The DynaGUI package : Documentation

B. E. Bolling, 2020-05-31

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

Different devices can have different attributes, depending on what type of device it is. For example, the Beam Position Monitors (BPMs) 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 BPMs' 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).

The package has then evolved to have the ability to analyse data from any file containing plot data and also live-streamed data from other sources, such as the stock market using Pandas[5].

I. 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, the DynaGUI (Dynamic Graphical User Interface) package was created. 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 yet powerful GUI toolkit for generating diagnostics- and control-system GUIs in accordance with any user's need.

In order to enhance the user-friendliness of this package, a launcher has been created which automatically checks such that all essential packages has been installed, followed by packages needed for the Tango control system[2], EPICS control system[4], and the Finance analysis package using Pandas[5]. See README.md for more information on the pre-requirements and on how to install the package.

In order to further enhance the user-friendliness of this package, a simple system for configuration files has been designed and is described in section III. 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 real-time plotting methods)). In the end of the documentation, the constructed Python and PyQt[3] algorithm for creating dynamic GUIs has been attached.

II. Licensing

The author supports open-source software. Therefore, the full DynaGUI project package is available via its GitHub page at https://github.com/benjaminbolling/DynaGUI. It may be redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

III. DynaGUI Package Overview

The user is first met by the Launcher screen, depicted in Figure 1 below. The Launcher screen contains buttons which launch the different DynaGUI applications: NV, TF, and Alarms, for each DynaGUI package 'Data Viewer', 'Epics'[4], 'Tango'[2], 'Finance' and 'Random'. Upon launching, there is a script that checks if the essential Python packages are included. If not, the terminal will output "Failed to import basic packages". If successful, however, the Launcher will check which DynaGUI packages' Python package requirements are fulfilled and which are not. The applications and packages which are not fulfilled will not be launchable from the Launcher screen by having their launch-buttons disabled. Each disabled button has a tooltip which gives the user information on why it is disabled.

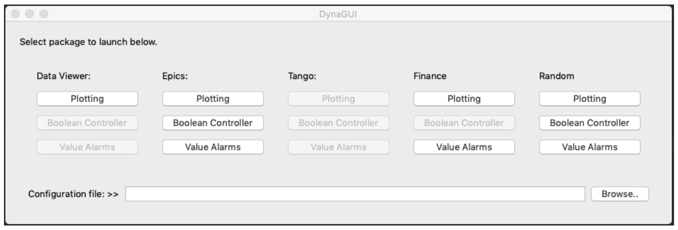


Figure 1. The layout of the DynaGUI Launcher with the file path to a configuration file inserted in and ready to be loaded. See Figure 6 to the left for the configuration file loaded with DynaGUI NV.

Each DynaGUI application's 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 application.

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 or the last error message (whichever was last). 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 will have this implemented as an option 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 pressing 'Load' to open up a filebrowser. 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.

IV. DynaGUI Applications

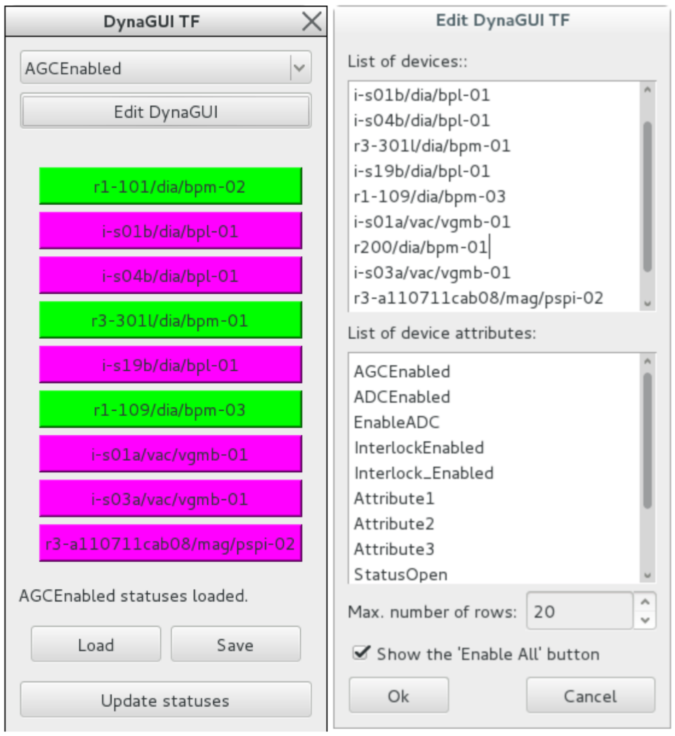
There are three different DynaGUI applications:
DynaGUI TF: A boolean controller application for generating tables with buttons that are colored green or red, symbolising true and false.
DynaGUI Alarms: A live-stream data monitor which compares real-time data scalar values with user-defined values and sounds an alarm if a condition is false.
DynaGUI NV: An application for analysing historical data and/or monitoring real-time scalar values and waveforms from data streams. A powerful tool as it has a 1D and a 2D plotting tool with the capability of e.g. plotting user-defined functions of both real-time data streams and historical data.

