DOCUMENTATION FOR THE DYNAGUI PACKAGE

B. E. Bolling*

Abstract

DynaGUI stands for Dynamic Graphical User Interface and is a method to construct temporary, permanent and/or a set of GUI:s for users in a simple and fast manner. Developed at MAX IV Laboratory[1] in Lund, Sweden, the tool has been designed to fit operational and diagnostics purposes of this synchrotron lightsource.

Different devices can have different attributes, depending on what type of device it is. For example, the Beam Position Monitors (BPM:s) of a particle accelerator reveal information about the beam position in the horizontal and vertical position, whilst magnets' power supplies both reads and can set the current setpoint of the magnet.

The BPM:s' device servers do, however, not only reveal information about the beam itself but also contain information of how their data should be treated, such as if they may generate an interlock (InterlockEnabled) or if they should have the Automatic Gain Control enabled or disabled. These signals are handled as true or false flags, meaning that a simple GUI can be constructed to read and also control the true or false-flags (see DynaGUI TF). For the case of having a GUI that reads numerical values, each device needs two fields: A field with the domain address for the device and another with the numerical value of the defined attribute (see DynaGUI NV).

BACKGROUND

At large research facilities and industrial complexes, there is a need of control system user interfaces. However, modern facilities also require continuous upgrading, maintenance, and development, which means that also the control systems need to be upgraded. In order to simplify the construction of control systems and diagnostics, I created the DynaGUI (Dynamic Graphical User Interface) package. The main idea of this package is to get rid of the middle-hand coding needed between hardware and the user by supplying the user with a simple GUI toolkit for generating diagnostics- and control-system GUI:s in accordance with any user's need.

In order to further enhance the user-friendliness of this package, I have designed a simple system for configuration files, described in the first section. In the four following sections, the system requirements is defined followed by the sections for the three DynaGUI package tools (DynaGUI TF (true/false controllers), DynaGUI Alarms (a diagnostics system to continuously monitor a set of attributes such that their values do not go above or below a user-specified limit) and DynaGUI NV (a system for observing attributes' values and a set of realtime plotting methods)). In the end of the documentation, the Python codes have been attached for each DynaGUI package tool.

* benjaminbolling@icloud.com

SYSTEM REQUIREMENTS

In this current version, the PyTango module is imported and used for monitoring and controlling the various devices since, at MAX IV Laboratory, the Tango Control System[2] is used for setting up and controlling device servers and clients (with PyTango being a python module that exposes to Python the complete Tango C++ API). The next control system to embed in the DynaGUI package will be EPICS.

The DynaGUI has been properly and successfully tested and deployed on two different Unix systems (CentOS 7 and Mac OS X). Further requirements are Python 2.7++ or Python 3.x, NumPy, Taurus[3], PyQtGraph, MatPlotLib and PyQt4 or PyQt5[4].

DYNAGUI PACKAGE CONFIGURATION FILES

A DynaGUI configuration file contains an identifier, which is: *IamaDynaGUIfile*. This must be the first row of any valid configuration file. A configuration file contains the device list, attributes list and number of rows, separated by the following key: *IamYourSeparator*. In total, a valid DynaGUI configuration file should contain 3 separators, separating the texts of the file into the following 4 sections: The DynaGUI identifier key, the list of devices, the list of attributes and the number of rows. An example of a valid DynaGUI configuration file can be seen in Figure 1. If a file is impossible to load, it can be two issues: Invalid number of separators or an invalid identifier key, which in any case will be printed in the DynaGUI's status bar.

```
 1 IamaDynaGUIfile
 2 ##IamYourSeparator##
 3 r1-101/dia/bpm-02
 4 i-s01b/dia/bpl-01
 5 i-s04b/dia/bpl-01
 6 r3-301l/dia/bpm-01
 7 i-s19b/dia/bpl-01
 8 r200
 9 r1-109/dia/bpm-01
10 r1-109/dia/bpm-02
11 r1-109/dia/bpm-03
12 i-s01a/vac/vgmb-01
13 i-s03a/vac/vgmb-01
14 r3-a110711cab08/mag/pspi-02
15 ##IamYourSeparator##
16 AGCEnabled
17 ADCEnabled
18 EnableADC
19 InterlockEnabled
20 Interlock_Enabled
21 Attribute1
22 Attribute2
23 Attribute3
24 StatusOpen
25 StatusClosed
26 OutputOn
27 ##IamYourSeparator##
28 4
```

Figure 1: An example of a valid DynaGUI configuration file, with an added non-existing device called 'r200' at row 8.

