

Table of contents

First Steps.....	6
All starts with a window.....	6
Listing 1.....	6
A window gets its content.....	7
Listing 2.....	7
About Windows.....	9
Windows - Function overview (Plugin - .DLL/.SO).....	9
Window - Parameter (Plugin).....	10
Window - Flags.....	10
Windows - Function overview (AGK-Basic - .AGC).....	11
Window - Parameter (AGK-Basic).....	11
Window - Example.....	12
Listing 3.....	12
About Layout & Group.....	14
Layout - Function overview (Plugin - .DLL/.SO).....	14
Layout - Parameter (Plugin).....	14
Function overview (AGK-Basic - .AGC).....	15
Layout - Parameter (AGK-Basic).....	15
Group - Function overview (Plugin - .DLL/.SO).....	15
Group - Parameter (Plugin).....	16
Layout & Group- Examples.....	16
Listing 4 (Layout).....	16
Listing 5 (Layout & Group).....	17
About Menus.....	21
Menu - Function overview (Plugin - .DLL/.SO).....	21
Parameter (Plugin).....	22
Symbol Table.....	22
Menu - Example.....	22

Listing 6.....	22
About Trees.....	26
Tree - Function overview (Plugin - .DLL/.SO).....	26
Parameter (Plugin).....	26
Tree - Example.....	27
Listing 7.....	27
About Text, Buttons and Toggles.....	34
Text - Function overview (Plugin - .DLL/.SO).....	34
Text - Parameter (Plugin).....	34
Button - Function overview (Plugin - .DLL/.SO).....	34
Button - Parameter (Plugin).....	35
Toggle Widgets.....	36
Option - Function overview (Plugin - .DLL/.SO).....	36
Checkbox - Function overview (Plugin - .DLL/.SO).....	36
Option/Checkbox Parameter (Plugin).....	36
Text/Button/Toggle Example.....	36
Listing 8.....	36
About Sliders, Progressbars and Properties.....	38
Slider - Function overview (Plugin - .DLL/.SO).....	38
Progressbar - Function overview (Plugin - .DLL/.SO).....	38
Property - Function overview (Plugin - .DLL/.SO).....	38
Slider/Progressbar/Property Parameter (Plugin).....	38
Slider/Progressbar/Property – Example.....	39
Listing 9.....	39
About Selectables, Combos and EditString.....	40
Selectable - Function overview (Plugin - .DLL/.SO).....	40
Selectable - Parameter (Plugin).....	40
Combo - Function overview (Plugin - .DLL/.SO).....	41
Combo - Parameter (Plugin).....	41
EditString - Function overview (Plugin - .DLL/.SO).....	42

EditString - Parameter (Plugin).....	42
EditString – Flags.....	43
EditString – Filter.....	43
EditString – Events.....	43
Selectable/Combos/EditString – Example.....	44
Listing 10.....	44
About Chart and Popups.....	48
Chart - Function overview (Plugin - .DLL/.SO).....	48
Chart - Parameter (Plugin).....	48
Popup - Function overview (Plugin - .DLL/.SO).....	49
Popup - Parameter (Plugin).....	49
Chart and Popup – Example.....	49
Listing 11.....	49
About ColorPicker, Widgets and Draw Commands.....	53
ColorPicker - Function Overview (Plugin - .DLL/.SO).....	53
Parameter (Plugin).....	53
Widget - Function Overview (Plugin - .DLL/.SO).....	53
Parameter (Plugin).....	53
Draw Commands - Function Overview (Plugin - .DLL/.SO).....	54
Parameter (Plugin).....	54
Chart and Popup – Example.....	56
Listing 11.....	56
About ColorPicker, Widgets and Draw Commands.....	60
ColorPicker - Function Overview (Plugin - .DLL/.SO).....	60
Parameter (Plugin).....	60
Widget - Function Overview (Plugin - .DLL/.SO).....	60
Parameter (Plugin).....	61
Draw Commands - Function Overview (Plugin - .DLL/.SO).....	61
Draw Commands - Function Overview (AGK-Basic - .agc).....	62
Parameter (Plugin/AGK-Basic).....	62

Colorpicker and Drawing – Example.....	64
Listing 12.....	64
About Input.....	68
Input - Function Overview (Plugin - .DLL/.SO).....	68
Input - Function Overview (AGK-Basic - .AGC).....	68
Parameter (Plugin).....	69
Input – Example.....	69
Listing 13.....	69
About Contextual.....	72
Contextual - Function Overview (Plugin - .DLL/.SO).....	72
Input - Function Overview (AGK-Basic - .AGC).....	72
Parameter (Plugin).....	72
About Styling.....	73
Manipulation of widget styles.....	73
Creating and using style templates.....	73
XML styling file.....	73
Styling - Function Overview (Plugin - .DLL/.SO).....	74
Parameter (Plugin).....	74
Widget type.....	76
Widget name.....	76
Cursorstyle.....	77
Styling - Function Overview (AGK-Basic - .AGC).....	77
Parameter (AGK-Basic).....	78
Style Examples.....	78
Listing 14.....	78
Listing 15.....	80
Listing 16.....	81
XML file “new_style”.....	82
XML file “other_style”.....	83
About Fonts and Images.....	84

Font - Function Overview (Plugin - .DLL/.SO).....	84
Parameter (Plugin).....	85
Glyphrange Table.....	85
Image - Function Overview (Plugin - .DLL/.SO).....	86
Parameter (Plugin).....	86
Font Example.....	87
Listing 17.....	87

First Steps

All starts with a window.

Let me show you the code first. After that come the explanations. Here is a simple example. It opens only a window without function and content.

Listing 1

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

do
    nk.HandleInput()

    if nk.WindowBegin("My First Window", 320, 50, 320, 240, NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||
NK_WINDOW_TITLE)
    endif
    nk.WindowEnd()

    nkSync()
loop
```

The first line includes the plugin. Just like any other. The following lines until `UseNewDefaultFonts` is the *AppGameKit* default code. I will not explain it now.

With `#insert "Nuklear.agc"` some important constants and functions written in *AGK-Basic* are included. So it is also important.

`nkInit()` initializes the GUI-engine.

Now comes the main loop. In order for **Nuklear** to respond to the user's input, the input must be passed from *AppGameKit* to *Nuklear*. This can be done with the command `nk.HandleInput()`.

Next we initialize our first window. For this `nk.WindowBegin` is responsible with the parameters (from left to right) window title, position of the window (x,y) width and height of the window and the window flags.

Why `WindowBegin` is in an if statement has the following reason: Nuclear remembers which status the window has. If it is minimized now, the content should not be shown anymore. So `WindowBegin` returns 0 (false) in this case.

Between *if* and *endif* there are all the controls that should appear in the window. With `nk.WindowEnd()` we finish the construction of the window.

In order for AppGameKit to display our window, `nkSync` must be called. The *AGK-Sync* does not have to be called anymore. This is done by `nkSync`.

Note that there are commands with and without a dot after `nk` (`nkSync` and `nk.WindowEnd()`). The ones with the dot are pure plugin commands and the ones without are commands in AGK-Basic that were inserted via `#insert "../Nuklear.agc"`.

A window gets its content

Now we fill our window with some controls. First of all we have to specify which space our controls are allowed to occupy. We do this with `nk.LayoutRowStatic` which waits for 3 parameters. Height and width of the controls and how many controls are placed next to each other.

```
nk.LayoutRowStatic(30,120,2)
```

Continue with two buttons. Because we want to react to the buttons, we put them into an if statement. We decide to use normal label buttons. Therefore you only need a string parameter as label for the button.

```
if nk.ButtonLabel("Say Hello") then hello_text$ = "Hello every body!"  
if nk.ButtonLabel("Clear Greetings") then hello_text$ = ""
```

The buttons set or clears the variable `hello_text$`.

Last but not least we want to bring our variable on the screen/window. For this we have `nk.Label` which shows a string. The parameters are the text to display and a flag how the text should be displayed.

```
nk.Label(hello_text$, NK_TEXT_CENTERED)
```

`NK_TEXT_CENTERED` only says that it will be centered vertically and horizontally in the available space. However, since we want to make the entire window width available to our text, we must first change the layout.

```
nk.LayoutRowDynamic(30,1)  
nk.Label(hello_text$, NK_TEXT_CENTERED)
```

This time we took a dynamic layout. This gives the layout the full width of the window. Again with a height of 30 pixels and the window width is divided into one column.

With `NK_TEXT_CENTERED` our text is finally displayed in the horizontal center of the window and in the vertical center of the row (30).

Listing 2

The code now looks like this.

```
#import_plugin Nuklear as nk
```

```

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

do
    nk.HandleInput()

    if nk.WindowBegin("My First Window", 320, 50, 320, 240, NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||
NK_WINDOW_TITLE)
        nk.LayoutRowStatic(30,120,2)
        if nk.ButtonLabel("Say Hello") then hello_text$ = "Hello every body!"
        if nk.ButtonLabel("Clear Greetings") then hello_text$ = ""

        nk.LayoutRowDynamic(30,1)
        nk.Label(hello_text$, NK_TEXT_CENTERED)
    endif
    nk.WindowEnd()

    nkSync()
loop

```


About Windows

Some of the marked functions are available in a shortened form in AGK-Basic version. In general, this concerns those functions where the parameters could also be passed as `nk_vec2` or `nk_rect`.

Others of these marked functions return only a Memblock-ID. This is then converted with the corresponding AppGameKit function in a UDT.

Windows - Function overview (Plugin - .DLL/.SO)

```
Integer WindowBegin(title as String, posx as Float, posy as Float, width as Float, height as Float,
flags as Integer)
Integer WindowBeginIntegeritled(name as String, title as String, posx as Float, posy as Float, width
as Float, height as Float, flags as Integer)
WindowEnd()
WindowStore(slot as Integer, name as String)
Integer WindowGetPosition()
Integer WindowGetSize()
Float WindowGetWidth()
Float WindowGetHeight()
Integer WindowGetContentRegion()
Integer WindowGetContentRegionMin()
Integer WindowGetContentRegionMax()
Integer WindowGetContentRegionSize()
WindowStoreCanvas(slot as Integer)
Integer WindowGetScroll()
Integer WindowHasFocus()
Integer WindowIsCollapsed(name as String)
Integer WindowIsClosed(name as String)
Integer WindowIsHidden(name as String)
Integer WindowIsActive(name as String)
Integer WindowIsHovered()
Integer WindowIsAnyHovered()
Integer WindowIsAnyActive()
WindowStorePanel(slot as Integer)
Integer WindowGetPanel()
Integer WindowGetPanelBounds()
Integer WindowGetBounds()
Float WindowGetBoundX()
Float WindowGetBoundY()
Float WindowGetBoundWidth()
Float WindowGetBoundHeight()
Float WindowGetContentRegionX()
Float WindowGetContentRegionY()
Float WindowGetContentRegionWidth()
Float WindowGetContentRegionHeight()
WindowSetBounds(name as String, x as Float, y as Float, w as Float, h as Float)
WindowSetPosition(name as String, x as Float, y as Float)
WindowSetSize(name as String, x as Float, y as Float)
WindowSetFocus(name as String)
WindowSetScroll(x as Integer, y as Integer)
WindowClose(name as String)
```

```

WindowCollapse(name as String , state as Integer)
WindowCollapseIf(name as String , state as Integer, cond as Integer)
WindowShow(name as String , state as Integer)
WindowShowIf(name as String , Integer state as , cond as Integer)

```

Window - Parameter (Plugin)

- **name**
Gives a name to a window. Used to identify different windows. Also outside a WindowBegin() and WindowEnd() command.
- **Title**
This is the title to be displayed. If the title stands alone without the parameter name, then the title is also the name.
- **posx, posy**
Position of a Windows.
- **width, height**
Size of a Windows.
- **Flags**
Flags for a window. Affects the look, behavior and functionality of a window.
- **Slot**
Is used as a workaround for the use of pointers. Some functions like the Draw/Stroke commands require such a 'slot' to access the pointer internally.
- **scrollx, scrolly**
Specifies the scroll offset.
- **State**
Describes a status - On or Off (1 or 0 / nk_true or nk_false).
- **Cond**
Condition that must be complied with in order to actually perform the state change.

Window - Flags

- `NK_WINDOW_BORDER`
- `NK_WINDOW_MOVABLE`
- `NK_WINDOW_SCALABLE`
- `NK_WINDOW_CLOSABLE`
- `NK_WINDOW_MINIMIZABLE`
- `NK_WINDOW_NO_SCROLLBAR`
- `NK_WINDOW_TITLE`
- `NK_WINDOW_SCROLL_AUTO_HIDE`
- `NK_WINDOW_BACKGROUND`
- `NK_WINDOW_SCALE_LEFT`
- `NK_WINDOW_NO_INPUT`
- `NK_WINDOW_PRIVATE`
- `NK_WINDOW_DYNAMIC`
- `NK_WINDOW_ROM`
- `NK_WINDOW_NOT_INTERACTIVE`

- `NK_WINDOW_HIDDEN`
- `NK_WINDOW_CLOSED`
- `NK_WINDOW_MINIMIZED`
- `NK_WINDOW_REMOVE_ROM`

Windows - Function overview (AGK-Basic - .AGC)

```
nk_rect nkWindowGetBounds()
nk_vec2 nkWindowGetSize()
nk_vec2 nkWindowGetPosition()
nk_rect nkWindowGetContentRegion()
nk_vec2 nkWindowGetContentRegionMin()
nk_vec2 nkWindowGetContentRegionMax()
nk_vec2 nkWindowGetContentRegionSize()
nk_rect nkWindowGetPanelBounds()
nk_vec2 nkWindowGetScroll()
nkWindowSetBounds(name as string as , bounds as nk_rect)
nkWindowSetPosition(name as string as , position as nk_vec2)
nkWindowSetSize(name as string as , size as nk_vec2)
nkWindowSetScroll(name as string as , scroll as nk_vec2)
```

Window - Parameter (AGK-Basic)

- **Name**
Gives a name to a window. Used to identify different windows. Also outside a `WindowBegin()` and `WindowEnd()` command.
- **Bounds**
Position and size of a window
- **Position**
Position of a window
- **Size**
Size of a window
- **Scroll**
Specifies the scroll offset.

A description of the most important functions can be found [here on the nuclear page](#). I think you can match the corresponding functions.

There are three functions which are not described because they were created by me as work around. This concerns the functions:

```
WindowStore(slot as Integer, name as String)
WindowStoreCanvas(slot as Integer)
WindowStorePanel(slot as Integer)
```

These functions simply store the pointers from the window, panel or canvas in an array. Other functions can then access this pointer.

`WindowStore` and `WindowStorePanel` are probably not necessary. I leave them in case there is a later use for them.

`WindowStoreCanvas` was used in the extended demo. This was necessary to write directly into the command buffer. For example if you want to write circles and lines directly into the command buffer.

The functions `FillCircle`, `StrokeLine`, `DrawImage` ect. finally use this pointer/slot.

Window - Example

Listing 3

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "Nuklear Window" )
SetWindowSize( 640, 480, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 640, 480 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

global boolean as string[1] = ["false","true"]
global collapse as integer = NK_MAXIMIZED
global window_open as integer = 1

do
    // pass agk inputs to nuklear
    nk.HandleInput()

    // check if the window should be displayed
    if window_open = 1
        // show our window
        if nk.WindowBegin("Window", 160, 50, 320, 240, NK_WINDOW_BORDER|NK_WINDOW_MOVABLE|
NK_WINDOW_TITLE|NK_WINDOW_SCALABLE|NK_WINDOW_MINIMIZABLE|NK_WINDOW_CLOSABLE)
            window_bounds as nk_rect
            window_bounds = nkWindowGetBounds()

            nk.LayoutRowDynamic(20,1)
            nk.Label("window position:"+str(window_bounds.x)+","+str(window_bounds.y), NK_TEXT_LEFT)
            nk.Label("window size:"+str(window_bounds.w)+","+str(window_bounds.h), NK_TEXT_LEFT)
            nk.Label("window has focus:"+boolean[nk.WindowHasFocus()], NK_TEXT_LEFT)
            nk.Label("window is hovered:"+boolean[nk.WindowIsHovered()], NK_TEXT_LEFT)
            nk.Label("window is active:"+boolean[nk.WindowIsActive("Window")], NK_TEXT_LEFT)
        endif
        nk.WindowEnd()
    endif

    // get new collapse state
    if nk.WindowIsCollapsed("Window")
        collapse = NK_MINIMIZED
    else
        collapse = NK_MAXIMIZED
    endif

    // check if our window was closed
    window_open = nk.WindowIsClosed("Window") = 0

    // F1 - toggle the collapse state
    if GetRawKeyPressed(112)
        collapse = 1 - collapse
    endif
enddo
```

```
        nk.WindowCollapse("Window", collapse)
    endif

    // F2 - bring back a closed window
    if GetRawKeyPressed(113) and window_open = 0 then window_open = 1

    nkSync()
loop
```

About Layout & Group

This time I show you an overview of the layouts and groups.

Layouts are used to place the widgets in your window, so you could say that every window that contains widgets MUST have a layout.

Also for this topic you can find a description of the commands on [the original nuclear documentation page](#).

Like last time, some functions are marked here for the same reasons as in the window topic.

Layout - Function overview (Plugin - .DLL/.SO)

```
LayoutSetMinRowHeight(height as Float)
LayoutResetMinRowHeight()
Integer LayoutWidgetBounds()
Float LayoutRatioFromPixel(pixel_width as Float)
LayoutRowStatic(height as Float, item_width as Integer, cols as Integer)
LayoutRowDynamic(height as Float, cols as Integer)
LayoutRowBegin(format as Integer, height as Float, cols as Integer)
LayoutRowEnd()
LayoutRow(format as Integer, height as Float, ratio_sz as String)
LayoutRowPush(width_ratio as Float)[/*]
LayoutRowTemplateBegin(height as Float)
LayoutRowTemplatePushDynamic()[/*]
LayoutRowTemplatePushVariable(min_width as Float)
LayoutRowTemplatePushStatic( width as Float)
LayoutRowTemplateEnd()
LayoutSpaceBegin(format as Integer, height as Float, count as Integer)
LayoutSpaceEnd()[/*]
Integer LayoutSpaceBounds()
LayoutSpacePush(posx as Float, posy as Float, width as Float, height as Float)
Integer LayoutSpaceToScreen(posx as Float, posy as Float)
Integer LayoutSpaceToLocal(posx as Float, posy as Float)
Integer LayoutSpaceRectToScreen(posx as Float, posy as Float, width as Float, height as Float)
Integer LayoutSpaceRectToLocal(posx as Float, posy as Float, width as Float, height as Float)
```

Layout - Parameter (Plugin)

- **Height**
Sets the height of the row.
- **pixel_width**
Pixel width to convert to window ratio.
- **item_width**
Holds the pixel width of each widget in the line.
- **cols**
Number of widgets in the row.
- **Format**
Either NK_DYNAMIC for the window ratio or NK_STATIC for fixed size columns.

- **ratio_sz**
A JSON string containing float values to split the line into individual columns. The string has the following format "[#.##,#.##,...]"
- **width_ratio**
Either a window ratio or a fixed width depending on format in the previous LayoutRowBegin call
- **min_width**
Holds the minimum pixel width, for the next column.
- **width**
Contains the absolute pixel width value, for the next column.
- **Count**
Number of widgets in the SpaceLayout.

