_Phase1Code
* MODAC\_Config: thoughts and experiments for configuring
* Postprocessing: python to process CSV files
* Scripts: copies of startup scripts (copy to desktop)
* setupPi: original scripts to setup Pi and libaries
* testCode: various test python
* Themocouple: module holding thermocouple API
* waveshareADAC: API for waveshare ADAC board

### 4.3.4 module modac

* + moData.py – Dictionary of all data
  + moKeys.py – functions to give keys into data
  + MoHardware.py
    - abstraction from implementation details
    - server initializes this, moHardware initializes rest
      * except Kiln
  + moNetwork.py
  + moClient.py
  + moServer.py
  + moCommand.py
  + moJson.py
  + moLogger.py

#### 4.3.4.1 Hardware Support Files

Each hardware sensor/effector used in Modac has its own python module. These all provide:

* init() : any hardware initialization, last action class update()
* update(): collects sensor data, posts it to moDict using module moKey
* values(), asDict(): methods to get values as arrays and dictionaries
* shutdown(): shutdown any hardware APIs

Modules may also contine internal classes and wrappers on other libraries/APIs.

Some hardware uses a python class, and/or external classes, while other hardware uses top level methods in the file (module)

* Ad16.py: internal class wrapping Adafruit ADS1115 library;
* Ad24.py: ADS1256/WaveshareADAC board [allRawArray(), all0to5Array()]
* binaryOutputs.py
* Enviro.py: wrapper over BME280 i2c environment sensor
* kType.py: reads AD channels and converts them to Temperatures in moData
* leicaDistroAsync.py: leica Distro support

### 4.3.5 package kilnControl

The kilnControl folder contains code that sits on top of basic MODAC and provides scripted control of the Kiln. Files used are:

* Kiln.py: runs control loop in separate Trio Thread
* kilnConfig.py: configuration variables used by kiln
* kilnScript.py: classes kilnScript, kilnScriptSegment; loading/saving/creating JSON of script for network
* kilnState.py: Enums for KilnState and KilnScriptState, default moData
* pidController.py: the core PID controller used

#### 4.3.5.1 Kiln.py

Kiln.py is the core of the kilnControl package. When initialized, it creates a separate Trio thread (from the Nursery), with its own control loop. Current implementation of modacServer starts this, but it could also delay and wait for a Client to send a run command. Note this is a different Run from the RunScript command - this one runs the core KilnControl loop.

*More details on Kiln process could be a whole section*

The kiln file has several top level methods, defines a Kiln class and holds a singleton of that class (kilnInstance). Most of the top level methods are used to control the Kiln Thread and the running of a Script. The kiln thread is started using the Trio Nursery to run the Kiln.kilnControlProcess() method.

##### 4.3.5.1.1 Kiln.KilnControlProcess()

KilnControlProcess() holds the main loop for the kiln thread (Trio). It is basically a forever-loop that does one KilnStep(), publishes status and goes back to sleep for the currently defined “sleepThisStep” time. That sleep value defaults to the “idleStateTimeStep” defined in kilnConfig.py (3Sec). It can be altered by the current ScriptStep to define the PID test/control period. Note that the kiln will effectively not respond to commands (other than emergencyOff) while the control thread is sleeping. This is one of the main contributors to the client/server latency. However, a short time step can cause the PID loop to malfunction. The Idle time is very short (3sec) to make the loop responsive to Start commands. During a script run, the step time should be 30sec or longer (GUI defaults to 30)

##### 4.3.5.1.2 Kiln.KilnStep()

Each iteration of the KilnControlProcess() does one call to KilnStep(), the heart of the KIln State Machine. Regardless of state, KilnStep() updates its Time and Sensor data. Then it checks some safety conditions (eStop, etc). If the KilnState is Idle, it then returns and the control process sleeps. If the KilnState is RunningScript, kilnStep() will examine its current ScriptSegment for action.

ScriptSegment activity is controlled by the secondary State variable “kiln.scriptState”. The initial state of an active scriptSegment is “Heating”, for which the kilnStep() will check for various end conditions (temperature/distance targets reached, maxTime exceeded), which would switch either to “Holding” state (target temp reached) or NextScriptSegment(). Both Heating and Holding will invoke the PID function, giving it the target and current temperatures. The result of this function controls whether the heaters are turned on/off. The kilnStep() sends the appropriate commands to moHardware to turn the corresponding heaters on/off.

##### 4.3.5.1.3 Kiln.DoSinglePID()

This method looks at the current average kilnTemperature (kilnTemps[0]) and uses a classic PID function to determine if the heaters should be on or off, setting the commandedHeaterStates appropriately.

##### 4.3.5.1.4 Kiln.DoMultiplePID()

This method is commented out for now. It is intended for when each thermocouple is controlling a single heater element

##### 4.3.5.1.5 Kiln.NextScriptSegment()

This method is called when the current script segment reaches a termination condition (time, temperature, distance). It checks if the script is over or loads the next step.

