PangoScript

Here you can find different example Pangoscripts to help you see different ways scripting can be used. They also should just work if copy and pasted into BEYOND, as-is, or modified to your needs.

Pangolin Internal Examples

These examples were made by internal team members at pangolin as examples for pangoscripts

Timeline AutoSave

```
CodeName "Autosave"
TimelineQuickSave
Sleep 60000 //Wait 1 Minute
Restart
```

This code is designed to create an autosave script for your timeline, since saving does pause output, it's not recommended to leave this on for operating your show. The Sleep is in milliseconds, so 60,000 is 60 seconds, you can change the timeframe to your desired save duration, just know it does save a separate backup, so more frequent saves will eat up more storage.

Daily Schedule

```
CodeName "Daily Schedule"
Autostart
WaitForTime 19,45,0,0 //Wait for 7:45
StartCue 1,1
WaitforCueStop 1,1
StartCue 1,10
WaitforCueStop 1,10
StartCue 1,21
WaitforCueStop 1,21
StartCue 1,41
WaitforTime 23,0,0,0
StopCue 1,41
WaitForTime 23,59,59,0
Sleep 10000
RunApp "C:\Windows\System32\shutdown", "-r -t 10" // full exe file name,
parameters
//These parameters on this shutdown, sets it to "restart" then "Time 10
seconds" These options can be found on the web.
```

```
ExitBeyond
```

This code is an example for an install where you want a sequence of cues (maybe timelines saved to cues) to run at specific times in sequence, And at the end the script waits for the clock to turn over, then restarts the machine, if you put beyond in the windows startup folder, it will auto start beyond, the script and start the cycle. You will need the “RunApp” command enabled, which is in configuration, and will also need to check the box which doesn't ask for confirmation on exit of beyond in the configuration window for this to be fully automatic, an example of automation scripting.

Flip Zones

```
CodeName "Flip Zones > - <"
Zone.0.SizeX = -100
Zone.1.SizeX = -100
Zone.2.SizeX = -100
Zone.3.SizeX = -100
Zone.4.SizeX = 100
Zone.5.SizeX = 100
Zone.6.SizeX = 100
Zone.7.SizeX = 100
Master.DISPLAYPOPUPTIMEOUT=.5
displaypopup "> - <"
Exit
```

This code exploits the fact that each zone has parameters associated with it on a level that is usually not seen by the user, technically there are these parameters, and also GEO parameters, these parameters are content values, and geo is for zoning, it's important to not use the GEO ones, as that will move your zone. These just affect the content within a zone. These parameters are also the same as what BEYOND server modifies, and by using numbers of zones instead of the names, it won't matter what the zone names are, just that they are in order. It also does a popup to let you know what you have selected, something that may be useful for a midi controller trigger, where the button has no label.

Pull Tempo from DMX

```
Codename "Pull Tempo From DMX"
WaitForDmx 501 //Channel at 1 will run code
beattap
sleep 200 // Make DMX reset to zero on console less then 100ms
restart
```

This code is imperfect, but it will tap the beat tap when the channel selected “updates” any change in value, so if you have a flash button with a cue that just sets the channel to full and off when you release, it will tap the temp, the sleep is to wait for your tap on the console to reset, 200ms is enough to still get 200bpm out of your taps, and a safe spacing for the reset. You could macro on your console to automatically do this from a speed master or just use it to tap the bpm. To make this code better, you

would really want a conditional trigger, where when it hits value of 255 it taps but, this works and is much simpler of a code.

Restart Show Softly

```
Codename "Restart Show Soft"
AnimateProp "Master.AudioVolume", 100,0,5000
sleep 5000
TimelineSetPos 0
Master.AudioVolume = 100
TimelinePlay
Exit
```

This script demonstrates the ability to animate objects over time, Animate Prop goes from value1, to value 2, over value in ms. In this case, a code is written to restart a timeline, but to fade out the audio before doing so. You could animate Volume and master brightness at the same time which could be interesting too. Then resets the values by just setting their value and plays the show again.