Function overview (AGK-Basic - .AGC)

```
nk_rect nkLayoutSpaceBounds()
nk_rect nkLayoutSpaceRectToScreen(rect ref as nk_rect)
nkLayoutRow(format as integer, height as float, ratio ref as float[])
nk_rect nkLayoutWidgetBounds()
nkLayoutSpacePush(rect ref as nk_rect)
nk_vec2 nkLayoutSpaceToScreen(pos ref as nk_vec2)
nk_vec2 nkLayoutSpaceToLocal(pos ref as nk_vec2)
nk_rect nkLayoutSpaceRectToLocal(rect ref as nk_rect)
```

Layout - Parameter (AGK-Basic)

- **Rect**
Contains the position and size of an area in an nk_rect UDT.
- **Format**
Either NK_DYNAMIC for the window ratio or NK_STATIC for fixed size columns.
- **Height**
Sets the height of the row.
- **Ratio**
A float array containing the column widths, either as ratios between 0.0 and 1.0 or as absolute numbers greater than 1.0.
- **pos**
Contains a position in an nk_vec2 UDT

Group - Function overview (Plugin - .DLL/.SO)

```
Integer GroupBegin(String title, Integer flags)
Integer GroupBeginTitled(String name, String title, Integer flags)
GroupEnd()
```

Group - Parameter (Plugin)

- **Title**
Holds the title of the group widget.
- **Flags**
These flags are the same as for WindowBegin. See there.
- **Name**
This name is used to identify the group internally. The name should be unique. If the title stands alone without the parameter name, then the title is also the name.

Layout & Group- Examples

Listing 4 (Layout)

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "Nuklear Window" )
SetWindowSize( 640, 480, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 640, 480 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

num_columns as integer = 4
ratio as float[3] = [0.15,0.35,0.35,0.15]
do
    // pass agk inputs to nuklear
    nk.HandleInput()

    // open a window
    if nk.WindowBegin("Window", 20, 50, 600, 240, NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||
NK_WINDOW_TITLE||NK_WINDOW_SCALABLE||NK_WINDOW_MINIMIZABLE||NK_WINDOW_CLOSABLE)
        // define the layout. row height of 20 with 1 widget of dynamic width.
        nk.LayoutRowDynamic(20,1)
        // put a widget in the first row
        nk.Label("drawing button grid (F1 - switch through number of columns)", NK_TEXT_CENTERED)

        // all next widgets have the same size from the defined layout.
        // each row with one widget. so that with each widget the number
        // of rows grows.
        nk.Label("DYNAMIC", NK_TEXT_LEFT)

        // The layout is changed. The line height remains at 20 pixels,
```



```

// with a variable number of widgets in the row.
nk.LayoutRowDynamic(20, num_columns)
i as integer
// put twenty buttons in the layout
for i=0 to 11
    nk.ButtonLabel("Button "+str(i))
next

// the layout is changed again. as above, except that the buttons
// are arranged statically.
nk.LayoutRowDynamic(20,1)
nk.Label("STATIC", NK_TEXT_LEFT)
nk.LayoutRowStatic(20, 110, num_columns)
for i=0 to 11
    nk.ButtonLabel("Button "+str(i))
next

// now a demonstration for ratio layouts.
nk.LayoutRowDynamic(20,1)
nk.Label("drawing a ratio layout (F2 - shuffle ratio)", NK_TEXT_CENTERED)
nkLayoutRow(NK_DYNAMIC, 20, ratio)
for i=0 to 3
    nk.ButtonLabel("Button "+str(trunc(ratio[i]*100))+"%")
next i
endif
nk.WindowEnd()

if nk.WindowIsClosed("Window") then exit

// F1 - switch through number of columns
if GetRawKeyPressed(112)
    if num_columns >= 4 then num_columns = 1 else inc num_columns,1
endif

// F2 - shuffle ratio
if GetRawKeyPressed(113)
    percent as integer = 100
    value as integer

    value = Random(10, 50)
    ratio[0] = value / 100.0
    dec percent, value

    value = Random(10, (percent-20))
    ratio[1] = value / 100.0
    dec percent, value

    value = Random(10, (percent-10))
    ratio[2] = value / 100.0
    dec percent, value

    ratio[3] = percent / 100.0
endif
nkSync()
loop

```

Listing 5 (Layout & Group)

Here is another example for the use of space layouts.

```
#option_explicit
```

```

#import_plugin Nuklear as nk

type connector
    source_index as integer
    dest_index as integer
endtype

type node
    name as string
    bounds as nk_rect
    link_index as integer
endtype

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "Nuklear Window" )
SetWindowSize( 1024, 768, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1024, 768 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

// init nodes
global node_array as node[]
global connector_array as connector[]

new_node as node
new_node.name = "component alpha"
new_node.bounds = nk_rect(10,50,150,150)
node_array.insert(new_node)

new_node.name = "component beta"
new_node.bounds = nk_rect(10,250,150,150)
node_array.insert(new_node)

new_node.name = "delta"
new_node.bounds = nk_rect(350,150,150,150)
node_array.insert(new_node)

LinkNode(node_array[0], node_array[2])
LinkNode(node_array[1], node_array[2])

// begin main loop
do
    // pass agk inputs to nuklear
    nk.HandleInput()

    // open a window
    if nk.WindowBegin("Window", 20, 50, 600, 540, NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||
NK_WINDOW_TITLE||NK_WINDOW_CLOSABLE||NK_WINDOW_NO_SCROLLBAR)
        region as nk_rect

```

```

// window canvas is stored in slot '0' for later use to draw on
// the window surface.
nk.WindowStoreCanvas(0)

// get the area of the available space of the window.
region = nkWindowGetContentRegion()

// give the layout the entire window area. with a defined number
// of widgets.
nk.LayoutSpaceBegin(NK_STATIC, region.h, node_array.length+1)

// loop all available nodes
i as integer
for i=0 to node_array.length
    // give the next widget a defined place on the window/panel.

nk.LayoutSpacePush(node_array[i].bounds.x,node_array[i].bounds.y,node_array[i].bounds.w,node_array[i].bounds.h)

    // now a widget, in this case a group is packed into the
    // layout. In this group more widgets can be placed.
    if nk.GroupBegin(node_array[i].name, NK_WINDOW_MOVABLE||NK_WINDOW_NO_SCROLLBAR||
NK_WINDOW_BORDER||NK_WINDOW_TITLE)
        // store group-window panel for later use.
        nk.WindowStorePanel(1)

        // give the next widget (label) a place.
        nk.LayoutRowDynamic(25, 1)
        nk.Label("pos:"+str(trunc(node_array[i].bounds.x))
+", "+str(trunc(node_array[i].bounds.y)), NK_TEXT_CENTERED)

        // give each group an end mark, but only if it was opened.
        nk.GroupEnd()
    endif

    // here we get the new stored group-window panel bounds
    bound0 as nk_rect
    bound0 = nkLayoutSpaceRectToLocal(nkWindowGetStoredPanelBounds(1))

    // copy the new position to the node description.
    node_array[i].bounds.x = bound0.x
    node_array[i].bounds.y = bound0.y
next

// now draw all connectors
for i=0 to connector_array.length
    DrawConnection(node_array[connector_array[i].source_index],
node_array[connector_array[i].dest_index])
next

    nk.LayoutSpaceEnd()
endif
nk.WindowEnd()

if nk.WindowIsClosed("Window") then exit

nkSync()
loop

function LinkNode(src ref as node, dst ref as node)
    new_link as connector
    new_link.source_index = node_array.find(src.name)

```

```

    new_link.dest_index = node_array.find(dst.name)

    connector_array.insert(new_link)
    src.link_index = connector_array.length
endfunction

function DrawConnection(src as node, dst as node)
    pos0 as nk_vec2
    pos1 as nk_vec2
    pos0.x = src.bounds.x + src.bounds.w
    pos0.y = src.bounds.y + src.bounds.h/2
    pos1.x = dst.bounds.x
    pos1.y = dst.bounds.y + dst.bounds.h/2

    pos0 = nkLayoutSpaceToScreen(pos0)
    pos1 = nkLayoutSpaceToScreen(pos1)

    nk.StrokeCurve(0, pos0.x,pos0.y, pos0.x+50,pos0.y, pos1.x-50,pos1.y, pos1.x,pos1.y, 1.5,
0xffff00ff)
endfunction

```

About Menus

There are a few workarounds here as well. Commands that expect an image as parameter must first save the corresponding images in slots or they must be created with an assigned name.

For storing in slots there are the commands:

```
Integer ImageToSlot0(slot, agk_img_id as Integer, x as Float, y as Float, w as Float, h as Float)
Integer ImageToSlot1(slot as Integer, agk_img_id as Integer)
```

For the name assignment are the commands:

```
Integer ImageCreate(name as String, agk_img_id as Integer, x as float, y as float, w as float, h as float)
Integer ImageCreate(name as String, agk_img_id as Integer)
```

Menu - Function overview (Plugin - .DLL/.SO)

```
MenubarBegin()
MenubarEnd()
Integer MenuBeginText(title as String, len as Integer, align nk_flags, width as Float, height as Float)
Integer MenuBeginLabel(label as String, align as Integer, width as Float, height as Float)
Integer MenuBeginImage(id as String, image_slot as Integer, width as Float, height as Float)
Integer MenuBeginImage(id as String, image_name as String, width as Float, height as Float)
Integer MenuBeginImageText(title as String, len as Integer, align as Integer, image_slot as Integer, width as Float, height as Float)
Integer MenuBeginImageText(title as String, len as Integer, align as Integer, image_name as String, width as Float, height as Float)
Integer MenuBeginImageLabel(title as String, align nk_flags, image_slot as Integer, width as Float, height as Float)
Integer MenuBeginImageLabel(title as String, align nk_flags, image_name as String, width as Float, height as Float)
Integer MenuBeginSymbol(id as String, sym as Integer, width as Float, height as Float)
Integer MenuBeginSymbolText(title as String, len as Integer, align as Integer, sym as Integer, width as Float, height as Float)
Integer MenuBeginSymbolLabel(title as String, align as Integer, sym as Integer, width as Float, height as Float)
Integer MenuItemText(title as String, len as Integer, align as Integer)
Integer MenuItemLabel(label as String, align as Integer)
Integer MenuItemImageLabel(image_slot as Integer, label as String, align as Integer)
Integer MenuItemImageLabel(image_name as String, label as String, align as Integer)
Integer MenuItemImageText(image_slot as Integer, text as String, len as Integer, align as Integer)
Integer MenuItemImageText(image_name as String, text as String, len as Integer, align as Integer)
Integer MenuItemSymbolText(sym as Integer, text as String, len as Integer, align as Integer)
Integer MenuItemSymbolLabel(sym as Integer, label as String, align as Integer)
MenuClose()
MenuEnd()
```

Parameter (Plugin)

- **title, label**
Title of the menu item to be displayed.
- **len**
Length of the title in characters.
- **align**
Alignment of the text.
- **width**
Width of the panel.
- **height**
Height of the panel.
- **id**
String id of the item. Serves for identification (internal).
- **image_slot**
A previously created image slot.
- **image_name**
A previously created association to an image.
- **sym**
A Symbol from the symbol table.

Symbol Table

- `NK_SYMBOL_NONE`
- `NK_SYMBOL_X`
- `NK_SYMBOL_UNDERSCORE`
- `NK_SYMBOL_CIRCLE_SOLID`
- `NK_SYMBOL_CIRCLE_OUTLINE`
- `NK_SYMBOL_RECT_SOLID`
- `NK_SYMBOL_RECT_OUTLINE`
- `NK_SYMBOL_TRIANGLE_UP`
- `NK_SYMBOL_TRIANGLE_DOWN`
- `NK_SYMBOL_TRIANGLE_LEFT`
- `NK_SYMBOL_TRIANGLE_RIGHT`
- `NK_SYMBOL_PLUS`
- `NK_SYMBOL_MINUS`
- `NK_SYMBOL_MAX`

The commands `Menu...Text` and `Menu...Label` do not differ in their functionality. With the `Label` variant only the length of the title is removed. So it is also the preferred command for creating menus.

Menu - Example

Listing 6

```
#option_explicit
```

```

#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "Nuklear Window" )
SetWindowSize( 1024, 768, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1024, 768 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

show_menu as Integer = nk_true
overwrite as Integer = nk_false
usage as float = 768.25
max_mem as float = 1024
show_as_mb as integer = nk_true
global chart_values as float[13]
chart_range as float = 300
// begin main loop
do
    // pass agk inputs to nuklear
    nk.HandleInput()

    // Get the mouse data to display in a chartwidget
    GetMouseMoveData()

    // open a window
    if nk.WindowBegin("Window", 20, 50, 600, 540, NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||
NK_WINDOW_TITLE||NK_WINDOW_CLOSABLE||NK_WINDOW_NO_SCROLLBAR)
        if show_menu
            // Start of the menubar
            nk.MenuBarBegin()

            // First menu to display common menu items
            // The menu bar is also a panel that can contain all kinds of
            // widgets. Therefore widgets must be inserted in a layout.
            // We have 3 menu entries with a menu bar size of 25 px.
            nk.LayoutRowBegin(NK_STATIC, 25, 3)

            // The first menubar item is 45px width.
            nk.LayoutRowPush(45)

            // Start of the menubar item. MenuBeginLabel' opens a panel,
            // in this case 120px width and 150px height. 'File' is the
            // title of the item and 'NK_TEXT_CENTERED' is the alignment
            // of the title.
            if(nk.MenuBeginLabel("File", NK_TEXT_CENTERED, 120, 150))

                // Since a panel has been opened, the widgets must be
                // added to a layout.
                nk.LayoutRowDynamic(25, 1)

```

```

        // Now the individual menu items for 'File' follow.
        // These are simple items. The last one with the
        // function to leave the main loop to end the program.
        nk.MenuItemLabel("New", NK_TEXT_LEFT)
        nk.MenuItemLabel("Open", NK_TEXT_LEFT)
        nk.MenuItemLabel("Save", NK_TEXT_LEFT)
        nk.MenuItemLabel("Close", NK_TEXT_LEFT)
        if nk.MenuItemLabel("Exit", NK_TEXT_LEFT) then exit

        // End of the first menubar item.
        nk.MenuEnd()
    endif

    // Now follows the second menubar item.
    nk.LayoutRowPush(45)
    if(nk.MenuBeginLabel("Edit", NK_TEXT_CENTERED, 150, 300))
        nk.LayoutRowDynamic(25, 1)
        nk.MenuItemLabel("Copy", NK_TEXT_LEFT)
        nk.MenuItemLabel("Delete", NK_TEXT_LEFT)
        nk.MenuItemLabel("Cut", NK_TEXT_LEFT)
        nk.MenuItemLabel("Paste", NK_TEXT_LEFT)

        // Insert a label as separator.
        nk.Label("~~~~~", NK_TEXT_CENTERED)

        // Remember: The 'MenuBegin...' command opens a panel and
        // the all items can be a widgets. Here you see how to
        // insert a checkox, Progressbar, Slider and option widgets.
        overwrite = nk.CheckboxLabel("Overwrite", overwrite)
        nk.LayoutRowDynamic(25, 2)
        nk.Progress(usage, max_mem, NK_FIXED)
        if show_as_mb
            nk.Label(str(usage,2)+" MB", NK_TEXT_LEFT)
        else
            nk.Label(str((usage/max_mem)*100,2)+" %", NK_TEXT_LEFT)
        endif
        nk.LayoutRowDynamic(25, 1)
        max_mem = nk.SliderFloat(usage, max_mem, 2048, 64)
        if nk.OptionLabel("Show as MegByte", show_as_mb) then show_as_mb =
nk_true

        if nk.OptionLabel("Show as Percent", 1 - show_as_mb) then show_as_mb
= nk_false

        // End of the first menubar item.
        nk.MenuEnd()
    endif

    // The third menubar item. Here you can see a chart widget
    // with live data. The procedure is the same as for the last
    // menu item.
    nk.LayoutRowPush(45)
    if(nk.MenuBeginLabel("Data", NK_TEXT_CENTERED, 400, 300))
        nk.LayoutRowDynamic(25, 1)
        nk.Label("Move Mouse", NK_TEXT_CENTERED)

        nk.LayoutRowDynamic(100,1)
        nk.ChartBegin(NK_CHART_LINES, chart_values.length, 0, chart_range)
        i as integer
        for i=0 to chart_values.length
            nk.ChartPush(chart_values[i])
        next
        nk.ChartEnd()
    endif

```



```

        nk.LayoutRowBegin(NK_DYNAMIC, 25, 3)
        nk.LayoutRowPush(0.1)
        nk.Label("range:", NK_TEXT_LEFT)
        nk.LayoutRowPush(0.8)
        chart_range = nk.SliderFloat(100, chart_range, 700, 1.0)
        nk.LayoutRowPush(0.1)
        nk.Label(str(chart_range, 0), NK_TEXT_LEFT)

        nk.LayoutRowDynamic(25, 1)
        nk.MenuItemLabel("Save Data", NK_TEXT_LEFT)
        nk.MenuEnd()
    endif
    nk.MenubarEnd()
endif
nk.WindowEnd()

if nk.WindowIsClosed("Window") then exit

nkSync()
loop

// Create mouse movement data. It calculates the average mouse movement
// distance every 200ms and stores it in a global variable (chart_values).
function GetMouseMoveData()
    global dist as float
    global updt as integer
    now as integer

    x as float
    y as float
    x = nk.InputGetMouseDeltaX()
    y = nk.InputGetMouseDeltaY()

    dist = dist + Sqrt(x * x + y * y)

    now = GetMilliseconds()
    if now > updt
        updt = now + 200

        i as integer
        for i=0 to chart_values.length-1
            chart_values[i] = chart_values[i+1]
        next i

        chart_values[chart_values.length] = dist
        dist = Sqrt(x * x + y * y)
    endif
endfunction
endfunction

```

About Trees

In the normal case only 3 - 4 commands are needed to build a tree structure. The commands `TreeElement...Push...` have an additional value parameter to mark an element as selected.

These functions return 1 when this node is open and 0 when it is closed. If the return value is 1 then the content should be drawn. So it is recommended to include these commands in an if query. After the content has been drawn an open node must be closed with `TreeStatePop`.

The original [documentation about Trees can be found here](#).

Tree - Function overview (Plugin - .DLL/.SO)