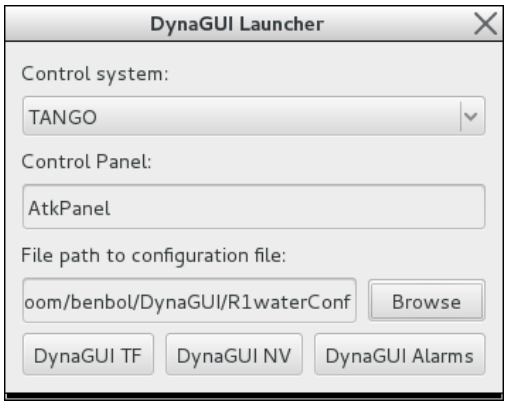

Figure 2: The layout of the DynaGUI Launcher.

DYNAGUI

The DynaGUI (not DynaGUI Alarms) layout is designed for having as high level of simplicity as possible. In the first row is a combobox with the list of attributes defined. Below the combobox is a button called 'Edit DynaGUI', which opens a window for configuring the DynaGUI (see the next section for how to configure a DynaGUI control panel).

Below the 'Edit DynaGUI'-button is the dynamic field, in which a dynamic control panel is generated. Below the dynamic control panel field is the DynaGUI status bar, showing the last action carried out and error messages. Below the status bar is the load and save buttons, which also have a tool-tip function showing the last loaded or saved file (in the current session).

The DynaGUI control panel does not overload a network as it has no continuous polling, but a future version might have this implemented combined with a system that monitors the network load from continuous polling together with the computer's RAM and CPU load to prevent any overload. Currently, however, the DynaGUI control panel updates the states when a button is clicked or another attribute is selected, or when the update statuses is clicked. The exception is DynaGUI Alarms (see DynaGUI Alarms section), which does have continuous polling.

A configuration can be loaded by launching a DynaGUI script in a terminal with the configuration file's filepath as input or by launching a DynaGUI script without any input and then loading the file via a user interface (press load to open up a filebrowser and find the desired configuration file). Saving a DynaGUI configuration is achieved by pressing the Save button, browsing into the destination folder and typing its desired name. Keyboard shortcuts ctrl+o and ctrl+s have been implemented for loading and saving a file.

The simplest method to launch DynaGUI is via its launcher (Launcher.py), in which the user can either directly define the path to the configuration file or browse to it (with blank meaning it will load with a predefined setup).

The DynaGUI launcher can then launch any application from the DynaGUI package. With the launcher, the user can also select which control system is to be used with which control panel (see Figure 2).

DYNAGUI TF

This GUI is dynamical in the sense that the user can insert the name of any device's servers and attributes that, in current state, are True or False. The GUI then builds itself up by creating buttons for each device server that will display the boolean of the in the combobox selected attribute (with TF being true or false). First, it will try to connect to the device's server entered, and then read the in the combobox selected attribute and paint the button's background color in accordance with Table 1 below. See Figure 3 for the layout of the DynaGUI TF.

