PangoScript Commands

New for 2026, a tool has been created with all new fresh documentation of every Pango script Command. The tool linked below allows you to read every command as well as see examples the script commands in use. The tools is searchable with the search bar on the top.

Click the link below to review this tool which will open in a new window.

Click here to view the PangoScript command list.

Information below is being rewritten

```
==== Intro ====


Currently, the main purpose of "scripting" is ability to add custom relation
on various events, like a MIDI, DMX, ArtNet, Channels. The script typically
has a few lines, and a line has one Command. Command may have a parameters.
Typically parameter is a number (constant).


==== Numbers ====

Supported float point , integer and hexadecimal numbers. As example:
100
-20
1.01
0xBF
All hexadecimal numbers much have 0x prefix, similar to C language, but
without ending H

There is not strict separation on integer and float point numbers.

Separator between command parameter could be space (" ") or a comma (",")


==== Special characters (separators) ====


 '.', ';', ',', ':', '!', '', ''', '^', '+', '-', '*', '\', ' ',
 '`', '[', ']', '(', ')', '{', '}', '?', '%','|','&','=':
```

```
==== Predefined constants ====


There are a few constants mostly for readability of the code. Each constant
will be transformed to a number
TRUE - numeric analog 1
FALSE - numeric analog 0
ANY equal to numeric -1. Used in a few WaitFor command
OFF equal to numeric 0
ON equal to numeric 1
TOGGLE equal to numeric 2
AsIs equal to -2.
This is required for commands like "MasterPause". I hope it should be handy to
see ON or OFF instead of 0 or 1.

A few examples of using commands:
  VirtualLJ off
  MasterPause Toggle
  WaitForMidi 0x90, 10, Any

==== Math operations (expressions) ====



Standard: +  -  /   *  %
Inversion: !
bit OR: |
bit AND: &
bit XOR: ^
bit right shift: >>
bit left shift: <<


==== Operators ====



IF operator.

Syntax: if ( expression )  operator

expression - covered with brackets. Must be an expression with
numerical result that gives "0" or not a "0". So, for numerical
variable it's possible to write "if (variable) ...".

Compare operators are: ">" ">=" "<" "<=" "=" "<>". They produces numerical
result "0" or "1".
```

```
You can combine comparing operations with "&" and "|" bit-wise operators (but be sure that left and right side are "0" or "1"). It's recommended to cover complex operations into brackets - like "if ((1>2) & ((3+2)>1))"

operator will be executed if condition gets non-zero result. If you want to place other operator after "if (condition) operator" - it must be divided with ";" - but it's better to place next operator on the next line ;-)

=== GOTO operator ===


Syntax:  goto label

"label" is  a name of position for a jump. The label can be placed before any operator and separated by ":"  character. As example:

mylabel: WaitForBeat 4
... do something...
goto mylabel


Operator GOTO can work with the labels inside the string variables. It means that you declare string variable, assign a label name to variable, and then you a variable in GOTO.
Example:

var s
s="mylabel"
goto s

DisplayPopup "It does not work"
exit

mylabel:
DisplayPopup "It works"




=== VAR operator ===


PangoScript allow define local variables. The lifetime for the local variable defined by lifetime of the Scripter that execute PangoScript. As soon as the Scripter freed, all its local variables are removed as well.

All variables must be declared before the using. No need to specify the type of variable. The variable automatically adjust the type depending on value.
```

```
Internally supported integer, float and string variables.
The declaration start from VAR operation and the follow one or more variable
names. As example:

var MyVariable
var a,b,c

Before using the variable must be initialized by some value. Otherwise BEYOND
will generate error and stop script execution. Example:
var MyInteger, MyString
MyInteger = 10
MyString = "Hello!"

All variables declared as VAR are local.


GLOBALVAR operator
The variable can be declared as global. In this case, it visible in ALL
scripts of BEYOND.




==== General functions ====


intstr(value:number):string - transform number to string
value - number or float number