Control Color Channels From DMX

One way to use your DMX console to control colors is by using RGB Generic profiles and linking those channels directly to the RGB objects of color channels in BEYOND. In Settings> DMX Settings> DMX to Pangoscript settings, you can load the following file to link the first 12 channels in your BEYOND install to these.

The script itself looks like the following, where “0” and “R” are replaced with channel numbers, and R,G,B respectively. This in conjunction with something like “Lyra's Magical Color Changing Workspace” linked below, you can recolor your entire workspace live from your console.

```
ColorChannel.0.R = ExtValue(0,255)
```

Examples from Users

The following examples are from the userbase, at the permission of the users. Scripts many not have been validated by pangolin internally, and opinions expressed, and explanations are directly from the user not pangolin.

Jeff Vyduna

Knob or fader to QuickFX 2^X speeds

```
// Quantize a continuous input (fader/potentiometer) to 2^x beat scaling in
FX.

// Take a 0-127 Midi value and select a 2^i multiplier
 //  for i in [-4, 4]
// Assumes a nominal FX duration of 2 beats to work best
// by producing multipliers that result in 32 beats at
// knob minimum (MIDI 0), 2 beats at knob center,
// and 1/16 of a beat (thirtysecond notes) at MIDI 127.

// If you want to consume the current metronome multiplier, add the next
// two lines to your MIDI device initializstion script in MIDI settings.
GlobalVar gMetroAx
gMetroAx = 0

Var metroScaleVal
metroScaleVal = 0
metroScaleVal = ExtValue(0,8.999) // Scale incoming MIDI value to 0..9

// Uncomment for dubugging values to the laser preview window
// DisplayPreview "Value:", metroScaleVal, 0xffffff


var metroAx // Metronome Multiplier (doc misnomer? Seems like a divisor)

// Assumes the FX duration is set to 2 beats.
// int(metroSlider): Beat duration, multiplier
// 0-1: 32 beats, 1/16
// 1: 16 beats, .125
// 2: 8 beats, .25
// 3: 4 beats, .5
// 4: 2 beats or nominal - 1<<4 /16 = 1x multiplier;
// 5: 1 beat: multiplier of 2
// 6: 1/2 (eigth note, *8 displayed), 4
// 7: 1/4 (sixteenth note, *16 displayed), 8
// 8: 1/8 (thirth-second note, *32 displayed), 16
metroAx = max(1 << int(metroScaleVal), 1) / 16

// DisplayPreview "metroAx", metroAx, 0xffffff

gMetroAx = metroAx // Make value available to other scripts

// Uncomment which QuickFX layers to apply the new metronome multipliers to.

// FXTimeScaleAx 4, 2, metroAx  // Layer 1 = Color 1
// FXTimeScaleAx 5, 2, metroAx  // Layer 2 = Color 2
// FXTimeScaleAx 6, 2, metroAx  // Layer 3 = Zone
```

```
FXTimeScaleAx 4, 2, metroAx  // Layer 4 = Intensity, Mask (1,2,3) 2=Metronome,
multiplier
FXTimeScaleAx 5, 2, metroAx  // Layer 5 = Structure
FXTimeScaleAx 6, 2, metroAx  // Layer 6 = Movements


// Display duration / period in beats on a Kontrol F1 7-seg display
// Second argument to MidiOutLong should be 41 for the controller's unshifted
page layer,
// or 78 for the shifted display. Since I map this to a shifted page, I'm
using 78.
if (metroAx <= 2) MidiOutLong(0xBC, 78, int(2.0 / metroAx)))
if (int(metroAx) = 4) MidiOutLong(0xBC, 78, 108)
if (int(metroAx) = 8) MidiOutLong(0xBC, 78, 116)
if (int(metroAx) = 16) MidiOutLong(0xBC, 78, 132)

exit
```

Takes a fader or rotary pot on a MIDI controller, such as on a APC40mk2 or Traktor F1, and scale the global cue speed to snap to a 2^X speed multiplier (25%, 50%, 100%, 200%, 400%, etc). For example, if you've made cues with a 4 beat duration, this lets you us a MIDI control to instantly change their duration to 8, 16, or 32 beats (as well as 2, 1, 1/2, 1/4 of a beat).

