The Journal of Open Source Software

The DynaGUI package

Benjamin Edward Bolling[1, 2]

1 European Spallation Source ERIC **2** MAX IV Laboratory

DOI: 10.21105/joss.01942

Software

- Review
- Repository
- Archive

Editor: Mark A. Jensen

Reviewers:

- @Chilipp
- @pythonpanda2

Submitted: 01 November 2019
Published: 31 May 2020

Summary

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. Initially developed at MAX IV Laboratory (Tavares, Leemann, Sjostrom, & Andersson, 2014), the initial and main user for this package is controlrooms at large-scale research facilities, such as particle accelerators.

In order to further enhance the user-friendliness of this package, a simple system for configuration files has been developed enabling users to configure it using any plain text-editor. The DynaGUI package consists of three applications: - DynaGUI TF, a true/false (boolean) dynamic control system, - DynaGUI Alarms, a dynamic diagnostics system for continuously monitoring of the values for a set of attributes, and - DynaGUI NV, a system for observing attributes' numerical-, string-, vector- or waveform values.

Each DynaGUI application's layout is designed for having as high level of simplicity as possible. At the top of each application 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. 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, error messages, or if an alarm is active (for DynaGUI Alarms). Below the status bar is the load and save buttons for loading or saving DynaGUI configuration files, which also have a tool-tip function showing the last loaded or saved file (in the current session). The simplest method to launch DynaGUI is via its launcher (Launcher.py), in which the user can select control system and either directly define the path to the configuration file or browse to it. Leaving it blank results in that DynaGUI will load with a predefined setup.

DynaGUI TF (True/False) 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. 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 (with green meaning True, red meaning False, fuchsia meaning attribute not existing or not boolean, and maroon meaning that the device cannot be connected to).

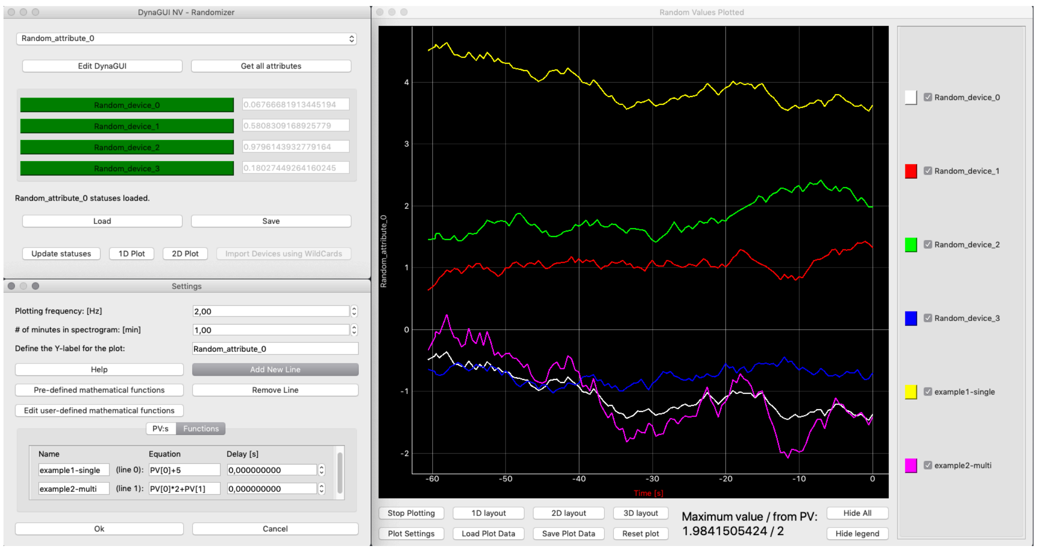
The DynaGUI Alarms (Dynamic Alarms GUI) has been designed to monitor numerical values and notify user(s) if a condition is not fulfilled. To edit the alarms GUI, the user has to press Edit DynaGUI to open an edit-panel. In the left window of the edit-panel, the user has to define the list of devices' server domains and signals to monitor. In the right window of the edit-panel, the user has to define descriptions of the different alarm signals as they should

appear in the DynaGUI Alarms control panel, and also the sweep-time has to be defined (how frequent the system should check the values). The DynaGUI Alarms' dynamic panel can be divided into three columns for each device and attribute it monitors. The first column contains the descriptions of the alarms, the second column contains the numerical values from the last sweep, the fourth column contains the alarm limit, and the third column contains a combobox with two gaps (larger than or smaller than) which points in the way it should be between the read value and the alarm limit. All gap criterias that are fulfilled results in that the background-color of the alarm description is painted lime-green. If a gap criteria is not fulfilled, an alarm will go off by using the computer's speakers, showing a message in the DynaGUI Alarms' status bar and the alarm description's background color becomes red.