A. *DynaGUI TF*

This GUI is dynamical in the sense that the user can insert the name of any device's servers and attributes that 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 I below. See Figure 2 for the layout of the DynaGUI TF.

Table I
DynaGUI TF colorscheme

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



DynaGUI TF

AGCEnabled

Edit DynaGUI

r1-101/dia/bpm-02
i-s01b/dia/bpl-01
i-s04b/dia/bpl-01
r3-301l/dia/bpm-01
i-s19b/dia/bpl-01
r1-109/dia/bpm-03
i-s01a/vac/vgmb-01
i-s03a/vac/vgmb-01
r3-a110711cab08/mag/pspi-02

AGCEnabled statuses loaded.

Load Save

Update statuses

Edit DynaGUI TF

List of devices::

i-s01b/dia/bpl-01
i-s04b/dia/bpl-01
r3-301l/dia/bpm-01
i-s19b/dia/bpl-01
r1-109/dia/bpm-03
i-s01a/vac/vgmb-01
r200/dia/bpm-01
i-s03a/vac/vgmb-01
r3-a110711cab08/mag/pspi-02

List of device attributes:

AGCEnabled
ADCEnabled
EnableADC
InterlockEnabled
Interlock_Enabled
Attribute1
Attribute2
Attribute3
StatusOpen

Max. number of rows: 20

✓ Show the 'Enable All' button

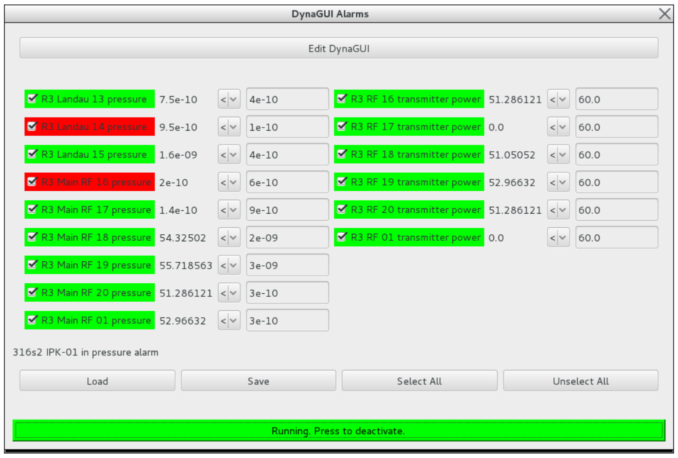
Ok Cancel

Figure 2. The DynaGUI TF layout. The main layout of the DynaGUI TF and its edit screen can be seen to the left and to the right, respectively.

To edit the list of devices and attributes, press the 'Edit DynaGUI' button. Each devices' server domains and attributes are separated by new rows, and the number of rows defines how many rows is the maximum for each column in the GUI. 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.

B. 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 in Figure 3.



DynaGUI Alarms

Edit DynaGUI

✓ R3 Landau 13 pressure	7.5e-10	<	4e-10	✓ R3 RF 16 transmitter power	51.286121	<	60.0
✓ R3 Landau 14 pressure	9.5e-10	<	1e-10	✓ R3 RF 17 transmitter power	0.0	<	60.0
✓ R3 Landau 15 pressure	1.6e-09	<	4e-10	✓ R3 RF 18 transmitter power	51.05052	<	60.0
✓ R3 Main RF 16 pressure	2e-10	<	6e-10	✓ R3 RF 19 transmitter power	52.96632	<	60.0
✓ R3 Main RF 17 pressure	1.4e-10	<	9e-10	✓ R3 RF 20 transmitter power	51.286121	<	60.0
✓ R3 Main RF 18 pressure	54.32502	<	2e-09	✓ R3 RF 01 transmitter power	0.0	<	60.0
✓ R3 Main RF 19 pressure	55.718563	<	3e-09				
✓ R3 Main RF 20 pressure	51.286121	<	3e-10				
✓ R3 Main RF 01 pressure	52.96632	<	3e-10				

