ResolveDevDoc

Release 0.1

Benoit Breault

Apr 25, 2022

UI FUSION/DAVINCIRESOLVE

CHAPTER

ONE

SUMMARY

ResolveDevDoc is an unofficial documentation for scripting using Python 3.6 and workflow integration with Davinci Resolve 17

This documentation was reformatted for ReadTheDocs to display additionnal informations about elements and API for Davinci Resolve. Please refer to the original documentation if needed. A reformated version or the Readme.txt is also available below (work in progress):

Scripting README.txt document | *Workflow Integrations README.txt document*

Warning: **Use at your own risk** Some errors might be present. This documentation project is still in developement.

Some of those samples were created with the help of the SteakUnderWater WSL community

Note: ResolveDevDoc has its documentation hosted on Read the Docs.

CHAPTER

TWO

CONTENTS

New in version Updated: as of 25 August, 2020

2.1 UI Introduction

In this package, you will find a brief introduction to the Workflow Integration Plugins support for DaVinci Resolve Studio. Apart from this README.txt file, this package contains following folders: Examples: containing some representative sample plugin, and a sample script. Scripts: containing some sample workflow scripts to interact with Resolve.

Overview

DaVinci Resolve Studio now supports Workflow Integration Plugins to be loaded and communicate with Resolve. Resolve can run one or more Workflow Integration Plugins at the same time. Users can write their own Workflow Integration Plugin (an Electron app) which could be loaded into DaVinci Resolve Studio. To interact with Resolve, Resolve's JavaScript APIs can be used from the plugin.

Alternatively, a Python or Lua script can be invoked, with the option of a user interface built with Resolve's built-in Qt-based UIManager, or with an external GUI manager. See the "Sample Workflow Integration Script" section below for details.

Sample Workflow Integration Plugin

A sample Workflow Integration Plugin is available in the "Examples/SamplePlugin" directory. In order for Resolve to register this plugin, this directory needs to be copied to 'Workflow Integration Plugins' root directory (mentioned in below section). Once a plugin is registered, plugin can be loaded from UI sub-menu under 'Workspace->Workflow Integrations'. This will load the plugin and show the plugin HTML page in a separate window.

Sample plugin helps to understand how a plugin should be structured and how it works with Resolve. Please refer to the directory/file structure, manifest file info, plugin loading, JavaScript API usage examples, etc. This sample plugin and scripts demonstrates few basic scriptable JavaScript API usages to interact with Resolve.

Loading Workflow Integration Plugin

On startup, DaVinci Resolve Studio scans the Workflow Integration Plugins root directory and enumerates all plugin modules. For each valid plugin module, it creates a UI sub-menu entry under 'Workspace->Workflow Integrations' menu. DaVinci Resolve Studio reads the basic details of the plugin from its manifest.xml file during load time. Once plugin is loaded, user can click on the 'Workflow Integrations' sub-menu to load the corresponding plugin.

Workflow Integration Plugin directory structure

```
com.<company>.<plugin_name>/
        package.js
        main.js
```

(continues on next page)

(continued from previous page)

```
        index.html
        manifest.xml
        node_modules/
                <Node.js modules>
        js/
                <supporting js files>
        css/
                <css files containing styling info>
        img/
                <image files>
```

Workflow Integration Plugins root directory

User should place their Workflow Integration Plugin under the following directory:

Note: Mac OS X: "/Library/Application Support/Blackmagic Design/DaVinci Resolve/Workflow Integration Plugins/"

Windows: "%PROGRAMDATA%Blackmagic DesignDaVinci ResolveSupportWorkflow Integration Plugins"

Supported platforms

- Plugins: Windows, Mac OS X (not supported on Linux currently)
- Scripts: Windows, Mac OS X, Linux

Using scriptable JavaScript API

Scriptable JavaScript API execution happens under HTML environment like any typical website. Once HTML page is loaded it can execute scriptable JavaScript API as needed (like clicking on a button, etc.)

This example JavaScript snippet creates a simple project in DaVinci Resolve Studio:

```
const WorkflowIntegration = require('./WorkflowIntegration.node');
isInitialized = WorkflowIntegration.Initialize('com.blackmagicdesign.resolve.sampleplugin
↪');
if (isInitialized) {
    resolve = WorkflowIntegration.GetResolve();
    resolve.GetProjectManager().CreateProject("Hello World");
}
```

The resolve object is the fundamental starting point for scripting via Resolve. As a native object, it can be inspected for further scriptable properties and functions in JavaScript.

WorkflowIntegration module API

To interact with Resolve you need to use 'WorkflowIntegration.node' Node.js module file in your plugin app. Below are the WorkflowIntegration (module) JavaScript API functions to communicate with Resolve.

WorkflowIntegration

- Initialize(<pluginId>) –> Bool # Returns true if initialization is successful, false otherwise. <pluginId> is the unique plugin id string configured in the manifest.xml file.
- GetResolve() –> Resolve # Returns Resolve object.
- RegisterCallback(callbackName, callbackFunc) –> Bool
  - Returns true if input callback name/function is registered successfully, false otherwise.

  - 'callbackName' should be a valid supported callback string name (refer to the below section 'Supported callbacks').
  - 'callbackFunc' should be a valid JavaScript function without any arguments.

- DeregisterCallback(callbackName) –> Bool # Returns true if input callback name is deregistered successfully, false otherwise.
- CleanUp() –> Bool # Returns true if cleanup is successful, false otherwise. This should be called during plugin app quit.
- SetAPITimeout(valueInSecs) –> Bool
  - By default, apis dont timeout. In order to enable timeout, set a non-zero positive integer value in the arg 'valueInSecs'.
  - Setting it to 0 will disable timeout. This function will return true if the timeout is set/reset successfully.

Supported callbacks

- 'RenderStart'
- 'RenderStop'

Please note that there is no console based support for JavaScript API.

Sample Workflow Integration Script

A sample Workflow Integration Python script is also available in the "Examples" directory. In order for Resolve to register this script, it needs to be copied to the 'Workflow Integration Plugins' root directory (mentioned in the above section).

Once a script is registered, it can be also loaded from the 'Workspace' menu, under 'Workflow Integrations'. This will invoke the script and show the sample UIManager window.

Workflow Integration scripts work similarly to other scripts in Resolve, and use the same scripting API. This example script provides a basic introduction into creating a popup Workflow application using a UIManager window, with simple layout of text fields and buttons, and event handlers to dispatch functions for integration with the user's facility. Alternatively, third-party UI managers such PyQt may be used instead, or no GUI at all.

When launched by Resolve, plugin scripts are automatically provided with 'resolve' and 'project' variables for immediate and easy access to Resolve's scripting API. Additional third-party modules may be imported for access to asset-management systems as desired.

UIManager Introduction

There are two main objects needed to manage a window, the UIManager that handles layout, and the UIDispatcher that manages interaction events, accessed as follows:

```
ui = fusion.UIManager()
dispatcher = bmd.UIDispatcher(ui)
```

Windows are created with the the UIDispatcher, passing a dictionary of attributes like ID and Text, with GUI elements in nested layouts all created with the UIManager.

2.2 UIDispatcher

The UIDispatcher object has a few important functions to manage processing of events. The most important are:

2.2.1 Functions

AddWindow({properties}, [children])

Description

This function is creating a window element.

Type: function: Accepts a dictionary of properties and a list of children, returns a Window object

```
ui = fusion.UIManager
dispatcher = bmd.UIDispatcher(ui)

win = dispatcher.AddWindow({ 'ID': 'myWindow' }, [ ui.Label({ 'Text': 'Hello World!' })␣
↪])
```

AddDialog({properties}, [children])

Description

This function is

Type: function: Accepts a dictionary of properties and a list of children, returns a Dialog object

---

Note: not tested yet

ui = fusion.UIManager dispatcher = bmd.UIDispatcher(ui)

win = dispatcher.AddDialog({ 'ID': 'myDialog' }, [ui.Label({ 'Text': 'Hello World!' })])

RunLoop()

Description

This function runs a Loop listening to user events.

Type: function: Call when your window is ready to receive user clicks and other events return int

---

Note: not tested yet

ui = fusion.UIManager dispatcher = bmd.UIDispatcher(ui)

win = dispatcher.AddDialog({ ‘ID’: ‘myDialog’ }, [ui.Label({ ‘Text’: ‘Hello World!’ })])

dispatcher.RunLoop()

ExitLoop(int)

Description

This function exits the running loop.

Type: function: Terminates the event processing, and returns int = any supplied exit code from RunLoop()

Note: not tested yet

dispatcher.ExitLoop(int)

Common usage is to create your window and set up any event handlers, including a Close handler for the window that calls ExitLoop(), then Show() your window and call RunLoop() to wait for user interaction:

```
ui = fusion.UIManager
dispatcher = bmd.UIDispatcher(ui)

win = dispatcher.AddWindow({ 'ID': 'myWindow' }, [ ui.Label({ 'Text': 'Hello World!' })␣
↪])

def OnClose(ev):
        dispatcher.ExitLoop()

win.On.myWindow.Close = OnClose

win.Show()
dispatcher.RunLoop()
```

AddWindow() will also accept a single child without needing a list, or a single dictionary containing both properties and child elements, for ease of use.

As well as constructing new child elements and layouts, the UIManager also offers a few useful functions:

2.2.2 UIManager

FindWindow(ID)

Description

This function returns the windows matching with ID

Type: Returns an element with matching ID

```
ui = fusion.UIManager
window = ui.FindWindow('win_id')
```

FindWindows(ID)

Description

This function is

Type: Returns a list of all elements with matching ID

```
ui = fusion.UIManager
windows = ui.FindWindows('win_id')
```

QueueEvent(element, event, {info})

Description

This function is

Type: element= , event= , info= : Calls the element's event handler for 'event', passing it the dictionary 'info'

Note: not tested yet

ui = fusion.UIManager ui.QueueEvent(element, event, info)

2.3 Layout

Additionally, elements can be nested to define layout, using the HGroup and VGroup elements. As with Window and other elements, you can pass a single dictionary or list with all properties and children, or separate them into a dict of properties and list of children, for convenience:

```
winLayout = ui.VGroup([
        ui.Label({ 'Text': "A 2x2 grid of buttons", 'Weight': 1 }),

        ui.HGroup({ 'Weight': 5 }, [
                ui.Button({ 'ID': "myButton1",  'Text': "Go" }),
                ui.Button({ 'ID': "myButton2",  'Text': "Stop" }),
                ]),
        ui.VGap(2),
        ui.HGroup({ 'Weight': 5 }, [
                ui.Button({ 'ID': "myButtonA",  'Text': "Begin" }),
                ui.Button({ 'ID': "myButtonB",  'Text': "End" }),
                ]),
        ]),
```

(continues on next page)

(continued from previous page)

```
win = dispatcher.AddWindow({ 'ID': "myWindow" }, winLayout)
```

HGap and VGap elements can included for finer spacing control. Note also the Weight attribute, which can be applied to most elements to control how they adjust their relative sizes.