result - string

floatstr(value:number):string - transform number to string
value - number
result - string

abs(value:number):number - value by modulus
value - integer or float number
result - same type as argument. result is absolute value

int(value:number):integer - return integer part of float value
value - a float point number

frac(value:number):float - return fractional part of float point value
value - float

round(value:float):integer - return rounded float point value (to integer)
value - a float point number

sqr(value:number):number - return a square of argument
```

```
value - a float point number or integer

sqrt(value:number):float - return a square root of argument
value - a float point number or integer

cos(value:float):float - co-sinus
value - a float point number or integer. angle in radians

sin(value: float):float - sinus
value - a float point number or integer. angle in radians

tan(value:float):float - tangents
value - a float point number or integer. angle in radians

arcsin(value:float)float - arc sinus
value - a float point number or integer. angle in radians

arccos(value: float):float - arc cosinus
value - a float point number or integer. angle in radians

arctan(value: float):float - arc tangents
value - a float point number or integer. angle in radians

arctan2(dx, dy:float):float - arc tangents
dx, dy - ....

min(a,b:number):number - return a minimum of two numeric values
a,b  float or integer

max(a,b:number):number - return a maximum of two numeric values
a,b  float or integer

pi:float - return PI value - 3.1415926...

invert(value:number):integer - invert value. Boolean operation, but can be
used with float point numbers. If value more than 0.5 then function return 0,
otherwise return 1.

==== Date and Time ====


now:float - date&time, calls now() function of Delphi

tickcount: integer - return number of millisecond from start of PC.

hms(hour, minute, second):integer - transform hour, minute and second into
seconds.
```

```
GetYear:integer - function return current year by PC clock. Result

GetMonth:integer - function return current month by PC clock

GetDay:integer - function return current day by PC clock.

timestr(now:float) :string
now - is variable representing time. Function return string with time in short
format such as "11:53", without seconds

timestrlong(now:float):string -
now - is variable representing time. Function return string with hours,
minutes, seconds, such as "11:53:10"

datestr(now:float):string - ********* ******* ****, * *** *** -
now - is variable representing date. Function return string short date format
such as "21.11.2012"

datestrlong(now:float):string - ******** ******* ****, **** -
now - is variable representing date. Function return string long date format
such as "21 ******* 2012 *."

dayofweek(now:float):string - short version day of the week

dayofweeklong(now:float):string - long version day of the week

==== String functions ====


uppercase(string):string  - transform input string to upper case. Result is a
string.

lowercase(string:string):string - transform input string to lower case. Result
is a string.

crlf:string - return a string, line separator (13,10)



==== Clock And Metronome ====


b2s(beats) - transform beats to seconds

b2ms(beats) - transform beats to milliseconds

s2b(seconds) - transform seconds to beats
```

```
b2s(seconds) - transform beats to seconds


==== Functions ====


There are a few functions for access of incoming data

Dmx ( Channel, OutputMin, OutputMax )
Parameters:
Channel - index of DMX channel, acceptable value from 1 to 2048
OutputMin, OutputMax - defines the range resulting value of the function.
Oupout value will start from OutputMin and increase up to OutputMax
result  = OutputMin +  (DmxChannelValue / 255) * (OutputMax-OutputMin)

Example:
  Position Dmx(10, -100, 100), Dmx(11, -100, 100), Dmx(12, -100, 100)

in this example we use Position command that has 3 arguments - X,Y,Z and we
use DMX IN values (three channels, 10,11,12), and map the values range -100 to
+100.

Dmx ( Channel  )
or
Dmx( Channel, OutputMin, OutputMax)
Parameters:
Channel - index of DMX channel, acceptable value from 1 to 2048
Result of the function is value of DMX channel as it is, without anr range
adjustments