Rotary relative encoder to 2^X cue speeds

```
// Quantize an relative encoder input to 100*2^n cue speed.

// SUMMARY

// Takes MIDI values 1 (incr) and 127 (decr) to select a 2^i speed
// multiplier for i in [-4, 4]. This produces multipliers of:
//  100% * [1/16, 1/8, 1/4, 1/2, 1, 2, 4, 8, 16]


// DETAILED DESCRIPTION

// This script lets you set your cue speed to multiples of 2. For example.
// if you had a cue designed for 1-beat duration, you can snap instantly to
// scaling the duration slower to 2, 4, 8 beats, or faster to 1/2, 1/4,
// etc., by setting MasterCueSpeed to 200%, 400%, 50%, etc.

// The script is configured to use input from a rotary incremental encoder
// in "relative" mode. These limitless-rotation encoders typically have a
// "click" feel during rotation, and send out MIDI valies of 1 (for
// clockwise increment) and 127 (counterclockwise decrrement).
```

```
// I commonly use a Traktor Kontrol F1 which has its default (unshifted)
// encoder assigned to MIDI Control Channel 13 value 41 (hex BC 29 xx),
// so this script is pasted into the MIDI-to-PangoScript slot there.
// This controller also allows you to push the encoder down as a momentary
// button. I use this to reset the master cue speed to 100% on receiving
// value 63. On Traktor, you configure this value (and turn-relative mode)
// in the Controller Editor software.

// This script also displays the current effect duration in beats on the
// MIDI contorller. The Traktor F1 has a 2-digit numeric display that I've
// configured to display values sent to it on the same MIDI channel.

// To store the current duration between inputs, you must use a Global
// Variable. To set this up, paste the following (uncommented) in your
// MIDI Controller's initialization script:

// GLOBALVAR gCueSpeedFrac
// gCueSpeedFrac = 1.0



var cueSpeedDelta
cueSpeedDelta = 0

// Scale incoming MIDI value to decr = -1, push = 0, incr = 1
cueSpeedDelta = ExtValue(1,-1)

// Uncomment for debugging. Displays text on the laser preview window.
// DisplayPreview "cueSpeedDelta:", cueSpeedDelta, 0xffffff

if (cueSpeedDelta > 0.5) gCueSpeedFrac = gCueSpeedFrac * 2 // Knob incr
if (cueSpeedDelta < -0.5) gCueSpeedFrac = gCueSpeedFrac / 2 // Knob decr
if ((cueSpeedDelta >= -0.5) & (cueSpeedDelta <= 0.5)) gCueSpeedFrac = 1 //
Knob push -> reset
if (gCueSpeedFrac > 17.0) gCueSpeedFrac = 16
if (gCueSpeedFrac < .06) gCueSpeedFrac = .0625

// Uncomment for debugging:
// DisplayPreview "gCueSpeedFrac:", gCueSpeedFrac, 0xffffff

// Display beat duration (period) on a Kontrol F1 7-segment display.
// This assumes a default effect duration of 1 beat, so 50% speed runs FX
// for 2 beats, and the display will show "2". 200% speed implies a 1/2 beat
// duration, but because there's no "/" character possible I add the
// leading period (.) indicator: ".2" means "1/2 beat", ".4" = 1/4 beat, etc.

var cueSpeedDispVal
cueSpeedDispVal = 0
cueSpeedDispVal = round(1.0 / gCueSpeedFrac) // .5 = 50%, display period of
```

```
"2"

// Display faster speeds over 100% as fractions of a beat.
// 102 = "(dot)2". 4 = 400%, send 104, display "(dot)4" (period is 1/4th
normal)
if (gCueSpeedFrac > 1.0) cueSpeedDispVal = 100 + round(gCueSpeedFrac)
MidiOutLong(0xBC, 41, cueSpeedDispVal)

// Finally, set the actual cue speed. You may wish to use AnimationSpeed
// instead, or MasterFXSpeed to only speed up FX and not cues.
MasterCueSpeed 100 * gCueSpeedFrac

exit
```