Note: A Weight of 0 will use the element's minimum size.

2.4 Elements

The element's ID is used to find, manage, and dispatch events for that element. GUI elements also support a set of common attributes including Enabled, Hidden, Visible, Font, WindowTitle, BackgroundColor, Geometry, ToolTip, StatusTip, StyleSheet, WindowOpacity, MinimumSize, MaximumSize, and FixedSize.

Note: For better management of elements, define an ID attribute. Not all example will contain an ID but keep that in mind.

You can then use the win.Find('ID') to find and update an element and update an attribute win.Find('myButton').Text = "Processing..."

Important: You need to create a Window to contain your element and attribute.

Check *UI Element Layout page* and *UI Dispatcher functions* for more details.

2.4.1 Window

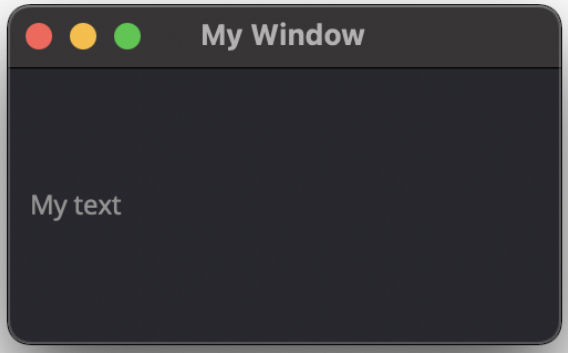
WindowTitle

Type: string

Description

This attribute is used to display a text on the window's title bar.

```
win = dispatcher.AddWindow({
    'ID': "my_window",
    'WindowTitle': 'My Window'
    },
    ui.VGroup([
            ui.Label({ 'ID': 'label_1', 'Text':'My text' })
    ])
)
```



WindowOpacity

Type: float

Description

This attribute is used to set the Windows's opacity, (default=1)

```
win = dispatcher.AddWindow({
    'ID': "my_window",
    'WindowOpacity': 0.5,  #50% opacity
    },
    ui.VGroup([
            ui.Label({ 'ID': 'label_1', 'Text':'My text' })
    ])
)
```

BackgroundColor

Type: dict RGBA

Description

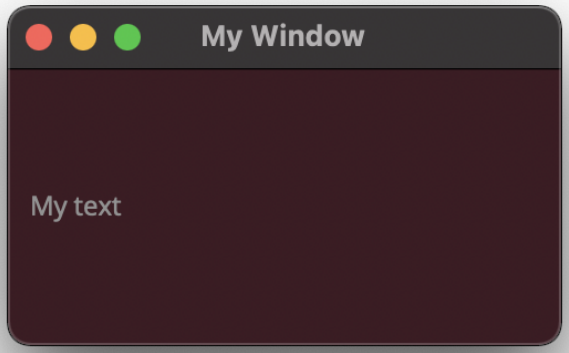
This attribute is used change the Window's background color.

```
win = dispatcher.AddWindow({
    'ID': "my_window",
    'BackgroundColor': {'R':0.6, 'G':0.1, 'B':0.2, 'A':0.2},
    },
```

(continues on next page)

(continued from previous page)

```
    ui.VGroup([
            ui.Label({ 'ID': 'label_1', 'Text':'My text' })
    ])
)
```



Geometry

Type: [posX, posY, width, height]

Description

This attribute is used to change the Window's position and size.

```
win = dispatcher.AddWindow({
    'ID': "my_window",
    'Geometry': [ 400,200,250,125 ],
    },
    ui.VGroup([
            ui.Label({ 'ID': 'label_1', 'Text':'My text' })
    ])
)
```

2.4.2 Label

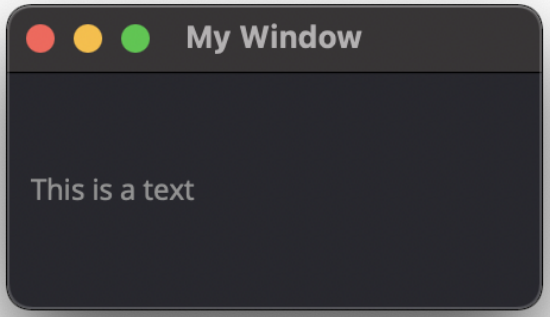
Text

Description

This label attribute is used to display Text on the element.

Type: string

```
ui.Label({ 'ID':'label_1', 'Text': "This is a text" })
```



Alignment

Type: ({'Parameter': bool})

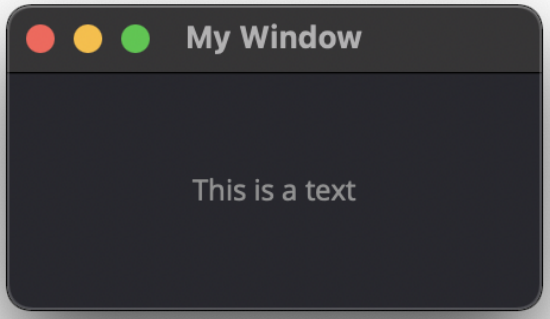
Description

This label attribute is used to align Text inside the Label element. Check out the qt5 documentation for more details

- AlignCenter
- AlignLeft
- AlignRight
- AlignHCenter
- AlignVCenter
- AlignTop
- AlignBottom
- AlignJustify

- AlignBaseline

```
ui.Label({ 'ID':'label_1', 'Text': "This is a text", 'Alignment': { 'AlignCenter' : True
↪} })
```



FrameStyle

Type: int

Description

This label attribute is used to Style the frame of the Label Element.

Check out the qt5 documentation for more details

- 0: NoFrame
- 1: Box
- 2: Panel
- 3: WinPanel
- 4: HLine
- 5: VLine
- 6: StyledPanel
- other to try

```
ui.Label({ 'ID': 'label_1', 'Text':'My text', 'FrameStyle': 1 })
```

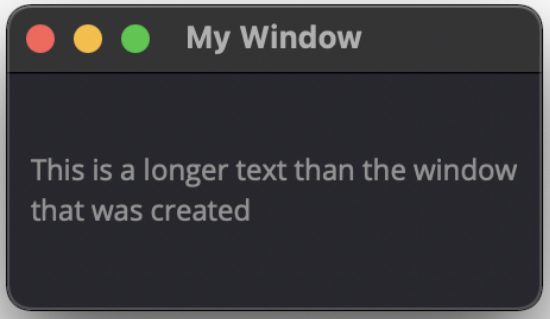
WordWrap

Type: bool

Description

This label attribute enable Wordwrap when the Text attribute is longer than the window's width

```
ui.Label({ 'ID':'label_1', 'Text': "This is a longer text than the window that was␣
↪created" , 'WordWrap': True })
```



Indent

Type: bool

Description

This label attribute

Note: Not yet tested

ui.Label({ 'ID':'label_1', 'Indent': "" })

Margin

Type:

Description
This label attribute

Note: Not yet tested

ui.Label({ 'ID':'label_1', 'Margin': "" })

StyleSheet

Type: string

Description
This attribute is set to apply a StyleSheet to the Element (similar to CSS)

```
css_style = f"""
color: rgb(205, 205, 245);
font-family: Garamond;
font-weight: bold;
font-size: 16px;
"""

ui.Label({ 'ID':'label_1', 'StyleSheet': css_style })
```

MinimumSize

Type: [width, height]

Description
This attribute is used to set a minimum width and height for the element if user resize the window.

```
ui.Label({ 'ID': 'label_1', 'Text':'My text','MinimumSize': [200, 200] })
```

MaximumSize

Type: [width, height]

Description

This attribute is used to set a maximum width and height for the element if user resize the window.

```
ui.Label({ 'ID': 'label_1', 'Text':'My text','MaximumSize': [400, 400] })
```

FixedSize

Type: [width, height]

Description

This attribute is used to set prevent users to resize the window.

Note: Not yet tested

ui.Label({ 'ID': 'label_1', 'Text':'My text','FixedSize': [250, 125] })

2.4.3 Button

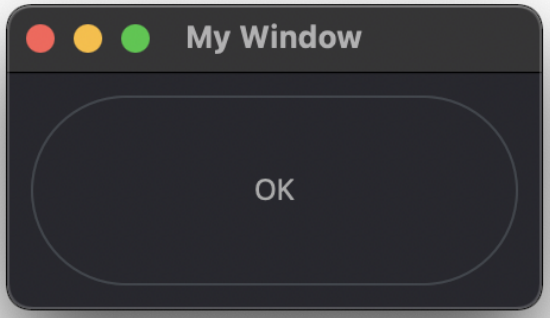
Text

Type: string

Description

This attribute is used to display Text on the element.

```
ui.Button({ 'ID': 'ok_btn',  'Text': "OK" })
```



Down

Type: bool

Description

This label attribute is used to

Note: Not yet tested

ui.Button({ 'ID': 'ok_btn', 'Down': "" })

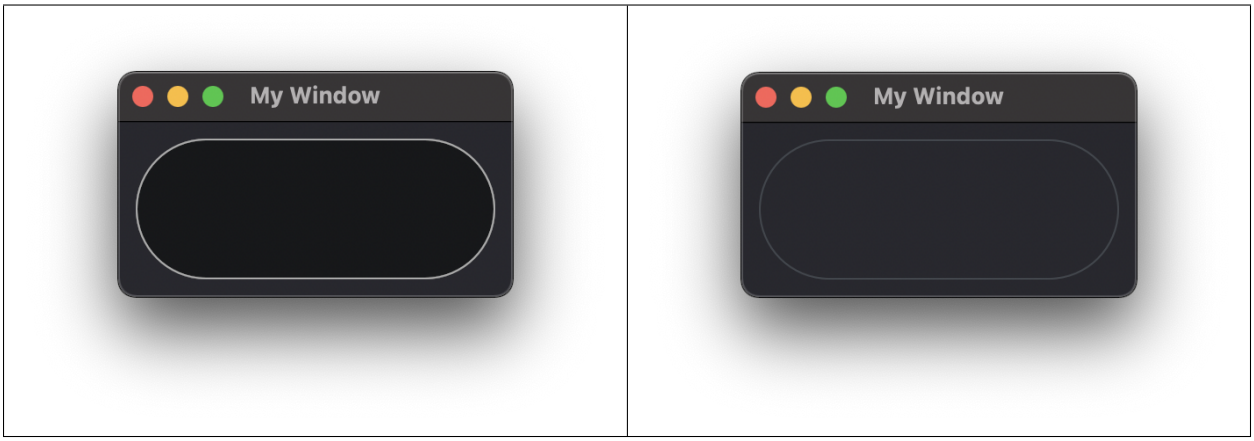
Checkable

Type: bool

Description

This label attribute is used to create a 2 states button.

```
ui.Button({ 'ID': 'ok_btn',  'Checkable': True })
```