Table 1: DynaGUI TF colorscheme

Color:	**Meaning for each device:**
Limegreen	True
Red	False
Fuchsia	Attribute not existing
Maroon	Device not possible to connect to

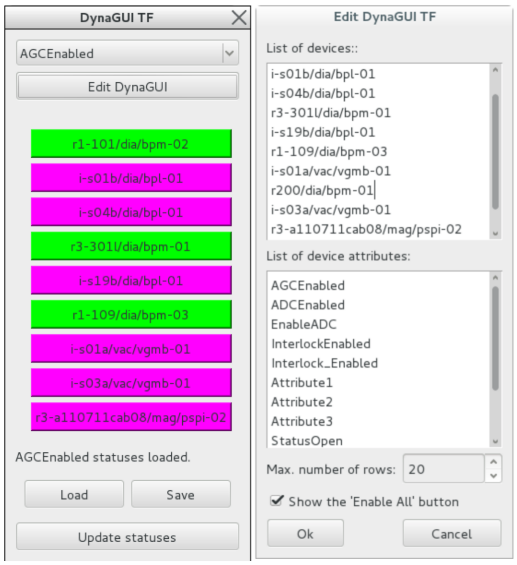

Figure 3: The DynaGUI TF layout. To the left, the main layout. To the right, the layout of the Edit DynaGUI TF.

To edit a list, press the 'Edit DynaGUI' button. Each devices' server domains or attributes are separated by new rows. The number of rows can also be defined for the GUI layout. The Enable all button can be removed or added by unticking the checkbox [Show the 'Enable All' button] (see Figure 1 (right)). Then use the new settings by pressing [Ok] or dismiss by pressing [Cancel], with [Ok] resulting in the dynamic control panel field restructuring itself accordingly.

DYNAGUI NV

DynaGUI NV (numerical values) is a dynamic viewer for not only numerical value attributes, but also string attributes. It is the most advance and powerful GUI creation tool in the DynaGUI package and works in similar ways as its sibling, DynaGUI TF. The differences are that it cannot be used to control the devices (for safety reasons) and that it contains 2 columns for each device. In the first column, the device server and its status can be found in accordance with the colorscheme described below.

Clicking on a device's button launches the controller for this device, with the exception if its status color is maroon (i.e. device disconnected). For Tango devices, the standard control panel is AtkPanel. The second column contains the numerical or string value of the selected attribute, or just a minus-sign if the attribute does not exist for that device.

Table 2: DynaGUI NV colorscheme

Color:	**Meaning for each device:**
Limegreen	Attribute valid and read
Fuchsia	Attribute not existing
Maroon	Device not possible to connect to

To get all available attributes for the loaded devices, one can click on the [Get All Attributes]-button. Its function checks for all the devices defined in the DynaGUI device list and logs each unique device type such that e.g. the attributes of a BPM-01 device is not loaded more than once, even if it is defined for more than one section. It also ensures that there are no attribute duplicates in the list. The list can then be edited, but note that the previous list of attributes will be overwritten.

The DynaGUI NV can also initialize live plotting via TaurusTrends. Pressing the 1D Plot button results in that the attribute selected will be plotted for the devices that have this attribute. In order to simplify with the naming of different curve lines, the TaurusTrend's window title will be the name of the plotted attribute whilst the curve lines' names is their respective server address. See Figure 5.

The 2D plotting tool launches a spectrogram that shows a colormap of how the attribute's value changes over time for all the devices of which that attribute is valid. The spectrogram tool allows the user to view the real values of that attribute or to store current values of all attributes and plot the difference between real and reference values, enabling the user to see how the values evolve with time. As an example, in Figure 4 below, the evolution of the read values of the horizontal beam position read by the BPM:s shown in green in Figure 5 is plotted as a function of time.

In order to see how the beam position looks like at a specific instance of time, one can move the yellow line in the spectrogram to the time of interest and then pressing the Plot Trace button then shows a plot of the beam position as seen by the defined and valid BPM:s (as shown on the left plot of Figure 4).

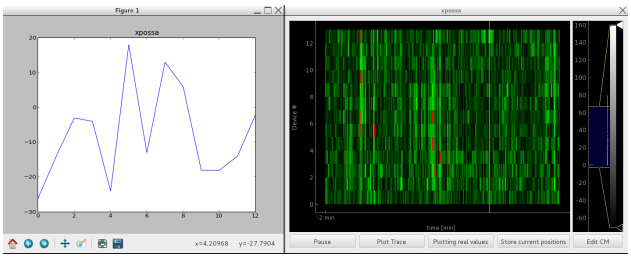

Figure 4: A 2D spectrogram showing how the xpossa (read values of the horizontal beam position) evolves with time for a set of BPM:s, seen in green in Figure 5. To the right is the live-running 2D spectrogram whilst at the left is a trace plotted at the time that the yellow marker is placed at.