OutputMin, OutputMax - defines the range resulting value of the function.
Output value will start from OutputMin and increase up to OutputMax.
Internally channels are normalized to 0...1 range
result  = OutputMin +  (ChannelValue) * (OutputMax-OutputMin)
Example:
  DisplayPopup Dmx(10) // display the value of 10th DMX channel.

Note: version with min/max used for simplification of use of function in
operators where you may need to transform DMX value range (0..255) to some
other range. As example Size operator, you may want to use size range like
0...100, or 10...100, or -100..100.

Channel ( Channel, OutputMin, OutputMax )
or
Channel( Channel )

Parameters:
Channel - index of Channel, acceptable value from 1 to 255
```

```
OutputMin, OutputMax - defines the range resulting value of the function.
Output value will start from OutputMin and increase up to OutputMax.
Internally channels are normalized to 0...1 range
result  = OutputMin +  (ChannelValue) * (OutputMax-OutputMin)

If function has one argument then result is normalized channel value (range
from 0 to 1)

Example:
  Size Channel(10, 5, 100), ChannelIn(10, 5, 100), ChannelIn(10, 5, 100),

in this example we use Size command that has 3 arguments - X, Y,Z and we use
Channel 10 value for all 3 axis.

Channel ( Channel )
Parameters:
Channel - index of Channel, acceptable value from 1 to 255
Result of the function is channel value. Channel value is normalized, the
range from 0 to 1.



CC (Channel, Controller, OutputMin, OutputMax)
Get a value of of MIDI controller. BEYOND memorize all values of incoming MIDI
controllers (Contrl Change Message) and you can get an access to it.
Channel - value 0..15
Controller - value 0..127. Thsi is Data1 in MIDI messages
OutputMin, OutputMax - defines the range resulting value. Result  = OutputMin
+  (ChannelValue/127) * (OutputMax-OutputMin)



ExtValue( OutputMin, OutputMax )
There are a few pretty big tables of code-sources. A tables such as DMX,
ArtNet, ControlChange, PitchBand, NoteOn initiated by channel/message that has
a value. This function allow get access to the channel that intiate the code.
Better explain on example. Lets say I want to connect DMX channel to Master
Live Control, Position X. In this case, code that will do ths is this:
    PositionIndex 1, ExtValue(-100,100)
In this example PositionIndex is a live control command, see below. 1 is index
of axis. And finally ExtValue() is a fuction that take DMX channel value in
map to rangle -100 to 100.

ExtDelta( Delta )
Function equal to ExtValue but made specially for MIDI encoders. There are
many controlled with wheels and knobs that generate Control Change message and
the data2 value is 00 or 7F. That is all, no values in between. For such cases
has a sense to use "delta" version of commands and ExtDelta() for getting
exact value of delta. Function return -Delta or +Delta. Only two values, no
exceptions. Example of use
```

```
PositionDelta 0, ExtDelta(1), 0   // move vertically.

another example:

AngleDelta 0,0, ExtDelta(45) // rotate by Z on 45 degrees.

Random( MaxValue )
Function return random value in range 0..MaxValue. Note, MaxValue included
into the range

RandomIn( MinValue, MaxValue )
Functino return random value in range MinValue..MaxValue

Param( ParamIndex ) - newer name of OscParam() function. It get the value of
parameter supplied into the script. See commend about OscParam()

OscParam( ParamIndex )
Specialy designed for use in "OSC to CODE" table.. When BEYOND receive OSC
message and supply it into interpreter to execution, then we can supply up to
10 parameters with OSC message. BEYOND put them into local array, and you may
use it inside the Code. Note, data is there only during execution of current
script. As soon  we exit from execution of this script, data will be lost. OK,
parameter index is from 1 up to 10. BEYOND check how many parameters the OSC
message has, and if you will try to access non existing paramater then script
will stop with error. So, if OSC message has 3 parameters, then you can use
index from 1 to 3. Accepts f, i, and s type of OSC parameters. You can freely
mix float and integer.

ParamRange( MinValue, MaxValue) function return TRUE (1)  if the first
parameter inside the specified range.

ParamRange(ParamIndex,  MinValue, MaxValue) function return TRUE (1)  if the