Checked

Type: bool

Description

This attribute is used to set the Checked status to a Checkable button

```
ui.Button({ 'ID': 'ok_btn', 'Checkable': True, 'Checked': True })
```

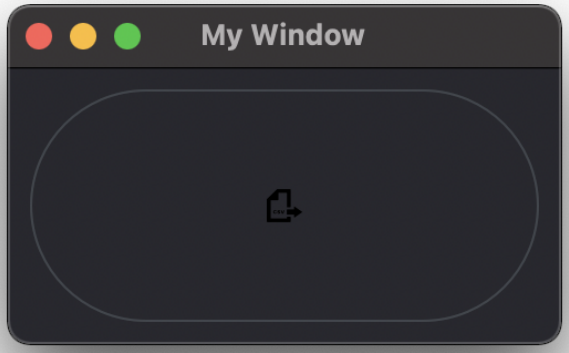
Icon

Type:

Description

This attribute is used to add ui.Icon object to the button.

```
ui.Button({ 'ID': 'ok_btn',  'Icon': ui.Icon({'File': r"UserData:/Scripts/images/csv.png
↪"}) })
```



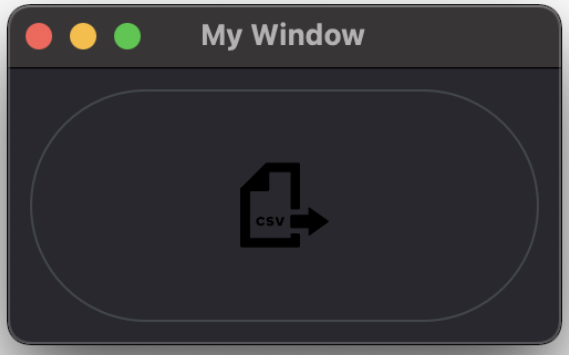
IconSize

Type: [int, int]

Description

This attribute is used to resize the Icon with lenght and height values.

```
ui.Button({'ID': 'ok_btn',  'Icon': ui.Icon({'File': r"UserData:/Scripts/images/csv.png"}
↪), 'IconSize': [40, 40]})
```



Flat

Type: bool

Description

This label attribute is used to

Note: Not yet tested

ui.Button({ 'ID': 'ok_btn', 'Flat': "" })

2.4.4 CheckBox

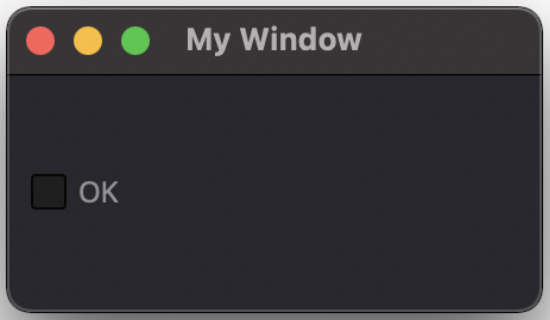
Text

Type: string

Description

This label attribute is used to display Text on the element.

```
ui.CheckBox({ 'ID': 'checkbox_1',  'Text': "OK" })
```



Down

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.CheckBox({ 'ID': 'checkbox_1', 'Down': "" })

Checkable

Type: bool

Description

This label attribute is used to disable the option to check. (default=True)

```
ui.CheckBox({ 'ID': 'checkbox_1',  'Checkable': False })
```

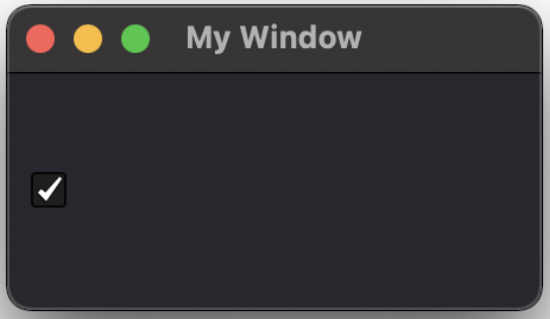
Checked

Type: bool

Description

This label attribute is used to change the checked status of the CheckBox.

```
ui.CheckBox({ 'ID': 'checkbox_1',  'Checked': True })
```

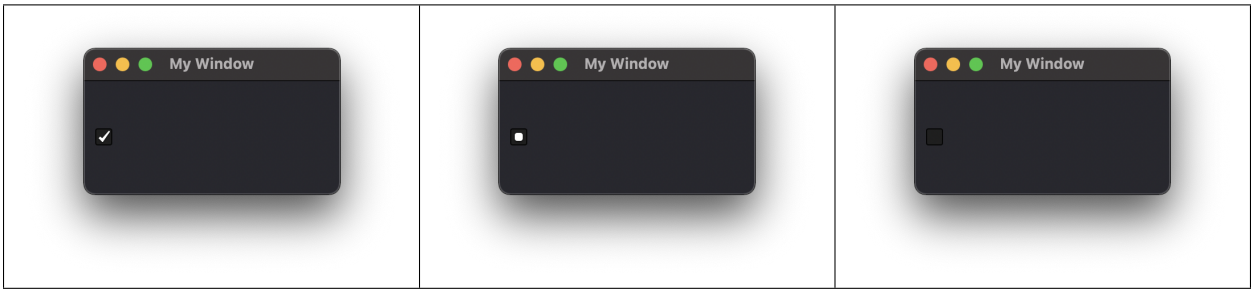


Tristate

Type:

Description
This label attribute is used to activate a 3 state checkbox

```
ui.CheckBox({ 'ID': 'checkbox_1',  'Tristate': True })
```



CheckState

Type:

Description
This label attribute is used to

Note: Not yet tested

ui.CheckBox({ ‘ID’: ‘checkbox_1’, ‘CheckState’: “” })

2.4.5 ComboBox

Refer to the UI Element Function page to AddItems to the ComboBox list

ItemText

Type:

Description

This label attribute is used to

Note: Not yet tested

ui.ComboBox({ ‘ID’: ‘combo_1’, ‘ItemText’: ‘test’ }) win.Find(“combo_1”).AddItems([“Blue”,”Cyan”,”Green”,”Yellow”])

Editable

Type: bool

Description

This attribute is used to allow users to add items to the ComboBox

Note that those items are not added permanently to the ComboBox list.

```
ui.ComboBox({ 'ID': 'combo_1',  'Editable': True })
```

CurrentIndex

Type:

Description

This attribute is used to get or change the selected item from the ComboBox

```
ui.ComboBox({ 'ID': 'combo_1' })
    win.Find("combo_1").AddItems(["Blue","Cyan","Green","Yellow","Red","Pink","Purple",
↪"Fuchsia","Rose","Lavender","Sky","Mint","Lemon","Sand","Cocoa","Cream"])

    print(win.Find("combo_1").CurrentIndex) #0 will be printed for the first item␣
↪(default)
```

(continues on next page)

(continued from previous page)

```
    win.Find("combo_1").CurrentIndex =  3 #"Yellow" will be selected
```

CurrentText

Type: string

Description

This attribute is used to get the Text from the selected Item

```
ui.ComboBox({ 'ID': 'combo_1' })

    win.Find("combo_1").AddItems(["Blue","Cyan","Green","Yellow","Red"])
print(win.Find("combo_1").CurrentText)  # print the first item by default "Blue"
```

Count

Type: int

Description

This label attribute is used to

Note: Not yet tested

ui.ComboBox({ 'ID': 'combo_1', 'Count': 3 })

2.4.6 SpinBox

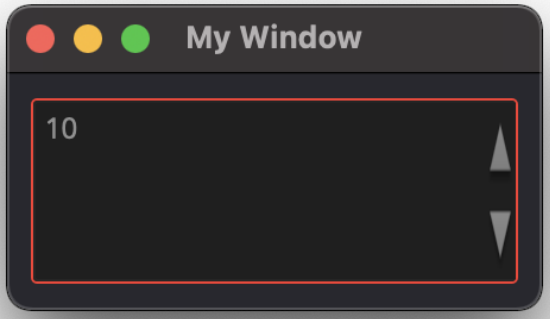
Value

Type: int

Description

This spinbox attribute is used to set the current SpinBox value (default max=99)

```
ui.SpinBox({ 'ID': 'spin_1',  'Value': 10 })
```



Minimum

Type: int

Description

This spinbox attribute is used to set a Minimum value to the SpinBox

```
ui.SpinBox({ 'ID': 'spin_1',  'Minimum': 5 })
```

Maximum

Type: int

Description

This spinbox attribute is used to set a Maximum value to the SpinBox

```
ui.SpinBox({ 'ID': 'spin_1',  'Maximum': 8 })
```

SingleStep

Type: int

Description

This spinbox attribute is used to set the step value of the SpinBox

```
ui.SpinBox({ 'ID': 'spin_1',  'SingleStep': 2 })
```

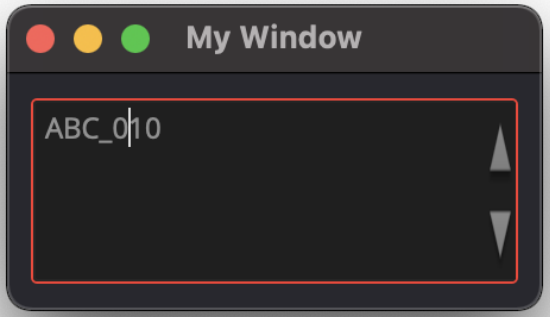
Prefix

Type: string

Description

This spinbox attribute is used add a text prefix to the spinbox value

```
ui.SpinBox({ 'ID': 'spin_1',  'Prefix': "ABC_0" })
```



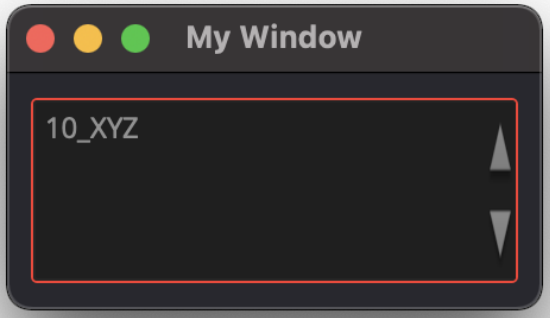
Suffix

Type: string

Description

This spinbox attribute is used add a text suffix to the spinbox value

```
ui.SpinBox({ 'ID': 'spin_1',  'Suffix': '_XYZ' })
```



Alignment

Type:

Description

This label attribute is used to

Note: Not yet tested

ui.SpinBox({ 'ID': 'spin_1', 'Alignment': "" })

ReadOnly

Type: bool

Description

This spinbox attribute is used limit the spinbox usage to the side arrows. Keyboard entry disabled

```
ui.SpinBox({ 'ID': 'spin_1',  'ReadOnly': True })
```

Wrapping

Type: bool

Description

This spinbox attribute is used to allow the value to return to the Minimum value when passed Maximum and vice-versa

```
ui.SpinBox({ 'ID': 'spin_1',  'Wrapping': True })
```