##### 4.3.5.1.6 Kiln.LoadScriptStep()

This method copies the ScriptSegment values to the control loop active values. If the segment is commanding any of the Power Outlets (12v, exhaust fan, support fan), the method will send appropriate commands to moHardware

### 4.3.6 package modac\_GUI

The modac\_GUI folder contains files used for building the GUIs. GLADE files used with graphical layout tool. A corresponding python file (or files) will load these XML descriptions and create active instances of the panels. Not all panels have Glade interface definitions. Many panels use a common Table/Graph form.

Each panel has an object named ‘Panel’, which is accessed by the Client GUI and put into a Tab.

The \_\_init\_\_() method uses Gtk.Builder.new\_from\_file(FILENAME) to load the GLADE description and return an instance of a Builder. The builder can then be queried to get references to named objects in the UI.

The first step of a panel is to builder.connect\_signals(), which connects signals from the UI to call back functions in the python module. Then it creates a self.box, to which it adds the Panel.

The init() module will also update any visibility and functionality that is easier to do in code than in the Glade Tool.

update(): generally calls getData(), which updates the UI from current data in moData Dictionary.

Details of each UI page are best understood by Reading The Code

#### 4.3.6.1 Table/Graph Data Panels

The Table/Graph Panel is a common form used on several MODAC PoC Panels. These are built in code rather than using a Glade Definition file. The panel has is a 2 part vertical box layout. The upper box contains a Table of values (date/time in first column), and a Graph of the selected datum below. These are all done off the same pattern code, but it is replicated in easy python module, so there may be differences. Most Table/Graph Panels select the actively graph’d datum by clicking on the column header. The date/time header was inactive. The KType panel (?) was slightly different as clicking on the Date/Time Column header will graph all 3 kType values.

The table is created using Gtk.ListStore, Gtk.TreeView and Gtk.TreeViewColumn held in a GtkViewport() and Gtk ScrolledWindow().

The graph is created using MatPlotLib - Figure, FigureCanvas, and and a lower scroll window.

The update() function calls two methods getData() and updatePlot(). When getData() is called, the current values for date/time and column data are retrieved from moData and appended to the ListStore. The oldest row will be removed from ListStore when its plotWidth limit is reached. The function updatePlot() (calls plotOne()) copies the column data to a Numpy array and MatPlotLib.and plots a line graph in the Graph.

Note these panels may use a lot of memory inefficiently, especially as they are constantly recreating two separate arrays - one for the table and a second for the current graph data. A better mechanism may be needed if these are heavily used. The AD24 panel with its 8 columns seemed to have some memory issues on longer data runs.

## 4.4 Software Artifacts (Files)

MODAC has several File artifacts. Kiln Scripts are used to step the system through a series of actions. Log files collect messages useful for debugging. CSV data files record sensor values. Configuration files can change how MODAC operates.

### **4.4.1 Kiln Script**

Kiln Script files are JSON syntax file holding a series of Script Segments/Steps

* name, description fields (auto filled in gui)
* Simulation Flag (allows testing MODAC logic)
* Series of Script Segments:
  + segment sequence number (index)
  + Target Temperature (degC)
  + Target Displacement (slump, mm)
  + Hold Time (after reach temperature)
  + Exhaust on/off
  + Pump on/off
  + 12v Heater Power on/off

[Here is an example Kiln Script](https://drive.google.com/file/d/1Qv0_Vp6CDvFj2TIiNejmkts55iJPkkxW/view?usp=sharing), used for a test run in early June 2020

### **4.4.2 Log Files**

Log messages are written to files by both server and client into folder “logs”

Messages are produced using the Python Logging framework, with various levels (Info, Debug, Warning, Error). They are used liberally in the code base to provide some level of execution tracing.

* file name has the form: prefixYYYMMDD\_HHMMSS.log
* rolls over and renames older files to log.1, log.2 … etc
* each file has max size of 1Mb,
* highest numbered log file is oldest.
* rate of messages and verbosity can be changed by editing code
* server: modac20200207\_160720.log.1
* client: modacGUI20200207\_160757.log.5

### **4.4.3 CSV Data**

CSV Data files are written by server and optionally gui

* server always on (saved in folder “logs”)
* gui only on User start (user selected location/name)
* modacServerData\_20200207\_1607.csv
* columns determined by code in moCSV.py and moData.py
* moData defines the column names and collects row contents as an array
* moCSV controls file access and rate of row addition (1/min is default)

### **4.4.4 JSON Message Log**

JSON messages written by server when compiled to use (off generally)

* each record is contents of moData dictionary
* by default this is turned off.
* a Logger Client was built but not maintained during development
* File name format: modacServer\_20200207\_1607.json

### **4.4.5 Configuration Files**

There are a lot of configuration variables for the Proof of Concept system. These are kept in python source files. A number of common ones are grouped into the

moKeys.py holds inline functions that return string constants for use as dictionary keys

configuration inline in hardware module support code

kiln control has a [kilnConfig.py](https://github.com/morphoptic/modac1/blob/master/kilnControl/kilnConfig.py) to hold variables shared with other code modules

# 5 Client Machine Setup

The TK GUI client runs on both Linux and Win10. Linux version could be setup using basically the same image as the server. Setting up the Windows 10 machine can be done two ways. The longer method is to install correct python, libraries, and modac code. Shorter method is to create a stand alone installation using a previously setup Win10 machine.