The DynaGUI NV's Spectrogram GUI

- Plotting can be paused.
- The histogram can be used to define the Spectrogram's colormap intensities.
- Store current positions sets the current values in the Spectrogram as reference values, which in the 'Plotting vs stored' mode is substracted from the real value.
- Plotting mode can be switched by pressing the plotting mode button, which always shows the current mode ('Plotting vs stored' or 'Plotting real values'). In 'Plotting real values' mode, the values are extracted from the device and plotted in the Spectrogram in accordance with its colormap and histogram. In 'Plotting vs stored', the same is done as in the other mode but with the reference values substracted. If no reference values exist, a reference will be taken when switching mode.
- The colormap can be edited using two methods: Either by right-clicking on the histogram markers (note that one can add more markers) and using the built-in PyQtGraph method for changing the colormap, or by pressing the 'Edit CM' button. The user then has the option to select if one wants to edit CM1 (background color), CM2 ('middle-layer color') and CM3 ('high-intensity color'). The colors are picked using the alpha-RGB color scheme.

DynaGUI NV applied on water cooling systems

A crucial element for large particle accelerators is its cooling system e.g. via a cold water inlet. In order to observe how water flows and water temperatures evolve with time, a control panel was configured and set up for the water flow gauges (FGE) leading to the magnets and temperature sensors (TSE) for the water flowing through different sections of the 3 GeV storage ring (R3). Thus, all these devices were loaded and the two attributes Temperature and Flow have been defined, where Temperature is valid for TSE:s and Flow is valid for FGE:s. The resulting control system GUI can be seen in Figure 6.

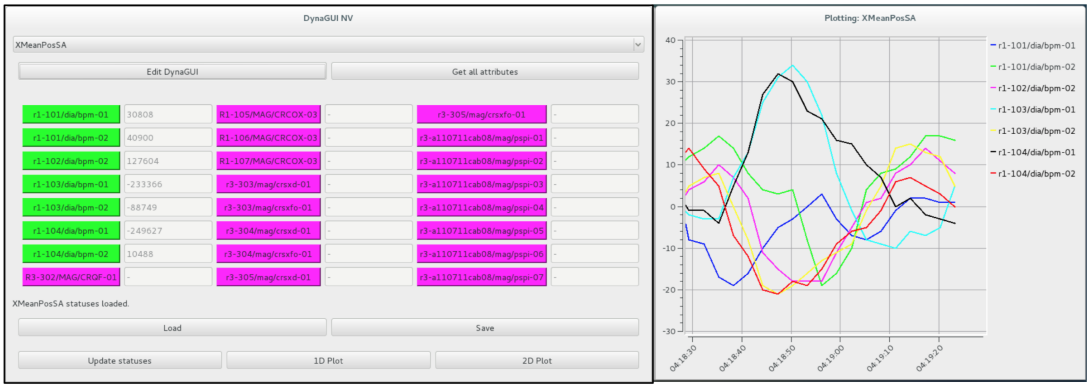

Figure 5: A dynamic control panel of DynaGUI has been configured (left), from which a TaurusTrend was launched for the devices of which the XMeanPosSA attribute is valid (see the green buttons which represent beam position monitors).

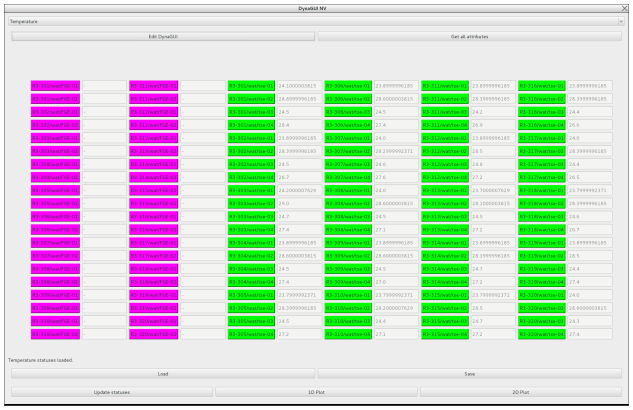

Figure 6: A dynamic control panel of DynaGUI has been configured for the water temperature and flow monitors in the MAX IV Laboratory 3 GeV storage ring.

