Bravo++ multi-mode

DISCLAIMER

This is a beta release of a script I developed for personal use in the hopes that others would find it useful and fun. I am distributing it for free personal use and I appreciate feedback, but please don't expect me to provide full-time support on this.

If the script doesn't work for you, you can submit the `log.txt` file, the configuration file you are using and a description of the problem by creating a GitHub issue or send me a PM and I will try to see if I can solve the problem, but it may take time. For platform users that are not on Windows, I can only provide limited help, since I only run X-Plane on Windows 11. That doesn't mean it won't work on other platforms and I encourage you to try, but if you get a platform specific problem I probably won't be able help you.

Description

Bravo++ allows you to configure multi-mode functionality, so that you get more out of your Honeycomb Bravo than just the basic autopilot. The default mode (AUTO) will retain the standard autopilot functionality, but you can configure additional modes so that you can use the selector switch, buttons and rotating button to control other functionality in the aircraft. I have minimized the use of the Honeycomb Configurator and the only two controls that still need to be configured there are the right knob and the trim wheel (these can be configured outside the Honeycomb configurator, but the behavior won't be as good). There are some configuration files provided for the default aircraft such as the Cessna 172, the King Air C90B, and the Cirrus SF50 along with configurations for the Aerobask DA42 and DA62. Hopefully these will be enough so that you can configure you're own aircraft and perhaps submit it to the collection.

Prerequisites:

- X-Plane 12
- Honeycomb configurator (optional, but I found it necessary for my installation runnning Windows 11)
- FlyWithLua NG
- DataRefTool or DataRefEditor plugin (if you want to customize or write your own configuration)

The functionality is provided as a FlyWithLua script and consists of 3 parts:

- BravoMultiMode.lua - This is the main script that provides all the functionality and is placed in the FlyWithLua/Scripts directory.
- log.lua - This is a log utility that is used by BravoMultiMode.lua and is located in the FlyWithLua/Modules directory.
- config file - The file can be called either bravo_multi-mode.cfg or bravo_multi-mode..cfg and is where you onfigure all the different modes you want to have on your specific aircraft. Some example files are included in the FlyWithLua/conf directory and should be placed directly under the corresponding aircraft folder.

There is also an extra utility called ButtonLogUtility.lua that is used to determine which buttons the selector knob is mapped to in X-Plane. By default it is set to 0 and will use the HID to determine the state of the left selector knob, but this will introduce lag (at least on Windows 11). So to have a more responsive update to the GUI it is better to determine the button number X-Plane has assigned to the selector knob when it is set to "alt".

Installation

You should begin by installing the FlyWithLua plugin for X-Plane 12. If you already have it installed make sure that it is the NG version.

Next you can either download the Bravo++ zip archive from the X-Plane forum or get the latest release from the GitHub repository. All relevant files are found under the FlyWithLua directory and the entire FlyWithLua directory should be copied under the `plugins` folder.

Finally, you can optionally install the Honeycomb configurator developed by Aerosoft. For Windows 11, I notice that through HID, the knob is polled roughly every second leading to "clicks" being missed which in turn make the knob and trim unusable. This is the reason I need to have it installed on my machine, but it is a minimal profile that just assigns the right knob and the trim wheel to the configurator. This may not be a problem on Mac and Linux, but I have not tested on those platforms. Anyhow, the links to Honeycomb Configurator download are:

- For Windows
- For Mac

Configuration

Determining the button number for the left selector knob

In order to determine which number X-Plane has assigned to the selector knob you need to enable the `ButtonLogUtil.lua` that should be located under the `Resources\plugins\FlyWithLua\Scripts` directory. You do this by opening the file and setting the `local write_log = false` to `local write_log = true`. This will allow the script to write output to the `log.txt` file located directly under the X-Plane 12 directory.