specified parameter (pram index) inside the specified range (from MinValue to
MaxValue)

GetTransitionName( index )
function return string, the name of transition. Index is number of transition.
Range 0..23

BeatTime
Function has no arguments. The result of function is float point number.
Integer part of function is number of beat from the start of BEYOND. The
fractional part is progress inside the beat.

GetMidiDeviceIndex
function return currently selected MIDI Device pair, Value range 1..4.

ObjectExists( AObjectName )  - function get object name a s string and return
```

```
1 if object exists, or zero if object is not found.


==== Timecode input ====


GetTimeCode - return latest timecode value as a float point value, value in
seconds

GetTimeCodeTick - return the tick when the last timecode value arrived. Use
function GetTick for getting the current tick value.

==== Cue related functions ====


CuePlaying( PageIndex, CueIndex) - function return 1 if cue is currently
playing in the Grid, otherwise return 0. Indexing of page and cue starts from
1.
Example: CuePlaying(1,1) //return state of first cue on a first page. Note,
function is not applicable for ProTracks and DMX server.

CueEmpty(PageIndex, CueIndex) - function return 1 if cue is empty, otherwise
return 0. Indexing of page and cue starts from 1.

GetCueCaptionColor(PageIndex, CueIndex)  - return cue caption color in GDI
format. Format is 24bit RGB, red is LSB. Overall formula Blue*256*256 + Green
* 256 + Red. Indexing of page and cue starts from 1.


==== Misc ====


GetBeyondBuild - return integer value, build of BEYOND application.

GetMidiDeviveIndex - return current MIDI device index associated with this
script. BEYOND can use 4 MIDI devices. By default script associated with 1st
device. There is a command for change device index. This function allows to
check current association. Indexing from 1.

GetMidiDeviceLayer - return current layer of MIDI device mapping.  Indexing
from 1. Layering introduced because BEYOND has big number of functions, and at
the same time, MIDI device has limited number of sliders and buttons. MIDI
settings allow to organize several layers, and as a result, MIDI controller
will do several functions (depending on current layer). GetMidiDeviceLayer
return currently active layer of MIDI device associated with this script.

==== User Interface ====
```

```
GetLcTabMode:integer - function return mode (destination of control) of Live Control tab. Basically, it tell what button selected - Master, Cue, Zone or ProTrack
Values are:
1 - Master
2 - Cue
3 - Zone
4 - ProTrack

GetTcTabMode:integer - function return mode of Time Control tab.Constant is same as in  GetLcTabMode

GetFxTabMode:integer - function return mode of  FX tab.Constant is same as in GetLcTabMode

GetGrid1Mode:integer - function return currently selected click mode of main Grid. Result is one of following values
0 - Select mode
1 - Flash mode
2 - FlashSolo mode
3 - Toggle mode
4 - Restart mode
5 - ProTrack mode

GetGrid2Mode:integer - function return currently selected click mode of secondary Grid. Constants same as for function GetGrid1Mode


==== Timeline mode functions ====


GetTimelinePos:float - function return time position of current timeline. Value in seconds. Fractional part contain millisecond part.

GetTimelineDuration:float - function return duration of current timeline. Value in seconds. Fractional part contain millisecond part.

GetTimelineOnline:integer function return 1 when "Enable laser output" is enabled in Timeline mode, otherwise return value is 0.

GetTImelinePlaying:integer function return 1 when timeline is currently playing, otherwise return value is 0.

GetTimlineTabIndex:integer. Timeline editor can work with multiple timelines. Function return index of currently selected timeline.

GetTimelineTabName:string - function return tab name of currently selected timeline.
```

```
==== FX and Live Control commands ====


The Destination

A few words about commands of BEYOND. Each command of BEYOND has information
about:
Sender - who initiated the command
Server - who is recipient / server of the command. As example - zone, cue,
master, and so on
Server index - index of server, of there are many such servers. As example -
Projection Zone
Command - the command itself
Arguments - depends on exact commands.

