All Yarns Are Beautiful Documentation

Release 1

Christian Obersteiner, Andreas Müller

August 17, 2014

Contents

Contents:

CHAPTER 1

AYAB API

1.1 Main AYAB GUI module

Provides an Interface for users to operate AYAB using a GUI.

class ayab.ayab.**GenericThread**(*function*, **args*, ***kwargs*)
A generic thread wrapper for functions on threads.

class ayab.ayab.**GuiMain**
GuiMain is the main object that handles the instance of AYAB's GUI from ayab_gui.UiForm .

GuiMain inherits from QMainWindow and instanciates a window with the form components form ayab_gui.UiForm.

apply_image_transform(*transform_type*, **args*)
Executes an image transform specified by key and args.

Calls a function from transform_dict, forwarding args and the image, and replaces the QtImage on scene.

flip_image()
Public mirror current Image function.

getSerialPorts()
Returns a list of all USB Serial Ports

invert_image()
Public invert current Image function.

load_image_from_string(*image_str*)
Loads an image into self.ui.image_pattern_view using a temporary QGraphicsScene

load_pil_image_on_scene(*image_obj*)
Loads the PIL image on a QtScene and sets it as the current scene on the Image View.

mirror_image()
Public mirror current Image function.

rotate_left()
Public rotate left current Image function.

rotate_right()
Public rotate right current Image function.

set_enabled_plugin(*plugin_name=None*)
Enables plugin, sets up gui and returns the plugin_object from the plugin selected on module_dropdown.

smart_resize()
Executes the smart resize process including dialog .

updateProgress(*progress*)
Updates the Progress Bar.

update_file_selected_text_field(*route*)
'Sets self.image_file_route and ui.filename_lineedit to route.

1.2 Knitting Plugin API

class ayab.plugins.knitting_plugin.**KnittingPlugin**(*callbacks_dict*)
A generic plugin implementing a state machine for knitting.

Subclasses inherit the basic State Machine defined in __init__.

cleanup_ui(*ui*)
Cleans up and reverts changes to ui done by setup_ui.

get_configuration_from_ui(*ui*)
Loads options dict with a given parent QtGui object. Required for save-load functionality.

Returns A dict with configuration.

Return type dict

onconfigure(*e*)
Callback when state machine executes configure(parent_ui = parent, options={})

This state gets called to configure the plugin for knitting. It can either be called when first configuring the plugin, when an error had happened and reset is necessary. The parent ui is expected to hold an object with properties.

Parameters **parent_ui** – An object having already been set up by setup_ui.

onfinish(*e*)
Callback when state machine executes finish().

When finish() gets called, the plugin is expected to be able to restore it's state back when configure() gets called. Finish should trigger a Process Completed notification so the user can operate accordingly.

onknit(*e*)
Callback when state machine executes knit().

Starts the knitting process, this is the only function call that can block indefinitely, as it is called from an instance of QThread, allowing for processes that require timing and/or blocking behaviour.

setup_ui(*parent_ui*)
Sets up UI, usually as a child of parent_ui.ui.knitting_options_dock.

While the whole parent_ui object is available for the plugin to modify, plugins authors are **strongly** encouraged to only manipulate the knitting_options_dock, plugins have full access to the parent UI as a way to fully customize the GUI experience.

Parameters **parent_ui** – A PyQt.QMainWindow with the property parent_ui.ui.knitting_options_dock, an instance of QDockWidget to hold the plugin's UI.

1.3 AYAB Control Plugin

Handles the serial communication protocol.

This module handles serial communication, currently works in a synchronous way. AyabCommunication uses an internal PySerial.Serial object to connect to the device. The initializer can also be overriden with a dummy serial object.

class ayab.plugins.ayab_plugin.ayab_communication.**AyabCommunication**(*serial=None*)
Class Handling the serial communication protocol.

__del__()
Handles on delete behaviour closing serial port object.

close_serial()
Closes serial port.

cnf_line(*lineNumber*, *lineData*, *flags*, *crc8*)
Sends a line of data via the serial port.

Sends a line of data to the serial port, all arguments are mandatory. The data sent here is parsed by the Arduino controller which sets the knitting needles accordingly.