Now you simply load up an aircraft and once it is loaded you should see a little text bubble next to the mouse cursor indicating the number of the button that wa last clicked. Now you can twist the left selector knob through the full range of selection and finally select the "Alt" setting. Note down the number which you will use in the next step.

Open the `BravoMultiMode.lua` file under the `Resources\plugins\FlyWithLua\Scripts` directory and look for the `local alt_selector_button = 0` and replace the "0" with the number you noted.

Go back to the `ButtonLogUtil.lua` and disable the logging by setting the `local write_log` back to false.

Configuring the buttons in X-Plane

Next you need to configure the Honeycomb Bravo buttons to use Bravo++. You may want to create a base profile (called Bravo++) that X-Plane uses, since this can be reused between aircraft configurations. Otherwise chose an existing profile and start configuring the buttons.

Here are the descriptions you should look for when configuring each button with their corresponding dataref:

- HDG = Bravo++ toggles HDG button (FlyWithLua/Bravo++/hdg_button)
- NAV = Bravo++ toggles NAV button (FlyWithLua/Bravo++/nav_button)
-

APR = Bravo++ toggles APR button (FlyWithLua/Bravo++/apr_button)
- REV = Bravo++ toggles REV button (FlyWithLua/Bravo++/rev_button)
- ALT = Bravo++ toggles ALT button (FlyWithLua/Bravo++/alt_button)
- VS = Bravo++ toggles VS button (FlyWithLua/Bravo++/vs_button)
- IAS = Bravo++ toggles IAS button (FlyWithLua/Bravo++/ias_button)
- AUTOPILOT = Bravo++ toggles AUTOPILOT button (FlyWithLua/Bravo++/autopilot_button)

For finding the corresponding command in X-Plane just search for "Bravo++" and you should see all the available options you can map to.

There are also datarefs that are used for toggling/scrolling through the modes. I would suggest having it on a button accessible to the hand that is not used for the Honeycomb Bravo (left hand for most people) on the joystick or yoke.

- For toggling through the modes in one direction you should map it to the command with description `Bravo++ toggles MODE`. So all you do is click on the button and it will move down one mode.
- If you prefer scrolling through the modes with the right twist knob located on the Honeycomb Bravo, then you need to map the following command with description `Bravo++ activates the mode select when button is held in` The way this works is that you need to keep the button pressed down while you scroll with the right knob. When you are done you release the button.
- If you don't like these options, you can also map the commands that move the selection up or down to any key or button you like using the provided commands with description `Bravo++ cycle mode up` and `Bravo++ cycle mode down`

Finally, there are two internal commands that are often assigned to one of the Honeycomb Bravo buttons.

- The `I/O` button (see oneof the example configs) is used for switching between the inner or outer scroll knob. The active state is shown in green on the Bravo++ window. The dataref is described as `Bravo++ toggles INNER/OUTER mode`.
- For switches a long click (above 250 ms, but below 400 msec) will toggle between up and down state . Each button that implements a switch will either have `^^` above or `vv` below the button label indicating what will happen if the button is pressed. This distinguishes it from buttons that just toggle between two states. So by initiating a long click you toggle how the switch will behave. The dataref is described as `Bravo++ toggles UP/DOWN switch mode`.

Finally, a small note on button behavior. A click (below 250 msec) will actuate the button or switch. A long click (between 250 - 400 msec) is use for switches and as described aboved will change the direction in which the switches will be actuated on the following click. Holding a button down for over 400 msec will sustain a switch and is useful for spring-loaded switches that cannot be activated with a simple click.

Configuring the rocker switches in X-Plane

The rocker switches have two Bravo++ commands each; one for the up position and one for down position. There are 7 rocker switches and they are named switch1, switch2, switch3, etc. As mentioned before, just search for "Bravo++" when binding the keys and you will find the 14 commands that need to be bound to the switches.