Lets consider example. We want to set the size of second projection zone to
50%. In this case, command will have such values
Sender - BEYOND set it automatically, no worry about
Server - Zone
Server index - 2
Command - Size
Arguments - 50,50,50

For the script, to make a complex command that include all fields is not
practical. Because the process has two parts. The first part - you define the
destination. By default it is Master. The second part is a Command itself.
BEYOND memorize the setting of destination and all consequent commands will
use it.



==== WaitFor command group ====


Sleep Time
Parameters: Time measured in milliseconds
Purpose: Allow to pause execution to defined time
Example 1: Sleep 100 // this will cause pause in execution on 1/10 of a
second.
Example 2:
MidiOut 0x90, 0x40, 0x7F
Sleep(500);
MidiOut 0x90, 0x40, 0x00
Sleep(500);
Restart
```

```
==== Timeline control ====


PlayTimeline
Start playback of current show in timeline editor

StopTimeline
Stop playback of current show in timeline editor

TimelineMarker
Add timeline marker. There are 3 modifications
TimelineMarker (no arguments) - add marker at current position and current color
TimelineMarker Color - addmarker with specified color at current time position
TimelineMarker Color, Time - add marker with specified color at specified time
Time - time in seconds, float point
Color  - index, 1..10;




==== Property Animation ====


Property animation designed for linear change of property from one state to another during some period of time. Technically, this is sort of micro script, but BEYOND does this job automatically. Internally BEYOND has internal list of the tasks for object property animation. The list is dynamic. The only way to create request for property animation is a script.

Note, the execution of property animation performed right after execution of all script by dedicated thread of BEYOND. The time resolution is ~ 40 "FPS". For animation of laser related properties it might be not perfect because laser projector frame rate may be much higher.

AnimateProp PropertyName,  StartValue, FinishValue, DurationMS, FinishEvent
PropertyName - string that contain complete name of object and its property. The property must be numeric.
StartValue - number. Specify start value of the property during animation
FinishValue - number. Specify final value of the property during animation.
DurationMS - duration of animation in milliseconds
FinishEvent - optional parameter, string. Specify name of Event that will be activated at the end of animation. Action equal to call of PulseEvent() procedure.

Exmaple:
AnimapeProp "Master.Brightness", 100, 0, 1000
// command change value of Master brightness from 100% to zero during 1000 ms
```

```
(one second)

Example2 :
AnimapeProp "Master.Brightness", Master.Brightness, 0, 500
// command change value of Master brightness from current value to zero during
500 ms (0.5 second)



AnimatePropDelta PropertyName, TotalDelta, DurationMS, FinishEvent
PropertyName - string that contain complete name of object and its property.
The property must be numeric.
TotalDelta- number. Specify how much will change the value of specified
property
DurationMS - duration of animation in milliseconds
FinishEvent - optional parameter, string. Specify name of Event that will be
activated at the end of animation. Action equal to call of PulseEvent()
procedure.

Example:
AnimatePropDelta "Master.Brightness", -25, 300    // decrease master
brightness on 25% during 0.3 second

Example 2:
AnimatePropDelta "Master.Color", -32, b2ms(1)    // shift color "slider" of
master on 32 during 1 beat

DeletePropAni PropertyName, PropertyName, PropertyName,,,,
Commamd delete existing animation-tasks from the pool. If no one parameter
specified then command delete ALL tasks. Command can contain one or more names
of properties.
PropertyName - string that contain complete name of object and its property.
The property must be numeric.

Example1:
DeletePropAni   // delete all

Example2:
DeletePropAni "Master.Brightness"

Example3:
DeletePropAni "Master.Brightness",  "Master.Color",


==== MIDI Surface Layer (MSL) ====


BEYOND offer more functions that can be mapped to MIDI sliders/buttons of
overage MIDI console has. The soluion is map a few function to one slider, and
```