316s2 IPK-01 in pressure alarm

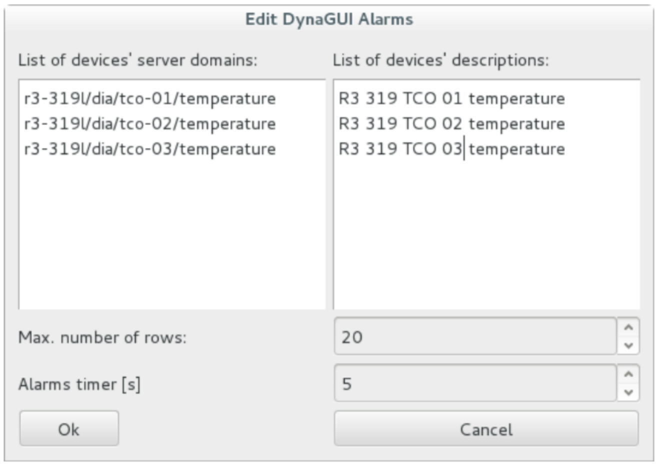
Load | Save | Select All | Unselect All

Running. Press to deactivate.

Figure 3. A DynaGUI Alarms layout in activated state with two pressure attributes being in alarm.

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. To begin monitoring, press activate. If an alarm criteria is fulfilled, DynaGUI Alarms will use the speakers of the computer to notify the user which devices are in alarm and what type of alarm (temperature, pressure, horizontal beam position, etc.) has been triggered and show the last alarm triggered. The triggered alarms also have their backgrounds turned red whilst the alarms not triggered have green backgrounds, as can be seen in Figure 3. To set up or edit the alarm signals to monitor, press Edit DynaGUI.

Pressing Edit DynaGUI opens up an edit panel for setting up the layout of the DynaGUI alarms application, the signals to monitor and their respective description. The list of the signals to monitor is defined in the left editbox whilst their descriptions is written in the right editbox, as can be seen in Figure 4. Each signal's description is then shown in the application, and their respective tooltip is the signal itself. The user can also define the number of rows for the layout and the alarms timer (or the polling frequency).



Edit DynaGUI Alarms

List of devices' server domains:	List of devices' descriptions:
r3-319l/dia/tco-01/temperature	R3 319 TCO 01 temperature
r3-319l/dia/tco-02/temperature	R3 319 TCO 02 temperature
r3-319l/dia/tco-03/temperature	R3 319 TCO 03 temperature

Max. number of rows: 20

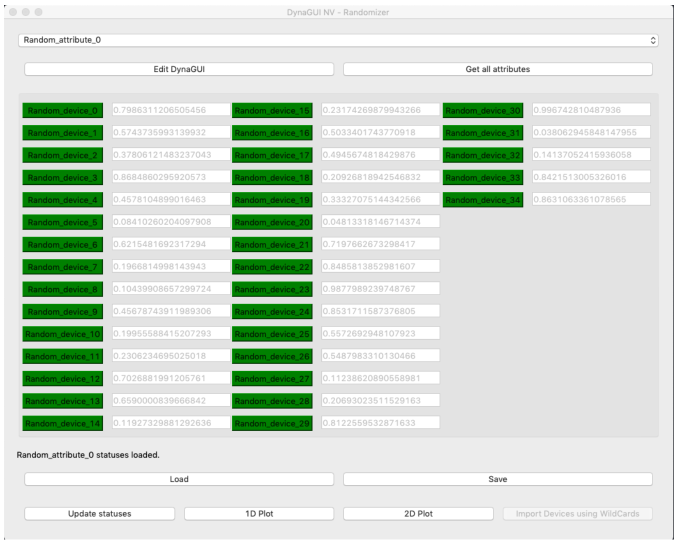
Alarms timer [s]: 5

Ok | Cancel

Figure 4. The DynaGUI Alarms edit panel.