Here are the descriptions you should look for when configuring each switch with their corresponding dataref:

- switch1 (position up) = Bravo++ command for rocker switch1 when it is positioned up (FlyWithLua/Bravo++/rocker_switch1_up)
- switch1 (position down) = Bravo++ command for rocker switch1 when it is positioned down (FlyWithLua/Bravo++/rocker_switch1_down)
-

- switch2 (position up) = Bravo++ command for rocker switch2 when it is positioned up (FlyWithLua/Bravo++/rocker_switch2_up)
- switch2 (position down) = Bravo++ command for rocker switch2 when it is positioned down (FlyWithLua/Bravo++/rocker_switch2_down)
- switch3 (position up) = Bravo++ command for rocker switch3 when it is positioned up (FlyWithLua/Bravo++/rocker_switch3_up)
- switch3 (position down) = Bravo++ command for rocker switch3 when it is positioned down (FlyWithLua/Bravo++/rocker_switch3_down)
- switch4 (position up) = Bravo++ command for rocker switch4 when it is positioned up (FlyWithLua/Bravo++/rocker_switch4_up)
- switch4 (position down) = Bravo++ command for rocker switch4 when it is positioned down (FlyWithLua/Bravo++/rocker_switch4_down)
- switch5 (position up) = Bravo++ command for rocker switch5 when it is positioned up (FlyWithLua/Bravo++/rocker_switch5_up)
- switch5 (position down) = Bravo++ command for rocker switch5 when it is positioned down (FlyWithLua/Bravo++/rocker_switch5_down)
- switch6 (position up) = Bravo++ command for rocker switch6 when it is positioned up (FlyWithLua/Bravo++/rocker_switch6_up)
- switch6 (position down) = Bravo++ command for rocker switch6 when it is positioned down (FlyWithLua/Bravo++/rocker_switch6_down)
- switch7 (position up) = Bravo++ command for rocker switch7 when it is positioned up (FlyWithLua/Bravo++/rocker_switch7_up)
- switch7 (position down) = Bravo++ command for rocker switch7 when it is positioned down (FlyWithLua/Bravo++/rocker_switch7_down)

Configuring the right twist knob and the trim wheel

You can try to configure the twist knob and the trim wheel directly in X-Plane, but I personally get issues with latency that results in not all the clicks getting registered as I turn the knob or wheel. I may delve deeper into this issue to see if I can resolve it, but for now I use the Honeycomb Configurator from Aerosoft to solve the problem. I am aware that Mac users have issues with the software, but perhaps it will work with the minimal setup config file I have provided.

You will find the Honeycomb Configurator file under `Resources\plugins\FlyWithLua\conf\Bravo++_honecomb_configurator.json` and you import it using the following steps:

- Select "Actions > Open settings"
- Click on "Import profiles"
- Select the file `Resources\plugins\FlyWithLua\Scripts\Bravo++_honecomb_configurator.json`
- Select the profile `Bravo++ Multi-mode`
- Click "Ok"

Once imported you need to activate the profile either before starting up X-Plane or if you do it while X-Plane is running, you need to ensure the current aircraft is using the profile by selecting in the X-Plane menu `Plugins > HoneyComb > BFC_Throttle > Reload bindings`.

Make sure that the right twist knob and trim wheel are not configured in X-Plane; i.e. they should be set to "Do nothing". On the other hand, if you want to try the functionality without using the Honeycomb Configurator, then you need to set the appropriate datarefs in X-Plane.

For reference, the relevant command descriptions that are configured are as follows:

- Increase value (turn knob to the right) = Handle button on bravo that increments values (FlyWithLua/Bravo++/knob_increase_handler)
- Decrease value (turn knob to the left) = Handle button on bravo that decrements values (FlyWithLua/Bravo++/knob_decrease_handler)
- Nose up (turn wheel up) = Handle trim on bravo for nose up (FlyWithLua/Bravo++/trim_nose_up_handler)
- Nose down (turn wheel down) = Handle trim on bravo for nose down (FlyWithLua/Bravo++/trim_nose_down_handler)

Configuring the aircraft

The easiest way to start is to use one of the predefined G1000 configurations (all aircraft, but the King Air C90B) like the for the Cessna 172. I won't go into the details on the content of the config file in this section and assume you want to get going as quickly as possible.