enable only one of them. For making this possible BEYOND intoduce conception of "layers". You organize some set of functions grouped into layers, and after that you can change the layer, what give quick access to the groups of functions.

MIDI Sufrace Layering allow connect multiple BEYOND features to one MIDI command (message). Layeing work with Main Grid, Secondary Grid, Sufrace Buttons, Surface Sliders, FX and Zone selection. Currently available 12 layers. Counting starts from 1 and up to 12. The function of BEYOND may belong to more than one layer. In this case this function stay active in all layers where it enabled. If the function of BEYOND is not enabled in current layer, then it will not react on assigned MIDI message, same as will not generate a feedback messages.

Simplified Layer control.

Each MIDI Mapping object has preperty Layer. When you write to this propetry then it change Layer of all tables. Such operation equal to using of 12 commands descrived below.

Example:
Midi1.Layer=1
Midi2.Layer=10
Midi3.Layer=5

Detailed control.

Each table has own Layer property that you can control independently.

==== Triggers ====

"Trigger" is a special mode for the scripter. The trigger as two part - Definition and Action. The the Definition part you define the type of the trigger, range of values and labels for corresponding sections of code. In the Action part has one or more code sections that will be activated - depending on the definition. Lets talk a bit more about when trigger may be in help and why it done this way.

The simplest and classic example is when DMX come into some range of values, and the fact that the value is in range now it create some reaction. If the is not the same as "if value is in range then we do something". No., It is like - we do it every time as value come into the range. The next time it will happen when the value will go out of range and back to the range.

No doubt it is possible to make a trigger by means for "standard" PangoScrtipt commands, but it require much more complex script code, more lines, and at the end it will be more slow. The Trigger use a hybrid method. You define what to check and the ranges, and BEYOND software do the idle job for you - BEYOND read the state and compare with ranges, take care about the state and other things. All it done in native code. If you will do it all in PangoScript then it will require much more CPU time. So, this part can be more optimal. What say in PangoScript is a reaction (action). You need to define a section(s) of code that will be activated.

Trigger definition commands

DefineDmxTrigger ChannelIndex - this command set the scripter into trigger mode, and define that trigger react on DMX channel number "ChannelIndex "

DefineMidiTrigger Message, Data1 - this command set the scripter into trigger mode, and define that trigger react on MIDI message. You need to specify the message n

DefineTrigger String-Expression, Caption - this command set the scripter into trigger mode, and define that trigger depends on expression. The expression is math formula. it has a syntax of string. The dumb exampe: DefineTrigger "2+2". In this example 2+2 is expression. But, according to syntax of this command we put expression inside "..". The Caption is just text string for PangoScript tab

The top level logic of these command is this. The most possible, the triggers will work with MIDI and DMX consoles. The trigger must be fix, and effective, because it will work on a high speed, because we have a trigger specially made for MIDI and DMX. BEYOND precalculate values and do it all in native code. But, not doubt will appear a need in some universal method, and in this case will help universal command DefineTrigger. This command work with expression, it is more slow because BEYOND need to calculate expression all the time, but it is very flexible, and work for all types of input data. You can use objects, variables, functions, expression and so on. It can work with Audio, Kinect or DMX, Universe or GamePad and any mix of this. So, for DefineTrigger we need to supply text of expression to trigger engine, and it is a string.

Trigger range commands

Here a formal description of commands, examples and logic after that.