Parameters

- **lineNumber** (*int*) – The line number to be sent.
- **lineData** (*bytes*) – The bytearray to be sent to needles.
- **flags** (*bytes*) – The flags sent to the controller.
- **crc8** (*bytes, optional*) – The CRC-8 checksum for transmission.

open_serial(*pPortname=None*)
Opens serial port communication with a portName.

read_line()
Reads a line from serial communication.

req_info()
Sends a request for information to controller.

req_start(*startNeedle*, *stopNeedle*)
Sends a start message to the controller.

CHAPTER 2

Serial Communication Protocol

Here is a sequence diagram with the serial communication knitting protocol.

CHAPTER 3

Building AYAB

3.1 Build process for Development

AYAB's GUI is built in Python, based on PyQt4. To develop for it, you must have a set of packages installed.

- Python 2.7+ (Python 3 is not yet supported.)
- PyQt4
- PySerial
- Fysom
- Yapsy
- Pillow (PIL)

You can install most of this packages (except for Python 2.7 and PyQt4) either using your distribution's package manager or using Pip if available. The full list of requirements is available on the `requirements.txt` file on the software directory. The `setup.py` file also includes the dependencies that can be instaled via pip so you can install using a Virtual Enviroment.

If you are in Windows, you can either install Python for Windows, PyQt4 for Windows and pip and easy_install. An easier alternative is installing a bundle such as WinPython.

After installing all requirements, you can run AYAB directly using `python2 ayab_devel_launch.py`.

To build you can use `python setup.py sdist`. It will generate a .tar.gz file on `dist/`. This is a tar.gz file that you can install using easy_install or the bundled `setup.py`.

3.2 Build process for Deploy

Windows packages are built using py2exe `python setup.py py2exe`.

CHAPTER 4

Plugin API Overview

4.1 Plugin Class

AYAB features a simple but powerful API for implementing APIs for Knitting Machine hacks. Plugins are based on Yapsy. While knowledge in Yapsy is useful, it is not complex to get started.

Plugin modules are specified by a `.yapsy-plugin` file, and a Python module, specified in said file.

A class extending `KnittingPlugin` is required to create a plugin. Methods not overriden on subclasses will raise `NotImplementedError`.

class `ayab.plugins.knitting_plugin.`**`KnittingPlugin`**(*callbacks_dict*)

A generic plugin implementing a state machine for knitting.

Subclasses inherit the basic State Machine defined in __init__.

`cleanup_ui`(*ui*)

Cleans up and reverts changes to ui done by setup_ui.

`get_configuration_from_ui`(*ui*)

Loads options dict with a given parent QtGui object. Required for save-load functionality.

Returns A dict with configuration.

Return type dict

`onconfigure`(*e*)

Callback when state machine executes configure(parent_ui = parent, options={})

This state gets called to configure the plugin for knitting. It can either be called when first configuring the plugin, when an error had happened and reset is necessary. The parent ui is expected to hold an object with properties.

Parameters **parent_ui** – An object having already been set up by setup_ui.

`onfinish`(*e*)

Callback when state machine executes finish().

When finish() gets called, the plugin is expected to be able to restore it's state back when configure() gets called. Finish should trigger a Process Completed notification so the user can operate accordingly.

`onknit`(*e*)

Callback when state machine executes knit().

Starts the knitting process, this is the only function call that can block indefinitely, as it is called from an instance of QThread, allowing for processes that require timing and/or blocking behaviour.

setup_ui (*parent_ui*)
Sets up UI, usually as a child of parent_ui.ui.knitting_options_dock.

While the whole parent_ui object is available for the plugin to modify, plugins authors are **strongly** encouraged to only manipulate the knitting_options_dock, plugins have full access to the parent UI as a way to fully customize the GUI experience.

Parameters parent_ui – A PyQt.QMainWindow with the property parent_ui.ui.knitting_options_dock, an instance of QDockWidget to hold the plugin's UI.

4.2 Building UIs

The setup_ui function gets called when the plugin gets selected on the main UI dropdown. The function clean_ui gets called on disposal. Usually the parent UI will only modify `parent_ui.ui.knitting_options_dock`, as it is the specified container for the knitting options. You can load generated modules using pyuic, by instanciating the dock, setting up autogenerated UI and then setting up behaviour.

UIs are expected to have a limited horizontal width, as children of the dock are not expected to overflow horizontally.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

a