So let's configure the Cessna 172 that uses the G1000.

Start by copying the file called `Resources\plugins\FlyWithLua\conf\bravo_multi-mode.Cessna_172SP_G1000.cfg` to the `Aircraft\Laminar Research\Cessna 172 SP` directory. Note that the Cessna has 3 .acf files and the configuration contains the name that is in `Cessna_172SP_G1000.acf`. This is how the script knows which configuration to use when it starts up. If you start any of the other two variants that aren't G1000 equipped, the script will just stop, since it can't find a corresponding config file.

Once the file is copied, you can load the aircraft and hopefully you will now see the Bravo++ window that contains the current mode and status of the buttons. If you don't then either the script couldn't find the config file, the Honeycomb Bravo device is not plugged in or something went wrong with the script. In the latter case you will probably hear FlyWithLua complaining and telling you that it has moved the bad script to `Script (Quarantine)` folder. This shouldn't happen, but if it does check the `log.txt` file for any errors.

The Bravo++ can be popped out as a separate window, which is useful if you have multiple monitors. I personally have 4 monitors, where I have the G1000 PFD, MFD and the Bravo++ window on the smallest monitor.

So initially you will see the default mode on the left (AUTO in green) and the currently selected value for the left selector knob. On the bottom you will see all the corresponding buttons in grey and if they are active they will be in white. If they are white the corresponding led on the Bravo device will also be lit. Finally on the right you have the "outer" and "inner" selection which are used when using the other modes. This controls whether the inner or outer knob is to be turned when using the right twist knob. So once you start up the aircraft you can test out the functionality by pressing the "HDG" button and if all is well you should see that the "HDG" button on the device will light up and the Bravo++ window will now show the text in white. By pressing the "HDG" again it should make the button inactive again.

To change the mode, you need to click the button you assigned to it. Clicking the button allows you to cycle through the different modes. For the Cessna 172, there are 3 modes (AUTO, PFD and MFD) and it is possible to add additional modes if desired, but that is for another day. If you are curious about additional modes you can look at the DA42 or DA62 configuration which contains an additonal mode called "SYS" that allows settings the lights, operating the anti-ice system, ignition and auxiliary pumps. You can basically configure whatever you want, but there are some known limitations which I won't take up here now.

If you select the "MFD" mode you will notice that the text changes for most of the content. If you turn the left selector knob to "ALT" you will notice that in the Bravo++ window it now indicates "COM". On the bottom, you will also notice that the labels for the buttons are now different. Some of the buttons do nothing, while others will

performa action. So from this selection your are able to tune the com radio frequencies using the right twist knob and the buttons. The "IAS" button on Bravo device now toggles whether the twist knob controls the inner or outer ring of the knob. So when it is set to outer it will control the MHz values of the frequency (118 - 136 MHz), while inner will control the KHz frequencies. The "VS" button will control which frequency is active by swapping the frequencies. The "ALT" button allows you to swicth between COM1 and COM2. Notice that the text for these buttons on the Bravo++ window are blue-green. This indicates that they toggle something without the causing the led light to go on. The "REV" button, on the other hand, is dark grey and this indicates that the led light will be activated if pressed. In this case it will unmute the COM2 speaker and cause the led light to go on. So try dialing in an ATIS/AWOS frequency at the airport you are at on COM2 and then unmute it by pressing the "REV" button. You should hear the ATIS/AWOS track.

I suggest you explore the rest of the functionality, especially the "FMS" selection, which allows you to access the flighplan menu and procedure menu without using the mouse.

For more advanced configuration I would suggest looking at King Air C90B configuration file.