```
Integer TreePush(type as Integer, title as String, state as Integer)
Integer TreePushID(type as Integer, title as String ,state as Integer, id as Integer)
Integer TreePushHashed(type as Integer, title as String, state as Integer, hash as String, len as Integer, seed as Integer)
Integer TreeImagePush(type as Integer, image_slot as Integer, title as String, state as Integer)
Integer TreeImagePush(type as Integer, image_name as String, title as String, state as Integer)
Integer TreeImagePushID(type as Integer, image_slot as Integer, title as String, state as Integer, id as Integer)
Integer TreeImagePushID( type as Integer, image_name as String, title as String, state as Integer, id as Integer)
Integer TreeImagePushHashed(type as Integer, image_slot as Integer, title as String, state as Integer, hash as String, len as Integer, seed as Integer)
Integer TreeImagePushHashed(type as Integer, image_name as String, title as String, state as Integer, hash as String, len as Integer, seed as Integer)
Integer TreeStatePush(type as Integer, title as String, state as Integer)
Integer TreeStateImagePush(type as Integer, image_name as String, title as String, state as Integer)
TreeStatePop()
Integer TreeElementPushID(type as Integer, title as String, state as Integer, value as Integer, id as Integer)
Integer TreeElementPushHashed(type as Integer, title as String, state as Integer, value as Integer, hash as String, len as Integer, seed as Integer)
Integer TreeElementImagePushID(type as Integer, image_slot as Integer, title as String, state as Integer, value as Integer, id as Integer)
Integer TreeElementImagePushID(type as Integer, image_name as String, title as String, state as Integer, value as Integer, id as Integer)
Integer TreeElementImagePushHashed(type as Integer, image_slot as Integer, title as String, state as Integer, value as Integer, hash as String ,len as Integer, seed as Integer)
Integer TreeElementImagePushHashed(type as Integer, image_name as String, title as String, state as Integer, value as Integer, hash as String, len as Integer, seed as Integer)
Integer TreeGetLastElementValue()
```

Parameter (Plugin)

- **type**
A value either `NK_TREE_NODE` or `NK_TREE_TAB`. As a highlighted collapsible UI section (`NK_TREE_TAB`) or as a tree node (`NK_TREE_NODE`).

- **title**
Text in the tree header.
- **state**
Initial value of the tree state, either `NK_MINIMIZED` or `NK_MAXIMIZED`.
- **id**
Loop counter, if this function is called in a loop.
- **hash**
String to generate the ID.
- **len**
Size of the passed string in hash.
- **seed**
Seed, if this function is called in a loop. Otherwise it can be 0.
- **image_slot**
A previously created image slot.
- **image_name**
A previously created connection to an image.
- **value**
A value 0 or 1, whether the header is selected

Tree - Example

In the example you can see that the tree nodes can also be used to create submenus for the menu bar.

Listing 7

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1024, 768, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1024, 768 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

parent_node as integer = nk_false
child_node as integer = nk_false
child_items as integer[3]
parent_items as integer[7]
```

```

selected as String

do
    // pass agk inputs to nuklear
    nk.HandleInput()

    if nk.WindowBegin("Tree Window", 80, 50, 400, 240, NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||
NK_WINDOW_TITLE||NK_WINDOW_SCALABLE)

        // First tree node. With the flag NK_TREE_TAB the node is slightly
        // highlighted. With NK_MINIMIZED the initial state of the node
        // is set, in this case minimized/collapsed.
        if (nk.TreePush(NK_TREE_TAB, "Root Tree", NK_MINIMIZED))

            // This is a child's node. The node type is set to NK_TREE_NODE.
            // The title of the node is not highlighted. The initial state
            // is also minimized/collapsed.
            if nk.TreePush(NK_TREE_NODE, "Selectable Tree", NK_MINIMIZED)

                // Another child node in the child node. However, this node
                // is selectable. The selected state is passed with the
                // variable parent_node. The last parameter is an ID so that
                // Nuklear can identify the control element. This should
                // be unique. After executing the TreeElementPushID function,
                // TreeGetLastElementValue can be used to query the last
                // state of the tree node.
                if nk.TreeElementPushID(NK_TREE_NODE, "Parent Tree Node", NK_MINIMIZED, parent_node,
0)

                    sel as integer = nk_false
                    parent_node = nk.TreeGetLastElementValue()

                    // Again a child's node. Parameters as before only
                    // with different ID.
                    if nk.TreeElementPushID(NK_TREE_NODE, "Child Tree Node", NK_MINIMIZED,
child_node, 1)

                        child_node = nk.TreeGetLastElementValue()

                        // Fill 'Child Tree Node' with selectable widgets
                        nk.LayoutRowStatic(18,100,1)
                        if parent_node then child_node = parent_node
                        for i=0 to child_items.length
                            sel = child_items[i]

                            if child_node then sel = child_node
                                if sel then selected = "Selected" else selected = "Unselected"

                                sel = nk.SelectableSymbolLabel(NK_SYMBOL_CIRCLE_SOLID,
selected, NK_TEXT_RIGHT, sel)

                                if sel = nk_false
                                    child_node = nk_false
                                    parent_node = nk_false
                                endif

                                child_items[i] = sel
                            next

                                nk.TreeElementPop()
                            endif

                            // Fill 'Parent Tree Node' with selectable widgets
                            nk.LayoutRowStatic(18,100,1)
                            for i=0 to parent_items.length
                                sel = parent_items[i]

```

```

        if parent_node then sel = parent_node
        if sel then selected = "Selected" else selected =
"Unselected"

        sel =
nk.SelectableSymbolLabel(NK_SYMBOL_CIRCLE_SOLID, selected, NK_TEXT_RIGHT, sel)
        if sel = nk_false then parent_node = nk_false

        parent_items[i] = sel
        next
        nk.TreeElementPop()
    endif
    nk.TreePop()
endif

// This is a sibling node to 'Selectable Tree'. The parameters
// are similar to there.
if nk.TreePush(NK_TREE_NODE, "Common Tree", NK_MINIMIZED)

    // And also this one gets a child's node. In this node
    // the area is divided into three sections with LayoutRow.
    // Each section gets a group and is filled with buttons.
    if nk.TreePush(NK_TREE_NODE, "Node filled with buttons!",
NK_MINIMIZED)

        // The three sections.
        nk.LayoutRowDynamic(250, 3)

        // First group/section/column is filled with 8 buttons.
        nk.GroupBegin("group 1", NK_WINDOW_NO_SCROLLBAR)
        nk.LayoutRowDynamic(20, 1)
        for i=0 to 7
            nk.ButtonLabel("Button "+str(i))
        next
        nk.GroupEnd()

        // Second group/section/column is also filled with buttons
but with different order

        nk.GroupBegin("group 2", NK_WINDOW_NO_SCROLLBAR)
        nk.LayoutRowDynamic(20, 1)
        nk.ButtonLabel("button 0")
        nk.LayoutRowDynamic(20, 2)
        nk.ButtonLabel("button 1")
        nk.ButtonLabel("button 2")
        nk.LayoutRowDynamic(20, 3)
        nk.ButtonLabel("button 3")
        nk.ButtonLabel("button 4")
        nk.ButtonLabel("button 5")
        nk.LayoutRowStatic(20, 50, 2)
        for i=6 to 13
            nk.ButtonLabel("Button "+str(i))
        next
        nk.GroupEnd()

        // Third group/section/column that has a menu bar.
        // Yes, it does not have to be at the top of the window.
        // Here you can see how to create a submenu in a menu.
        nk.GroupBegin("group 3", NK_WINDOW_NO_SCROLLBAR)
        nk.MenuBarBegin()

        nk.LayoutRowBegin(NK_STATIC, 25, 1)
        nk.LayoutRowPush(45)
        if nk.MenuBeginLabel("File", NK_TEXT_CENTERED, 150, 150)

```

```

                                nk.LayoutRowDynamic(20, 1)
                                if nk.TreePush(NK_TREE_TAB, "New", NK_MINIMIZED)
                                    nk.MenuItemLabel("Document", NK_TEXT_LEFT)
                                    nk.MenuItemLabel("Project", NK_TEXT_LEFT)
                                endif
                                nk.Label("~~~~~",
NK_TEXT_CENTERED)

                                if nk.MenuItemLabel("Exit", NK_TEXT_LEFT) then exit
                                    nk.MenuEnd()
                                endif
                                nk.MenubarEnd()
                                nk.GroupEnd()

                                nk.TreePop()
                            endif
                            nk.TreePop()
                        endif
                        nk.TreePop()
                    endif
                    nk.WindowEnd()

                nkSync()
loop#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1024, 768, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1024, 768 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

parent_node as integer = nk_false
child_node as integer = nk_false
child_items as integer[3]
parent_items as integer[7]
selected as String

do
    // pass agk inputs to nuklear
    nk.HandleInput()

    if nk.WindowBegin("Tree Window", 80, 50, 400, 240, NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||
NK_WINDOW_TITLE||NK_WINDOW_SCALABLE)

        // First tree node. With the flag NK_TREE_TAB the node is slightly
        // highlighted. With NK_MINIMIZED the initial state of the node
        // is set, in this case minimized/collapsed.
        if (nk.TreePush(NK_TREE_TAB, "Root Tree", NK_MINIMIZED))

```

```

// This is a child's node. The node type is set to NK_TREE_NODE.
// The title of the node is not highlighted. The initial state
// is also minimized/collapsed.
if nk.TreePush(NK_TREE_NODE, "Selectable Tree", NK_MINIMIZED)

// Another child node in the child node. However, this node
// is selectable. The selected state is passed with the
// variable parent_node. The last parameter is an ID so that
// Nuklear can identify the control element. This should
// be unique. After executing the TreeElementPushID function,
// TreeGetLastElementValue can be used to query the last
// state of the tree node.
if nk.TreeElementPushID(NK_TREE_NODE, "Parent Tree Node", NK_MINIMIZED, parent_node, 0)
    sel as integer = nk_false
    parent_node = nk.TreeGetLastElementValue()

// Again a child's node. Parameters as before only
// with different ID.
if nk.TreeElementPushID(NK_TREE_NODE, "Child Tree Node", NK_MINIMIZED, child_node, 1)
    child_node = nk.TreeGetLastElementValue()

// Fill 'Child Tree Node' with selectable widgets
nk.LayoutRowStatic(18,100,1)
if parent_node then child_node = parent_node
    for i=0 to child_items.length
        sel = child_items[i]

        if child_node then sel = child_node
        if sel then selected = "Selected" else selected = "Unselected"

        sel = nk.SelectableSymbolLabel(NK_SYMBOL_CIRCLE_SOLID, selected, NK_TEXT_RIGHT,
sel)

        if sel = nk_false
            child_node = nk_false
            parent_node = nk_false
        endif

        child_items[i] = sel
    next
        nk.TreeElementPop()
    endif

// Fill 'Parent Tree Node' with selectable widgets
nk.LayoutRowStatic(18,100,1)
for i=0 to parent_items.length
    sel = parent_items[i]

    if parent_node then sel = parent_node
    if sel then selected = "Selected" else selected = "Unselected"

    sel = nk.SelectableSymbolLabel(NK_SYMBOL_CIRCLE_SOLID, selected,
NK_TEXT_RIGHT, sel)

    if sel = nk_false then parent_node = nk_false

    parent_items[i] = sel
next
    nk.TreeElementPop()
endif
nk.TreePop()
endif

// This is a sibling node to 'Selectable Tree'. The parameters
// are similar to there.

```

```

if nk.TreePush(NK_TREE_NODE, "Common Tree", NK_MINIMIZED)

    // And also this one gets a child's node. In this node
    // the area is divided into three sections with LayoutRow.
    // Each section gets a group and is filled with buttons.
    if nk.TreePush(NK_TREE_NODE, "Node filled with buttons!", NK_MINIMIZED)

        // The three sections.
        nk.LayoutRowDynamic(250, 3)

        // First group/section/column is filled with 8 buttons.
        nk.GroupBegin("group 1", NK_WINDOW_NO_SCROLLBAR)
        nk.LayoutRowDynamic(20, 1)
        for i=0 to 7
            nk.ButtonLabel("Button "+str(i))
        next
        nk.GroupEnd()

        // Second group/section/column is also filled with buttons but with different
order
        nk.GroupBegin("group 2", NK_WINDOW_NO_SCROLLBAR)
        nk.LayoutRowDynamic(20, 1)
        nk.ButtonLabel("button 0")
        nk.LayoutRowDynamic(20, 2)
        nk.ButtonLabel("button 1")
        nk.ButtonLabel("button 2")
        nk.LayoutRowDynamic(20, 3)
        nk.ButtonLabel("button 3")
        nk.ButtonLabel("button 4")
        nk.ButtonLabel("button 5")
        nk.LayoutRowStatic(20, 50, 2)
        for i=6 to 13
            nk.ButtonLabel("Button "+str(i))
        next
        nk.GroupEnd()

        // Third group/section/column that has a menu bar.
        // Yes, it does not have to be at the top of the window.
        // Here you can see how to create a submenu in a menu.
        nk.GroupBegin("group 3", NK_WINDOW_NO_SCROLLBAR)
        nk.MenuBarBegin()

        nk.LayoutRowBegin(NK_STATIC, 25, 1)
        nk.LayoutRowPush(45)
        if nk.MenuBeginLabel("File", NK_TEXT_CENTERED, 150, 150)
            nk.LayoutRowDynamic(20, 1)
            if nk.TreePush(NK_TREE_TAB, "New", NK_MINIMIZED)
                nk.MenuItemLabel("Document", NK_TEXT_LEFT)
                nk.MenuItemLabel("Project", NK_TEXT_LEFT)
            endif
            nk.Label("~~~~~", NK_TEXT_CENTERED)
            if nk.MenuItemLabel("Exit", NK_TEXT_LEFT) then exit
            nk.MenuEnd()
        endif
        nk.MenuBarEnd()
        nk.GroupEnd()

        nk.TreePop()
    endif
    nk.TreePop()
endif
nk.TreePop()
endif
endif

```



```
endif
nk.WindowEnd()

nkSync()
loop
```

About Text, Buttons and Toggles

The text commands offer you different ways how a text should be displayed on the screen.

Text - Function overview (Plugin - .DLL/.SO)

```
Text(text as String, len as Integer, align as Integer)
TextColored(text as String, len as Integer, align as Integer, color as Integer)
TextWrap(text as String, len as Integer)
TextWrapColored(text as String, len as Integer, color as Integer)
Label(label as String, align as Integer)
LabelColored(label as String, align as Integer, color as Integer)
LabelWrap(label as String)
LabelColoredWrap(label as String, color as Integer)
```

Text - Parameter (Plugin)

- **text, label**
The text to display.
- **len**
Length of the text in characters.
- **align**
Label/text alignment.
- **color**
Color of the text/label.

Button - Function overview (Plugin - .DLL/.SO)

There are a lot of button commands. But actually they all do the same thing. Sometimes there is only a label, a symbol or an image displayed in the button. Of course it is also possible to display a combination of image and label or symbol and label.

A little more flexibility in the appearance is provided by the Styled-Buttons. You can give them your own styles.

```
Integer ButtonText(title as String, len as Integer)
Integer ButtonLabel(title as String)
Integer ButtonColor(color as Integer)
Integer ButtonSymbol(symbol as Integer)
Integer ButtonImage0(image_slot as Integer)
Integer ButtonImage1(img_name as String)
Integer ButtonMemblockImage(memid as Integer)
Integer ButtonSymbolLabel(symbol as Integer, label as String, align as Integer)
Integer ButtonSymbolText(symbol as Integer, label as String, len as Integer, align as Integer)
Integer ButtonImageLabel0(image_slot as Integer, label as String, align as Integer)
```

```

Integer ButtonImageLabel1(image_name as String, label as String, align as Integer)
Integer ButtonImageText0(image_slot as Integer, label as String, len as Integer, align as Integer)
Integer ButtonImageText1(image_name as String, label as String, len as Integer, align as Integer)
Integer ButtonTextStyled(style_name as String, title as String, len as Integer)
Integer ButtonLabelStyled(style_name as String, title as String)
Integer ButtonSymbolStyled(style_name as String, symbol as Integer)
Integer ButtonImageStyled0(style_name as String, image_slot as Integer)
Integer ButtonImageStyled1(style_name as String, image_name as String)
Integer ButtonSymbolTextStyled(style_name as String, symbol as Integer, title as String, len as Integer, align as Integer)
Integer ButtonSymbolLabelStyled(style_name as String, symbol as Integer, title as String, align as Integer)
Integer ButtonImageLabelStyled0(style_name as String, image_slot as Integer, title as String, align as Integer)
Integer ButtonImageLabelStyled1(style_name as String, image_name as String, title as String, align as Integer)
Integer ButtonImageTextStyled0(style_name as String, image_slot as Integer, title as String, len as Integer, align as Integer)
Integer ButtonImageTextStyled1(style_name as String, image_name as String, title as String, len as Integer, align as Integer)
ButtonSetBehavior(behavior as Integer)
Integer ButtonPushBehavior(behavior as Integer)
Integer ButtonPopBehavior()

```

Button - Parameter (Plugin)

- **title, label**
The text to display.
- **len**
Length of the text in characters.
- **color**
Color of the text/label.
- **symbol**
Type of symbol to be displayed.
- **image_slot**
An image slot that was previously created.
- **img_name**
The name of a previously created image.
- **mem_id**
Image data in a Memblock.
- **align**
Alignment of the label/text in the button.
- **style_name**
The name of a previously created style.

Toggle Widgets

Toggle widgets are either option widgets or checkbox widgets. Like all other widgets that offer a text and label variant. The Label variant is always preferable.

Option - Function overview (Plugin - .DLL/.SO)

```
Integer OptionLabel(label as String, active as Integer)
```

```
Integer OptionText(text as String, len as Integer, active as Integer)
```

Checkbox - Function overview (Plugin - .DLL/.SO)

```
Integer CheckboxLabel(label as String, active as Integer)
```

```
Integer CheckboxText(text as String, len as Integer, active as Integer)
```

```
Integer CheckboxFlagsLabel(label as String, flags as Integer, value as Integer)
```

```
Integer CheckboxFlagsText(text as String, len as Integer, flags as Integer, value as Integer)
```

Option/Checkbox Parameter (Plugin)

- **label, text**
The text to display.
- **len**
Length of the text in characters.
- **active**
Current state
- **flags**
A flag for comparison with value.
- **value**
A value that is compared to the flags.

Text/Button/Toggle Example

Listing 8

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
```

```

UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

#constant OPTION_YES = 1
#constant OPTION_NO = 2
#constant OPTION_MAYBE = 3
option as integer = OPTION_NO
flags as integer = 0
flags = NK_WINDOW_BORDER | NK_WINDOW_MOVABLE | NK_WINDOW_TITLE | NK_WINDOW_SCALABLE
do
    // pass agk inputs to nuklear
    nk.HandleInput()

    if nk.WindowBegin("Nuklear Window - Text, Buttons & Toggles", 40, 50, 500, 240, flags)

        // Three buttons are created that are evenly divided over the whole
        // row. The first button is a common label button. The second one
        // is a color button. The last one is a label button with symbol.
        // The button commands return 1 when they have been pressed. If
        // you want to react to them, you should query them with an
        // if-statement.
        nk.LayoutRowStatic(25, 150, 3)
        nk.ButtonLabel("Label Button")
        nk.ButtonColor(MakeColor(225,128,0))
        nk.ButtonSymbolLabel(NK_SYMBOL_TRIANGLE_RIGHT, "Label+Symbol", NK_TEXT_RIGHT)

        // 'Label' is used for the text display. For longer texts
        // 'LabelWrap' is suitable.
        nk.LayoutRowDynamic(20, 1)
        nk.Label("Common Label", NK_TEXT_LEFT)
        nk.LabelColored("Common Color Label", NK_TEXT_RIGHT, makecolor(255,255,0))
        nk.LayoutRowDynamic(40, 1)
        nk.LabelWrap("And a very long text can be wrapped. If this text would take several lines.
However, the layout should also provide the space. There is no scrollbar to display the text
completely.")