C. 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. See Figure 5.



DynaGUI NV - Randomizer

Random_attribute_0

Edit DynaGUI | Get all attributes

Random_device_0	0.7986311206505456	Random_device_15	0.23174269879943266	Random_device_30	0.996742810487936
Random_device_1	0.5743735993139932	Random_device_16	0.5033401743770918	Random_device_31	0.038062945848147955
Random_device_2	0.37806121483237043	Random_device_17	0.4945674818429876	Random_device_32	0.14137052415936058
Random_device_3	0.8684860295920573	Random_device_18	0.20926818942546832	Random_device_33	0.8421513005326016
Random_device_4	0.4578104899016463	Random_device_19	0.33327075144342566	Random_device_34	0.8631063361078565
Random_device_5	0.08410260204097908	Random_device_20	0.04813318146714374		
Random_device_6	0.6215481692317294	Random_device_21	0.7197662673298417		
Random_device_7	0.1966814998143943	Random_device_22	0.8485813852981607		
Random_device_8	0.10439908657299724	Random_device_23	0.9877989239748767		
Random_device_9	0.45678743911989306	Random_device_24	0.8531711587376805		
Random_device_10	0.19955588415207293	Random_device_25	0.5572692948107923		
Random_device_11	0.2306234695025018	Random_device_26	0.5487983310130466		
Random_device_12	0.7026881991205761	Random_device_27	0.11238620890558981		
Random_device_13	0.6590000839666842	Random_device_28	0.20693023511529163		
Random_device_14	0.11927329881292636	Random_device_29	0.8122559532871633		

Random_attribute_0 statuses loaded.

Load | Save

Update statuses | 1D Plot | 2D Plot | Import Devices using WildCards

Figure 5. The DynaGUI NV panel configured with 35 artificial ('random') devices and 8 attributes with maximum number of rows being 15.

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[2], whilst for EPICS[4] and Finance, new control panels are currently under development. 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. Note that for a scalar attribute, the value is printed whilst for a vector or a waveform attribute, the shown value is information that it is not a scalar and the length of the vector/waveform.

Table II
DYNAGUI NV COLORSCHEME

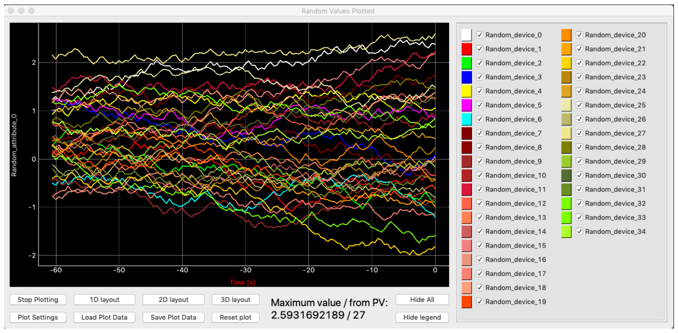
Color:	**Meaning for each device:**
Limegreen	Attribute valid and read
Fuchsia	Attribute not existing
Maroon	Device not possible to connect to

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 control system also allows for importation of devices by using wildcards, i.e. the user names part of a or some device(s) with an asterisk resulting in that it is interpreted as wildcard. Currently this works with Tango control systems[2], but will be extended to support more control systems.

The DynaGUI NV's 1D Graph Tool

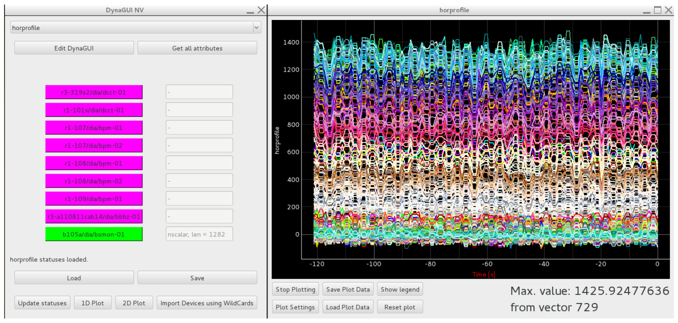
The DynaGUI NV can initialize live plotting as well as save and load plotting data, and is initialized by pressing the [1D Plot] button. The 1D graph will then plot the selected attribute for the devices of which this attribute is valid (whilst ignoring the devices for which the attribute is invalid). Before plotting, the user is prompted to define update frequency and periodicity to plot and store. In order to simplify the naming of different curve lines, the Graph Tool's window title becomes the name of the plotted attribute whilst the curve lines' names corresponds to the name of the device it represents. See Figure 6 for plotting a scalar attribute and Figure 10 for plotting a vector/waveform attribute. For any case, the maximum value is shown in the bottom-right corner and which line returns this value.