2.4.7 Slider

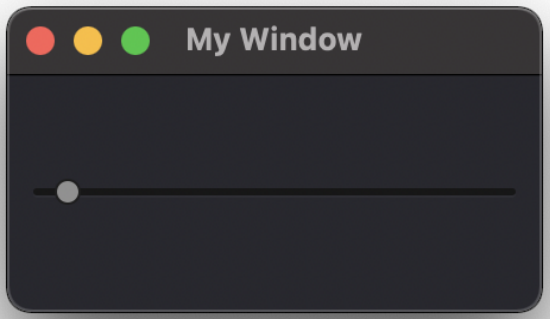
Value

Type: int

Description

This slider attribute is used to set the slider value

```
ui.Slider({ 'ID': 'slider_1',  'Value': 5 })
```



Minimum

Type: int

Description

This slider attribute is used to set a Minimum value to the Slider

```
ui.Slider({ 'ID': 'slider_1',  'Minimum': 2 })
```

Maximum

Type: int

Description

This slider attribute is used to set a Maximum value to the Slider

```
ui.Slider({ 'ID': 'slider_1',  'Maximum': 8 })
```

SingleStep

Type: int

Description

This slider attribute is used to set the step value of the slider

```
ui.Slider({ 'ID': 'slider_1',  'SingleStep': 2 })
```

PageStep

Type:

Description

This label attribute is used to

Note: Not yet tested

ui.Slider({ 'ID': 'slider_1', 'PageStep': "" })

Orientation

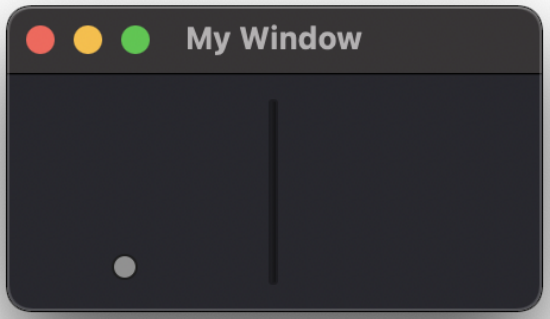
Type: string

Description

This slider attribute is used to set the orientation of the slider

- Vertical
- Horizontal

```
ui.Slider({ 'ID': 'slider_1',  'Orientation': 'Vertical' })
```



Tracking

Type: bool

Description

This label attribute is used to… (default=False)

Note: Not yet tested

ui.Slider({ 'ID': 'slider_1', 'Tracking': "" })

SliderPosition

Type:

Description

This label attribute returns the current Slider value.

```
print(win.Find('slider_1').SliderPosition)  #default=0
```

2.4.8 LineEdit

Text

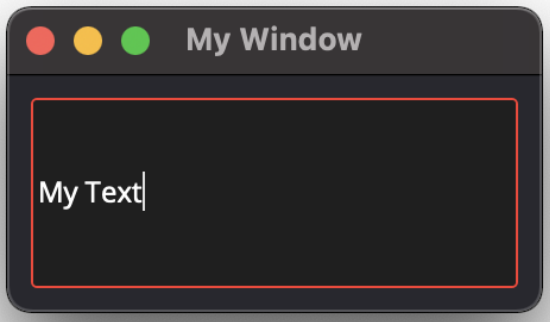
Type: string

Description

This attribute is used to set and display the Text in the LineEdit box. For Multi-Line text, use the *TextEdit* element.

Note: Not yet tested

ui.LineEdit({ 'ID': 'le_1', 'Text': "My Text" })

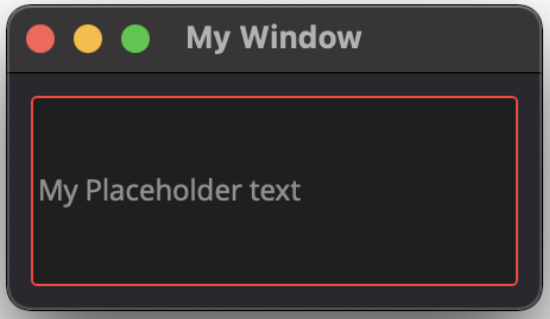


PlaceholderText

Type: string

Description
This attribute is used to display a text in the lineEdit box. The PlaceholderText will be replaced by user input.

```
ui.LineEdit({ 'ID': 'le_1',  'PlaceholderText': "My Placeholder text" })
```



Font

Type:

Description
This attribute is used to

Note: Not yet tested

ui.LineEdit({ 'ID': 'le_1', 'Font': "" })

MaxLength

Type: int

Description

This attribute is used to limit the user input to x(int) character

```
ui.LineEdit({ 'ID': 'le_1',  'MaxLength': 10 })
```

ReadOnly

Type: bool

Description

This attribute is used to set the LineEdit to be Read-Only.

```
ui.LineEdit({ 'ID': 'le_1',  'ReadOnly': True })
```

Modified

Type:

Description

This label attribute is used to

Note: Not yet tested

ui.LineEdit({ 'ID': 'le_1', 'Modified': "" })

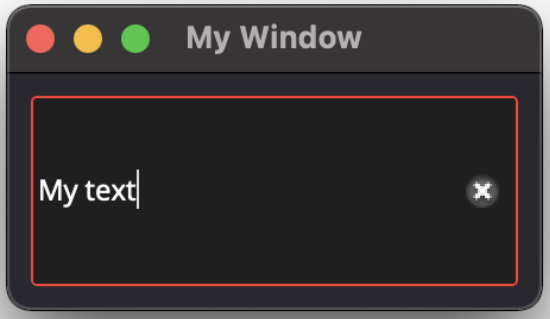
ClearButtonEnabled

Type: bool

Description

This attribute is used to add a button to clear the text field

```
ui.LineEdit({ 'ID': 'le_1', 'ClearButtonEnabled': True })
```



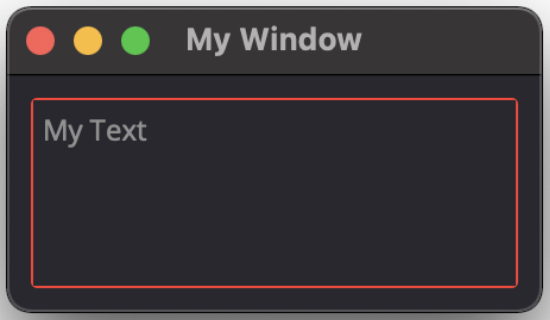
2.4.9 TextEdit

Text

Type: string

Description
This attribute is used to set and display the Text in the TextEdit box.

```
ui.TextEdit({ 'ID': 'te_1',  'Text': "My Text" })
```



PlaceholderText

Type: string

Description

This attribute is used to display a text in the lineEdit box.

The PlaceholderText will be replaced by user input.

```
ui.TextEdit({ 'ID': 'te_1',  'PlaceholderText': "My Placeholder Text" })
```

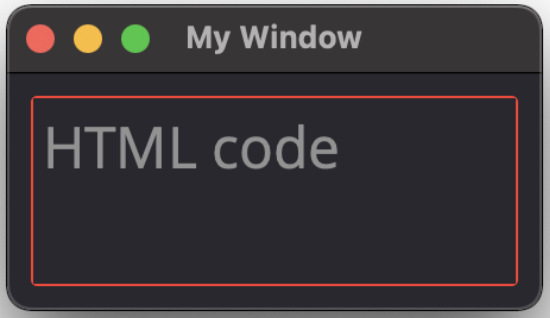
HTML

Type: string

Description

This attribute is used render HTML code inside the TextEdit box

```
ui.TextEdit({ 'ID': 'te_1',  'HTML': "<h1>HTML code</h1>" })
```



Font

Type: ui.Font

Description

This attribute is used to specify a Font element with parameters

```
ui.TextEdit({ 'ID': 'te_1',  'Font': ui.Font({ 'Family': "Times New Roman" }) })
```

Alignment

Type: dict

Description

This label attribute is used to

Note: Not yet tested

ui.TextEdit({ 'ID': 'te_1', 'Alignment': "" })

ReadOnly

Type: bool

Description

This label attribute is used to set the TextEdit to ReadOnly. User cannot add or remove text.

```
ui.TextEdit({ 'ID': 'te_1',  'ReadOnly': True })
```

TextColor

Type: dict(r,g,b, a) ?

Description

This label attribute is used to

Note: Not yet tested

ui.TextEdit({ 'ID': 'te_1', 'TextColor': { 'R':1, 'G': 0, 'B':0, 'A':1 })

TextBackgroundColor

Type: string

Description

This label attribute is used to

Note: Not yet tested

ui.TextEdit({ 'ID': 'te_1', 'TextBackgroundColor': "blue" })

TabStopWidth

Type: int

Description

This attribute is used to set the width of the Tab when inserted.

```
ui.TextEdit({ 'ID': 'te_1',  'TabStopWidth': 50 })
```

Lexer

Type:

Description

This attribute is used to

Note: Not yet tested

ui.TextEdit({ 'ID': 'te_1', 'Lexer': })

LexerColors

Type:

Description

This attribute is used to

Note: Not yet tested

ui.TextEdit({ 'ID': 'te_1', 'LexerColors': })

2.4.10 ColorPicker

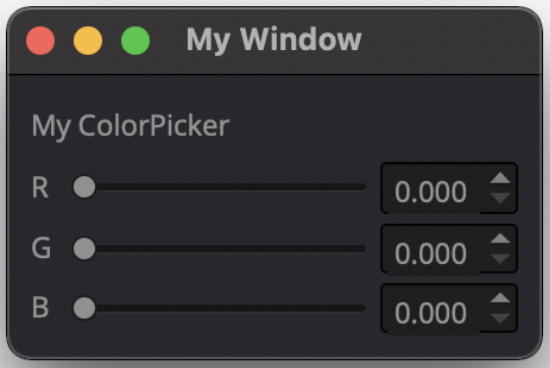
Text

Type: string

Description

This attribute is used to display a Text with the ColorPicker

```
ui.ColorPicker({ 'ID': 'colorpicker_1',  'Text': "My ColorPicker" })
```



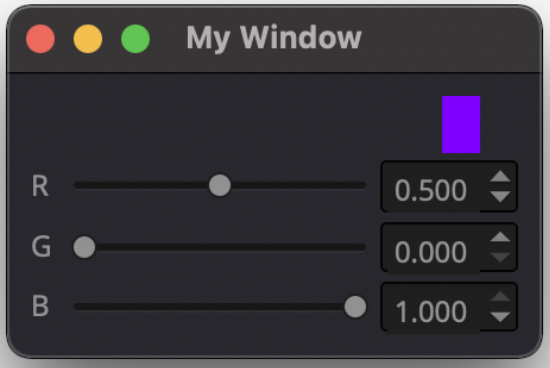
Color

Type: dict

Description

This attribute is used to set a default color to the ColorPicker. Each RGB color using a float value betwee 0 and 1.

```
ui.ColorPicker({ 'ID': 'colorpicker_1', 'Color': {'R':0.5, 'G':0, 'B':1.0} })
```



Tracking

Type: bool

Description
This label attribute is used to

Note: Not yet tested

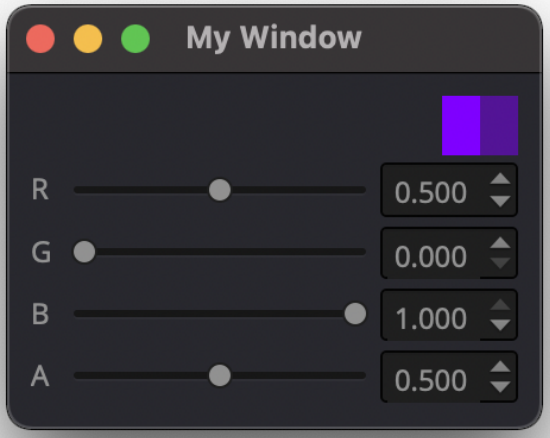
ui.ColorPicker({ 'ID': 'colorpicker_1', 'Tracking': True })

DoAlpha

Type: bool

Description
This attribute is used to include Alpha value in the RGB ColorPicker

```
ui.ColorPicker({ 'ID': 'colorpicker_1',  'DoAlpha': True })
```



2.4.11 Font

Family

Type: string

Description

This attribute is used to set the font family. Combine with an element using text.