        // Option widgets and Checkbox widget are both toggles. While
        // Options follows an - either or - strategy, checkboxes allow
        // combinations. Or they stand alone and give the possibility to
        // turn something on or off.
        nk.LayoutRowDynamic(20, 3)
        if nk.OptionLabel("YES", option = OPTION_YES) then option = OPTION_YES
        if nk.OptionLabel("NO", option = OPTION_NO) then option = OPTION_NO
        if nk.OptionLabel("MAYBE", option = OPTION_MAYBE) then option = OPTION_MAYBE

        flags = nk.CheckboxFlagsLabel("Border", flags, NK_WINDOW_BORDER)
        flags = nk.CheckboxFlagsLabel("Movable", flags, NK_WINDOW_MOVABLE)
        flags = nk.CheckboxFlagsLabel("Scalable", flags, NK_WINDOW_SCALABLE)
        flags = nk.CheckboxFlagsLabel("Closable", flags, NK_WINDOW_CLOSABLE)
        flags = nk.CheckboxFlagsLabel("Minimizable", flags, NK_WINDOW_MINIMIZABLE)
        flags = nk.CheckboxFlagsLabel("No Scrollbar", flags, NK_WINDOW_NO_SCROLLBAR)
        flags = nk.CheckboxFlagsLabel("Title", flags, NK_WINDOW_TITLE)
    endif
    nk.WindowEnd()

    nkSync()
loop

```

About Sliders, Progressbars and Properties

All functions have one thing in common. Above the parameter `val` the current value is always passed. The return value is the adjusted new value.

Slider - Function overview (Plugin - .DLL/.SO)

```
Float SliderFloat(min as float ,val as Float ,max as Float ,step as Float)
```

```
Integer SliderInteger(min as Integer, val as Integer, max as Integer, step as Integer)
```

Progressbar - Function overview (Plugin - .DLL/.SO)

```
Integer Progress(val as Integer, max as Integer, modifyable as Integer)
```

Property - Function overview (Plugin - .DLL/.SO)

```
Integer PropertyInteger(label as String, min as Integer, val as Integer, max as Integer, step as Integer, inc_per_pixel as Float)
```

```
Float PropertyFloat(label as String, min as Float, val as Float, max as Float, step as Float, inc_per_pixel as Float)
```

Slider/Progressbar/Property Parameter (Plugin)

- **Min, max**
Lower and upper limit of the adjustable range.
- **Val**
Aktueller Wert (liegt meistens zwischen der unteren und oberen Grenze).
- **Step**
Step size to adjust.
- **Modifyable**
Flag, if the progress bar can be adjusted. (NK_FIXED or NK_MODIFIABLE)
- **label**
Text to be displayed in the property.
- **inc_per_pixel**
By how much the value per pixel is adjusted when moving with the mouse. Units per pixel.

Slider/Progressbar/Property – Example

Listing 9

```
#import_plugin Nuklear as nk

SetErrorMode(2)

SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

do
    // pass agk inputs to nuklear
    nk.HandleInput()

    if nk.WindowBegin("Nuklear Window - Sliders, Progressbars & Properties", 40, 50, 480, 240,
NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||NK_WINDOW_TITLE)

        // Two sliders are created here. The first is a float slider with
        // a value range of 0-10.0 with a step size of 0.1.
        // The second is an integer slider with the value range 0-100 and
        // a step size of 1.
        // Both functions return the changed value and must be filled with
        // the current value (float_value#/int_value).
        ratio as float[1] = [0.9,0.1]
        nklayoutRow(NK_DYNAMIC, 25, ratio)
        float_value# = nk.SliderFloat(0.0, float_value#, 10.0, 0.1)
        nk.Label(str(float_value#,2), NK_TEXT_CENTERED)
        int_value = nk.SliderInteger(0, int_value, 100, 1)
        nk.Label(str(int_value), NK_TEXT_CENTERED)

        // A progressbar can be changeable (NK_MODIFIABLE) or fixed
        // (NK_FIXED). It needs the current value (int_value) and the
        // maximum value. If it is a changeable progressbar the changed
        // value is returned.
        nk.LayoutRowDynamic(30, 1)
        nk.Progress(int_value, 100, NK_FIXED)

        // For Property widget a label is needed as well as min-, max-
        // and the current value. At the end there is the step size for
        // pressing the inc/dec buttons and the step size for dragging the mouse.
        float_value# = nk.PropertyFloat("The FLOAT value", 0.0, float_value#, 10.0, 0.25, 0.1)
        int_value = nk.PropertyInteger("The INTEGER value", 0, int_value, 100, 5, 1)
    endif
    nk.WindowEnd()

    nkSync()
loop
```

About Selectables, Combos and EditString

Selectables are similar to labels. Only with the difference that they are selectable. So you could also call them "Toggable" labels. The functions return the changed state of the widget. 0 For not selected and 1 for selected.

Combos are generally known. They offer a drop-down list of different elements. Like most ...Begin... functions, the ComboBegin... functions return the status. 0 for folded and 1 for unfolded. The item functions ComboItem... return nk_true (1) if selected.

For text input there is EditString. With the different flags that can be set, there are different input fields available. GetLastEditEvent returns the current status of the edit widget.

Selectable - Function overview (Plugin - .DLL/.SO)

```
Integer SelectableLabel(label as String, align as Integer, value as Integer)
Integer SelectableText(text as String, len as Integer, align as Integer, value as Integer)
Integer SelectableImageLabel(image_slot as Integer, label as String, align as Integer, value as Integer)
Integer SelectableImageLabel(image_name as String, label as String, align as Integer, value as Integer)
Integer SelectableImageText(image_slot as Integer, text as String, len as Integer, align as Integer, value as Integer)
Integer SelectableImageText(image_name as String, text as String, len as Integer, align as Integer, value as Integer)
Integer SelectableSymbolLabel(symbol as Integer, label as String, align as Integer, value as Integer)
Integer SelectableSymbolText(symbol as Integer, text as String, len as Integer, align as Integer, value as Integer)
```

Selectable - Parameter (Plugin)

- **label, text**
Darzustellender Text.
- **Align**
Ausrichtung des Textes.
- **Value**
Selected=1, Unselected=0
- **len**
Länge des Textes.
- **image_slot**
Einen Image-Slot der zuvor erstellt wurde.
- **image_name**
Der Name eines zuvor erstelltem Images.
- **Symbol**
Symbol das dargestellt werden soll.

Combo - Function overview (Plugin - .DLL/.SO)

```
Integer ComboSeparator(items_separated_by_separator as String, separator as Integer, selected as Integer, count as Integer, item_height as Integer, width as Float, height as Float)
Integer ComboBeginText(selected_sz as String, len as Integer, width as Float, height as Float)
Integer ComboBeginLabel(selected_sz as String, width as Float, height as Float)
Integer ComboBeginColor(r as Integer, g as Integer, b as Integer, a as Integer, width as Float, height as Float)
Integer ComboBeginColor(rgba as Integer, width as Float, height as Float)
Integer ComboBeginSymbol(symbol as Integer, width as Float, height as Float)
Integer ComboBeginSymbolLabel(selected_sz as String, symbol as Integer, width as Float, height as Float)
Integer ComboBeginSymbolText(selected_sz as String, len as Integer, symbol as Integer, width as Float, height as Float)
Integer ComboBeginImage(image_slot as Integer, width as Float, height as Float)
Integer ComboBeginImage(image_name as String, width as Float, height as Float)
Integer ComboBeginImageLabel(selected_sz as String, image_slot as Integer, width as Float, height as Float)
Integer ComboBeginImageLabel(selected_sz as String, image_name as String, width as Float, height as Float)
Integer ComboBeginImageText(selected_sz as String, len as Integer, image_slot as Integer, width as Float, height as Float)
Integer ComboBeginImageText(selected_sz as String, len as Integer, image_name as String, width as Float, height as Float)
Integer ComboItemLabel(label as String, align as Integer)
Integer ComboItemText(text as String, len as Integer, align as Integer)
Integer ComboItemImageLabel(image_slot as Integer, label as String, align as Integer)
Integer ComboItemImageLabel(image_name as String, label as String, align as Integer)
Integer ComboItemImageText(image_slot as Integer, text as String, len as Integer, align as Integer)
Integer ComboItemImageText(image_name as String, text as String, len as Integer, align as Integer)
Integer ComboItemSymbolLabel(symbol as Integer, label as String, align as Integer)
Integer ComboItemSymbolText(symbol as Integer, text as String, len as Integer, align as Integer)
ComboClose()
ComboEnd()
```

Combo - Parameter (Plugin)

- **items_separated_by_separator**
A string that contains the complete label items to be displayed. These items are separated with 'separator'.
- **separator**
A character that is used as separator for the item list.
- **selected**
The item that should be currently selected. Usually it is between 0 and 'count'.
- **count**
Number of items in the item list.
- **item_height**
Höhe das jedem Item zur Verfügung gestellt wird.

- **width, height**
Width and height of the drop-down field.
- **selected_sz**
The selected text that is displayed when the combo box is not expanded.
- **r, g, b, a, rgba**
For ColorCombo fields, the RGB color that is displayed in the unfolded state.
- **symbol**
A symbol from the symbol table. ([see in section - Menu](#))
- **image_slot**
An image slot that was previously created.
- **image_name**
The name of a previously created image.
- **label, text**
Text of the item to be displayed.
- **align**
Alignment of the text of the item.

EditString - Function overview (Plugin - .DLL/.SO)

```
Integer GetLastEditEvent()
String EditString(flags as Integer ,text as String, max as Integer, filter as Integer)
Integer EditGetCursorPosition()
Integer EditGetMarkStart()
Integer EditGetMarkEnd()
EditSetCursorPosition(pos as Integer)
EditSetMarkStart(pos as Integer)
EditSetMarkEnd(pos as Integer)
```

EditString - Parameter (Plugin)

- **flags**
Flags that can be set to determine the behaviour of the widget. (see EditString - Flags)
- **text**
The current text that is displayed. The modified text is returned.
- **max**
Maximum length of the text.
- **filter**
A filter that specifies which characters can be entered.
- **pos**
Position of the cursor/mark in the text.

EditString – Flags

- `NK_EDIT_DEFAULT`
- `NK_EDIT_READ_ONLY`
- `NK_EDIT_AUTO_SELECT`
- `NK_EDIT_SIG_ENTER`
- `NK_EDIT_ALLOW_TAB`
- `NK_EDIT_NO_CURSOR`
- `NK_EDIT_SELECTABLE`
- `NK_EDIT_CLIPBOARD`
- `NK_EDIT_CTRL_ENTER_NEWLINE`
- `NK_EDIT_NO_HORIZONTAL_SCROLL`
- `NK_EDIT_ALWAYS_INSERT_MODE`
- `NK_EDIT_MULTILINE`
- `NK_EDIT_GOTO_END_ON_ACTIVATE`

- `NK_EDIT_SIMPLE` (`NK_EDIT_ALWAYS_INSERT_MODE`)
- `NK_EDIT_FIELD` (`NK_EDIT_SIMPLE|NK_EDIT_SELECTABLE|NK_EDIT_CLIPBOARD`)
- `NK_EDIT_BOX` (`NK_EDIT_ALWAYS_INSERT_MODE| NK_EDIT_SELECTABLE| NK_EDIT_MULTILINE| NK_EDIT_ALLOW_TAB|NK_EDIT_CLIPBOARD`)
- `NK_EDIT_EDITOR` (`NK_EDIT_SELECTABLE|NK_EDIT_MULTILINE|NK_EDIT_ALLOW_TAB| NK_EDIT_CLIPBOARD`)

EditString – Filter

- `NK_FILTER_DEFAULT`
- `NK_FILTER_ASCII`
- `NK_FILTER_FLOAT`
- `NK_FILTER_DECIMAL`
- `NK_FILTER_HEX`
- `NK_FILTER_OCT`
- `NK_FILTER_BINARY`

EditString – Events

- | | |
|------------------------------------|--|
| • <code>NK_EDIT_ACTIVE</code> | edit widget is currently being modified |
| • <code>NK_EDIT_INACTIVE</code> | edit widget is not active and is not being modified |
| • <code>NK_EDIT_ACTIVATED</code> | edit widget went from state inactive to state active |
| • <code>NK_EDIT_DEACTIVATED</code> | edit widget went from state active to state inactive |
| • <code>NK_EDIT_COMMITED</code> | edit widget has received an enter and lost focus |

Selectables/Combos/EditString – Example

Listing 10

```
#import_plugin Nuklear as nk

#constant DEMO_COMBO = 1
#constant DEMO_EDIT = 2
#constant DEMO_SELECTABLE = 3
// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

tab as integer = 0
cmb_sel0 as String
cmb_sel1 as integer = 0
vehicles as string[5] = ["Bike","Motor Bike","Car","Truck","Airplane","Ship"]
ratio as float[1] = [0.1,0.9]
color as nk_color
color.a = 255
text as String
selected as integer[15]
do
    // pass agk inputs to nuklear
    nk.HandleInput()

    // open the window
    if nk.WindowBegin("Nuklear Window - Selectables, Combos & EditString", 40, 50, 480, 240,
NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||NK_WINDOW_TITLE)

        // here you can see how notetabs can be made.Three tabs are to be
        // created to demonstrate the widgets combo, editstring and
        // selctables.
        // First we divide the line into three parts and set a selectable
        // for each part to switch the tabs.
        nk.LayoutRowDynamic(20, 3)

        // first tab 'Combo Tab'. Together with a label it should show a
        // triangle to the right when folded and a triangle to the bottom
        // when unfolded. The variable 'tab' will be set according to the
        // tab to show the content of the tab later.
        symbol = NK_SYMBOL_TRIANGLE_RIGHT
        if tab = DEMO_COMBO then symbol = NK_SYMBOL_TRIANGLE_DOWN
        if nk.SelectableSymbolLabel(symbol, "Combo Tab", NK_TEXT_CENTERED, tab = DEMO_COMBO)
            tab = DEMO_COMBO
```

```

        symbol = NK_SYMBOL_TRIANGLE_RIGHT
    endif

    // second tab 'Edit Tab'
    if tab = DEMO_EDIT then symbol = NK_SYMBOL_TRIANGLE_DOWN
    if nk.SelectableSymbolLabel(symbol, "Edit Tab", NK_TEXT_CENTERED, tab = DEMO_EDIT)
        tab = DEMO_EDIT
        symbol = NK_SYMBOL_TRIANGLE_RIGHT
    endif

    // third tab 'Selectable Tab'
    if tab = DEMO_SELECTABLE then symbol = NK_SYMBOL_TRIANGLE_DOWN
    if nk.SelectableSymbolLabel(symbol, "Selectable Tab", NK_TEXT_CENTERED, tab =
DEMO_SELECTABLE)
        tab = DEMO_SELECTABLE
        symbol = NK_SYMBOL_TRIANGLE_RIGHT
    endif

    // in dependency of the 'tab' variable the content is displayed.
    select(tab)
        // the combo demo
        case DEMO_COMBO:
            nk.LayoutRowDynamic(60, 1)
            nk.GroupBegin("ComboGrp", NK_WINDOW_BORDER||NK_WINDOW_NO_SCROLLBAR)
            nk.LayoutRowDynamic(20, 4)
            nk.Label("Vehicle 1:", NK_TEXT_RIGHT)

            // A normal combo widget. The text is displayed in the
            // variable 'cmb_sel'. The combo widget opens a panel of
            // 120 x 200 pixels when activated.
            if nk.ComboBeginLabel(cmb_sel0, 120, 200)
                // The opened panel must also have a layout.
                nk.LayoutRowDynamic(18,1)

                // Now all items of the combo widget are set. If one
                // of the items is selected, 'ComboItemLabel' returns
                // 'nk_true' (1). And cmb_sel0 is set to the new value.
                if nk.ComboItemLabel("Bike", NK_TEXT_LEFT) then cmb_sel0 = "Bike"
                if nk.ComboItemLabel("Motor Bike", NK_TEXT_LEFT) then cmb_sel0 = "Motor Bike"
                if nk.ComboItemLabel("Car", NK_TEXT_LEFT) then cmb_sel0 = "Car"
                if nk.ComboItemLabel("Truck", NK_TEXT_LEFT) then cmb_sel0 = "Truck"
                if nk.ComboItemLabel("Airplane", NK_TEXT_LEFT) then cmb_sel0 = "Airplane"
                if nk.ComboItemLabel("Ship", NK_TEXT_LEFT) then cmb_sel0 = "Ship"
            nk.ComboEnd()
        endif

        // This is another way to fill a combo widget. The string
        // array 'vehicles' contains all items and is passed to
        // the function 'nkComboArray'. As return value you get
        // the selected array index.
        nk.Label("Vehicle 2:", NK_TEXT_RIGHT)
        cmb_sel1 = nkComboArray(vehicles, cmb_sel1, 18, 120, 200)

        // This way a Color Combo Widget is created. The parameters
        // should be self-explanatory. Furthermore the widget
        // behaves similar to the first described combo widget.
        nk.Label("Color-Combo:", NK_TEXT_RIGHT)
        if nk.ComboBeginColor(color.r, color.g, color.b, color.a, 120, 100)
            nkLayoutRow(NK_DYNAMIC, 18, ratio)
            nk.Label("R:", NK_TEXT_RIGHT)
            color.r = nk.Progress(color.r, 255, NK_MODIFIABLE)
            nk.Label("G:", NK_TEXT_RIGHT)
            color.g = nk.Progress(color.g, 255, NK_MODIFIABLE)

```

```

        nk.Label("B:", NK_TEXT_RIGHT)
        color.b = nk.Progress(color.b, 255, NK_MODIFIABLE)
        nk.Label("A:", NK_TEXT_RIGHT)
        color.a = nk.Progress(color.a, 255, NK_MODIFIABLE)
        nk.ComboEnd()
    endif

    // The same as before except that a colorpicker opens.
    nk.Label("Combo-Picker:", NK_TEXT_RIGHT)
    if nk.ComboBeginColor(color.r, color.g, color.b, color.a, 200, 400)
        nk.LayoutRowDynamic(120, 1)
        color = nk.ColorFromInt(nk.ColorPicker(color.r/255.0, color.g/255.0,
color.b/255.0, color.a/255.0, NK_RGBA))
        nk.ComboEnd()
    endif
    nk.GroupEnd()
endcase

// the edit demo
case DEMO_EDIT:
    // this demo shows one input field and several buttons to
    // change the text within the input field. marks are read
    // and characters are inserted before and after the marks.
    nk.LayoutRowDynamic(200, 1)
    nk.GroupBegin("EditGrp", NK_WINDOW_BORDER|NK_WINDOW_NO_SCROLLBAR)
    ratio[0] = 0.25
    ratio[1] = 0.75
    nkLayoutRow(NK_DYNAMIC, 190, ratio)
    nk.GroupBegin("EditBtnGrp", NK_WINDOW_NO_SCROLLBAR)
    nk.LayoutRowDynamic(20, 1)
    // Button creation
    if nk.ButtonLabel("Italic")
        // read the beginning of the marker
        pos = nk.EditGetMarkStart()

        // insert text on the readed position
        text = Left(text, pos) + "[i]" + Right(text, len(text)-pos)

        // read the ending of the marker and add the length of
        // the inserted text. because the insertion of the
        // text takes place in a string. The widget has no
        // knowledge of this.
        pos = nk.EditGetMarkEnd()+3

        // now insert a text at the end of the marker.
        text = Left(text, pos) + "[\i]" + Right(text, len(text)-pos)
    endif

    // same as above
    if nk.ButtonLabel("Bold")
        pos = nk.EditGetMarkStart()
        text = Left(text, pos) + "[b]" + Right(text, len(text)-pos)
        pos = nk.EditGetMarkEnd()+3
        text = Left(text, pos) + "[\b]" + Right(text, len(text)-pos)
    endif

    // same as above
    if nk.ButtonLabel("Underline")
        pos = nk.EditGetMarkStart()
        text = Left(text, pos) + "[u]" + Right(text, len(text)-pos)
        pos = nk.EditGetMarkEnd()+3
        text = Left(text, pos) + "[\u]" + Right(text, len(text)-pos)
    endif
endcase

```

```

// same as above
if nk.ButtonLabel("Center")
    pos = nk.EditGetMarkStart()
    text = Left(text, pos) + "[center]" + Right(text, len(text)-pos)
    pos = nk.EditGetMarkEnd()+8
    text = Left(text, pos) + "[\center]" + Right(text, len(text)-pos)
endif

// same as above
if nk.ButtonLabel("Code")
    pos = nk.EditGetMarkStart()
    text = Left(text, pos) + "[code]" + Right(text, len(text)-pos)
    pos = nk.EditGetMarkEnd()+6
    text = Left(text, pos) + "[\code]" + Right(text, len(text)-pos)
endif
nk.GroupEnd()