Takes a rotary incremental (relative mode) encoder on a MIDI controller, such as on a APC40mk2 or Traktor F1, and scale the metronome for beat-based effects in selected QuickFX rows to snap to a 2^X time scale multiplier. For example, if you've made effects with a 2 beat nominal duration, this lets you twist a knob and instantly change their duration to 2, 4, 8, or 16 beats (as well as 1, 1/2, 1/4, or 1/16th of a beat).

Advance a QuickFX, toggling through multiple layers

```
// Advance the currently running QuickFX from the color layers,
// which are layers 1 and 2 in many QuickFX layouts.
// Use this script in a MIDI-to-PangoScript slot.

// As configured below, it will toggle advance through the first 24 colors
// in layer 1, then in layer 2, then back to layer one (including a state
where both layers are off)

// Scripts like this that store some state in a golbal variable should have
// that variable defined in the controller initialization script. Copy these
these and uncomment:
// GLOBALVAR gColorFXIdx
// gColorFXIdx = 0

gColorFXIdx = gColorFXIdx + 1 // advance on MIDI trigger
gColorFXIdx = gColorFXIdx % 49

MidiOutLong(0xBC, 41, gColorFXIdx) // Display on Traktor F1 the color index

if (gColorFXIdx > 24) GOTO Layer2

SetFX 2, 0
SetFX 1, gColorFXIdx
```

```
exit


Layer2:

SetFX 1, 0
SetFX 2, gColorFXIdx - 24

exit
```

Many workspaces keep the default scheme of having the first two QuickFX layers apply colors. On some compact MIDI controllers, I've wanted to have a single button advance through the effects in a row. This script advances through the first 24 colors on layer 1, then disables layer 1 and advances through the first 24 effects in layer 2, finally ending with a state where both layers are off.

Relative encoder for fine Channel adjustment

```
// Fine control from a relative encoder
// Place this script in a MIDI-to-PangoScript slot.
// and be sure to change the channel number in several places below

// Channels are displayed in the UI as 0 to 100
// but can be set as 0-1000 values using ChannelOut
// Channel.<N>.Value is a fractional value 0...1
VAR ChannelNumber, ChVal
ChannelNumber = 5
ChVal = int(Channels.5.Value * 1000)
//DisplayPreview "Ch " + intstr(ChannelNumber) + " name iss: " +
Channels.1.Name
//DisplayPreview "Ch " + intstr(ChannelNumber) + " value (0-1000) was: " +
intstr(ChVal)

VAR MidiVal
MidiVal = ExtValue(0,127)
DisplayPreview "MidiVal: " + intstr(MidiVal)

if (MidiVal > 64) GOTO Incr

if (ChVal <= 0) GOTO Done
ChannelOut ChannelNumber, int(ChVal - 1)
//DisplayPreview "Decremented"
GOTO Done

Incr:
if (ChVal >= 1000) GOTO Done
ChannelOut ChannelNumber, ChVal + 1
```

```
//DisplayPreview "Incremented"

Done:
//DisplayPreview "Ch " + intstr(ChannelNumber) + " value set to: " +
intstr(ChVal) + "/1000"
```