- Times New Roman
- Arial
- list available font...

```
ui.Label({'Text': "My Label", "Font": ui.Font({ 'Family': "Times New Roman" }),
```

StyleName

Type: string

Description

This label attribute is used to

Note: Not yet tested

ui.Font({ 'StyleName': "" })

PointSize

Type: int

Description

This attribute is used to set a size to the Font (pt).

```
ui.Label({'Text': "My Label", "Font": ui.Font({ 'PointSize': 36 }),
```

PixelSize

Type: int

Description

This attribute is used to set a size to the Font (px).

```
ui.Label({'Text': "My Label", "Font": ui.Font({ 'PixelSize': 36 }),
```

Bold

Type: bool

Description

This attribute is used to apply **bold** to the text

Note: Do not seems to apply on all fonts

ui.Label({‘Text’: “My Label”, “Font”: ui.Font({ ‘Bold’: True }),

---

Italic

Type: bool

Description

This attribute is used to apply *Italic* to the text

```
ui.Label({'Text': "My Label", "Font": ui.Font({ 'Italic': True }),
```

Underline

Type: bool

Description

This attribute is used to add a line under the text

```
ui.Label({'Text': "My Label", "Font": ui.Font({ 'Underline': True }),
```

Overline

Type: bool

Description

This attribute is used to add a line on top of the text

```
ui.Label({'Text': "My Label", "Font": ui.Font({ 'Overline': True }),
```

StrikeOut

Type: bool

Description

This attribute is used to add a line through the text

```
ui.Label({'Text': "My Label", "Font": ui.Font({ 'StrikeOut': True }),
```

Kerning

Type:

Description

This attribute is used to

Note: Not yet tested

ui.Font({ 'Kerning': 24 })

Weight

Type: int, float

Description

This attribute is used to set a size relative to other element of the group. Element with Weight 0.5 will be twice the size of an element with Weight 0.25

Note: Not yet tested

ui.Font({ 'Weight': 0.25 })

Stretch

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Font({ 'Stretch': True })

MonoSpaced

Type: bool

Description

This label attribute is used to

Note: Not yet tested

ui.Font({ 'MonoSpaced': True })

2.4.12 Icon

File

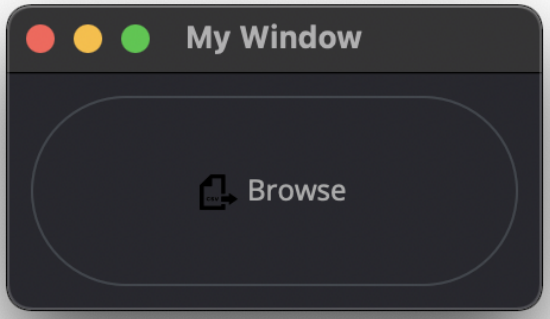
Type: string

Description

This attribute is used to point to an image file path to use for the Icon Element. Need to be associated with an element supporting Icon attribute. (ie: ui.Button)

- .png
- .jpg

```
ui.Button({ 'ID': "Browse",  'Text': " Browse", "Icon": ui.Icon({'File': r"UserData:/
↪Scripts/images/csv.png"})})
```



2.4.13 TabBar

Note: Before you can edit TabBar attributes, you need to create a TabBar element, then use the UI Element function AddTab()

Also note that TabBar has *TabBar Property Array*

CurrentIndex

Type: int

Description
This attribute is used to set the current TabBar index

Note: Not yet tested

ui.TabBar({ 'ID':'tabbar_1', 'CurrentIndex': 3 }) win.Find('tabbar_1').AddTab('Tab1')

win.Find('tabbar_1').AddTab('Tab2')

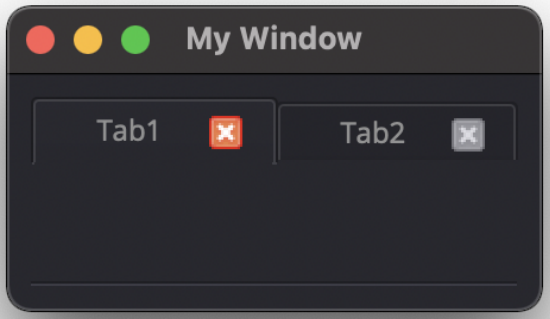
TabsClosable

Type: bool

Description

This attribute is used to add a button to close tabs

```
ui.TabBar({ 'ID':'tabbar_1', 'TabsClosable': True })
    win.Find('tabbar_1').AddTab('Tab1')
win.Find('tabbar_1').AddTab('Tab2')
```



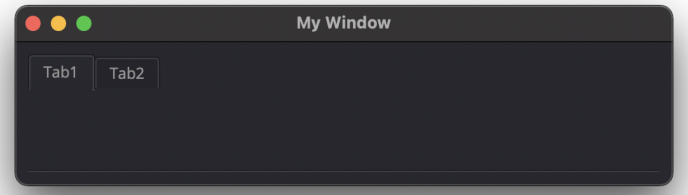
Expanding

Type: bool

Description

This attribute is used to force tabs to expand or not on Window resize. (default=True)

```
ui.TabBar({ 'ID':'tabbar_1', 'Expanding': False })
```



AutoHide

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.TabBar({ 'AutoHide': True })

Movable

Type: bool

Description

This attribute is used to enable Drag'n Drop to reorder tabs (default=False)

```
ui.TabBar({ 'ID':'tabbar_1', 'Movable': True })
```

DrawBase

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tabbar({ 'DrawBase': True })

UsesScrollButtons

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tabbar({ 'ID':'tabbar_1', 'UsesScrollButtons': True })

DocumentMode

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tabbar({ 'DocumentMode': True })

ChangeCurrentOnDrag

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tabbar({ 'ChangeCurrentOnDrag': True })

2.4.14 Stack

Description #NotInReadme

Stack are groups of Elements used with TabBar to manage each pages

```
ui.Stack({'ID':'stack_1'})
```

CurrentIndex

toolbox_items[‘Stack’].CurrentIndex = 0

AddChild()

toolbox_items[‘Stack’].AddChild(ui.Button({‘ID’: “Browse”, “Icon”: ui.Icon({‘File’: r”UserData:/Scripts/images/test.gif”}), ‘IconSize’ : [15, 15]}))

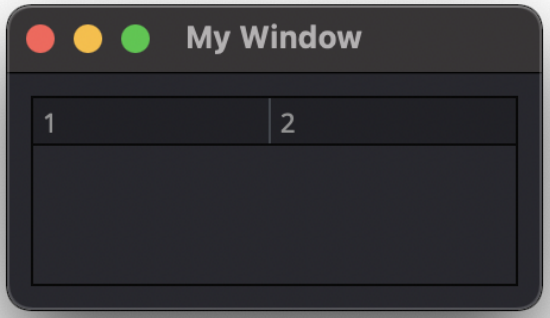
2.4.15 Tree

ColumnCount

Type: int

Description
This attribute is used to set the number of column in the Tree

```
ui.Tree({ 'ID':'my_tree', 'ColumnCount': 2 })
```



SortingEnabled

Type: bool

Description
This attribute enables sorting of the TreeItems elements. (default=False)

```
ui.Tree({ 'ID':'my_tree', 'SortingEnabled': True })
```

ItemsExpandable

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'ItemsExpandable': True })

ExpandsOnDoubleClick

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'ExpandsOnDoubleClick': True })

AutoExpandDelay

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'AutoExpandDelay': True })

HeaderHidden

Type: bool

Description

This attribute is used to hide the header row.

```
ui.Tree({ 'ID':'my_tree', 'HeaderHidden': True })
```

IconSize

Type: int

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'Icon': ui.Icon({'File': r"UserData:/Scripts/images/logo.png"}, 'IconSize': 12 })

RootIsDecorated

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'RootIsDecorated': True })

Animated

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'Animated': True })

AllColumnsShowFocus

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'AllColumnsShowFocus': True })

WordWrap

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'WordWrap': True }) itm = win.Find('my_tree').NewItem()

itm.Text[0] = "too long text for the cell" itm.Text[1] = "this is also too long"

win.Find('my_tree').AddTopLevelItem(itm)

TreePosition

Type:

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'TreePosition': })

SelectionBehavior

Type:

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'SelectionBehavior': })

SelectionMode

Type:

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'SelectionMode': })

UniformRowHeights

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'UniformRowHeights': True })

Indentation

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'Indentation': True })

VerticalScrollMode

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'VerticalScrollMode': True })

HorizontalScrollMode

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'HorizontalScrollMode': True })

AutoScroll

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'AutoScroll': True })

AutoScrollMargin

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'AutoScrollMargin': True })

TabKeyNavigation

Type: bool

Description

This attribute is used to allow Tab to go to next row, Shift+Tab to previous. (default=False)

```
ui.Tree({ 'ID':'my_tree', 'TabKeyNavigation': True })
```

AlternatingRowColors

Type: bool

Description

This attribute is used activate atlerning row colors on the Tree (default=False)

```
ui.Tree({ 'ID':'my_tree', 'AlternatingRowColors': True })
```

FrameStyle

Type: int

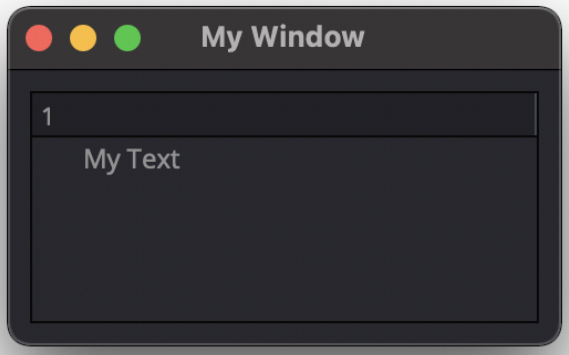
Description

This attribute is used to Style the frame of the Tree Element.