// now display EditString with the content of 'text', a
// maximum text length of 1024 characters, a default
// filter and an edit field style.
// The changed text is saved by 'EditString' back to text.
text = nk.EditString(NK_EDIT_BOX, text, 1024, NK_FILTER_DEFAULT)
nk.GroupEnd()
endcase

case DEMO_SELECTABLE:
    // this demo is very simple. It shows a grid and remembers
    // the responsible of the selectables. The logic only
    // changes the symbol.
    nk.LayoutRowDynamic(150, 1)
    nk.GroupBegin("SelectableGrp", NK_WINDOW_BORDER||NK_WINDOW_NO_SCROLLBAR)
    nk.LayoutRowStatic(30,30,4)
    for i=0 to 15
        if selected[i] then symbol = NK_SYMBOL_X else symbol = NK_SYMBOL_CIRCLE_OUTLINE
        selected[i] = nk.SelectableSymbolLabel(symbol, " ", NK_TEXT_CENTERED,
selected[i])
    next
    nk.GroupEnd()
endcase
endselect
endif
nk.WindowEnd()

nkSync()
loop

```

About Chart and Popups

Chart - Function overview (Plugin - .DLL/.SO)

```
Integer ChartBegin(type as Integer, num as Integer, min as Float, max as Float)
```

```
Integer ChartBeginColored(type as Integer, color as Integer, active as Integer, num as Integer, min as Float, max as Float)
```

```
ChartAddSlot(type as Integer, count as Integer, min_value as Float, max_value as Float)
```

```
ChartAddSlotColored(type as Integer, color as Integer, active as Integer, count as Integer, min_value as Float, max_value as Float)
```

```
Integer ChartPush(value as Float)
```

```
Integer ChartPushSlot(value as Float, slot as Integer)
```

```
ChartEnd()
```

Chart - Parameter (Plugin)

- **type**
A chart type `NK_CHART_LINES`, `NK_CHART_COLUMN` or `NK_CHART_MAX`
- **num**
Number of values to be filled in.
- **min, max**
The lowest and highest value.
- **color**
Colour of the chart.
- **active**
Colour of the selected value. (Highlightcolor)
- **count**
Number of values to be filled in a slot.
- **min_value, max_value**
The lowest and highest value in a slot.
- **value**
A value that is inserted into a chart.
- **slot**
An integer value that specifies a slot.

Popup - Function overview (Plugin - .DLL/.SO)

```
Integer PopupBegin(type as Integer, title as String, flags as Integer, x as Float, y as Float, w as Float, h as Float)
```

```
PopupClose()
```

```
PopupEnd()
```

```
Integer PopupGetScrollX()
```

```
Integer PopupGetScrollY()
```

```
PopupSetScroll(offset_x as Integer, offset_y as Integer)
```

Popup - Parameter (Plugin)

- **type**
A pop-up type `NK_POPUP_STATIC` or `NK_POPUP_DYNAMIC`
- **title**
A title. Also used internally to identify a pop-up.
- **flags**
These are normal windowflags [see „About Window“](#).
- **x,y**
Position of the popup panel.
- **w,h**
Size of the popup panel.
- **offset_x, offset_y**
Scrolloffset position of the panel.

Chart and Popup – Example

Listing 11

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

#constant POPUP_1 = 1
```

```

#constant POPUP_2 = 2
#constant FNC_SINUS = 0
#constant FNC_COSINUS = 1

global steps as integer = 32
global popup as integer = 0
global mode as integer = NK_CHART_COLUMN
global fnc as integer = 0
global chart_array as float[23]
global pop2_min as float = -25.0
global pop2_max as float = 100.0

// For the second chart the values are stored in an array.
i as integer
for i=0 to chart_array.length
    chart_array[i] = (Random(0, 1250)-250) / 10.0
next

do
    // pass agk inputs to nuklear
    nk.HandleInput()

    // open window
    if nk.WindowBegin("Chart and Popup Demo", 80, 50, 640, 360, NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||
NK_WINDOW_TITLE)
        client as nk_rect
        bounds as nk_rect

        // Get client area from the window.
        client = nkWindowGetContentRegion()
        nk.LayoutRowDynamic(120, 1)

        // Get area from the widget.
        bounds = nkWidgetBounds()
        min as float = 0.0

        // If fnc is set to GNC_COSINUS, the minimum value must be set to
        // -1. cos returns the value range -1.0 to 1.0.
        if fnc = FNC_COSINUS then min = -1.0

        // Start drawing the chart parameters are (from left to right)
        // drawing mode (line- or bar- graphic), number of values minimum
        // value and maximum value.
        nk.ChartBegin(mode, steps, min, 1.0)

        // Insert all values between 0 and 180.
        for i=0 to steps-1
            if fnc = FNC_SINUS
                nk.ChartPush(Sin((i*(180.0/steps))))
            else
                nk.ChartPush(Cos((i*(180.0/steps))))
            endif
        next
        nk.ChartEnd()

        // If the right mouse button was pressed on the chart, the position
        // of the mouse pointer is remembered so that one of the popup
        // fields can be opened there. The popup variable is set so that
        // the correct popup window can be opened later.
        if nkInputIsMouseClicked(NK_BUTTON_RIGHT, bounds)
            popup = POPUP_1

            // Calculate the upper left corner of the pop-up window.

```

```

        mx = GetRawMouseX() - client.x-3
        my = GetRawMouseY() - client.y-3
    endif

    // Get the next widget area.
    bounds = nkWidgetBounds()

    // Now another way to pass the values to a chart. Here it is
    // done with an array.
    nkChartColoredArray(NK_CHART_COLUMN, nkRGB(255, 255, 0), nkRGB(0,255,255), chart_array,
pop2_min, pop2_max)

    // Also here a query for the right mouse button to open a popup
    // window for the second chart.
    if nkInputIsMouseClickedInRect(NK_BUTTON_RIGHT, bounds)
        popup = POPUP_2
        mx = GetRawMouseX() - client.x-3
        my = GetRawMouseY() - client.y-3
    endif

    // Depending on the variable popup, a corresponding popup window
    // is opened.
    select popup
        case POPUP_1:
            PopupChart1(mx, my)
        endcase
        case POPUP_2:
            PopupChart2(mx, my)
        endcase
    endselect
endif
nk.WindowEnd()

nkSync()
loop

function PopupChart1(mx as float, my as float)
    // Like a normal window, a pop-up window is opened here. It requires
    // a popup type, a title, the window flags, the position of the popup
    // window and the size.
    if nk.PopupBegin(NK_POPUP_STATIC, "popup_chart_1", NK_WINDOW_BORDER||NK_WINDOW_DYNAMIC||
NK_WINDOW_NO_SCROLLBAR, mx, my, 220,140)
        // give it a layout of one dynamic column and a row height of
        // 20 pixels.
        nk.LayoutRowDynamic(20, 1)

        // let the user choose between a bar graph or a line graph
        if nk.OptionLabel("Bar Chart", mode = NK_CHART_COLUMN) then mode = NK_CHART_COLUMN
        if nk.OptionLabel("Line Chart", mode = NK_CHART_LINES) then mode = NK_CHART_LINES

        // Selection of the accuracy. Determines the number of steps how
        // the sine/cosine curve is displayed.
        label as String
        select steps
            case 8: label = "LOW - Resolution ": endcase
            case 16: label = "MED - Resolution ": endcase
            case 32: label = "HIGH - Resolution ": endcase
            case 64: label = "ULTRA - Resolution ": endcase
        endselect
        label = label + str(steps)

        ratio as float[2] = [0.1,0.8,0.1]
    end
endfunction

```

```

nkLayoutRow(NK_DYNAMIC, 20, ratio)

if nk.ButtonSymbol(NK_SYMBOL_TRIANGLE_LEFT) And steps > 8 then steps = steps / 2
nk.Label(label, NK_TEXT_CENTERED)
if nk.ButtonSymbol(NK_SYMBOL_TRIANGLE_RIGHT) And steps < 64 then steps = steps * 2

// Again as option widget selection of the curve.
nk.LayoutRowDynamic(20, 1)
if nk.OptionLabel("Sinus Fnc", fnc = FNC_SINUS) then fnc = FNC_SINUS
if nk.OptionLabel("Cosinus Fnc", fnc = FNC_COSINUS) then fnc = FNC_COSINUS

// If the mouse cursor is moved outside the pop-up, the pop-up
// window should close.
if nk.WindowIsHovered() = NK_FALSE
    popup = 0
    nk.PopupClose()
endif
endif

// Closes the pop-up window
nk.PopupEnd()
endfunction

// The construction of the second popup window is similar to the first,
// but with other widgets inside. It offers the possibility to add new
// random values and to change the limits min/max.
function PopupChart2(mx as float, my as float)
    if nk.PopupBegin(NK_POPUP_STATIC, "popup_chart_2", NK_WINDOW_BORDER||NK_WINDOW_DYNAMIC||
NK_WINDOW_NO_SCROLLBAR, mx, my, 220,140)
        nk.LayoutRowDynamic(20, 1)

        if nk.ButtonLabel("Randomize")
            for i=0 to chart_array.length
                chart_array[i] = (Random(0, (pop2_max-pop2_min)*10.0)+(pop2_min*10.0)) / 10.0
            next
            popup = 0
        endif

        pop2_min = nk.PropertyFloat("Minimum", -500.0, pop2_min, 0.0, 2.5,1.0)
        pop2_max = nk.PropertyFloat("Maximum", 0.0, pop2_max, 500.0, 2.5,1.0)

        if nk.WindowIsHovered() = NK_FALSE And nk.InputIsMouseDown(NK_BUTTON_LEFT) = NK_FALSE
            popup = 0
            nk.PopupClose()
        endif
    endif
    nk.PopupEnd()
endfunction

```

About ColorPicker, Widgets and Draw Commands

ColorPicker - Function Overview (Plugin - .DLL/.SO)

```
Integer ColorPicker(r as Float, g as Float, b as Float, a as Float, format as Integer)
```

Parameter (Plugin)

- **r,g,b,a**
Current selected color.
- **format**
Format of the colorPicker, with alpha channel (NK_RGBA) or without alpha channel (NK_RGB)

Widget - Function Overview (Plugin - .DLL/.SO)

Widget and WidgetFitting return the same result. The parameters *paddingx*, *paddingy* are ignored.

```
Integer Widget()
```

```
Integer WidgetFitting(paddingx as Float, paddingy as Float)
```

```
Integer WidgetGetLastState()
```

```
Integer WidgetBounds()
```

```
Integer WidgetPosition()
```

```
Integer WidgetSize()
```

```
Float WidgetWidth()
```

```
Float WidgetHeight()
```

```
Integer WidgetIsHovered()
```

```
Integer WidgetIsMouseClicked(btn as Integer)
```

```
Integer WidgetHasMouseClicked(btn as Integer, down as Integer)
```

```
Spacing(cols as Integer)
```

Parameter (Plugin)

- **paddingx, paddingy**
Ignored by the function.
- **btn**
Which mouse button should be checked?
- **down**
The status that is being checked. 1 (NK_TRUE) button pressed or 0 (NK_FALSE) button not pressed.
- **cols**
Columns that are skipped in the layout.

Draw Commands - Function Overview (Plugin - .DLL/.SO)

Whenever it is necessary to use these drawing commands, the area to be drawn on must be copied into a designated *slot* (`windowStoreCanvas`). For each of these commands it is necessary to specify this *slot* as the first parameter.

<code>StrokeLine(slot as Integer, x0 as Float, y0 as Float, x1 as Float, y1 as Float, thickness as Float, col as Integer)</code>
<code>StrokeCurve(slot as Integer, ax as Float, ay as Float, ctrl0x as Float, ctrl0y as Float, ctrl1x as Float, ctrl1y as Float, bx as Float, by as Float, thickness as Float, col as Integer)</code>
<code>StrokeRect(slot as Integer, x as Float, y as Float, w as Float, h as Float, rounding as Float, thickness as Float, col as Integer)</code>
<code>FillRect(slot as Integer, x as Float, y as Float, w as Float, h as Float, rounding as Float, col as Integer)</code>
<code>FillRectMultiColor(slot as Integer, x as Float, y as Float, w as Float, h as Float, col_left as Integer, col_top as Integer, col_right as Integer, col_bottom as Integer)</code>
<code>StrokeCircle(slot as Integer, x as Float, y as Float, w as Float, h as Float, thickness as Float, col as Integer)</code>
<code>FillCircle(slot as Integer, x as Float, y as Float, w as Float, h as Float, col as Integer)</code>
<code>StrokeArc(slot as Integer, cx as Float, cy as Float, radius as Float, a_min as Float, a_max as Float, thickness as Float, col as Integer)</code>
<code>FillArc(slot as Integer, cx as Float, cy as Float, radius as Float, a_min as Float, a_max as Float, col as Integer)</code>
<code>StrokeTriangle(slot as Integer, x0 as Float, y0 as Float, x1 as Float, y1 as Float, x2 as Float, y2 as Float, thickness as Float, col as Integer)</code>
<code>FillTriangle(slot as Integer, x0 as Float, y0 as Float, x1 as Float, y1 as Float, x2 as Float, y2 as Float, col as Integer)</code>
<code>DrawImage0(slot as Integer, x as Float, y as Float, w as Float, h as Float, img_slot as Integer, col as Integer)</code>
<code>DrawImage1(slot as Integer, x as Float, y as Float, w as Float, h as Float, img_name as String, col as Integer)</code>
<code>DrawText(slot as Integer, x as Float, y as Float, w as Float, h as Float, text as String, fnt_name as String, colb as Integer, colf as Integer)</code>

Parameter (Plugin)

- **slot**
A slot where the drawing area was copied with `windowStoreCanvas`.
- **x0, y0**
First point of a line or triangle.
- **x1, y1**
Second point of a line or triangle.
- **x2, y2**
Third point of a triangle.
- **thickness**
Line thickness
- **col**
Color in which is drawn.
- **ax, ay**
First point of a curved line.

- **bx, by**
Second point of a curved line.
- **ctrl0x, ctrl0y**
A control point to the first point (ax,ay). Defines how the curve will look like.
- **ctrl1x, ctrl1y**
A control point to the second point (bx,by). Defines how the second curve will look like.
- **x, y**
A point that defines a position on the drawing area.
- **w, h**
Specify a size - width and height.
- **rounding**
How much are the corners of the rect rounded.
- **col_left**
Color of the left edge of the rect.
- **col_top**
Color of the top edge of the rect.
- **col_right**
Color of the right edge of the rect.
- **col_bottom**
Color of the bottom edge of the rect.
- **cx, cy**
Center position of an Arc-Command.
- **radius**
Radius of an Arc-Command
- **a_min, a_max**
Beginning angle and the end angle of the arc in radians.
- **img_slot**
An image slot where an image has been copied. (ImageToSlot)
- **img_name**
An image name where an image was created. (ImageCreate)
- **text**
The text to be displayed.
- **fnt_name**
The font name in which the text should be displayed. (FontAtLasAdd...)
- **colb**
Back color.
- **colf**
Front (text) color.

Chart and Popup – Example

Listing 11

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

#constant POPUP_1 = 1
#constant POPUP_2 = 2
#constant FNC_SINUS = 0
#constant FNC_COSINUS = 1

global steps as integer = 32
global popup as integer = 0
global mode as integer = NK_CHART_COLUMN
global fnc as integer = 0
global chart_array as float[23]
global pop2_min as float = -25.0
global pop2_max as float = 100.0

// For the second chart the values are stored in an array.
i as integer
for i=0 to chart_array.length
    chart_array[i] = (Random(0, 1250)-250) / 10.0
next

do
    // pass agk inputs to nuklear
    nk.HandleInput()

    // open window
    if nk.WindowBegin("Chart and Popup Demo", 80, 50, 640, 360, NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||
NK_WINDOW_TITLE)
        client as nk_rect
        bounds as nk_rect

        // Get client area from the window.
        client = nkWindowGetContentRegion()
        nk.LayoutRowDynamic(120, 1)

        // Get area from the widget.
```



```

bounds = nkWidgetBounds()
min as float = 0.0

// If fnc is set to GNC_COSINUS, the minimum value must be set to
// -1. cos returns the value range -1.0 to 1.0.
if fnc = FNC_COSINUS then min = -1.0

// Start drawing the chart parameters are (from left to right)
// drawing mode (line- or bar- graphic), number of values minimum
// value and maximum value.
nk.ChartBegin(mode, steps, min, 1.0)

// Insert all values between 0 and 180.
for i=0 to steps-1
    if fnc = FNC_SINUS
        nk.ChartPush(Sin((i*(180.0/steps))))
    else
        nk.ChartPush(Cos((i*(180.0/steps))))
    endif
next
nk.ChartEnd()

// If the right mouse button was pressed on the chart, the position
// of the mouse pointer is remembered so that one of the popup
// fields can be opened there. The popup variable is set so that
// the correct popup window can be opened later.
if nkInputIsMouseClicked(NK_BUTTON_RIGHT, bounds)
    popup = POPUP_1

    // Calculate the upper left corner of the pop-up window.
    mx = GetRawMouseX() - client.x-3
    my = GetRawMouseY() - client.y-3
endif

// Get the next widget area.
bounds = nkWidgetBounds()

// Now another way to pass the values to a chart. Here it is
// done with an array.
nkChartColoredArray(NK_CHART_COLUMN, nkRGB(255, 255, 0), nkRGB(0,255,255), chart_array,
pop2_min, pop2_max)

// Also here a query for the right mouse button to open a popup
// window for the second chart.
if nkInputIsMouseClicked(NK_BUTTON_RIGHT, bounds)
    popup = POPUP_2
    mx = GetRawMouseX() - client.x-3
    my = GetRawMouseY() - client.y-3
endif

// Depending on the variable popup, a corresponding popup window
// is opened.
select popup
case POPUP_1:
    PopupChart1(mx, my)
endcase
case POPUP_2:
    PopupChart2(mx, my)
endcase
endselect
endif
nk.WindowEnd()

```

```

    nkSync()
loop

function PopupChart1(mx as float, my as float)
    // Like a normal window, a pop-up window is opened here. It requires
    // a popup type, a title, the window flags, the position of the popup
    // window and the size.
    if nk.PopupBegin(NK_POPUP_STATIC, "popup_chart_1", NK_WINDOW_BORDER||NK_WINDOW_DYNAMIC||
NK_WINDOW_NO_SCROLLBAR, mx, my, 220,140)
        // give it a layout of one dynamic column and a row height of
        // 20 pixels.
        nk.LayoutRowDynamic(20, 1)

        // let the user choose between a bar graph or a line graph
        if nk.OptionLabel("Bar Chart", mode = NK_CHART_COLUMN) then mode = NK_CHART_COLUMN
        if nk.OptionLabel("Line Chart", mode = NK_CHART_LINES) then mode = NK_CHART_LINES

        // Selection of the accuracy. Determines the number of steps how
        // the sine/cosine curve is displayed.
        label as String
        select steps
            case 8: label = "LOW - Resolution ": endcase
            case 16: label = "MED - Resolution ": endcase
            case 32: label = "HIGH - Resolution ": endcase
            case 64: label = "ULTRA - Resolution ": endcase
        endselect
        label = label + str(steps)

        ratio as float[2] = [0.1,0.8,0.1]
        nkLayoutRow(NK_DYNAMIC, 20, ratio)

        if nk.ButtonSymbol(NK_SYMBOL_TRIANGLE_LEFT) And steps > 8 then steps = steps / 2
        nk.Label(label, NK_TEXT_CENTERED)
        if nk.ButtonSymbol(NK_SYMBOL_TRIANGLE_RIGHT) And steps < 64 then steps = steps * 2

        // Again as option widget selection of the curve.
        nk.LayoutRowDynamic(20, 1)
        if nk.OptionLabel("Sinus Fnc", fnc = FNC_SINUS) then fnc = FNC_SINUS
        if nk.OptionLabel("Cosinus Fnc", fnc = FNC_COSINUS) then fnc = FNC_COSINUS

        // If the mouse cursor is moved outside the pop-up, the pop-up
        // window should close.
        if nk.WindowIsHovered() = NK_FALSE
            popup = 0
            nk.PopupClose()
        endif
    endif

    // Closes the pop-up window
    nk.PopupEnd()
endfunction

// The construction of the second popup window is similar to the first,
// but with other widgets inside. It offers the possibility to add new
// random values and to change the limits min/max.
function PopupChart2(mx as float, my as float)
    if nk.PopupBegin(NK_POPUP_STATIC, "popup_chart_2", NK_WINDOW_BORDER||NK_WINDOW_DYNAMIC||
NK_WINDOW_NO_SCROLLBAR, mx, my, 220,140)
        nk.LayoutRowDynamic(20, 1)

        if nk.ButtonLabel("Randomize")
            for i=0 to chart_array.length

```

```

        chart_array[i] = (Random(0, (pop2_max-pop2_min)*10.0)+(pop2_min*10.0)) / 10.0
    next
    popup = 0
endif

pop2_min = nk.PropertyFloat("Minimum", -500.0, pop2_min, 0.0, 2.5,1.0)
pop2_max = nk.PropertyFloat("Maximum", 0.0, pop2_max, 500.0, 2.5,1.0)

if nk.WindowIsHovered() = NK_FALSE And nk.InputIsMouseDown(NK_BUTTON_LEFT) = NK_FALSE
    popup = 0
    nk.PopupClose()
endif
endif
nk.PopupEnd()
endfunction

```

About ColorPicker, Widgets and Draw Commands

ColorPicker - Function Overview (Plugin - .DLL/.SO)

There is only one command here. Creates a box with a gradient in which the desired color can be picked. The return value is an integer with the **RGBA** values.