Beyond channels show up as values between 0 and 100 in the Channels tab, but are internally stored as a normalized 0..1 float. This makes them useful tools for linking to any parameter that takes an input. Sometimes you want much finer control over a parameter than 127 values from MIDI (and few controllers support 14-bit MSB+LSB). You may have experienced this when linking MIDI controllers to Z rotation - it's steppy and I've resorted to enabling physics to dampen the motion which helps a little but is still imprecise. I was recently preparring for a film shoot where I needed fine control over some large value ranges (precise points counts in a resampler to help tune rolling shutter banding). By configuring my rotary encoder to send incremental up/down values, you can route it to parameter inputs through Channels using arbitrary fine resolution using the internal float value.

Select Groups mode Destination cues from APC40 mkii Track/Clip Stop buttons

```
// Activate a Destination cue (Groups) on APC40 mk2's Track Select button or
Clip Stop button

// Requires GlobalVar lastDestinationCueNumber - Define these in the
controller init scripts!
// GlobalVar lastDestinationCueNumber
// lastDestinationCueNumber = 0

// For this script, Destination Cues page MUST be in page 1 of the whole
workspace
// Select a desitnation cue; CueDown is the only trigger possible -- MouseDown
doesn't seem to work
// Cancel all other track and clip stop indicator LEDs
// Illuminate the correct light
// Handle if a destination cue was clicked again by holding internal state

// You must set this correctly for each button you add this script to
Var destinationCueNumber
destinationCueNumber = 1

Var isSecondRow
isSecondRow = (destinationCueNumber > 8) & (destinationCueNumber <=16)

// StartCue doesn't work for Destination cues
CueDown 1, destinationCueNumber // Page 1 must be "Destinations" Cue sheet

// Turn off all lights in this group
// Using a loop was too slow for the device or PangoScript. :-/
```

```
MidiOut 0x90, 0x33, 0x00
MidiOut 0x91, 0x33, 0x00
MidiOut 0x92, 0x33, 0x00
MidiOut 0x93, 0x33, 0x00
MidiOut 0x94, 0x33, 0x00
MidiOut 0x95, 0x33, 0x00
MidiOut 0x96, 0x33, 0x00
MidiOut 0x97, 0x33, 0x00

MidiOut 0x90, 0x34, 0x00
MidiOut 0x91, 0x34, 0x00
MidiOut 0x92, 0x34, 0x00
MidiOut 0x93, 0x34, 0x00
MidiOut 0x94, 0x34, 0x00
MidiOut 0x95, 0x34, 0x00
MidiOut 0x96, 0x34, 0x00
MidiOut 0x97, 0x34, 0x00



// Turn on this button's light
MidiOut 0x90 + ((destinationCueNumber - 1) % 8), 0x33 + isSecondRow, 0x7F //
33 = track row, 90 where 0 is first
light

// Handle repeat press of same cue
if (lastDestinationCueNumber = destinationCueNumber) Goto
ToggleThisDestination

destinationPlaying = 1 // Global. A new destination was clicked, so it's
playing

Goto FinishUp



ToggleThisDestination:
destinationPlaying = 1 - destinationPlaying // global
if (destinationPlaying = 0) MidiOut 0x90 + ((destinationCueNumber - 1) % 8),
0x33 + isSecondRow, 0x00

FinishUp:
lastDestinationCueNumber = destinationCueNumber
```

I like Groups mode, though it still has several bugs around destination/chase cues being brittle. I like the default assignment of zone destinations, and there seem to be many effects across zones that can only be accomplished with MultiFX. However, you cannot select destination cues from MIDI controllers. This is likely a temporary bug that will eventually be fixed, but I wanted to be able to select them from an APC40 mkii now. Since the Clip Stop and Channel Select rows are mapped to Zone Mute and Output by default, I

reassigned those buttons to use this script, which also manages the LED state. Unfortunately it currently requires your page of destination cues to be the first page in your workspace, but there's likely some way around this limitation with more scripting.

Return to Guided learning