With the 1D graph tool, attributes representing scalars and vectors/waveforms can be plotted in real-time or analyzed from either an archiving database or from a file. When loading from a database, the user is prompted



Maximum value / from PV:
2.5931692189 / 27

Figure 6. 1D real-time plotting launched for the 35 artificial devices' scalar attribute Random_attribute_0 (as configured in Figure 5).



nscalar, len = 1282

horprofile statuses loaded.

Max. value: 1425.92477636
from vector 729

Figure 7. A dynamic control panel of DynaGUI has been configured (left), from which a 1D real-time plot has been launched for the devices of which the HorProfile vector attribute (or waveform) is valid. This vector attribute reveals the horizontal profile of a particle beam, see Figure 8.

for the start- and the end-date that is to be grabbed from the archive. Currently, the Cassandra database archiving system is supported (used at MAX IV Laboratory[1]) but more will be added together together with a method for users to add their own. When data has been loaded, the user is prompted if the 1D graph tool should open the file in data analysis mode (replace all data in the graph window and analyze using all tools of the 1D graph tool) or if the loaded data should only be plotted in a

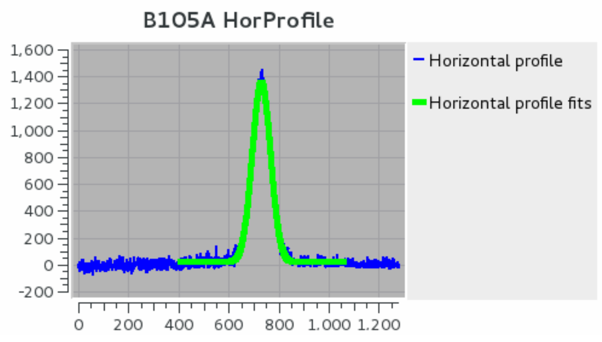
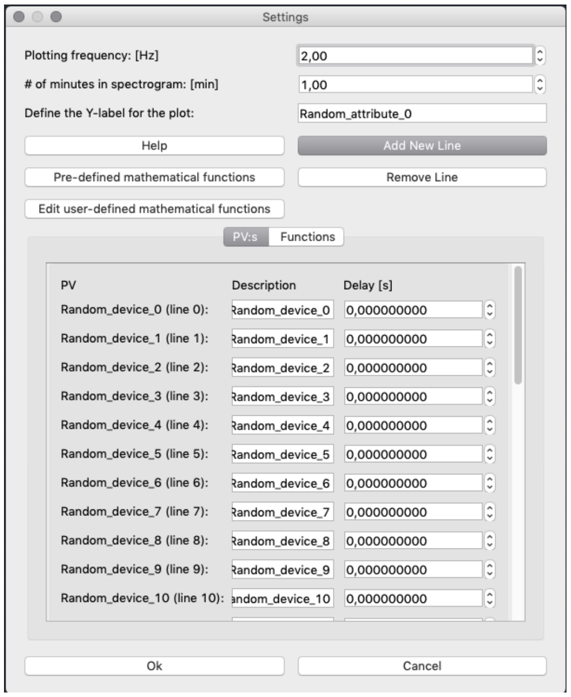


Figure 8. The horizontal profile of the beam as seen from the diagnostic beamline b105a at the 1.5 GeV storage ring at MAX IV Laboratory[1].