```
Integer ColorPicker(r as Float, g as Float, b as Float, a as Float, format as Integer)
```

Parameter (Plugin)

- **r,g,b,a**
Color components of the respective color including alpha value.
- **format**
*Which format the colorpicker should have. Should the Colorpicker be displayed with an alpha channel (**NK_RGBA**) or without (**NK_RGB**).*

Widget - Function Overview (Plugin - .DLL/.SO)

The **Widget** and **WidgetFitting** functions do not differ. They both return the range that the widget occupies. But the column counter will be incremented and the next widget will be displayed in the following column/row. If this should not happen, the functions **WidgetBounds**, **WidgetPosition**, **WidgetSize**, **WidgetWidth** and **WidgetHeight** are available.

The **Spacing** function can be used to skip the number of columns specified with the **cols** parameter.

```
Integer Widget()
```

```
Integer WidgetFitting(paddingx as Float, paddingy as Float)
```

```
Integer WidgetGetLastState()
```

```
Integer WidgetBounds()
```

```
Integer WidgetPosition()
```

```
Integer WidgetSize()
```

```
Float WidgetWidth()
```

```
Float WidgetHeight()
```

```
Integer WidgetIsHovered()
```

```
Integer WidgetIsMouseClicked(btn as Integer)
```

```
Integer WidgetHasMouseDown(btn as Integer, down as Integer)
```

```
Spacing(cols as Integer)
```

Parameter (Plugin)

- **paddingx, paddingy**
Padding currently has no effect on this command.
- **btn**
Specifies the mouse button to be checked. (see in section Input)
- **down**
Checks the status of a button. Returns true if the status matches the parameter.
- **cols**
Number of columns that are skipped.

Draw Commands - Function Overview (Plugin - .DLL/.SO)

These functions are only used to perform various drawing operations on a created window. All these functions need a **slot** in which the drawing area is specified with *WindowStoreCanvas*. There is also a counterpart to each of these functions in '**nuklear.agc**' which uses the corresponding UDT's to set the position and size (*nk_vec2*) or the whole area (*nk_rect*). Also the color is passed there with *nk_color*.

```
StrokeLine(slot as Integer, x0 as Float, y0 as Float, x1 as Float, y1 as Float, thickness as Float, color as Integer)
```

```
StrokeCurve(slot as Integer, x0 as Float, y0 as Float, ctrl0x as Float, ctrl0y as Float, ctrl1x as Float, ctrl1y as Float, x1 as Float, y1 as Float, thickness as Float, color as Integer)
```

```
StrokeRect(slot as Integer, x as Float, y as Float, w as Float, h as Float, rounding as Float, thickness as Float, color as Integer)
```

```
FillRect(slot as Integer, x as Float, y as Float, w as Float, h as Float, rounding as Float, color as Integer)
```

```
FillRectMultiColor(slot as Integer, x as Float, y as Float, w as Float, h as Float, lt as Integer, rt as Integer, rb as Integer, lb as Integer)
```

```
StrokeCircle(slot as Integer, x as Float, y as Float, w as Float, h as Float, thickness as Float, color as Integer)
```

```
FillCircle(slot as Integer, x as Float, y as Float, w as Float, h as Float, color as Integer)
```

```
StrokeArc(slot as Integer, cx as Float, cy as Float, radius as Float, a_min as Float, a_max as Float, thickness as Float, color as Integer)
```

```
FillArc(slot as Integer, cx as Float, cy as Float, radius as Float, a_min as Float, a_max as Float, color as Integer)
```

```
StrokeTriangle(slot as Integer, x0 as Float, y0 as Float, x1 as Float, y1 as Float, x2 as Float, y2 as Float, thickness as Float, color as Integer)
```

```
FillTriangle(slot as Integer, x0 as Float, y0 as Float, x1 as Float, y1 as Float, x2 as Float, y2 as Float, color as Integer)
```

```
DrawImage(slot as Integer, x as Float, y as Float, w as Float, h as Float, img_slot as Integer, color as Integer)
```

```
DrawImage(slot as Integer, x as Float, y as Float, w as Float, h as Float, img_name as String, color
```

```
as Integer)
```

```
DrawText(slot as Integer, x as Float, y as Float, w as Float, h as Float, label as String, font_name  
as String, back_col as Integer, front_col as Integer)
```

Draw Commands - Function Overview (AGK-Basic - .agc)

```
nkStrokeLine(canvas_slot as integer, p0 as nk_vec2, p1 as nk_vec2, thickness as float, color as  
nk_color)
```

```
nkStrokeCurve(canvas_slot as integer, p0 as nk_vec2, ctrl0 as nk_vec2, ctrl1 as nk_vec2, p1 as  
nk_vec2, thickness as float, color as nk_color)
```

```
nkStrokeRect(canvas_slot as integer, rect as nk_rect, rounding as float, thickness as float, color as  
nk_color)
```

```
nkFillRect(canvas_slot as integer, rect as nk_rect, rounding as float, color as nk_color)
```

```
nkFillRectMultiColor(canvas_slot as integer, rect as nk_rect, lt as nk_color, rt as nk_color, rb as  
nk_color, lb as nk_color)
```

```
nkStrokeCircle(canvas_slot as integer, center as nk_vec2, radius as float, thickness as float, color  
as nk_color)
```

```
nkFillCircle(canvas_slot as integer, center as nk_vec2, radius as float, color as nk_color)
```

```
nkStrokeArc(canvas_slot as integer, center as nk_vec2, radius as float, a_min as float, a_max as  
float, thickness as float, color as nk_color)
```

```
nkFillArc(canvas_slot as integer, center as nk_vec2, radius as float, a_min as float, a_max as float,  
color as nk_color)
```

```
nkStrokeTriangle(canvas_slot as integer, p0 as nk_vec2, p1 as nk_vec2, p3 as nk_vec2, thickness as  
float, color as nk_color)
```

```
nkFillTriangle(canvas_slot as integer, p0 as nk_vec2, p1 as nk_vec2, p3 as nk_vec2, color as  
nk_color)
```

```
nkDrawText(canvas_slot as integer, rect as nk_rect, label as String, font_name as String, back_col as  
nk_color, front_col as nk_color)
```

```
nkDrawImage(canvas_slot as integer, rect as nk_rect, img_slot as integer, color as nk_color)
```

Parameter (Plugin/AGK-Basic)

- **canvas_slot,slot**
A slot where a window space was previously registered with the `WindowsStoreCanvas` command
- **p0,x0,y0**
First window coordinate on the registered wind space.
- **p1,x1,y1**
Second window coordinate on the registered wind space.
- **p2,x2,y2**
Third window coordinate on the registered wind space.

- **Thickness**
Thickness of the line to be drawn.
- **Color**
Color in which to draw.
- **ctrl0,ctrl0x,ctrl0y**
*A control point to p1 for the curved line. Controls the direction and strength of the curve.
ATTENTION: The AGK-Basic version of this function expects a coordinate relative to p1.*
- **ctrl1,ctrl1x,ctrl1y**
*A control point to p2 for the curved line. Controls the direction and strength of the curve.
ATTENTION: The AGK-Basic version of this function expects a coordinate relative to p2.*
- **rect,x,y,w,h**
Specifies the position and size of the area to be drawn. The ...rect... functions draw a rectangle. The ...Image functions draw an image into this rectangle. DrawText will pass a valid area, everything that goes over it will be clipped. With the Plugin functions for drawing circles it behaves differently. The parameter rect is used to calculate a center point ($rect.x+rect.w/2$, $rect.y+rect.h/2$) and a radius ($rect.w/2$) with these values a circle is drawn.
- **Rounding**
Indicates how much the corners of a rectangle are rounded.
- **lt,rt,rb,lb**
Color for the respective corners of a rectangle. For the plugin functions they are integer values and for the AGK basic functions they are nk_color values.
- **center,cx,cy**
Center of the circle and arc function.
- **radius**
Radius of the circle or arc.
- **a_min,a_max**
The angular range of the arc.
- **label**
A text to be output.
- **font_name**
A font name that was previously generated during initialization.
- **back_col,front_col**
Background and draw color of the text function.
- **img_slot**
An image slot that was previously generated with ImageToSlot.
- **img_name**
An image name that was previously generated with CreateImage.

Colorpicker and Drawing – Example

Listing 12

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "Nuklear Demo" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// constants for the tabs.
#constant PICKER_TAB = 1
#constant EXTRA_TAB = 2
#constant DRAWING_TAB = 3

// basic nuklear initialization
nkInit()

nk.FontStashBegin()
    nk.FontAtlasAddFromSystem("arial", "arial.ttf", 24)
nk.FontStashEnd()

LoadImage(1, "tools.png")
nk.ImageToSlot(0, 1)

do
    // pass agk inputs to nuklear
    nk.HandleInput()

    // open the window
    if nk.WindowBegin("ColorPicker, Widget and Drawing Commands Demo", 40, 50, 1024, 576,
        NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||NK_WINDOW_TITLE||NK_WINDOW_NO_SCROLLBAR)

        // Here is a simple way to make notetabs.
        nk.LayoutRowStatic(20, 150, 3)
        if nk.SelectableLabel("Colorpicker", NK_TEXT_CENTERED, selected = PICKER_TAB) then selected =
PICKER_TAB
        if nk.SelectableLabel("Extra", NK_TEXT_CENTERED, (selected = EXTRA_TAB)) then selected =
EXTRA_TAB
        if nk.SelectableLabel("Drawing", NK_TEXT_CENTERED, selected = DRAWING_TAB) then selected =
DRAWING_TAB

        // depending on the selected tab the corresponding function is
        // branched to. These functions define the remaining window
        // structure.
        select selected
            case PICKER_TAB: do_picker_tab():endcase
            case EXTRA_TAB: do_extra_tab():endcase
```



```

        case DRAWING_TAB: do_drawing_tab():endcase
    endselect
endif
nk.WindowEnd()

nkSync()
loop

// this demonstrates the colorpicker. The currently selected color is
// passed as nk_colorf and with NK_RGBA/NK_RGB you can choose whether the
// colorpicker should be displayed with or without alpha channel. The
// return value is the changed color as integer.
function do_picker_tab()
    global sel_color as nk_colorf
    result as integer

    // layout and the call of the colorpicker.
    nk.LayoutRowStatic(256, 256, 1)
    result = nkColorPicker(sel_color, NK_RGBA)

    // this displays the currently selected color.
    nk.LayoutRowStatic(25, 256, 1)
    nk.ButtonColor(result)

    // all color channels as values.
    nk.LayoutRowStatic(20, 64, 4)
    nk.Label("R:"+str(GetColorRed(result)), NK_TEXT_CENTERED)
    nk.Label("G:"+str(GetColorGreen(result)), NK_TEXT_CENTERED)
    nk.Label("B:"+str(GetColorBlue(result)), NK_TEXT_CENTERED)
    nk.Label("A:"+str(GetColorAlpha(result)), NK_TEXT_CENTERED)
endfunction

// here you can see how to use the draw commands to draw over the widget.
function do_extra_tab()
    global value as integer

    // store window canvas in slot 0
    nk.WindowStoreCanvas(0)

    // prepare and display the info text
    nk.LayoutRowDynamic(25, 1)
    nk.Label("Here is an example of how to pimp a standard widget. Do you want to label the progress
bar?", NK_TEXT_CENTERED)

    // prepare the Progressbar
    nk.LayoutRowDynamic(40, 1)
    bounds as nk_rect

    // get the widget bounds of the following widget.
    bounds = nkWidgetBounds()

    // draw the progressbar
    value = nk.Progress(value, 1000, nk_true)

    // build the label
    label as String
    label = str(value/10.0,1)+"% Processed"

    // set the label in the center of the widget
    inc bounds.x,(bounds.w-nk.FontGetLabelWidth(label))/2.0
    inc bounds.y,(40.0-nk.FontGetHeight())/2.0

    // and draw the text/label

```

```

    nkDrawText(0, bounds, label, "default", nkRGBA(0,0,0,0), nkRGB(192,225,225))
endfunction

function do_drawing_tab()
    white as nk_color
    black as nk_color
    red as nk_color
    green as nk_color
    blue as nk_color
    yellow as nk_color
    bounds as nk_rect
    txt_rect as nk_rect
    draw_rect as nk_rect

    // Store window canvas to slot 0.
    nk.WindowStoreCanvas(0)

    // get position and size of the whole window content.
    bounds = nkWindowGetContentRegion()
    txt_rect.x = bounds.x+25
    txt_rect.y = bounds.x+60
    txt_rect.w = 128
    txt_rect.h = 20
    draw_rect = txt_rect
    inc draw_rect.y, 30
    draw_rect.w = 120
    draw_rect.h = 90

    // define some default variables
    white = nkRGB(255,255,255)
    black = nkRGB(0,0,0)
    red = nkRGB(255,0,0)
    green = nkRGB(0,255,0)
    blue = nkRGB(0,0,255)
    yellow = nkRGB(255,255,0)

    // draw a rect around the entire drawable window space.
    nkStrokeRect(0, bounds, 15.0, 2.0, white)

    // draw the text and the corresponding line.
    nkDrawText(0, txt_rect, "Line", "arial", black, yellow)
    nkStrokeLine(0, nk_vec2(draw_rect.x,draw_rect.y),
nk_vec2(draw_rect.x+draw_rect.w,draw_rect.y+draw_rect.h), 2, red)

    // move the Text-Rect and Drawable-Rect to the right.
    // draw the text and the corresponding curve.
    inc txt_rect.x, 128:inc draw_rect.x, 128
    nkDrawText(0, txt_rect, "Curve", "arial", black, yellow)
    nkStrokeCurve(0, nk_vec2(draw_rect.x,draw_rect.y), nk_vec2(50,-30), nk_vec2(-50, 30),
nk_vec2(draw_rect.x+draw_rect.w,draw_rect.y+draw_rect.h), 2.0, red)

    // ...
    inc txt_rect.x, 128:inc draw_rect.x, 128
    nkDrawText(0, txt_rect, "Rect", "arial", black, yellow)
    nkStrokeRect(0, draw_rect, 0, 2, red)

    // ...
    inc txt_rect.x, 128:inc draw_rect.x, 128
    nkDrawText(0, txt_rect, "FillRect", "arial", black, yellow)
    nkFillRect(0, draw_rect, 10, red)

    // ...
    inc txt_rect.x, 128:inc draw_rect.x, 128

```

```

nkDrawText(0, txt_rect, "FillRectMC", "arial", black, yellow)
nkFillRectMultiColor(0, draw_rect, red, green, blue, black)

// ...
inc txt_rect.x, 128:inc draw_rect.x, 128
nkDrawText(0, txt_rect, "Circle", "arial", black, yellow)
nkStrokeCircle(0, nk_vec2(draw_rect.x+60,draw_rect.y+45), 45.0, 2, red)

// ...
inc txt_rect.x, 128:inc draw_rect.x, 128
nkDrawText(0, txt_rect, "FillCircle", "arial", black, yellow)
nkFillCircle(0, nk_vec2(draw_rect.x+60,draw_rect.y+45), 45.0, red)

// move the Text-Rect and Drawable-Rect under the StrokeLine demo.
// draw the text and the coresponding arc.
txt_rect.x = bounds.x+25:draw_rect.x=txt_rect.x:inc txt_rect.y, 128:inc draw_rect.y, 128
nkDrawText(0, txt_rect, "Arc", "arial", black, yellow)
nkStrokeArc(0, nk_vec2(draw_rect.x+60,draw_rect.y+45), 60.0,-37.5*(3.14/180.0),37.5*(3.14/180.0),
2, red)

// move the Text-Rect and Drawable-Rect to the right.
// draw the text and the coresponding filled arc.
inc txt_rect.x, 128:inc draw_rect.x, 128
nkDrawText(0, txt_rect, "FillArc", "arial", black, yellow)
nkFillArc(0, nk_vec2(draw_rect.x+60,draw_rect.y+45), 60.0,-37.5*(3.14/180.0),37.5*(3.14/180.0),
red)

// ...
inc txt_rect.x, 128:inc draw_rect.x, 128
nkDrawText(0, txt_rect, "Triangle", "arial", black, yellow)
nkStrokeTriangle(0, nk_vec2(draw_rect.x, draw_rect.y), nk_vec2(draw_rect.x+draw_rect.w,
draw_rect.y), nk_vec2(draw_rect.x+(draw_rect.w/2), draw_rect.y+draw_rect.h), 2, red)

// ...
inc txt_rect.x, 128:inc draw_rect.x, 128
nkDrawText(0, txt_rect, "FillTriangle", "arial", black, yellow)
nkFillTriangle(0, nk_vec2(draw_rect.x, draw_rect.y), nk_vec2(draw_rect.x+draw_rect.w,
draw_rect.y), nk_vec2(draw_rect.x+(draw_rect.w/2), draw_rect.y+draw_rect.h), red)

// ...
inc txt_rect.x, 128:inc draw_rect.x, 128
nkDrawText(0, txt_rect, "Image", "arial", black, yellow)
nkDrawImage(0, draw_rect, 0, red)
endfunction

```

About Input

Input - Function Overview (Plugin - .DLL/.SO)

```
Input - Function Overview (Plugin - .DLL/.SO)
Integer InputHasMouseClicked( id as Integer)
Integer InputHasMouseClickedInRect( id as Integer, x as Float, y as Float, w as Float, h as Float)
Integer InputHasMouseClickedDownInRect( id as Integer, x as Float, y as Float, w as Float, h as Float, down as Integer)
Integer InputIsMouseClickedInRect( id as Integer, x as Float, y as Float, w as Float, h as Float)
Integer InputIsMouseClickedDownInRect( id as Integer, x as Float, y as Float, w as Float, h as Float, down as Integer)
Integer InputAnyMouseClickedInRect( x as Float, y as Float, w as Float, h as Float)
Integer InputIsMousePrevHoveringRect( x as Float, y as Float, w as Float, h as Float)
Integer InputIsMouseHoveringRect( x as Float, y as Float, w as Float, h as Float)
Integer InputMouseClicked( id as Integer, x as Float, y as Float, w as Float, h as Float)
Integer InputIsMouseDown( id as Integer)
Integer InputIsMousePressed( id as Integer)
Integer InputIsMouseReleased( id as Integer)
Integer InputIsKeyPressed( key as Integer)
Integer InputIsKeyReleased( key as Integer)
Integer InputIsKeyDown( key as Integer)
Float InputGetMouseDeltaX()
Float InputGetMouseDeltaY()
```

Input - Function Overview (AGK-Basic - .AGC)

```
nkInputIsMouseClickedInRect(button as integer, rect ref as nk_rect)
nkInputHasMouseClickedInRect(button as integer, rect ref as nk_rect)
nkInputHasMouseClickedDownInRect(button as integer, rect ref as nk_rect, down as integer)
nkInputIsMouseClickedDownInRect(button as integer, rect ref as nk_rect, down as integer)
nkInputAnyMouseClickedInRect(rect ref as nk_rect)
nkInputIsMousePrevHoveringRect(rect ref as nk_rect)
nkInputIsMouseHoveringRect(rect ref as nk_rect)
nkInputMouseClicked(button as integer, rect ref as nk_rect)
nkInputGetMouseDelta()
```

Parameter (Plugin)

- **id**
ButtonId which should be checked.
- **x,y,w,h**
Indicates a position and size of a rectangle
- **down**
Can be `nk_true` (1) or `nk_false` (0). State of a button to be checked.
- **key**
Key code.

