VisualK-OS Control Center (VKOS_CC)
Documentation [v1.1 – 11/03/2016]

System design and technical documentation: Kevin Slaughter
slaughkj@miamioh.edu

Table of Contents

[Revision History]

v1.1 - Typo corrections, corrected FBA linear diagram to include the 25 pin cable going to the Arduino.
v1.0 – Initial publication, to accompany the initial release of the source code.

Section 1 – System Explanation

The VisualK-OS Control Center (hereafter referred to as "VKOS_CC") drives an array of Arduino-based "slave" devices, allowing for management of the entire lighting system from one computer. The Control Console, a custom setup that houses a Raspberry Pi (Model B) and various other goodies, is what will generally be running the Python scripts that comprise the VKOS_CC that this documentation focuses on. The Control Console and the FBA device are kind of one-in-the-same, but separate systems. Unlike Thor, the drum lighting module, the FBA device cannot be used as a stand-alone device (yet…).

The Control Console currently contains the following:

- Dedicated Raspberry Pi (B)
  - |-- *Runs the VKOS_CC, controls all slave devices*
  - |-- *PiFace attached by secured breadboard ties, but not yet used.*
- 3.5" TFT LCD (composite input)
  - |-- *Kinda shitty display… It works, but everything is tiny.*
- Powered USB hub
  - |-- *Helps the rPi properly power numerous slave devices.*
- 2 Auxiliary 110/120 AC mains outlets.
- 350W desktop PSU (For powering the FBA, explained later)
- Arduino Uno (For the above-mentioned FBA slave device)

* "FBA" stands for "Flood Bank Array", which is a dual set of 4 channel RGB LED banks: Each lamp is approximately $12in^2$, constructed of linked rows of LED strips. Each board also includes a white LED strip between each line of RGB LEDs for general lighting and strobe usage. All components are mounted on a board and controlled by a custom straight-through RJ-45 connection. Each 4 channel board contains an etched PCB with the necessary dual 74HC595 shift registers and TIP120 transistors to drive all 12 RGB connections of the 4 channel array. Each board is then connected by said RJ-45 to a printed enclosure containing RJ-45 ports linked and broken out from a 25 pin DSUB connection, which links everything neatly back to the host Arduino device housed within the Control Console.

In summary:
[LED Boards] → [RJ-45] → [DSUB25 Breakout Box] → [25 pin LPT cable] → [Arduino] → [rPi]

Section 2 – Modules (Python Sources)

The Python script that drives the Control Center is comprised of several systems, in order to achieve more fluidity via object-oriented design. The main program hosts several subsystem objects:

Subsystems

File: *VKCC_objGamePad.py*
Desc: GamePad I/O. Used for rapid/programmable handheld control of the lighting system. Currently, only the FBA is driven by a hardwired PS3 controller.

File: *VKCC_objSerialManager.py*
Desc: Serial COMMS bridge. Handles all communications/interpretations to and from the slave devices.

File: *VKCC_objScreenManager.py*
Desc: GUI management class. Governs all GUI tasks, directing user input as needed to the currently active screen class. All GUI drawing/blanking is done here.

These subsystems allow the main Python script to remain clean and topical, so that the source of a given problem within the software can be more easily identified and remedied. The main program only processes command line arguments, sets the display mode (console vs. Curses GUI) and systematically loops through the system's normal flow until either an error is encountered or we choose to exit the program via <CTRL+C>.

The non-subsystem modules consist of the various screens used by the Curses GUI, global object/constant definitions and file containing constant definitions only applicable to VKOS_CC in a local scope (ie: The Arduino devices are not involved whatsoever).

All Other Modules

File	*Description*
COMMS_Constants.py	Definitions of constants shared by the Arduino devices and the VKOS_CC Python script… Basically, a way for the Arduino devices and the Python script to easily communicate without complicated overhead.
globals.py	Global object definitons, accessible to the majority of the VKOS_CC Python script. These include the Screen Manager, GamePad and Serial Manager object classes, as well as the slave device wrapper object classes (explained below).
_screenCfgFBA.py	Configuration screen for the FBA device.
_screenCfgMain.py	Main configuration menu
_screenCfgSS.py	SynthStation configuration menu. (Asthetically implemented)
_screenCfgThor.py	Thor configuration menu (Aesthetically implemented)
_screenClientScan.py	If specified when scanning for slave devices, this screen appears to indicate progress.

_screenDisplayMapping.py	Screen to display the current gamepad mapping profile. (Aesthetically implemented)
_screenMain.py	Main menu
_screenManualFBA.py	Manual (keyboard-based) control of the FBA device.
VKCC.py	Main Python script.
VKCC_Constants.py	Definitions of constants specific and local to the VKOS_CC program. These constants do not concern the Arduino devices whatsoever.
VKCC_objFBA.py	Wrapper class for controlling the FBA slave device.
VKCC_objSynthStation.py	Wrapper class for controlling the yet-to-be-implemented Synth Station lighting controller.
VKCC_objThor.py	Wrapper class for controlling the Thor acoustic drum lighting module.

Section 3 – Program Topology

This section will address the general flow of the VKOS_CC Python script. In short, the gamepad (PS3 controller) will almost always be used to control the FBA device while the keyboard attached to the rPi running the main VKOS_CC Python script is used to navigate the configuration and monitoring screens of the software. The gamepad I/O subsystem allows for different profiles to be used for directing how the system is to respond to input from the gamepad, including cycling between gamepad profiles.

The Python script begins by processing any command line arguments, initializes the gamepad and screen manager subsystems and then enters the main program loop until either an error or escape sequence is received (Such as <CTRL+C>). There is an option to mute *all* lighting effects while on the main menu by pressing “*”, though this is largely meant as an immediate override for whatever automation may be taking place. I may add in a condition or alternate key that will enable the white floods on the FBA boards when mute is active, but that is not a priority at the moment.

Currently, the system only works via the Manual FBA screen or gamepad input. Future revisions will include support for automation scripts, though there are currently a few routines in place (such as the ColorWall FBA function).