File format

Once, I have finalized the the configuration file I will wite a more formal specification, but for now I will provide a quick run down.

Modes, selector labels and button labels

The first section defines what will be shown in the Bravo++ window. These are modes, selector labels and the button labels.

So for the Cessna 172 we have the following:

```
# List all the modes. AUTO must always be included.
MODES = "AUTO,PFD,MFD"

# Set up the labels
PFD_SELECTOR_LABELS = "COM,NAV,BARO/CRS,RNG,FMS"
MFD_SELECTOR_LABELS = "COM,NAV,BARO/CRS,RNG,FMS"
```

There are 3 modes (AUTO,PFD and MFD) and the selector labels (the ones corresponding to the left twist knob on the Honeycomb Bravo) for each mode is specified by using the mode name and then adding `_SELECTOR_LABELS`. The selector labels should have 5 values. Note that the `AUTO` will use the default selector names (ALT, VS, HDG, CRS and IAS), but can be overridden if explicitly specified in the config file.

After that we assign the button labels to each selector name.

```
PFD_ALT_BUTTON_LABELS = "   ,   ,COM1,COM2,1&2,<->,O/I,   "
PFD_VS_BUTTON_LABELS  = "MKR,DME,NAV1,NAV2,1&2,<->,O/I,   "
PFD_HDG_BUTTON_LABELS = "   ,   ,   ,   ,SYNC,STD,O/I,   "
PFD_CRS_BUTTON_LABELS = "   ,   ,   ,   ,   ,   ,   ,   "
PFD_IAS_BUTTON_LABELS = "MNU,FPL,PRC,CLR,ENT,PSH,O/I,DIR"

MFD_ALT_BUTTON_LABELS = "   ,   ,COM1,COM2,1&2,<->,O/I,   "
MFD_VS_BUTTON_LABELS  = "MKR,DME,NAV1,NAV2,1&2,<->,O/I,   "
MFD_HDG_BUTTON_LABELS = "   ,   ,   ,   ,SYNC,STD,O/I,   "
MFD_CRS_BUTTON_LABELS = "   ,   ,   ,   ,   ,   ,   ,   "
MFD_IAS_BUTTON_LABELS = "MNU,FPL,PRC,CLR,ENT,PSH,O/I,DIR"
```

A button label is specifed by using the `mode name` + `selector name` separated by _ and ending in `_BUTTON_LABELS`. Note that the selector name is not the same as the selector label name. Here we have to use the default selector names which are: ALT, VS, HDG, CRS and IAS.

For `AUTO`, it will use default button labels, but like the selector labels they can be overridden by specifying them in the config file.

Note that even if you do not want to have all the buttons assigned to something you need to assign a blank label as seen in the example above.

Actions for right twist knob

The twist knob on the right of the Honeycomb Configurator is used for incrementing or decrementing values for various components of the cockpit. For the GNS530/GNS430 and the G1000 the twist knobs have an inner and outer wheel, and this is implemented in the Bravo++.

In the case where you just want to configure a simple knob you just specify the name of the `mode name` + `selector name` (separated by _) and then add `_UP` or `_DOWN`.

So for the autopilot mode (AUTO), to assign incrementing and decrementing the altitude when the selector is on `ALT` we specify the following:

```
AUTO_ALT_UP = "sim/autopilot/altitude_up"
AUTO_ALT_DOWN = "sim/autopilot/altitude_down"
AUTO_ALT_KNOB_LABELS="feet"
```

Here we specify the mode AUTO and when the left selector knob is set to ALT we want it to increase and decrease the altitude by assigning the datarefs corresponding datarefs. DataRefs can be found using DataRef Editor or DataRef Tool as specified at the beginnign of the document under `Prereqisites`. It is also optional to specify a text that will be displayed using the `AUTO_ALT_KNOB_LABELS`. The text value can either be one value or two values (separated by comma) depending on whether the knob has both an inner/outer functionality or not.