Input – Example

Listing 13

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "Input Demo" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 10, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

// some variables for displaying actions.
cur_name as string = ""
cur_bound as nk_rect
cur_state as string = ""
do

    // pass agk inputs to nuklear
    nk.HandleInput()

    // make a data view window
    if nk.WindowBegin("Input Data Display", 0, 0, 1280, 90, NK_WINDOW_BORDER|NK_WINDOW_TITLE||
NK_WINDOW_NO_SCROLLBAR)
        nk.LayoutRowDynamic(20, 10)
        // first row
        // labeling of the data
        nk.Label("Mouse X", NK_TEXT_CENTERED)
        nk.Label("Mouse Y", NK_TEXT_CENTERED)
        nk.Label("Delta X", NK_TEXT_CENTERED)
        nk.Label("Delta Y", NK_TEXT_CENTERED)
        nk.Label("Button Left", NK_TEXT_CENTERED)
        nk.Label("Button Right", NK_TEXT_CENTERED)
```

```

nk.Label("Button Middle", NK_TEXT_CENTERED)
nk.Label("Current Window", NK_TEXT_CENTERED)
nk.Label("Current Bound", NK_TEXT_CENTERED)
nk.Label("Current Bu.State", NK_TEXT_CENTERED)

// second row
// display mouse position and mouse move (delta)
nk.Label(str(GetRawMouseX(),0), NK_TEXT_CENTERED)
nk.Label(str(GetRawMouseY(),0), NK_TEXT_CENTERED)
nk.Label(str(nk.InputGetMouseDeltaX(),0), NK_TEXT_CENTERED)
nk.Label(str(nk.InputGetMouseDeltaY(),0), NK_TEXT_CENTERED)

// display mouse buttons
down_text as string = ""
if nk.InputIsMouseDown(NK_BUTTON_LEFT) then down_text = " Hold "
if nk.InputIsMousePressed(NK_BUTTON_LEFT) then down_text = " Down "
if nk.InputIsMouseReleased(NK_BUTTON_LEFT) then down_text = " Up "
nk.Label(down_text, NK_TEXT_CENTERED)

down_text = ""
if nk.InputIsMouseDown(NK_BUTTON_RIGHT) then down_text = " Hold "
if nk.InputIsMousePressed(NK_BUTTON_RIGHT) then down_text = " Down "
if nk.InputIsMouseReleased(NK_BUTTON_RIGHT) then down_text = " Up "
nk.Label(down_text, NK_TEXT_CENTERED)

down_text = ""
if nk.InputIsMouseDown(NK_BUTTON_MIDDLE) then down_text = " Hold "
if nk.InputIsMousePressed(NK_BUTTON_MIDDLE) then down_text = " Down "
if nk.InputIsMouseReleased(NK_BUTTON_MIDDLE) then down_text = " Up "
nk.Label(down_text, NK_TEXT_CENTERED)

// and window data
nk.Label(cur_name, NK_TEXT_CENTERED)

down_text = ""
if len(cur_name) > 0 then down_text =
str(cur_bound.x,0)+", "+str(cur_bound.y,0)+": "+str(cur_bound.w,0)+"x"+str(cur_bound.h,0)
nk.Label(down_text, NK_TEXT_CENTERED)

nk.Label(cur_state, NK_TEXT_CENTERED)
endif
nk.WindowEnd()

win1_bound as nk_rect
win2_bound as nk_rect
win3_bound as nk_rect

cur_name = ""
cur_state = ""

// display first window
if nk.WindowBegin("Window 1", 50, 150, 320, 240, NK_WINDOW_BORDER||NK_WINDOW_TITLE||
NK_WINDOW_MOVABLE||NK_WINDOW_SCALABLE)
// get the bounds of this window
win1_bound = nkWindowGetBounds()

// if the mouse pointer is within the boundaries of this window,
// the window name, window boundaries and button status are set
// to variable.
if nk.InputIsMouseHoveringRect(win1_bound.x,win1_bound.y,win1_bound.w,win1_bound.h)
cur_name = "Window 1"
cur_bound = win1_bound

```

```

        if nk.InputIsMouseDown(NK_BUTTON_LEFT) then cur_state = "Left "
        if nk.InputIsMouseDown(NK_BUTTON_RIGHT) then cur_state = cur_state+"Right "
        if nk.InputIsMouseDown(NK_BUTTON_MIDDLE) then cur_state = cur_state+"Middle "
    endif
endif
nk.WindowEnd()

// display second window
if nk.WindowBegin("Window 2", 400, 420, 240, 240, NK_WINDOW_BORDER||NK_WINDOW_TITLE||
NK_WINDOW_MOVABLE||NK_WINDOW_SCALABLE)
    win2_bound = nkWindowGetBounds()
    if nk.InputIsMouseHoveringRect(win2_bound.x,win2_bound.y,win2_bound.w,win2_bound.h)
        if len(cur_name) = 0 or nk.WindowHasFocus()
            cur_name = "Window 2"
            cur_bound = win2_bound
            if nk.InputIsMouseDown(NK_BUTTON_LEFT) then cur_state = "Left "
            if nk.InputIsMouseDown(NK_BUTTON_RIGHT) then cur_state = cur_state+"Right "
            if nk.InputIsMouseDown(NK_BUTTON_MIDDLE) then cur_state = cur_state+"Middle "
        endif
    endif
endif
nk.WindowEnd()

// display third window
if nk.WindowBegin("Window 3", 670, 150, 240, 320, NK_WINDOW_BORDER||NK_WINDOW_TITLE||
NK_WINDOW_MOVABLE||NK_WINDOW_SCALABLE)
    win3_bound = nkWindowGetBounds()
    if nk.InputIsMouseHoveringRect(win3_bound.x,win3_bound.y,win3_bound.w,win3_bound.h)
        if len(cur_name) = 0 or nk.WindowHasFocus()
            cur_name = "Window 3"
            cur_bound = win3_bound
            if nk.InputIsMouseDown(NK_BUTTON_LEFT) then cur_state = "Left "
            if nk.InputIsMouseDown(NK_BUTTON_RIGHT) then cur_state = cur_state+"Right "
            if nk.InputIsMouseDown(NK_BUTTON_MIDDLE) then cur_state = cur_state+"Middle "
        endif
    endif
endif
nk.WindowEnd()

if len(cur_state) > 0 then cur_state = cur_state + "down"

nkSync()
loop

```

About Contextual

This is a dropdown menu that appears when the right mouse button is pressed. It is also possible to create a menu without these commands. But these functions simplify a lot.

Contextual - Function Overview (Plugin - .DLL/.SO)

```
Integer ContextualBegin(flags as Integer, width as Float, height as Float, bx as Float, by as Float, bw as Float, bh as Float)
```

```
Integer ContextualItemText(text as String, len as Integer, align as Integer)
```

```
Integer ContextualItemLabel(label as String, align as Integer)
```

```
Integer ContextualItemImageLabel0(image_slot as Integer, label as String, align as Integer)
```

```
Integer ContextualItemImageLabel1(image_name as String, label as String, align as Integer)
```

```
Integer ContextualItemImageText0(image_slot as Integer, text as String, len as Integer, align as Integer)
```

```
Integer ContextualItemImageText1(image_name as String, text as String, len as Integer, align as Integer)
```

```
Integer ContextualItemSymbolLabel(symbol as Integer, label as String, align as Integer)
```

```
Integer ContextualItemSymbolText(symbol as Integer, text as String, len as Integer, align as Integer)
```

```
ContextualClose()
```

```
ContextualEnd()
```

Input - Function Overview (AGK-Basic - .AGC)

```
nkContextualBegin(flags as integer, width as float, height as float, bound as nk_rect)
```

Parameter (Plugin)

- **Flags**
Windowflags, how the pop-up window is displayed.
- **width, height**
Height and width of the pop-up window
- **bx,by,bw,bh,bound**
Area that reacts to the right mouse button.
- **text,label**
Labeling of the menu entry
- **len**
Length of the text.
- **Align**
Alignment of the Text.
- **image_slot,image_name**
Image slot or image name of the image to be inserted in the menu.
- **symbol**
The symbol to be inserted in the menu.

About Styling

There are a lot of functions to adjust the style of its interface. Basically there are three different ways how you can change the look of the UI

1. via an XML styling file.
2. direct manipulation of the widget styles.
3. create individual widgets as templates and then replace them in a nuclear context.

So it's up to you which variant you prefer. Usually the styling file is preferred, because it can be changed and the change can be tested without recompiling the program. I present you the three variants.

Manipulation of widget styles

When the nuclear engine is initialised. the nuclear context is automatically read and stored in the global variable `nk_ctx` of type `nk_style`. To change a widgetstyle the best way would be to make a copy of the widgetstyle, change the desired parameters there and then write it back into the context.

Here a code snippet

```
// set default values.
current_style as nk_style_button
current_style = nk_ctx.button

// set current style text alignment to left
current_style.text_alignment = NK_TEXT_LEFT

// write the values from UDT current_style to the memblock
nkStyleSetButtonToMemblock(current_style, nk_ctx.memblocks[NK_WIDGETTYPE_BUTTON], 0)

// write the memblock to nuklear engines context.
nk.StyleSetFromMemblock(NK_WIDGETTYPE_BUTTON, nk_ctx.memblocks[NK_WIDGETTYPE_BUTTON])
```

Creating and using style templates

There are different ways to create a style template. I try to present each way here.

The first way would be to describe the widget using XML tags. This would be done by the `styleMake` function. The first parameter is a string containing the template name. So the template can be accessed at any time with this name.

The second parameter is the XML-tag that finally describes the widget. This XML tag is the same as it is found in the XML styling file.

The second way is to define the templates via the XML styling file. How this looks like you will see later in the examples.

XML styling file

The best and most flexible and also the recommended way to customize the UI style is via an XML styling file. This is loaded and set with `StyleLoadFromFile` or `StyleLoadFromString`. Again see the example at the end of the chapter.

Styling - Function Overview (Plugin - .DLL/.SO)

```
Integer StyleCreateMemblock(widget_type as Integer, memID as Integer)
Integer StyleCreateMemblock(nameID as String, type as Integer, memID as Integer)
Integer StyleCreateMemblock(type as Integer)
Integer StyleCreateMemblock(nameID as String, type as Integer)
StyleSetFromMemblock(nameID as String)
StyleSetFromMemblock(type as Integer, memID as Integer)
StyleDeleteMemblock(nameID as String)
Integer StyleMake(name as String, style_tag as String)
Integer StyleMakeItem(name as String, color as Integer)
Integer StyleMakeItem(style_tag as String)
Integer StyleMakeItem(name as String, img_name as String, sub_x as Integer, sub_y as Integer, sub_w as Integer, sub_h as Integer)
Integer StyleMakeItem(name as String, img_name as String, sub_x as Integer, sub_y as Integer, sub_w as Integer, sub_h as Integer, bl as Float, bt as Float, br as Float, bb as Float)
Integer StyleGetItemSize()
Integer StyleMakeImage(name as String, agk_id as Integer)
Integer StyleMakeImage(name as String, agk_id as Integer, sub_x as Integer, sub_y as Integer, sub_w as Integer, sub_h as Integer)
StylePush(style_name as String)
StylePushMemblock(mem_name as String)
StylePop()
StyleSetContext(name as String)
Integer StyleSetContextProperty(widget as String, prop_name as String, value as String)
String StyleGetContextProperty(widget as String, prop_name as String)
StyleColorTableBegin(flag as Integer)
StyleColorTableEnd()
StyleColorTablePush(widget_type as Integer, color as Integer)
String StyleGetColorTableTags()
String StyleGetColorTable()
StyleLoadFromString(xml as String)
StyleLoadFromFile(file as String)
Integer StyleMakeCursor(cursor_name as String, item_name as String, width as Float, height as Float, offsetx as Float, offsety as Float)
Integer StyleSetCursor(cursor_style as Integer)
StyleShowCursor()
StyleHideCursor()
StyleReplaceCursor(cursor_style as Integer, cursor_name as String)
```

Parameter (Plugin)

- **widget_type**
One of the constant from table 'widget type'. (NK_WIDGETTYPE_...)
- **memID**
A memoryblock id.
- **nameID**
A name ID to identify a memblock.

- **name**
A Name for a Style, Item or an Image.
- **style_tag**
This is a XML-Styletag. In the form of "<checkbox normal='checkbox' hover='checkbox' active='checkbox_cursor' text_normal='col_white' text_hover='col_white' text_active='col_white'/>"
- **color**
A color as an Integer value.
- **img_name**
An image name previously created with CreateImage.
- **sub_x, sub_y, sub_w, sub_h**
Position and size of a sub-image of the specified image.
- **bl, bt, br, bb**
Width of the non stretchable border (left, top, right, bottom).
- **agk_id**
AGK-ImageID that contains an image.
- **widget**
A widget name from the 'widget name' table.
- **propname**
A property name is the same as in the corresponding UDT of the specified widget.
- **value**
The value to be set.
- **flag**
If the flag is set (1), then the Colortable is filled with default values. If it is not set (0), the colortable is filled with 'empty' colours.
- **xml**
A string containing the entire XML style text.
- **file**
The file containing the XML style text.
- **cursor_name**
A name for the cursor to be created or used.
- **cursor_style**
The cursor type to be set. Is a value from the table 'Cursorstyle'.
- **item_name**
An item name that was previously created with styleItemMake. The item should be an image or a patch9 image.
- **width, height**
The width and height of the cursor.
- **offsetx, offsety**
The offset to the upper left corner, which serves as an action point.

Widget type

• NK_WIDGETTYPE_UNKNOWN	= 0
• NK_WIDGETTYPE_FONT	= 1
• NK_WIDGETTYPE_CURSORS	= 2
• NK_WIDGETTYPE_TEXT	= 3
• NK_WIDGETTYPE_BUTTON	= 4
• NK_WIDGETTYPE_CONTEXT_BUTTON	= 5
• NK_WIDGETTYPE_MENU_BUTTON	= 6
• NK_WIDGETTYPE_CHECKBOX	= 7
• NK_WIDGETTYPE_OPTION	= 8
• NK_WIDGETTYPE_TOGGLE	= 7
• NK_WIDGETTYPE_SELECTABLE	= 9
• NK_WIDGETTYPE_SLIDER	= 10
• NK_WIDGETTYPE_PROGRESS	= 11
• NK_WIDGETTYPE_PROPERTY	= 12
• NK_WIDGETTYPE_EDIT	= 13
• NK_WIDGETTYPE_CHART	= 14
• NK_WIDGETTYPE_SCROLLV	= 15
• NK_WIDGETTYPE_SCROLLH	= 16
• NK_WIDGETTYPE_SCROLLBAR	= 15
• NK_WIDGETTYPE_TAB	= 17
• NK_WIDGETTYPE_COMBO	= 18
• NK_WIDGETTYPE_WINDOW	= 19
• NK_WIDGETTYPE_WINDOW_HEADER	= 20

Widget name

- font
- cursors
- text
- button
- context_button
- menu_button
- checkbox
- option
- selectable
- slider
- progress
- property
- edit
- chart
- scrollv
- scrollh
- tab
- combo
- window

Cursorstyle

- NK_CURSOR_ARROW = 0
- NK_CURSOR_TEXT = 1
- NK_CURSOR_MOVE = 2
- NK_CURSOR_RESIZE_VERTICAL = 3
- NK_CURSOR_RESIZE_HORIZONTAL = 4
- NK_CURSOR_RESIZE_TOP_LEFT_DOWN_RIGHT = 5
- NK_CURSOR_RESIZE_TOP_RIGHT_DOWN_LEFT = 6

Styling - Function Overview (AGK-Basic - .AGC)

```
nkStyleGetPropertyItem(result ref as nk_style_item, property as String)
```

```
Integer nkStyleGetPropertyType(property as string)
```

```
Integer nkStyleGetPropertyItemType(property as String)
```

```
nkStyleGetPropertyItemColor(result ref as nk_color, property as string)
```

```
nk_image nkStyleGetPropertyItemImage(result ref as nk_image, property as string)
```

```
nk_patch9 nkStyleGetPropertyItemPatch9(result ref as nk_patch9, property as string)
```

```
nk_vec2 nkStyleGetPropertyVec2(result ref as nk_vec2, property as string)
```

```
Integer nkStyleGetPropertyInteger(property as string)
```

```
Float nkStyleGetPropertyFloat(property as string)
```

```
Integer nkStyleGetContextPropertyAsInteger(widget_sz as string, property as string)
```

```
Integer nkStyleSetTextFromMemblock(out ref as nk_style_text, memID as integer, offset as Integer)
```

```
Integer nkStyleSetTextToMemblock(widget as nk_style_text, memID as integer, offset as integer)
```

```
Integer nkStyleSetButtonFromMemblock(out ref as nk_style_button, memID as integer, offset as Integer)
```

```
Integer nkStyleSetButtonToMemblock(widget ref as nk_style_button, memID as integer, offset as Integer)
```

```
Integer nkStyleSetToggleFromMemblock(out ref as nk_style_toggle, memID as integer, offset as Integer)
```

```
Integer nkStyleSetToggleToMemblock(widget ref as nk_style_toggle, memID as integer, offset as Integer)
```

```
Integer nkStyleSetSelectableFromMemblock(out ref as nk_style_selectable, memID as integer, offset as Integer)
```

```
Integer nkStyleSetSelectableToMemblock(widget ref as nk_style_selectable, memID as integer, offset as Integer)
```

```
Integer nkStyleSetProgressFromMemblock(out ref as nk_style_progress, memID as integer, offset as Integer)
```

```
Integer nkStyleSetProgressToMemblock(widget ref as nk_style_progress, memID as integer, offset as Integer)
```

```
Integer nkStyleSetChartFromMemblock(out ref as nk_style_chart, memID as integer, offset as Integer)
```

```
Integer nkStyleSetChartToMemblock(widget ref as nk_style_chart, memID as integer, offset as Integer)
```

```
Integer nkStyleSetComboFromMemblock(out ref as nk_style_combo, memID as integer, offset as Integer)
```

```
Integer nkStyleSetComboToMemblock(widget ref as nk_style_combo, memID as integer, offset as Integer)
```

```
Integer nkStyleSetSliderFromMemblock(out ref as nk_style_slider, memID as integer, offset as Integer)
```

```
Integer nkStyleSetSliderToMemblock(widget ref as nk_style_slider, memID as integer, offset as Integer)
```