DynaGUI NV is the most advance tool in the package, which for each device has two columns. The first column is a button with the server address of the device, whilst the value of the selected attribute is shown in the second column and hence enables users to inspect values in a simple and fast manner. DynaGUI NV also allows for launching a controller for the device (using AtkPanel for Tango Controls (Chaize et al., 1999)) or initializing 1D plots or 2D colormaps for the selected attribute for all devices for which the attribute is valid. Each device's control panel button becomes painted in lime-green if the attribute is valid, in fuchsia if the attribute is not existing for the device and maroon if a connection to the device cannot be established. The plot initialization automatically launches for all devices for which the attribute is valid. The plotting tool has been elaborately described in the DynaGUI documentation, with features included in the 1D graph tool being plotting read values in real time and also setting up and plotting equations (or functions) of the read values as well as combining the read values with one another, with examples of such attached in Figure 1. The 1D graph tool supports both scalar and vector (or waveform) plotting in real time, and plot data can be saved and loaded.

The package aims to enable users to construct dynamic GUI:s for multiple purposes, and the author is open for implementing new functions and control systems on demand. For testing purposes, an artificial control system 'Randomizer' has been created displaying only random values. In this DynaGUI version, the DynaGUI panels are constructed using PyQt (Riverbank Computing Limited, 2016), with versions 4 and 5 supported. The author is working on fully implementing the PyEPICS ("PyEpics," 2014) package as well as a new Finance package built on Pandas (McKinney, 2011) in order to have more sources to monitor live-stream data from and hence demonstrate the openness that serves as the core of the package.

Figures



Random_attribute_0

Random_device_0	0.06766681913445194
Random_device_1	0.5808309168925779
Random_device_2	0.9796143932779164
Random_device_3	0.18027449264160245

Random_attribute_0 statuses loaded.

Plotting frequency: [Hz] 2,00
# of minutes in spectrogram: [min] 1,00
Define the Y-label for the plot: Random_attribute_0

Name		Equation	Delay [s]
example1-single	(line 0):	PV[0]+5	0,000000000
example2-multi	(line 1):	PV[0]*2+PV[1]	0,000000000

Maximum value / from PV:
1.9841505424 / 2

Figure 1: A dynamic control panel of DynaGUI NV has been configured (top-left), from which a 1D realtime plot has been launched for 4 artificial devices for a made-up attribute (right). Using this tool, two other lines have been set up as functions of two input data streams, with equations defined in the bottom-left figure and then plotted.

Acknowledgements

The author wants to thank Bernhard Meirose, Accelerator Operator at the MAX IV Laboratory, for the discussion that inspired and lead me into developing this package. The author also recognises and wants to thank Jonas Petersson and Robin Svard, 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.

References

Chaize, J. M., Gotz, A., Klotz, W.-D., Meyer, J., Perez, M., & Taurel, E. (1999). TANGO - an object oriented control system based on CORBA. *ICALEPCS'99*.

McKinney, W. (2011). Pandas: A foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, *14*.

PyEpics: Interface for the Channel Access (CA) library of the EPICS Control System to the Python Programming language. (2014). Retrieved from https://cars9.uchicago.edu/software/python/pyepics3/

Riverbank Computing Limited. (2016). PyQt5: Python bindings for the Qt cross platform UI and application toolkit. Retrieved from https://www.riverbankcomputing.com/software/pyqt/

Tavares, P. F., Leemann, S. C., Sjostrom, M., & Andersson, A. (2014). The MAX IV storage ring project. *Journal of Synchrotron Radiation*, *21*. doi:10.1107/S1600577514011503