In the case where you want to simulate a knob that has an inner or outer portion to control coarse and fine values, you need to use the addtiional keywords OUTER and INNER. So again lookin at an example:

```
PFD_ALT_OUTER_UP = "sim/GPS/g1000n1_com_outer_up"
PFD_ALT_OUTER_DOWN = "sim/GPS/g1000n1_com_outer_down"
PFD_ALT_INNER_UP = "sim/GPS/g1000n1_com_inner_up"
PFD_ALT_INNER_DOWN = "sim/GPS/g1000n1_com_inner_down"
```

Here we are specifying the PFD mode with the left selector set to ALT, but we also want different behaviour based on the value of the CF (coarse/fine) selector. The CF selector is internal to Bravo++ and is made available through its own dataref.

Actions for the buttons and button leds

The Honeycomb Bravo has 8 buttons that can be configured to trigger a command depending on the mode and the selector that has been set. This gives the possibility of configuring up to 40 buttons per mode!

The buttons also support simple click and long press (i.e. holding down the button). This is enabled on all the

buttons and holding down the button simulates spring-loaded switches which are often used for tests or actions that should not be activated continually.

To specify a command to a button you use a similar pattern to what has been used for the previous sections: `mode name` + `selector name` (optional) + `button name` + `switch direction` (optional) separated with ``` ` ``` and ending with `_BUTTON`. Note that the `selector name` and `switch direction` are optional and the reason is that it is possible that you want an a button to trigger the same action regardless of what the selector is set to. This is how you want the autopilot to behave for example. The `switch direction` is only used when the same command cannot be used to toggle the state of a switch and requires 2 separate commands. So just like with the twist knob in the previous section, after the name you specify the name of the dataref to use after the = sign.

To specify the dataref and value to use for deciding when a led light on the button should light up or not you first specify the name: `mode name` + `selector name` (optional) + `button name` and ending with `_BUTTON_LED`. Again the `selector name` is optional because there are cases where you want the same led to light (or not) irregardless of the selection on the selector knob. Next you need to specify the dataref that contains the value of the state of whatever the corresponding command triggered and you need to provide a value to check against. In this case the value is what it should be in order to turn off the led.

Let's look at some examples:

```
PFD_ALT_APR_BUTTON = "sim/audio_panel/select_audio_com1"
PFD_ALT_APR_BUTTON_LED = "sim/cockpit2/radios/actuators/audio_selection_com1,0"
PFD_ALT_REV_BUTTON = "sim/audio_panel/select_audio_com2"
PFD_ALT_REV_BUTTON_LED = "sim/cockpit2/radios/actuators/audio_selection_com2,0"
PFD_ALT_ALT_BUTTON = "sim/GPS/g1000n1_com12"
PFD_ALT_VS_BUTTON = "sim/GPS/g1000n1_com_ff"
PFD_ALT_IAS_BUTTON = "FlyWithLua/Bravo++/cf_mode_button"
```

Here we have an example from the PFD of the G1000. The first line says that when the `PFD` mode is selected and the selector knob is set to `ALT` and the `APR` button is pressed, it should either mute or unmute the speaker. The second line specifies the same conditions as the first, but here it relates to the led light. So here it checks the value of the dataref `sim/cockpit2/radios/actuators/audio_selection_com1` and if the value is 0, then it should turn off the led light. Note that the led light specification is optional and you can see in line 5-6 that there is no led specified. This will result in the text color in the Bravo++ window being a different color (green-blue) than those with led lights. The last line shows how to specify the toggling of the INNER/OUTER mode using a Bravo++ dataref.