```
Integer nkStyleSetScrollbarFromMemblock(out ref as nk_style_scrollbar, memID as integer, offset as Integer)
```

```
Integer nkStyleSetScrollbarToMemblock(widget ref as nk_style_scrollbar, memID as integer, offset as Integer)
```

Integer)

```
Integer nkStyleSetEditFromMemblock(out ref as nk_style_edit, memID as integer, offset as Integer)
```

```
Integer nkStyleSetEditToMemblock(widget ref as nk_style_edit, memID as integer, offset as Integer)
```

```
Integer nkStyleSetPropertyFromMemblock(out ref as nk_style_property, memID as integer, offset as Integer)
```

```
Integer nkStyleSetPropertyToMemblock(widget ref as nk_style_property, memID as integer, offset as Integer)
```

```
Integer nkStyleSetTabFromMemblock(out ref as nk_style_tab, memID as integer, offset as Integer)
```

```
Integer nkStyleSetTabToMemblock(widget ref as nk_style_tab, memID as integer, offset as Integer)
```

```
Integer nkStyleSetWindowHeaderFromMemblock(out ref as nk_style_window_header, memID as integer, offset as Integer)
```

```
Integer nkStyleSetWindowHeaderToMemblock(widget ref as nk_style_window_header, memID as integer, offset as Integer)
```

```
Integer nkStyleSetWindowFromMemblock(out ref as nk_style_window, memID as integer, offset as Integer)
```

```
Integer nkStyleSetWindowToMemblock(widget ref as nk_style_window, memID as integer, offset as Integer)
```

Parameter (AGK-Basic)

- **result**
This is an output parameter. Depending on the function the requested property is stored in 'result'.
- **property**
A string to be converted into the corresponding property, an integer in the form of "12345", floats of "[3.1415]", nk_vec2 of "[10,10.5]" or "10, 10. 5", nk_color of "[255,255,255]", nk_image of "[img_id,w,h,region0,region1,region2,region3]" and nk_patch9 of "[img_id,w,h,region0,region1,region2,region3,border0,border1,border2,border3]".
- **widget_sz**
A widget name from the 'widget name' table.
- **mem_id**
A memory block for reading or writing.
- **out**
The widget that gets the data from a Memblock.
- **widget**
The widget which is written in a Memblock.

Style Examples

Listing 14

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
```

```

SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

// set default values.
default_style as nk_style_button
current_style as nk_style_button
default_style = nk_ctx.button
current_style = nk_ctx.button

do
    // pass agk inputs to nuklear
    nk.HandleInput()

    if nk.WindowBegin("Manipulate Context Style Part 1", 30, 50, 1220, 240, NK_WINDOW_BORDER||
NK_WINDOW_MOVABLE||NK_WINDOW_TITLE)
        change as integer = nk_false

        // layout for three buttons (reset, set color, set alignment
        nk.LayoutRowDynamic(30, 3)

        // the reset button
        if nk.ButtonLabel("Reset")
            // copy button default style to current button style
            current_style = default_style

            change = nk_true
        endif

        if nk.ButtonLabel("Set Color")
            // change back color of current style
            current_style.normal.color = nkRGB(32,32,64)
            current_style.hover.color = nkRGB(32,64,32)
            current_style.active.color = nkRGB(64,64,32)

            // change text color of current style
            current_style.text_normal = nkRGB(192,192,192)
            current_style.text_hover = nkRGB(255,255,255)
            current_style.text_active = nkRGB(255,255,255)

            change = nk_true
        endif

        if nk.ButtonLabel("Set Alignment")
            // set current style text alignment to left
            current_style.text_alignment = NK_TEXT_LEFT

            change = nk_true
        endif

        if change
            // write the values from UDT current_style to then context memblock
            nkStyleSetButtonToMemblock(current_style, nk_ctx.memblocks[NK_WIDGETTYPE_BUTTON], 0)

```

```

        // write the memblock to nuklear engines context.
        nk.StyleSetFromMemblock(NK_WIDGETTYPE_BUTTON, nk_ctx.memblocks[NK_WIDGETTYPE_BUTTON])
    endif
endif
nk.WindowEnd()

nkSync()
loop

```

Listing 15

```

#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

// first images for the styles are loaded and set.
nk.StyleMakeImage("button_normal", LoadImage("button_n.png"))
nk.StyleMakeImage("button_disabled", LoadImage("button_d.png"))

// now each of these style images is assigned to an item. As the entire
// image is assigned to the item, the position of the section is set to
// zero (0) and the width and height to minus one (-1) to indicate the
// full width and height of the image.
nk.StyleMakeItem("item_btn_n"      , "button_normal"      , 0,0,-1,-1)
nk.StyleMakeItem("item_btn_d"      , "button_disabled"   , 0,0,-1,-1)

// weitere farbitems werden erstellt.
nk.StyleMakeItem("item_black"      , MakeColor(0,0,0))
nk.StyleMakeItem("item_white"      , MakeColor(255,255,255))
nk.StyleMakeItem("item_cyan"       , MakeColor(0,255,255))
nk.StyleMakeItem("item_red"        , MakeColor(255,0,0))

// here the styles are created. in the xml tags the properties of the
// widgets are assigned to the newly created items.
// another way to set a style permanently is to change the widget properties
// in context directly. This is done with the command StyleSetContextProperty.
// This would look like this -
// nk.StyleSetContextProperty("button", "normal", "item_btn_n")
nk.StyleMake("default_button", "<button normal='item_btn_n' hover='item_btn_n' active='item_btn_n'
border='3' text_normal='item_black' text_hover='item_white' text_active='item_cyan'/>")
nk.StyleMake("disabled_button", "<button normal='item_btn_d' hover='item_btn_d' active='item_btn_d'
text_normal='item_red' text_hover='item_red' text_active='item_red' />")

```



```

mode as Integer = 1

// for a temporary setting of the style i use here StylePush which stays
// until StylePop is called. If several styles are pushed onto the stack,
// the same amount of StylePop commands must be called.
// if you want a permanent change of the style, it is useful to use
// StyleSetContext.
nk.StylePush("default_button")

do
    // pass agk inputs to nuklear
    nk.HandleInput()

    if nk.WindowBegin("Manipulate Context Style Part 2", 30, 50, 1220, 240, NK_WINDOW_BORDER||
NK_WINDOW_MOVABLE||NK_WINDOW_TITLE)
        // layout for three buttons.
        nk.LayoutRowStatic(40, 260, 3)

        // set buttons to default style.
        if nk.ButtonLabel("Button Default") And mode = 1
            nk.StylePop()
            mode = 0
        endif

        // no matter which style is currently active this button uses
        // its own style.
        nk.ButtonLabelStyled("disabled_button", "Button Inaktive")

        // set buttons to user style
        if nk.ButtonSymbolLabel(NK_SYMBOL_TRIANGLE_UP, "Button Style", NK_TEXT_CENTERED) And mode = 0
            nk.StylePush("default_button")
            mode = 1
        endif

    endif
    nk.WindowEnd()

    nkSync()
loop

// for clean end we must pop our style if needed.
if mode = 1 then nk.StylePop()

```

Listing 16

```

#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders

```

```

UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

new_style as String
other_style as String

// load two styles in separates strings
OpenToRead(1,"new_style.skin"): new_style = ReadString(1): CloseFile(1)
OpenToRead(1,"other_style.skin"): other_style = ReadString(1): CloseFile(1)

// set the style to new_style at the beginning
nk.StyleLoadFromString(new_style)

mode as Integer = 1
do
    // pass agk inputs to nuklear
    nk.HandleInput()

    if nk.WindowBegin("Manipulate Context Style Part 2", 30, 50, 1220, 240, NK_WINDOW_BORDER||
NK_WINDOW_MOVABLE||NK_WINDOW_TITLE)
        // layout for three buttons.
        nk.LayoutRowStatic(40, 260, 3)

        // set buttons to new_style.
        if nk.ButtonLabel("New Style") And mode = 1
            nk.StyleLoadFromString(new_style)
            mode = 0
        endif

        // no matter which style is currently active this button uses
        // its own style.
        nk.ButtonLabelStyled("disabled_button", "Button Inaktive")

        // set buttons to other_style
        if nk.ButtonLabel("Other Style") And mode = 0
            nk.StyleLoadFromString(other_style)
            mode = 1
        endif

    endif
    nk.WindowEnd()

    nkSync()
loop

```

XML file "new_style"

```

<skin>
    <image name="button_normal"    file="button_n.png" filter_min="nearest" filter_mag="nearest"/>
    <image name="button_disabled"  file="button_d.png" filter_min="nearest" filter_mag="nearest"/>

    <styleitem name="item_black"    color="0,0,0,255"/>
    <styleitem name="item_white"    color="0xffffffff"/>
    <styleitem name="item_cyan"     color="0,255,255,255"/>
    <styleitem name="item_red"      color="255,0,0,255"/>

    <styleitem name="item_btn_n"    image="button_normal"/>

```

```

<styleitem name="item_btn_d" image="button_disabled"/>

<stylecontrol name="disabled_button">
  <button normal='item_btn_d' hover='item_btn_d' active='item_btn_d' text_normal='item_red'
text_hover='item_red' text_active='item_red' />
</stylecontrol>

<style>
  <button normal='item_btn_n' hover='item_btn_n' active='item_btn_n' border='3'
text_normal='item_black' text_hover='item_white' text_active='item_cyan' />
</style>
</skin>

```

XML file “other_style”

```

<skin>
  <image name="button_normal" file="button_n.png" filter_min="nearest" filter_mag="nearest"/>
  <image name="button_disabled" file="button_d.png" filter_min="nearest" filter_mag="nearest"/>

  <styleitem name="item_black" color="0,0,0,255"/>
  <styleitem name="item_white" color="0xffffffff"/>
  <styleitem name="item_cyan" color="0,255,255,255"/>
  <styleitem name="item_red" color="255,0,0,255"/>
  <styleitem name="item_green_n" color="0,128,0,255"/>
  <styleitem name="item_green_h" color="0,192,0,255"/>
  <styleitem name="item_green_a" color="0,225,0,255"/>
  <styleitem name="item_yellow" color="225,225,0,255"/>

  <styleitem name="item_btn_d" image="button_disabled"/>

  <stylecontrol name="disabled_button">
    <button normal='item_btn_d' hover='item_btn_d' active='item_btn_d' text_normal='item_red'
text_hover='item_red' text_active='item_red' />
  </stylecontrol>

  <style>
    <button normal='item_green_n' hover='item_green_h' active='item_green_a' border='3'
text_normal='item_black' text_hover='item_white' text_active='item_cyan' border_color="item_yellow"
rounding='10' />
  </style>
</skin>

```

About Fonts and Images

Sometimes the nuclear standard font is no longer sufficient. Then another font must be loaded. Here I present the functions for this.

To prepare additional fonts for use, the `FontAtlasAdd...` functions are used. These must be placed between `FontStashBegin()` and `FontStashEnd()`. After `StyleSetFont` all widget calls will be drawn with this font.

```
nk.FontStashBegin()
  nk.FontAtlasAddFromFile("roboto22", "media/ttf/Roboto-Regular.ttf", 22)
  nk.FontAtlasAddFromSystem("segoe32", "segoepr.ttf", 32)
  nk.FontAtlasAddAgkFont("imact_y42", "imact_yellow_42.png")
nk.FontStashEnd()
nk.StyleSetFont("segoe32")
```

For the new UTF-8 support, the corresponding code ranges must be added. Since the Engine renders every printable character in a font atlas, it would be a bad idea to provide the code ranges for all languages. This would be a very large megatexture.

When using UTF-8 characters it is important to make sure that the selected font supports the desired characters.

The above example with different language support looks like this:

```
nk.FontStashBegin()
  // cyrillic supports
  nk.FontSetGlyphRange(NK_GLYPHRANGE_CYRILLIC)
  nk.FontAtlasAddFromSystem("cyrillic", "segoeui.ttf", 24)

  // japanese supports
  nk.FontSetGlyphRange(NK_GLYPHRANGE_JAPANESE)
  nk.FontAtlasAddFromFile("japanese", "media/ume-tmo3.ttf", 28)

  nk.FontSetGlyphRange(NK_GLYPHRANGE_DEFAULT)
  nk.FontAddGlyphRange(0x0102, 0x0103)
  nk.FontAddGlyphRange(0x0110, 0x0111)
  nk.FontAddGlyphRange(0x0128, 0x0129)
  nk.FontAddGlyphRange(0x0168, 0x0169)
  nk.FontAddGlyphRange(0x01a0, 0x01b0)
  nk.FontAddGlyphRange(0x1ea0, 0x1ef9)
  nk.FontAtlasAddFromSystem("vietnamese", "segoeui.ttf", 24)
nk.FontStashEnd()

// use cyrillic font
nk.StyleSetFont("cyrillic")
```

In the last chapters functions were introduced that required an image slot or an image name. Now the functions to create these images are shown here.

Font - Function Overview (Plugin - .DLL/.SO)

```
FontStashBegin()
```

```
FontStashEnd()
```

```
FontAddFromFile(name as String, file as String, height as Float)
```

```
StyleSetFont(name as String)
```

```
Float FontGetTextWidth(text as String, len as Integer)
```

```

Float FontGetLabelWidth(label as String)
Float FontGetHeight()
Integer FontAtlasAddFromFile(name as String, file as String, height as Float)
Integer FontAtlasAddAgkFont(agk_id as Integer, name as String, file as String)
Integer FontAtlasAddAgkFont(name as String, file as String)
Integer FontAtlasAddFromSystem(name as String, file as String, height as Float)
FontSetGlyphRange(preset as Integer)
FontAddGlyphRange(from as Integer, to as Integer)

```

Parameter (Plugin)

- **name**
A name that is used as an identifier.
- **file**
File name of the font or image.
- **height**
Size of the font.
- **text, label**
The text where the output width is measured.
- **len**
The length of the text.
- **agk_id**
The AGK-ID in which the loaded image is to be saved.
- **preset**
One of six coderanges. See glyphrange table.
- **from, to**
Indicate the code range (from / to)

Glyphrange Table

- | | |
|--------------------------|-----|
| • NK_GLYPHRANGE_DEFAULT | = 0 |
| • NK_GLYPHRANGE_CHINESE | = 1 |
| • NK_GLYPHRANGE_CYRILLIC | = 2 |
| • NK_GLYPHRANGE_KOREAN | = 3 |
| • NK_GLYPHRANGE_JAPANESE | = 4 |
| • NK_GLYPHRANGE_GREEK | = 5 |

Image - Function Overview (Plugin - .DLL/.SO)

```
Integer ImageToSlot(slot as Integer, agk_img_id as Integer, x as Float, y as Float, w as Float, h as Float)
```

```
Integer ImageToSlot(slot as Integer, agk_img_id as Integer)
```

```
Integer ImageCreate(name as String, agk_img_id as Integer, x as Float, y as Float, w as Float, h as Float)
```

```
Integer ImageCreate(name as String, agk_img_id as Integer)
```

```
Integer ImageCreate(agk_img_id as Integer)
```

```
Integer ImageExists(slot as Integer)
```

```
Integer ImageExists(name as String)
```

Parameter (Plugin)

- **slot**
The slot where the Agk image is to be saved.
- **agk_img_id**
A valid AGK image ID.
- **x, y, w, h**
Bildausschnitt aus dem AGK-Image welches als Nuklear Image gespeichert werden soll.
- **name**
A name that is used to identify the image.

Font Example

Listing 17

```
#import_plugin Nuklear as nk

// show all errors
SetErrorMode(2)

// set window properties
SetWindowTitle( "First Nuklear Window" )
SetWindowSize( 1280, 720, 0 )
SetWindowAllowResize( 1 ) // allow the user to resize the window

// set display properties
SetVirtualResolution( 1280, 720 ) // doesn't have to match the window
SetOrientationAllowed( 1, 1, 1, 1 ) // allow both portrait and landscape on mobile devices
SetSyncRate( 0, 0 ) // 30fps instead of 60 to save battery
SetScissor( 0,0,0,0 ) // use the maximum available screen space, no black borders
UseNewDefaultFonts( 1 ) // since version 2.0.22 we can use nicer default fonts

#insert "../Nuklear.agc"

// basic nuklear initialization
nkInit()

nk.FontStashBegin()
  nk.FontSetGlyphRange(NK_GLYPHRANGE_CYRILLIC)
  nk.FontAtlasAddFromSystem("cyrillic14", "segoeui.ttf", 24)

  nk.FontSetGlyphRange(NK_GLYPHRANGE_CHINESE)
  if nk.FontAtlasAddFromFile("chinese14", "media/NotoSansSC-Regular.otf", 24) <= 0 then
message("error")

  nk.FontSetGlyphRange(NK_GLYPHRANGE_KOREAN)
  nk.FontAtlasAddFromFile("korean14", "media/BMYEONSUNG_ttf.ttf", 24)

  nk.FontSetGlyphRange(NK_GLYPHRANGE_JAPANESE)
  nk.FontAtlasAddFromFile("japanese14", "media/ume-hgo4.ttf", 24)

  nk.FontSetGlyphRange(NK_GLYPHRANGE_GREEK)
  nk.FontAtlasAddFromSystem("greek14", "segoeui.ttf", 24)

  // vietnamese unicode range
  nk.FontSetGlyphRange(NK_GLYPHRANGE_DEFAULT)
  nk.FontAddGlyphRange(0x0102, 0x0103)
  nk.FontAddGlyphRange(0x0110, 0x0111)
  nk.FontAddGlyphRange(0x0128, 0x0129)
  nk.FontAddGlyphRange(0x0168, 0x0169)
  nk.FontAddGlyphRange(0x01a0, 0x01b0)
  nk.FontAddGlyphRange(0x1ea0, 0x1ef9)
  nk.FontAtlasAddFromSystem("vietnamese14", "segoeui.ttf", 24)
nk.FontStashEnd()

do
  // pass agk inputs to nuklear
  nk.HandleInput()

  nk.StyleSetFont("cyrillic14")
  if nk.WindowBegin("Мое первое окно (Cyrillic)", 40, 25, 400, 200, NK_WINDOW_BORDER||
```

```

NK_WINDOW_MOVABLE||NK_WINDOW_TITLE)
    endif
    nk.WindowEnd()

    nk.StyleSetFont("chinese14")
    if nk.WindowBegin("迈恩斯特斯·芬斯特 (Chinese)", 450, 25, 400, 200, NK_WINDOW_BORDER||
NK_WINDOW_MOVABLE||NK_WINDOW_TITLE)
    endif
    nk.WindowEnd()

    nk.StyleSetFont("korean14")
    if nk.WindowBegin("내 첫 창 (Korean)", 860, 25, 400, 200, NK_WINDOW_BORDER||NK_WINDOW_MOVABLE||
NK_WINDOW_TITLE)
    endif
    nk.WindowEnd()

    nk.StyleSetFont("japanese14")
    if nk.WindowBegin("私の最初のウィンドウ (Japanese)", 40, 250, 400, 200, NK_WINDOW_BORDER||
NK_WINDOW_MOVABLE||NK_WINDOW_TITLE)
    endif
    nk.WindowEnd()

    nk.StyleSetFont("greek14")
    if nk.WindowBegin("Το πρώτο μου παράθυρο (Greek)", 450, 250, 400, 200, NK_WINDOW_BORDER||
NK_WINDOW_MOVABLE||NK_WINDOW_TITLE)
    endif
    nk.WindowEnd()

    nk.StyleSetFont("vietnamese14")
    if nk.WindowBegin("Cửa sổ đầu tiên của tôi (Vietnamese)", 860, 250, 400, 200, NK_WINDOW_BORDER||
NK_WINDOW_MOVABLE||NK_WINDOW_TITLE)
    endif
    nk.WindowEnd()

    nkSync()
loop

```