Check out the qt5 documentation for more details

- 0: NoFrame
- 1: Box
- 2: Panel
- 3: WinPanel
- 4: HLine
- 5: VLine
- 6: StyledPanel
- other to try

```
ui.Tree({ 'ID':'my_tree', 'FrameStyle': 1 })
```



LineWidth

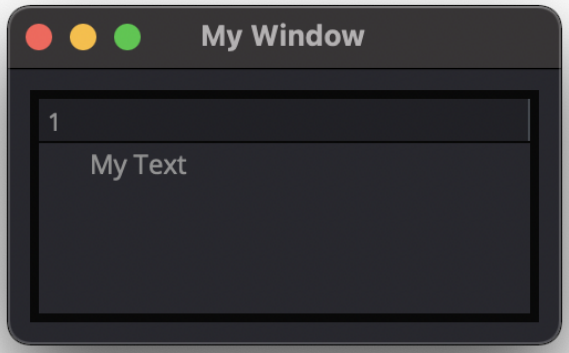
Type: int

Description

This attribute is used to adjust the line with of the selected FrameStyle

`FrameStyle is required`

```
ui.Tree({ 'ID':'tree_1', 'FrameStyle': 1, 'LineWidth': 4 })
```



MidLineWidth

Type: int

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'MidLineWidth': 2 })

FrameRect

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'FrameRect': True })

FrameShape

Type:

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'FrameShape': })

FrameShadow

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Tree({ 'ID':'my_tree', 'FrameShadow': True })

2.4.16 TreeItem

Note: Before you can edit TreeItem attributes, you need to create a Tree element, then use the UI Element function to add Item to the Tree

```
itm = win.Find('my_tree').NewItem()

win.Find('my_tree').AddTopLevelItem(itm)
```

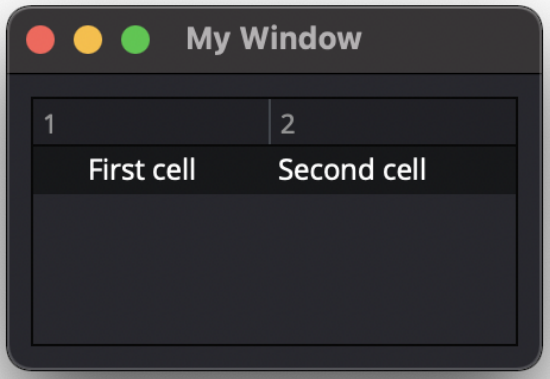
Selected

Type: bool

Description

This attribute is used to define the selected status to an item of the Tree. (default=False)

```
itm = win.Find('my_tree').NewItem()
win.Find('my_tree').AddTopLevelItem(itm)

itm.Selected = True
```



Hidden

Type: bool

Description

This attribute is used to define the selected status to an item of the Tree. (default=False)

```
itm = win.Find('my_tree').NewItem()
win.Find('my_tree').AddTopLevelItem(itm)

itm.Hidden = True
```

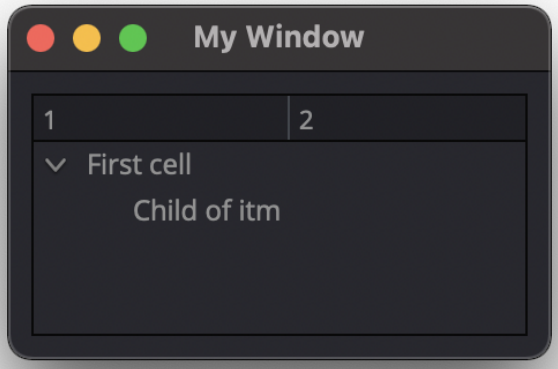
Expanded

Type: bool

Description

This attribute is used to define the expanded status to an item of the Tree. (default=False) `TreeItem must have child to display.`

```
itm = win.Find('my_tree').NewItem()
itm2 = win.Find('my_tree').NewItem()

itm.Text[0] = "First cell"
itm2.Text[0] = "Child of itm"
itm.AddChild(itm2)

win.Find('my_tree').AddTopLevelItem(itm)
itm.Expanded = True
```



Disabled

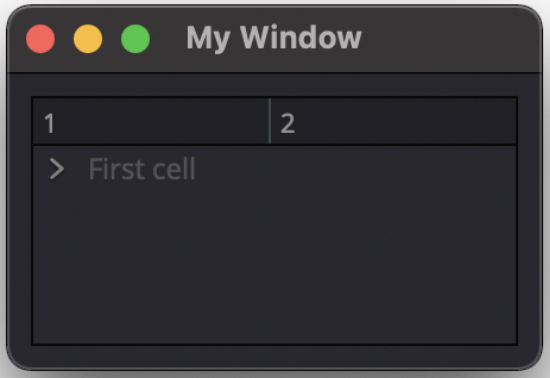
Type: bool

Description

This attribute is used to define the disabled status to an item of the Tree. (default=False)

`TreeItem will be grayed out.`

```
itm = win.Find('my_tree').NewItem()
itm2 = win.Find('my_tree').NewItem()

itm.Text[0] = "First cell"
itm2.Text[0] = "Child of itm"
itm.AddChild(itm2)

win.Find('my_tree').AddTopLevelItem(itm)
itm.Disabled = True
```



FirstColumnSpanned

Type: bool

Description
This attribute is used to

Note: Not yet tested

ui.TreeItem({ 'FirstColumnSpanned': True })

Flags

Type: bool

Description
This attribute is used to

Note: Not yet tested

ui.TreeItem({ 'Selected': True })

ChildIndicatorPolicy

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.TreeItem({ 'Selected': True })

Important: Some elements also have property arrays, indexed by item or column (zero-based), e.g. newItem.Text[2] = 'Third column text'

2.4.17 Combo

ItemText[index]

Type: string

Description

This attribute is used to get the Item Text of the ComboBox at specified index.

```
win.Find("combo_1").AddItems(["Blue","Cyan","Green","Yellow","Red","Pink","Purple",
↪"Fuchsia","Rose","Lavender","Sky","Mint","Lemon","Sand","Cocoa","Cream"])
first_color = win.Find('combo_1').ItemText[0]
# Blue
```

2.4.18 TabBar Property Array

TabText[index]

Type: string

Description

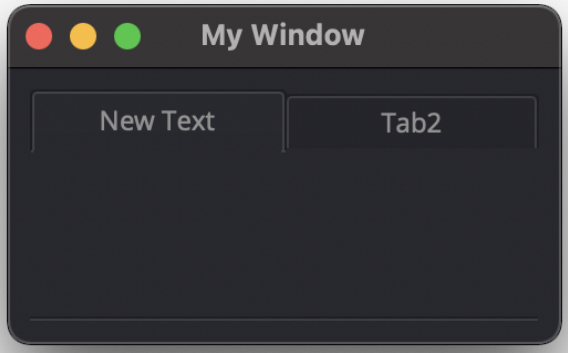
This attribute is used to get or set the Tab Text of the selected Tab index.

```
ui.TabBar({'ID':'tabbar_1'})

win.Find('tabbar_1').AddTab('Tab1')
win.Find('tabbar_1').AddTab('Tab2')
```

(continues on next page)

(continued from previous page)

```
print(win.Find('tabbar_1').TabText[0])  #Tab1

win.Find('tabbar_1').TabText[0] = 'New Text'
```



TabToolTip[index]

Type: string

Description

This attribute is used to display a text when mouse hover the tab

```
ui.TabBar({'ID':'tabbar_1'})
    win.Find('tabbar_1').AddTab('Tab1')

win.Find('tabbar_1').TabToolTip[0] = 'Tool tip'
```

TabWhatsThis[ ]

Type: string

Description

This attribute is used to

Note: Not yet tested

newItem.TabWhatsThis[2] = “Third Tab WhatsThis Text”

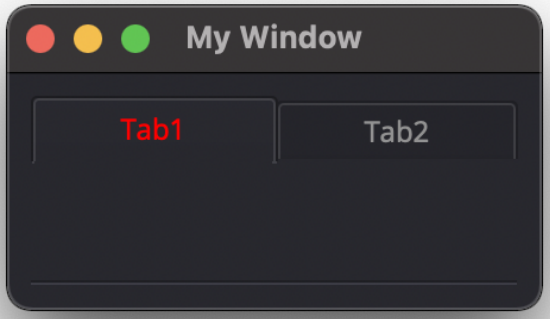
TabTextColor[index]

Type: dict

Description

This attribute is used to change the Tab Text color with RGBA dictionary values.

```
ui.TabBar({'ID':'tabbar_1'})
    win.Find('tabbar_1').AddTab('Tab1')
    win.Find('tabbar_1').TabTextColor[0] = { 'R':1, 'G': 0, 'B':0, 'A':1 }
```



2.4.19 Tree Property Array

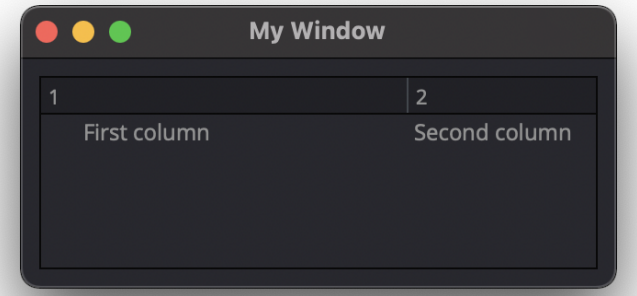
ColumnWidth[index]

Type: int

Description

This attribute is used change the Width of a Tree column

```
itm = win.Find('my_tree').NewItem()
itm.Text[0] = "First column"
itm.Text[1] = "Second column"
win.Find('my_tree').AddTopLevelItem(itm)

    win.Find('my_tree').ColumnWidth[0] = 200
```



2.4.20 Treeitem Property Array

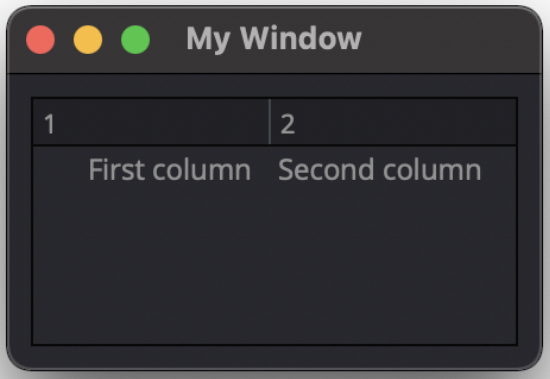
Text[index]

