PangoScript

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. 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”.

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

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

Syntax: goto label

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

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.*