The DynaGUI-generated water control panel in Figure 6 shows all TSE:s in green and all FGE:s in magenta for the temperature attribute, and shows last updated values. By launching a 2D plot and plotting the changes in water temperature, it becomes easy to spot abnormalities. As an example, in Figure 7, the changes in temperatures and in water flows has been plotted against time for all TSE:s and FGE:s, resp. In Figure 8, the trace has been plotted at the time selected by the yellow line in Figure 7 for both the relative and real water temperatures, useful for investigating e.g. abnormalities at specific points in time.

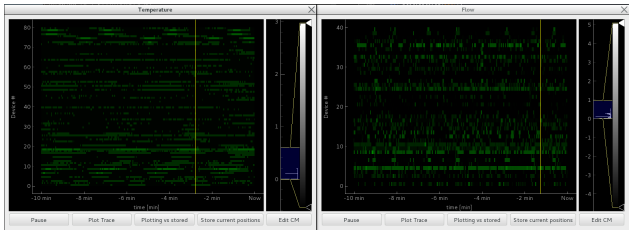

Figure 7: 2D spectrograms showing how the water temperatures and water flows for the MAX IV Laboratory's 3 GeV storage ring's magnets change with time.

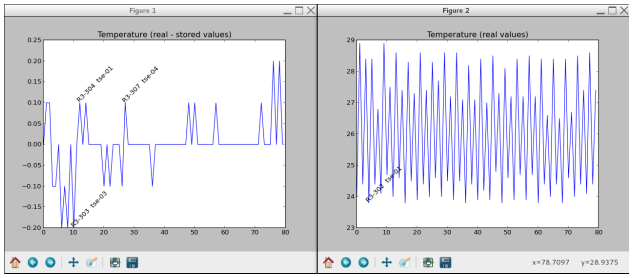

Figure 8: The relative and real water temperatures plotted for each device at the point in time selected by the yellow line in Figure 7.

DYNAGUI ALARMS

The Dynamic Alarms GUI works in similar ways as its predecessors but has been designed to monitor numerical values and notify user(s) if a condition is not fulfilled. The home screen of DynaGUI Alarms can be seen on the left in Figure 9. To set up some alarms for some signals, one can press Edit DynaGUI to open an edit-panel. In the left window, the user has to define the list of devices' server domains and signals to monitor in the left editbox, as shown in Figure 9 on the right. The user also has to define the number of rows for the control panel and the alarms timer, which is the time between each sweep.

To begin monitoring, press activate (Figure 9). In the first column are all alarms' descriptions. In the second column are the current values. In the third column is a combobox with which the user can select if the attribute should be smaller or larger than the alarm limit. The way the gap points is the way it should be, otherwise – the alarm will go off. The checkboxes are used to select which alarms to monitor.

If an alarm goes off, it will read which devices are in alarm and what type of alarm (temperature, pressure, horizontal beam position, etc.) goes off.

A DynaGUI Alarms configuration file is slightly different, with its identifier being *IamaDynaGUIalarmFile* and with the configuration file containing the full path to the attribute to monitor, a description of the monitored attributes, the limits for the alarms and number of rows for the user interface, separated with the same key as for a regular DynaGUI configuration file. An example of a valid DynAlamrsGUI configuration file can be seen in Figure 10.