Type: string

Description

This attribute is used to set the TreeItem text at column index

```
itm = win.Find('my_tree').NewItem()
itm.Text[0] = "First column"
itm.Text[1] = "Second column"

win.Find('my_tree').AddTopLevelItem(itm)
```



StatusTip[ ]

Type: string

Description

This attribute is used to

Note: Not yet tested

newItem.StatusTip[2] = 'StatusTip'

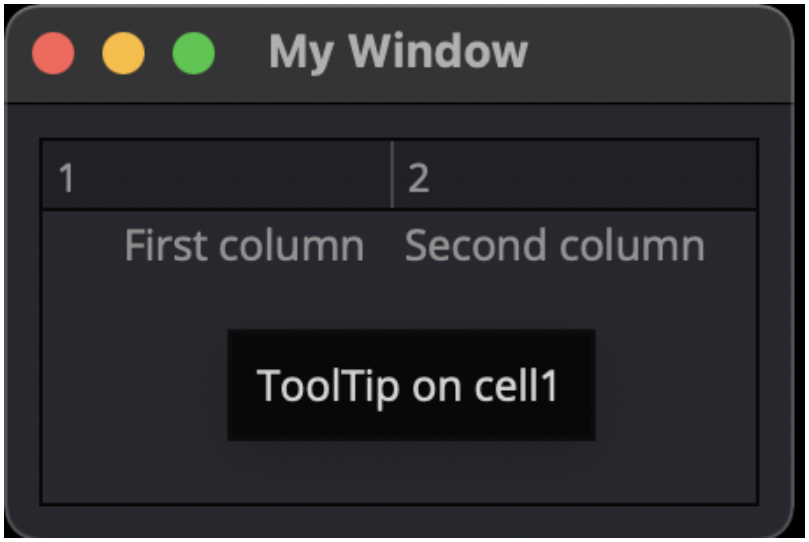
ToolTip[index]

Type: string

Description

This attribute is used to display a text when mouse hover a cell

```
itm = win.Find('my_tree').NewItem()

itm.Text[0] = "First column"
itm.Text[1] = "Second column"
itm.ToolTip[0] = 'ToolTip on cell1'

win.Find('my_tree').AddTopLevelItem(itm)
```



WhatsThis[ ]

Type: string

Description

This attribute is used to . . .

Note: Not yet tested

newItem.WhatsThis[2] = 'WhatsThis'

SizeHint[ ]

Type: int

Description

This attribute is used to

Note: Not yet tested

newItem.SizeHint[2] = 'SizeHint inside Tree in third row'

TextAlignment[ ]

Type:

Description

This attribute is used to

Note: Not yet tested

newItem.TextAlignment[2] = 'TextAlignment inside Tree in third row'

CheckState[ ]

Type: bool

Description
This attribute is used to

Note: Not yet tested

newItem.CheckState[2] = 'CheckState inside Tree in third row'

BackgroundColor[index]

Type: dict

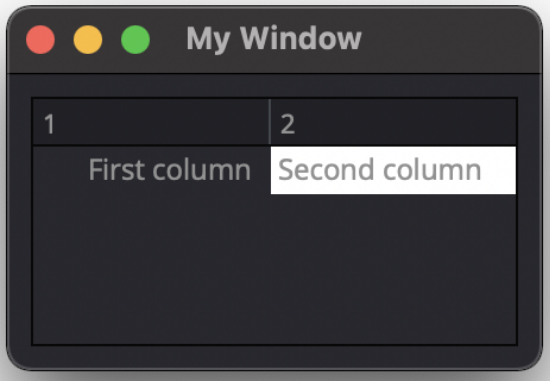
Description
This attribute is used to set a BackgroundColor to a cell using RGBA dictionary.

Note: Not yet tested

itm = win.Find('my_tree').NewItem()

itm.Text[0] = "First column" itm.Text[1] = "Second column" itm.BackgroundColor[1] = {'R':1, 'G':1, 'B':1, 'A':1}

win.Find('my_tree').AddTopLevelItem(itm)



TextColor[index]

Type: dict

Description

This attribute is used to change the color of the text using RGBA dictionary

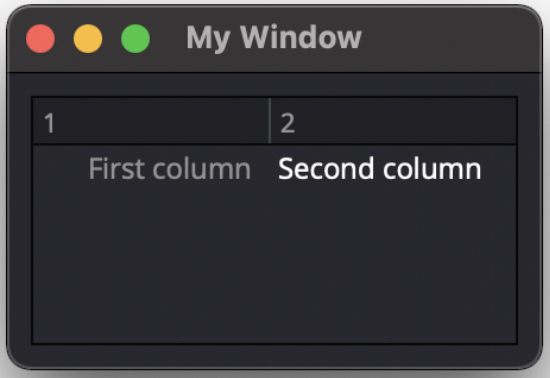
---

Note: Not yet tested

itm = win.Find('my_tree').NewItem()

itm.Text[0] = "First column" itm.Text[1] = "Second column" itm.TextColor[1] = {'R':1, 'G':1, 'B':1, 'A':1}

win.Find('my_tree').AddTopLevelItem(itm)



Icon[index]

Type: ui.Icon

Description

This attribute is used to add an icon image into a cell.

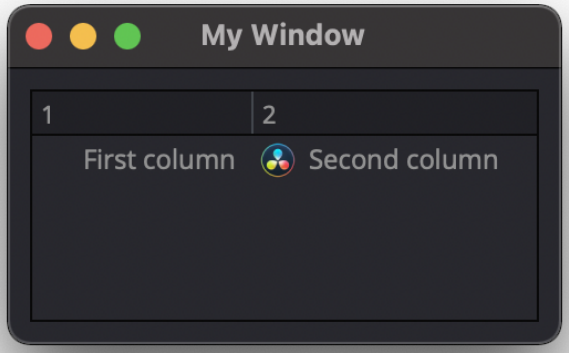
Refer to *Icon* for property list.

```
itm = win.Find('my_tree').NewItem()

itm.Text[0] = "First column"
itm.Text[1] = "Second column"
```

(continues on next page)

(continued from previous page)

```
itm.Icon[1] =  ui.Icon({'File': r"UserData:/Scripts/images/logo.png"})

win.Find('my_tree').AddTopLevelItem(itm)
```



Font[index]

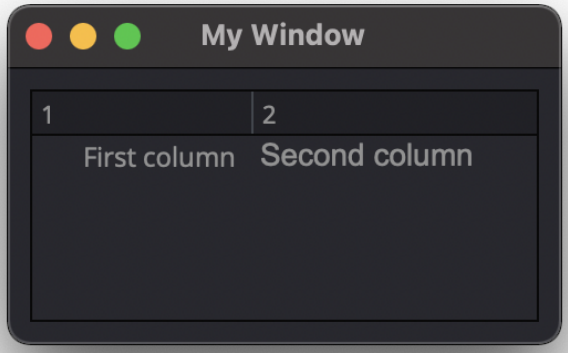
Type: ui.Font

Description

This attribute is used to modify the Font used inside a cell.

Refer to *Font* for property list.

```
itm = win.Find('my_tree').NewItem()

itm.Text[0] = "First column"
itm.Text[1] = "Second column"
itm.Font[1] =  ui.Font({ 'Family': "Arial", 'PointSize': 14})

win.Find('my_tree').AddTopLevelItem(itm)
```



Some elements like Label and Button will automatically recognise and render basic HTML in their Text attributes, and TextEdit is capable of displaying and returning HTML too. Element attributes can be specified when creating the element, or can be read or changed later:

```
win.Find('myButton').Text = "Processing..."
```

2.4.21 Timer

Interval

Type: int

Description #NotInReadme

This attribute is used to set a time in milisecs to the ui.Timer Element.

```
    ui.Timer({ 'ID': 'MyTimer', 'Interval': 1000 })  # 1000 millisecs
mytimer.Start()
    dispatcher['On']['Timeout'] = OnTimer  #this create a loop each 1000ms
```

Singleshot

Type: int

Description

This attribute is used to

Note: Not yet tested

ui.Timer({ 'ID': 'MyTimer', 'Singleshot': 1000 })

RemainingTime

Type: int

Description

This attribute is used to

Note: Not yet tested

ui.Timer({ 'ID': 'MyTimer', 'RemainingTime': 1000 })

IsActive

Type: bool

Description

This attribute is used to

Note: Not yet tested

ui.Timer({ 'ID': 'MyTimer', 'IsActive': True })

2.5 Functions

Most elements have functions that can be called from them as well:

- Show()
- Hide()
- Raise()
- Lower()
- Close() Returns boolean
- Find(ID) Returns child element with matching ID
- GetChildren() Returns list
- AddChild(element)
- RemoveChild(element)
- SetParent(element)
- Move(point)

- Resize(size)
- Size() Returns size
- Pos() Returns position
- HasFocus() Returns boolean
- SetFocus(reason) Accepts string “MouseFocusReason”, “TabFocusReason”, “ActiveWindowFocusReason”, “OtherFocusreason”, etc
- FocusWidget() Returns element
- IsActiveWindow() Returns boolean
- SetTabOrder(element)
- Update()
- Repaint()
- SetPaletteColor(r,g,b)
- QueueEvent(name, info) Accepts event name string and dictionary of event attributes
- GetItems() Returns dictionary of all child elements

Some elements have extra functions of their own:

2.5.1 Label

SetSelection(int, int)

Description

This function used to

Type: int= int=

Note: Not tested yet

win[‘mylabel’].SetSelection(0,1)

HasSelection()

Description

This function return True if Label has selection

Type: return bool

Note: Not tested yet

win[‘mylabel’].HasSelection()

SelectedText()

Description

This function return SelectedText string

Type: return string

Note: Not tested yet

win['mylabel'].SelectedText()

SelectionStart()

Description

This function return the index of the selection

Type: return int

Note: Not tested yet

win['mylabel'].SelectionStart()

2.5.2 Button

Click()

Description

This function is

Type: func

Note: Not tested yet

win['mybutton'].Click()

Toggle()

Description

This function is

Type: func

Note: Not tested yet

win['mybutton'].Toggle()

AnimateClick()

Description

This function is

Type: func

Note: Not tested yet

win['mybutton'].AnimateClick()

2.5.3 CheckBox

Click()

Description

This function is

Type: func

Note: Not tested yet

win['mycheckbox'].Click()

Toggle()

Description

This function is

Type: func

Note: Not tested yet

win['mycheckbox'].Toggle()

AnimateClick()

Description

This function is

Type: func

Note: Not tested yet

win['mycheckbox'].AnimateClick()

2.5.4 ComboBox

AddItem(string)

Description

This function add the item to the ComboBox list.

Type: func

```
win.Find('combo_1').AddItem('Item Name')
```

InsertItem(int, string)

Description

This function is inserting an item at the specified index.