Settings

Plotting frequency: [Hz] 2,00

# of minutes in spectrogram: [min] 1,00

Define the Y-label for the plot: Random_attribute_0

Help | Add New Line | Pre-defined mathematical functions | Remove Line | Edit user-defined mathematical functions

PV:s | Functions

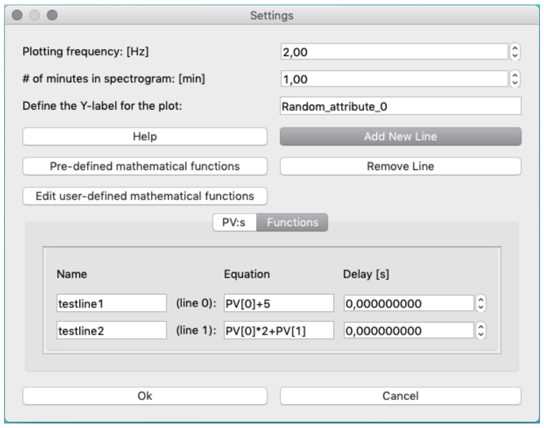
PV	Description	Delay [s]
Random_device_0 (line 0):	Random_device_0	0,000000000
Random_device_1 (line 1):	Random_device_1	0,000000000
Random_device_2 (line 2):	Random_device_2	0,000000000
Random_device_3 (line 3):	Random_device_3	0,000000000
Random_device_4 (line 4):	Random_device_4	0,000000000
Random_device_5 (line 5):	Random_device_5	0,000000000
Random_device_6 (line 6):	Random_device_6	0,000000000
Random_device_7 (line 7):	Random_device_7	0,000000000
Random_device_8 (line 8):	Random_device_8	0,000000000
Random_device_9 (line 9):	Random_device_9	0,000000000
Random_device_10 (line 10):	andom_device_10	0,000000000

Ok | Cancel

Figure 9. Settings for the 35 lines being plotted in real-time.

simple window without overwriting any data. Note that in the latter case, real-time plotting continues without being affected of any loaded data that is opened in a new window. During analysis of either real-time or historical data, it is possible to show or hide lines by ticking or unticking the corresponding legend(s). It is also possible to change colors of the lines by clicking on the color of the legend-items. Settings include changing the plotting frequency and amount of minutes to store and show, the Y-label of the plot, device descriptions as on how to show them in the legend and delays for each line, see Figure 9.

Further functionalities in the 1D plot settings includes an equation handler with which it is possible to plot functions of the input (real-time or archived) data as well as combining different input data, whether the data is scalars or a vector/waveform. In Figure 10, two example equations are being set up. Equations are set up by pressing [Add New Line], and each input data stream is given as 'PV[line-number]'. When setting the values, the user will be prompted if the equations should apply for all historical values in the graph or only for future readvalues. A simple example for combining two lines in real-time is '$PV[0]*2+PV[1]$', as can be seen by 'testline2' in Figure 11. Mathematical functions available can be seen by clicking the button [Pre-defined mathematical functions], which includes e.g. trigonometric functions. Users can also define mathematical functions by clicking the button [Edit mathematical functions].



Settings

Plotting frequency: [Hz] 2,00

# of minutes in spectrogram: [min] 1,00

Define the Y-label for the plot: Random_attribute_0

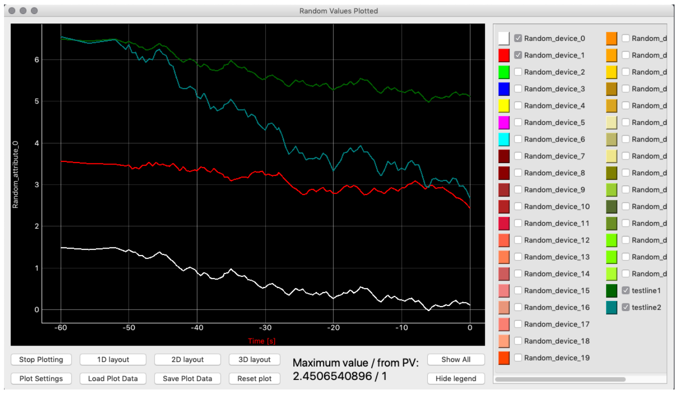
Help | Add New Line | Pre-defined mathematical functions | Remove Line | Edit user-defined mathematical functions

PV:s | Functions

Name		Equation	Delay [s]
testline1	(line 0):	PV[0]+5	0,000000000
testline2	(line 1):	PV[0]*2+PV[1]	0,000000000

Ok | Cancel

Figure 10. Setting up equations for the real-time data input with two input data lines being plotted and two function lines being plotted.