InRangeTrigger MinValue, MaxValue, LabelName
The action will be activated when values comes in range between MinValue and MaxValue. When it happen the scripter does goto to LabelName
MinValue - number, a minimum value of the range

```
MaxValue - number, a maximum value of the range
LabelName - string that contain label name

InRangeTriggerCmd MinValue, MaxValue, Command
The action will be activated when values comes in range between MinValue and
MaxValue. When it happen the scripter execute Command.
MinValue - number, a minimum value of the range
MaxValue - number, a maximum value of the range
Command- string that contain a PangoScript command


OutOfRangeTrigger MinValue, MaxValue, LabelName
The action will be activated when values comes out of range of MinValue and
MaxValue. When it happen the scripter does goto to LabelName.
MinValue - number, a minimum value of the range
MaxValue - number, a maximum value of the range
LabelName - string that contain label name

OutOfRangeTriggerCmd MinValue, MaxValue, Command
The action will be activated when values goes out of range of MinValue and
MaxValue. When it happen the scripter execute Command.
MinValue - number, a minimum value of the range
MaxValue - number, a maximum value of the range
Command- string that contain a PangoScript command


IncreaseTrigger MinValue, MaxValue, LabelName
The action will be activated when value increase and stay in range of MinValue
and MaxValue. When it happen the scripter does goto to LabelName.
MinValue - number, a minimum value of the range
MaxValue - number, a maximum value of the range
LabelName - string that contain label name


DecreaseTrigger MinValue, MaxValue, LabelName
The action will be activated when value decrease and stay in range of MinValue
and MaxValue. When it happen the scripter does goto to LabelName.
MinValue - number, a minimum value of the range
MaxValue - number, a maximum value of the range
LabelName - string that contain label name

About the logic

The most simple is InRangeTrigger command. When values comes in range, then
something happen. We can define a few ranges, and when value comes into the
range then something will happen. What will happen? We considered a few
options, and appeared that in simplest case one simple command will be enough.
I mean, Blackout, or EnableLaserOutput, or StopAllCue , who knows. In many
```

cases one command will be enough. But, what is not enough? I this case we should do goto to a second of code. In fact, you simply put a name of label, and BEYOND will do a goto (this process is optimized too). So, we have two options - Command or Label for Goto.

There is inverted version of InRangeTrigger - OutOfRange trigger. The only different is goes to be activated when value goes out of range. All the rest is equal. There in Min and Max value, and there version of command for Command and for Label.

The third variation of trigger allow define reaction on the increase or decrease of the value. it work like this: scripter memorize current value, and if the new value is bigger (or smaller) than current state then it active the trigger. Command IncreaseTrigger create a reaction when new value is more than current. Command DecreaseTrigger create a reaction when new value is less than current.

No doubt, we can add more commands, add more functionality to trigger. The range oriented commands is a classic. The increase/decrease has a practical use too.

Rule of 3 words: Type, Range, Interaction
T - type of the trigger - DMX, or MIDI, or universal
R - range, when value in range the we do something
I - interaction, the trigger is only a way to do some action, we need define what it will do.

Example 1: Enable/Disable laser output from MIDI

```
DefineMidiTrigger 0xB0, 0x00, "Enable Laser test"
InRangeTriggerCmd 0,63,"DisableLaserOutput"
InRangeTriggerCmd 64,127,"EnableLaserOutput"
```