Let's look at another example from the King Air C90B configuration.

```
SYS_HDG_HDG_BUTTON = "laminar/c90/electrical/switch/auto_ignition_L"
SYS_HDG_HDG_BUTTON_LED = "sim/cockpit2/annunciators/igniter_on,0,1"
SYS_HDG_NAV_BUTTON = "laminar/c90/electrical/switch/auto_ignition_R"
SYS_HDG_NAV_BUTTON_LED = "sim/cockpit2/annunciators/igniter_on,0,2"
SYS_HDG_APR_UP_BUTTON = "laminar/c90/powerplant/switch/autofeather_switch_up"
SYS_HDG_APR_DOWN_BUTTON = "laminar/c90/powerplant/switch/autofeather_switch_dn"
SYS_HDG_APR_BUTTON_LED = "sim/cockpit2/switches/prop_feather_mode,0"
SYS_HDG_IAS_BUTTON = "FlyWithLua/Bravo++/switch_mode_button"
```

So the first line specified the command to turn on the left engine's auto ignition and the second specifies the dataref to check for the led condition, but notice that there are two numbers specified. Here the dataref is an array that contains more than one value, so on this case we need to specify an index starting from 1, so that we know which value to compare to. So in this case it will check if the value in the first place in the array is equal to 0. If it is it will turn off the led light. Looking at line 4 you see the same dataref, but this time it specified 2 instead of

1. This is because in line 3 we turn on the right engine's auto ignition so we need to check the corresponding value in the array.

On line 5 -6 we see an example of a switch where we specify the dataref for the UP and DOWN command. This will result in a slightly different rendering of the button in the Bravo++ window where you will either see a ^^ above or vv below the button. Line 7 shows the Bravo++ dataref for toggling between UP or DOWN when using switches.

Actions for rocker switches and leds

The Honeycomb Bravo has 7 rocker switches that can be assigned 2 dataref commands each. The principles of assigning datarefs is pretty much the same as with the buttons.

Let's look at an example for the DA42:

```
SWITCH1_UP="sim/ice/pitot_heat0_on"
SWITCH1_DOWN="sim/ice/pitot_heat0_off"
SWITCH1_LED = "aerobask/sw_pitot_heat,0"

SWITCH2_UP="aerobask/eng/master1_up"
SWITCH2_DOWN="aerobask/eng/master1_dn"
SWITCH2_LED = "aerobask/eng/sw_master1,0"

SWITCH3_UP="aerobask/eng/master2_up"
SWITCH3_DOWN="aerobask/eng/master2_dn"
SWITCH3_LED = "aerobask/eng/sw_master2,0"

SWITCH4_UP="sim/electrical/battery_1_on"
SWITCH4_DOWN="sim/electrical/battery_1_off"
SWITCH4_LED = "sim/cockpit/electrical/battery_on,0"

SWITCH5_UP="sim/systems/avionics_on"
SWITCH5_DOWN="sim/systems/avionics_off"
SWITCH5_LED = "aerobask/sw_avionics,0"

SWITCH6_UP="aerobask/eng/fuel_pump1_on"
SWITCH6_DOWN="aerobask/eng/fuel_pump1_off"
SWITCH6_LED = "sim/cockpit/engine/fuel_pump_on,0,1"

SWITCH7_UP="aerobask/eng/fuel_pump2_on"
SWITCH7_DOWN="aerobask/eng/fuel_pump2_off"
SWITCH7_LED = "sim/cockpit/engine/fuel_pump_on,0,2"
```

Each switch is distinguished by a number and then by either appending "_UP", "_DOWN" or "_LED". The first two just specify a command dataref to activate depending on whether the switch is up or down. The third is a datref to monitor state in order to determine whether the "led" for the switch on the Bravo++ window should be turned off.