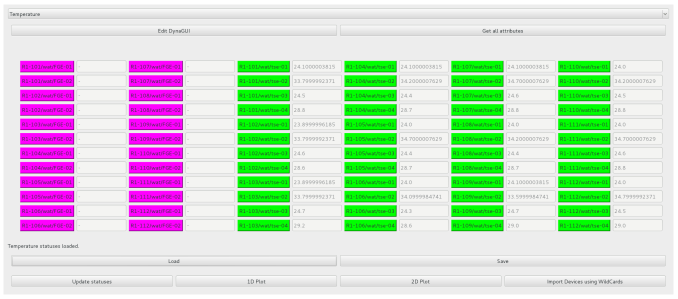
Maximum value / from PV: 2.4506540896 / 1

Figure 11. Two user-defined functions defined in Figure 10 plotted together with the data inputs they are a function of.

The DynaGUI NV's Spectrogram GUI

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 1.5 GeV storage ring (R1). Thus, all these devices were loaded and the two attributes Temperature and Flow have been defined, where Temperature is valid for TSEs and Flow is valid for FGEs. The resulting panel for the control system GUI can be seen in Figure 12.

The DynaGUI-generated water control panel in Figure 12 shows all TSEs in green and all FGEs 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. Clicking on any datapoint in the trace plot (Figure 12 to the left) adds a data marker revealing the name of the device.



R1-101/wat/FGE-01	-	R1-107/wat/FGE-01	-	R1-101/wat/tse-01	24.1000003815	R1-104/wat/tse-01	24.1000003815	R1-107/wat/tse-01	24.1000003815	R1-110/wat/tse-01	24.0
R1-101/wat/FGE-02	-	R1-107/wat/FGE-02	-	R1-101/wat/tse-02	33.7999992371	R1-104/wat/tse-02	34.2000007629	R1-107/wat/tse-02	34.7000007629	R1-110/wat/tse-02	34.2000007629
R1-102/wat/FGE-01	-	R1-108/wat/FGE-01	-	R1-101/wat/tse-03	24.5	R1-104/wat/tse-03	24.4	R1-107/wat/tse-03	24.6	R1-110/wat/tse-03	24.5
R1-102/wat/FGE-02	-	R1-108/wat/FGE-02	-	R1-101/wat/tse-04	28.8	R1-104/wat/tse-04	28.7	R1-107/wat/tse-04	28.8	R1-110/wat/tse-04	28.8
R1-103/wat/FGE-01	-	R1-109/wat/FGE-01	-	R1-102/wat/tse-01	23.8999996185	R1-105/wat/tse-01	24.0	R1-108/wat/tse-01	24.0	R1-111/wat/tse-01	24.0
R1-103/wat/FGE-02	-	R1-109/wat/FGE-02	-	R1-102/wat/tse-02	33.7999992371	R1-105/wat/tse-02	34.7000007629	R1-108/wat/tse-02	34.2000007629	R1-111/wat/tse-02	34.7000007629
R1-104/wat/FGE-01	-	R1-110/wat/FGE-01	-	R1-102/wat/tse-03	24.6	R1-105/wat/tse-03	24.4	R1-108/wat/tse-03	24.4	R1-111/wat/tse-03	24.6
R1-104/wat/FGE-02	-	R1-110/wat/FGE-02	-	R1-102/wat/tse-04	28.6	R1-105/wat/tse-04	28.7	R1-108/wat/tse-04	28.7	R1-111/wat/tse-04	28.8
R1-105/wat/FGE-01	-	R1-111/wat/FGE-01	-	R1-103/wat/tse-01	23.8999996185	R1-106/wat/tse-01	24.0	R1-109/wat/tse-01	24.1000003815	R1-112/wat/tse-01	24.0
R1-105/wat/FGE-02	-	R1-111/wat/FGE-02	-	R1-103/wat/tse-02	33.7999992371	R1-106/wat/tse-02	34.0999984741	R1-109/wat/tse-02	33.5999984741	R1-112/wat/tse-02	34.7999992371
R1-106/wat/FGE-01	-	R1-112/wat/FGE-01	-	R1-103/wat/tse-03	24.7	R1-106/wat/tse-03	24.3	R1-109/wat/tse-03	24.7	R1-112/wat/tse-03	24.5
R1-106/wat/FGE-02	-	R1-112/wat/FGE-02	-	R1-103/wat/tse-04	29.2	R1-106/wat/tse-04	28.6	R1-109/wat/tse-04	29.0	R1-112/wat/tse-04	29.0

Temperature statuses loaded.