This trigger react in the MDI slider 0xB0, 0x00, and has values in range of 0..127. When slider in lower half the laser output will be disables, and when in higher half, then output will be disables. Now command bby comamnd:

```
DefineMidiTrigger 0xB0, 0x00, "Enable Laser test"
DefineMidiTrigger  is  command
0xB0, 0x00 - this is slider parameter
"Enable Laser test"  - this is optional parameter - caption of this scripter


InRangeTriggerCmd 0,63,"DisableLaserOutput"
InRangeTriggerCmd - command, define the range and command
0,63 - the range
"DisableLaserOutput" - command as string.
```

```
InRangeTriggerCmd 64,127,"EnableLaserOutput"
InRangeTriggerCmd - command that define second range
64,127 - second range
"EnableLaserOutput" - command that will be executed.



Example 2: Same as example 1, but with goto

DefineMidiTrigger 0xB0, 0x00, "Enable Laser test 2"
InRangeTrigger 0,63, "D1"  // D1 is a name of  label, see below
InRangeTrigger 64,127, "D2" // D2 is a name of  label, see below
exit // we need to exit because we should stop script execution after the
declaration

D1:  // this is label used 1st range
DisplayPreview "Disable" // visual indication in Preview panel
DisableLaserOutput // Commamd
exit // we should stop script execution, otherwise it will do below

D2: // this is label used 2nd range
DisplayPreview "Ensable" // visual indication in Preview panel
EnableLaserOutput
exit // we should stop script execution. This is good practice to have it.

Commentaries for this example. The script itself is bigger, and there are
sections for definition and for corresponding scripts. There are two typical
mistakes
1. Do not forger to put exit instruction
2. Ensure that label name in code and in definition is the same. Otherwise, it
will not work. Label is case sensitive.

And general advice - use DisplayPreview, DisplayPopup or qlog for debug
purpose. It might be good idea to create a trigger definition with all jumps,
and put inside commands like DipsplayPreview only, without real functionality.
Once you see that the logic works fine, the put there the rest of code.



Example 3: Slider at first time do Enable Laser, and and second time Disable
Laser

This code based on previous, but it use variable. Variable organize a

DefineMidiTrigger 0xB0, 0x00, "Enable Laser test 3"
InRangeTrigger 64,127, "Trick1"
var counter // declare variable
counter=0 // initialize variable
exit
```

```
Trick1:
DisplayPreview Counter  // debug action
if (Counter=0) EnableLaserOutput  // action when Count is 0
if (Counter=1) DisableLaserOutput // action when Count is 1
Counter=(Counter+1) % 2 // increment counter and divide by modulus 2, so, it
will be 0,1,0,1,0,1 and so on
exit // stop this section
```

Example 4: The same as example 4, but with Goto

```
DefineMidiTrigger 0xB0, 0x00, "Enable Laser test 3"
InRangeTrigger 64,127, "Trick1"
var counter
counter=1
exit

Trick1:
DisplayPreview Counter
Counter=(Counter+1) % 2
if (Counter=0) goto When0
if (Counter=1) goto When1
exit

When0:
EnableLaserOutput
exit

When1:
DisableLaserOutput
exit
```

Well, example, is similar, and show that you can use additional Goto instructions.

Example 5

What I want to demonstrate by this example - BEYOND does a goto when it execute a trigger. It means this. You can do goto by means of script command. But trigger engine use the same goto. Look a the code, comments below

```
DefineMidiTrigger 0xB0, 0x00, "Enable Laser test 3"
InRangeTrigger 64,127, "Trick1"
InRangeTrigger 0,63, "StopIt"
exit
```

```
Trick1:
DisplayPopup "Yes"
Sleep 1000
DisplayPopup "No"
Sleep 1000
goto Trick1

StopIt:
DisplayPopup "I dont care"
exit
```

Two ranges, and main action in range of 64..127. The main action is after label Trick1. It has a dead loop. At least, it looks like dead loop. You see message panel with Yes and No and it will work until you move the slider down. It will break this endless loop.

Example 6: Lets try to use expressions... BPM notification

```
DefineTrigger "Master.BPM" // expression...
InRangeTrigger 0,50,"WhenSlow"
InRangeTrigger 250,500,"WhenFast"
exit

WhenSlow:
DisplayPopup "Hey, dont sleep"
exit

WhenFast:
DisplayPopup "Hey, too fast!"
exit
```

This example use simplest expression - read the value of object property - Master.BPM, and depending on the BPM show the message.

Example 7: Out of range.

This example based on previous. The only change is OutOfRangeTrigger. This section will be activate when BPM is too slow or too fast. This is only example without big sense, but it it easy to test, because you can see BPM value on toolbar and have the slider that control it.

```
DefineTrigger "Master.BPM" // expression...
OutOfRangeTrigger 60,250,"WhenOutOf"
exit

WhenOutOf:
DisplayPopup "Hmm, this is not normal"
```

```
exit

Example 8: Starts/Stop timeline

DefineMidiTrigger 0xB0, 0x00, "Timeline control"
InRangeTriggerCmd  0, 63, "StopTimeline"
InRangeTriggerCmd 64,127, "PlayTimeline"
```