Annunciator and gear leds

The annunciator leds and gear leds are pretty straight forward and hopefully this example from the King Air C90B should be clear enough:

```
GEAR_DEPLOYMENT_LED = "sim/flightmodel2/gear/deploy_ratio,0"

MASTER_WARNING_LED = "sim/cockpit2/annunciators/master_warning,0"
FIRE_WARNING_LED = "sim/cockpit2/annunciators/engine_fires,0"
OIL_LOW_PRESSURE_LED = "sim/cockpit2/annunciators/oil_pressure_low,0"
```

```
FUEL_LOW_PRESSURE_LED = "sim/cockpit2/annunciators/fuel_pressure_low,0"
ANTI_ICE_1_LED = "sim/cockpit2/ice/ice_pitot_heat_on_pilot,0"
ANTI_ICE_2_LED = "sim/cockpit2/ice/ice_pitot_heat_on_copilot,0"
STARTER_ENGAGED_LED = "sim/cockpit2/engine/actuators/starter_hit,0"
APU_LED = "sim/cockpit2/electrical/APU_running,0"

MASTER_CAUTION_LED = "sim/cockpit2/annunciators/master_caution,0"
VACUUM_LED = "sim/cockpit2/annunciators/low_vacuum,0"
HYD_LOW_PRESSURE_LED = "sim/cockpit2/annunciators/hydraulic_pressure,0"
AUX_FUEL_PUMP_1_LED = "sim/cockpit2/fuel/transfer_pump_left,0"
AUX_FUEL_PUMP_2_LED = "sim/cockpit2/fuel/transfer_pump_right,0"
PARKING_BRAKE_LED = "sim/cockpit2/controls/parking_brake_ratio,0"
VOLTS_LOW_LED = "sim/cockpit2/annunciators/low_voltage,0"
DOOR_1_LED = "sim/flightmodel2/misc/canopy_open_ratio,0"
DOOR_2_LED = "sim/flightmodel2/misc/door_open_ratio,0"
DOOR_3_LED = "sim/cockpit2/annunciators/cabin_door_open,0"
```

Each led has its own unique name that matches the corresponding led ligh on the Honeycomb Bravo. Just like in the previous section the value of the led will be a dataref and a value to check against. The only additional feature we have here is that we can tie several datarefs to the same annunciator led. This is done by adding a _# between the annunciator name and the trailing _LED. So this means that if any of these datarefs have a value other than what is specified the led light will light up.

Note that the GEAR_DEPLOYMENT_LED will most likely always be this value for retractable gears. For fixed gear aircraft you should not specify the GEAR_DEPLOYMENT_LED in the configuration.

The trim wheel

The trim wheel overrides the default values used in X-Plane to better simulate a manual trim wheel. The two parameters that can be set are TRIM_INCREMENT and TRIM_BOOST. The value of the trim is any value between -1 and 1, so the trim increment specifies how much one "click" of the wheel will change the value of the trim. The default value, if nothing is specified, is 0.01.

The trim boost is a value that is applied to the trim increment if the wheel is turned quickly. So if there are many "clicks" in short succession, the trim increment will be multiplied by the trim boost value. The default, if not specified, is 3.

To specify your own values you can just add the follwoing to the config file:

```
TRIM_INCREMENT=0.005
TRIM_BOOST=6
```

Troubleshooting

If the Bravo++ window does not appear or the window has no buttons or the selector values don't change when toggling/scrolling, do the following:

- Open the log.txt file under the X-Plane directory and search for "BRAVO++ ERROR"
- If no Bravo++ error is found, look for other possible issues in the log.txt file
- If all else fails, send me a PM on the X-Plane forum or create an issue on GitHub with the log.txt and config file if you are not using one of the examples, and I will do my best to help out.

The most common issues are:

- The Honeycomb Bravo wasn't found; is it plugged in?
- The button number assigned to the `alt_selector_button` is wrong; make sure that you select the `ALT` in teh selector before noting the number.
- The name is invalid; i.e. you have used the wrong combination of `mode name` + `selector name` + `button name`.
- The dataref doesn't exist. You will usually see this in the log.

Known bugs

I am aware of some minor annoying bugs:

- The button for switching the com frequency doesn't work all the time. You just have to be persistent and press it multiple times. This doesn't happen with the nav frequency, so I am not sure why it's an issue with the com frequency.