Figure 12. A dynamic control panel of DynaGUI has been configured for the water temperature and flow monitors in the MAX IV Laboratory[1] 1.5 GeV storage ring.

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 13 below, the evolution of the read values of the horizontal beam position read by some BPMs is plotted as a function of time.

In order to see how the beam position looks like at a specific instance of time, the yellow line in the spectrogram can be moved to the time of interest and then pressing the Plot Trace button shows a plot of the beam position as seen by the defined and valid BPMs (as shown on the left plot of Figure 13).

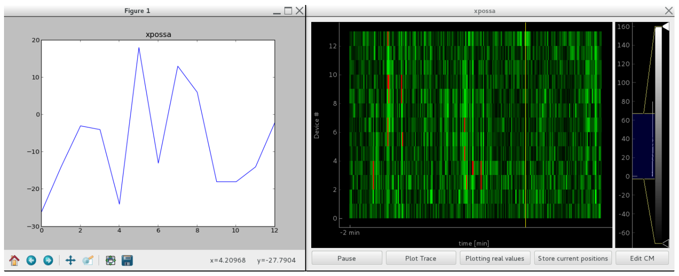


Figure 13. A 2D spectrogram showing how the xpossa (read values of the horizontal beam position) evolves with time for a set of BPMs, seen in green in Figure 6. To the right is the live-running 2D spectrogram whilst at the left is a trace plotted at the time that the yellow marker is placed at.

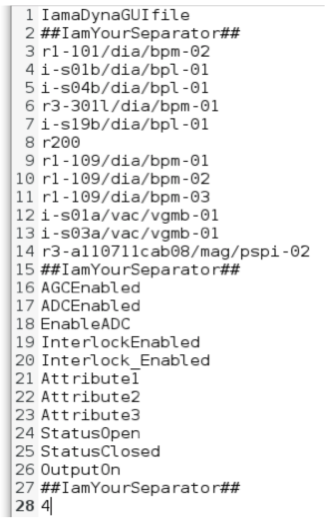
Functionalities of the current 2D spectrogram:

- 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[3] 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.

V. DynaGUI package configuration files

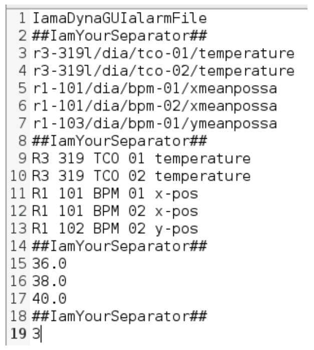
In order to allow users to create and store their own control systems configuration file, each DynaGUI application allows users to save and load configurations. A valid DynaGUI configuration file always contains and has to contain an identifier (*IamaDynaGUIfile*) in the first row. A configuration file also always contains a device list, attributes list and number of rows for the control panel, separated by *##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 14. If a file is impossible to load, it can be either an invalid number of separators or an invalid identifier key (which in any case will be printed in the DynaGUI's status bar).

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 DynaGUI Alarms configuration file can be seen in Figure 15.



```
 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 14. An example of a DynaGUI configuration file, with an added non-existing device called 'r200' at row 8.



```
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 15. An example of a DynaGUI Alarms configuration file.

VI. 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[3] objects (pushbuttons, textedits, etc.). In this version of DynaGUI, the dynamic buttons' identifiers is the label of each individual button, pointing to different server addresses. Each button are connected to the same function (hClick) which identifies which button was clicked and carries out thet requested action.

VII. Future Development Plans

Future developments include having a method to import different device types using wildcards, to embed the EPICS control system[4], 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 Svard at the MAX IV Laboratory for developing the original 2D Spectrogram Application (for monitoring transverse beam position via Beam Position Monitors). Their codes were used as an inspiration for developing a generic algorithm to initialize a 2D spectrogram for any given list of subscriptable live-value variables. The author also wants to thank Bernhard Meirose at the MAX IV Laboratory for giving the inspiration and awakening the idea to creating this package.

References

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

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

[3] PyQt packages, Riverbank Computing Limited.

[4] PyEpics: Interface for the Channel Access (CA) library of the EPICS Control System to the Python Programming language, https://cars9.uchicago.edu/software/python/pyepics3/.

[5] W. McKinney, *pandas: a foundational Python library for data analysis and statistics*, Python for High Performance and Scientific Computing, 14, 2011.