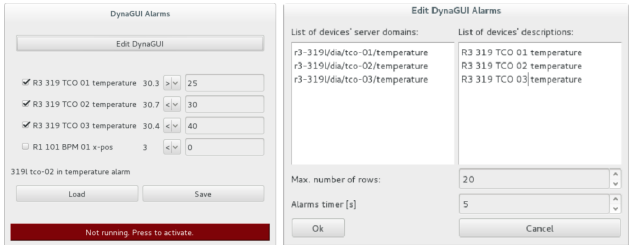

Figure 9: The DynaGUI Alarms layout to the left and a DynaGUI Alarms edit panel to the right.

```
IamaDynaGUIalarmFile
##IamYourSeparator##
r3-319l/dia/tco-01/temperature
r3-319l/dia/tco-02/temperature
r1-101/dia/bpm-01/xmeanpossa
r1-101/dia/bpm-02/xmeanpossa
r1-103/dia/bpm-01/ymeanpossa
##IamYourSeparator##
R3 319 TCO 01 temperature
R3 319 TCO 02 temperature
R1 101 BPM 01 x-pos
R1 101 BPM 02 x-pos
R1 102 BPM 02 y-pos
##IamYourSeparator##
36.0
38.0
40.0
##IamYourSeparator##
3
```

Figure 10: An example of a valid DynaGUI Alarms configuration file.

THE HEART OF DYNAGUI

The main structure of the DynaGUI algorithm is its generic control panel constructor, which constructs a control panel from a list (*self.devlist*) and has in a simple manner been inserted below.

```
    rowc = -1
    colc = 0
    for index in self.devlist:
     rowc += 1
     button = QtGui.QPushButton(index,self.groupBox)
     self.sublayout.addWidget(button,rowc,colc,1,1)
     self.groupBox.setStyleSheet("text-align:center")
     if rowc == self.Nrows - 1:
      rowc = -1
      colc += 1
    self.bGr = QtGui.QButtonGroup(self)
    self.bGr.buttonClicked.connect(self.hClick)
    for button in self.groupBox.findChildren(QtGui.
     QPushButton):
     if self.bGr.id(button) < 0:
      self.bGr.addButton(button)

```

In the code, the layout for the dynamic control panel has been defined as *sublayout* which contains only the dynamic buttongroup, which is the container and eventhandler for all dynamically constructed PyQt objects (pushbuttons, texteditis, etc.). In this version of DynaGUI, the dynamic buttons' identifiers is the label of each individual button, pointing to different server addresses.

FUTURE DEVELOPMENT PLANS

Future developments include having a method to import different device types using wildcards, to embed the EPICS control system, and include plugin options for using the package with any controls system. Another future development plan is to have different object labels and identifiers, with the identifier containing the server address for the device it points to and hence enabling user-defined labels not being the same as the server address.

ACKNOWLEDGEMENTS

The author recognizes and wants to thank Jonas Petersson and Robin Svärd, Accelerator Operators at the MAX IV Laboratory, for developing the original 2D Spectrogram Application for monitoring transverse beam position via Beam Position Monitors at the MAX IV Laboratory storage rings. The author used their codes as inspiration for developing a generic algorithm which can initialize a 2D spectrogram for any given list of devices with attributes. The author also wants to thank Bernhard Meirose, also an Accelerator Operator at the MAX IV Laboratory, for giving the inspiration and idea to creating this package by giving the suggestion to me to code a GUI with in-line commentary such that other Accelerator Operators may use it to construct other control systems as well.

LICENSING

The author supports open-source software. This package may be freely developed further as long as its derivatives are also open-source and developed under different GitHub branches. The package may be employed anywhere, with a direct gratitude from the author if users would notify the author where the package is used (only for statistics).

REFERENCES

[1] P.F. Tavares, S.C., Leemann, M. Sjöström, Å. Andersson, *The MAX IV storage ring project*, Journal of Synchrotron Radiation, 21 (5): 862–77, 2014, doi:10.1107/S1600577514011503.

[2] J-M. Chaize, A. Götz, W-D. Klotz, J. Meyer, M. Perez, E.Taurel, *TANGO - an object oriented control system based on CORBA*, ICALEPCS 99, 1999, Trieste, Italy.

[3] Taurus Python Module, http://taurus-scada.org.

[4] PyQt packages, Riverbank Computing Limited.