Type: func

```
win.Find('combo_1').InsertItem(1, 'New item')
```

AddItems(list)

Description

This function is adding a list of item to the ComboBox list.

Type: func

```
win.Find('combo_1').AddItems(['Item 1', 'Item 2', 'Item 3'])
```

InsertItems(int, list)

Description

This function is inserting a list of items at the specified index.

Type: int= index, list=[string]

```
win.Find('combo_1').InsertItems(1, ['Item 1', 'Item 2'])
```

InsertSeparator(int)

Description

This function inserts a Seprator in the list at the specified index.

Type: int= index

```
win.Find('combo_1').InsertSeparator(2)  #insert after second item
```

RemoveItem(int)

Description

This function is

Type: int= index

```
win.Find('combo_1').RemoveItem(2)  #remove third item
```

Clear()

Description

This function removes all item from the ComboBox list

Type: func

```
win.Find('combo_1').Clear()
```

SetEditText(string)

Description

This function sets the Text to appear in the editable Combox Item.

`ComboBox must be Editable`

Type: func

```
ui.ComboBox({'ID':'combo_1', 'Editable': True }),

win.Find('combo_1').SetEditText('My text')
```

ClearEditText()

Description

This function clears the EditText box in the Combox Item.

`ComboBox must be Editable`

Type: func

```
    ui.ComboBox({'ID':'combo_1', 'Editable': True }),

win.Find('combo_1').ClearEditText()
```

Count()

Description

This function returns the number of item in the ComboBox list.

Type: func

```
ui.ComboBox({'ID':'combo_1'}),
    win.Find("combo_1").AddItems(["Item 1","Item 2","Item 3"])
    item_count = win.Find('combo_1').Count()
    print(item_count)  # 3
```

ShowPopup()

Description

This function opens the ComboBox list to display content

Type: func

```
win.Find('combo_1').ShowPopup()
```

HidePopup()

Description

This function closes the ComboBox list to hide content

Type: func

```
win.Find('combo_1').HidePopup()
```

2.5.5 SpinBox

SetRange(int, int)

Description

This function is setting a Minimum and Maximum value to the SpinBox.

Type: func

```
win.Find('spinbox_1').SetRange(0, 4)  #min=0, max=4
```

StepBy(int)

Description

This function adding the specified value to the SpinBox.

Type: func

```
win.Find('spinbox_1').StepBy(2)  #adds 2
```

StepUp()

Description

This function is adding the current Step value to the SpinBox (default=1)

Type: func

```
win.Find('spinbox_1').StepUp()
```

StepDown()

Description

This function is removing the current Step value to the SpinBox (default=1)

Type: func

```
win['myspinbox'].StepDown()
```

SelectAll()

Description

This function is selecting all the numbers in the SpinBox

Type: func

```
win.Find('spinbox_1').SelectAll()
```

Clear()

Description

This function clears the SpinBox display

Type: func

```
win.Find('spinbox_1').Clear()
```

2.5.6 Slider

SetRange(int, int)

Description

This function is setting a Minimum and Maximum value to the Slider.

Type: func

```
win.Find('slider_1').SetRange(0, 4)  #min=0, max=4
```

TriggerAction(string)

Description

This function is

Type: func

Note: Not tested yet

win[‘myslider’].TriggerAction(string)

2.5.7 LineEdit

SetSelection(int, int)

Description

This function is selecting a range of characters in the LineEdit.

Type: func int = index start, index end

```
win.Find('le_1').SetSelection(0, 4)  #selects the first 4 characters
```

HasSelectedText()

Description

This function is

Type: return bool

Note: Not tested yet

win['le_1'].HasSelectedText()

SelectedText()

Description

This function is

Type: return string

Note: Not tested yet

win['le_1'].SelectedText()

SelectionStart()

Description

This function is

Type: return int

Note: Not tested yet

win['le_1'].SelectionStart()

SelectAll()

Description

This function is selecting all the text in the LineEdit element.

Type:

```
win.Find('le_1').SelectAll()
```

Clear()

Description

This function deletes all the text in the LineEdit element.

Type: return

```
win.Find('le_1').Clear()
```

Cut()

Description

This function will copy to clipboard and remove the selected LineEdit characters. `A selection in the LineEdit is required`

Type:

```
win.Find('le_1').SetSelection(0, 4)
win.Find('le_1').Cut()  #this will cut the first 4 characters
```

Copy()

Description

This function will copy to clipboard the selected LineEdit characters. `A selection in the LineEdit is required`

Type: return

```
win.Find('le_1').SetSelection(0, 4)
win.Find('le_1').Copy()  #this will copy the first 4 characters
```

Paste()

Description

This function paste the clipboard content to the LineEdit element.

Type:

```
win.Find('le_1').Paste()
```

Undo()

Description

This function wil undo the last action made in the TextEdit element.

Type:

```
win.Find('le_1').Undo()
```

Redo()

Description

This function is

Type:

Note: Not tested yet

win[‘le_1’].Redo()

Deselect()

Description

This function will deselect the selected text of the LineEdit element.

Type:

```
win.Find('le_1').Deselect()
```

Insert(string)

Description

This function insert the text string at the cursor position in the LineEdit element.

Type:

```
win.Find('le_1').Insert('New Text')
```

Backspace()

Description

This function remove the last character from the cursor position in the LineEdit element.

Type:

```
win.Find('le_1').Backspace()
```

Del()

Description

This function remove the next character from the cursor position in the LineEdit element.

Type:

```
win.Find('le_1').Del()
```

Home(bool)

Description

This function is selecting all characters from cursor to beginning when set to True.

Type:

```
win.Find('le_1').Home(True)
```

End(bool)

Description

This function is selecting all characters from cursor to end when set to True.

Type:

```
win.Find('le_1').End(True)
```

CursorPositionAt(point)

Description

This function is

Type: return int

Note: Not tested yet

win[‘le_1’].CursorPositionAt(point)

2.5.8 TextEdit

InsertPlainText(string)

Description

This function insert the text string at the cursor position in the TextEdit element.

Type: func

```
win.Find('te_1').InsertPlainText('New text')
```

InsertHTML(string)

Description

This function insert the HTML code string at the cursor position in the TextEdit element.

Type: func

```
win.Find('te_1').InsertHTML('<h1>My title</h1>')
```

Append(string)

Description

This function is adding the string on the next line of the TextEdit box.

Type: func

```
win.Find('te_1').Append('My text')
```

SelectAll()

Description

This function is selecting all the text in the TextEdit element.

Type: func

```
win.Find('te_1').SelectAll()
```

Clear()

Description

This function deletes all the text in the LineEdit element.

Type: func

```
win.Find('te_1').Clear()
```

Cut()

Description

This function will copy to clipboard and remove the selected LineEdit characters. `A selection in the LineEdit is required`

Type:

```
win.Find('te_1').SetSelection(0, 4)
win.Find('te_1').Cut()  #this will cut the first 4 characters
```

Copy()

Description

This function will copy to clipboard the selected characters. `A selection in the TextEdit is required`

Type: return

```
win.Find('te_1').SelectAll()
win.Find('te_1').Copy()  #this will copy all text to clipbboard
```

Paste()

Description

This function paste the clipboard content to the cursor position in the TextEdit element.

Type:

```
win.Find('te_1').Paste()
```

Undo()

Description

This function wil undo the last action made on each line of the TextEdit element.

Type:

```
win.Find('te_1').Undo()
```

Redo()

Description

This function is

Type:

Note: Not tested yet

win[‘te_1’].Redo()

ScrollToAnchor(string)

Description

This function is

Type: func

Note: Not tested yet

win[‘te_1’].ScrollToAnchor(‘My text’)

ZoomIn(int)

Description

This function is increase the displayed text size by defined number.

Type: func

```
win.Find('te_1').ZoomIn(14)  #will display text with 14pt size
```

ZoomOut(int)

Description

This function is decrease the displayed text size by defined number.

Type: func

Note: Not tested yet

win.Find(‘te_1’).ZoomOut(2) #will reduce text size of 2pt

EnsureCursorVisible()

Description

This function is

Type: func

Note: Not tested yet

win['te_1'].EnsureCursorVisible()

MoveCursor(moveOperation, moveMode)

Description

This function is

Type: moveOperation = , moveMode =

Note: Not tested yet

win['te_1'].MoveCursor(moveOperation, moveMode)

CanPaste()

Description

This function is

Type: return bool

Note: Not tested yet

win['te_1'].CanPaste()

AnchorAt(point)

Description

This function is

Type: return string

Note: Not tested yet

win['te_1'].AnchorAt(point)

Find(string, findFlags)

Description

This function is

Type: string= , findFlags= : return bool

Note: Not tested yet

win['te_1'].Find('my text', findFlags)

2.5.9 TabBar

AddTab(string)

Description

This function adds a Tab with specified name to the TabBar

Type: string

```
win.Find('tabbar_1').AddTab('Tab 1')
```

InsertTab(int, string)

Description

This function insert a Tab in the TabBar at specified index.

Type: returns tab index (int)

```
win.Find('tabbar_1').InsertTab(0, 'Tab 0')  #insert tab at index 0
```

Count()

Description

This function counts the number of Tabs

Type: return number of Tab

```
print(win.Find('tabbar_1').Count())
```

RemoveTab(int)

Description

This function is

Type: int= Tab index

```
win.Find('tabbar_1').RemoveTab(0)  #remove first tab
```

MoveTab(int, int)

Description

This function moves a Tab to another position

Type: int=tab index to move int=tab index destination

```
win.Find('tabbar_1').MoveTab(1, 0)  #move second tab to first position
```

2.5.10 Tree

AddTopLevelItem(item)

Description

This function adds the item at the top of the Tree.

Type: item= TreeItem

```
item = win.Find('tree_1').NewItem()
item.Text[0] = 'My Text'
win.Find('tree_1').AddTopLevelItem(item)
```

InsertTopLevelItem(int, item)

Description

This function insert the item at specified position.

Type: int= index, item= TreeItem

```
item = win.Find('tree_1').NewItem()
item.Text[0] = 'Insert'
win.Find('tree_1').InsertTopLevelItem(0, item)
```

SetHeaderLabel(string)

Description

This function is setting the name for the first header

Type: string= header label

```
win.Find('tree_1').SetHeaderLabel('New header')
```

CurrentColumn()

Description

This function returns the selected column index.

Type: returns int

```
print(win.Find('tree_1').CurrentColumn())
```

SortColumn()

Description

This function is

Type: return int

Note: Not tested yet

win[‘mytree’].SortColumn()

TopLevelItemCount()

Description

This function return the number of Item in the Tree. (row)

Type: returns

```
print(win.Find('tree_1').TopLevelItemCount())
```

CurrentItem()

Description

This function returns the selected Item in the Tree (row)

Type: return the UITreeItem

```
print(win.Find('tree_1').CurrentItem().Text[0])  #print first column Text of the␣
↪selected TreeItem (row)
```

TopLevelItem(int)

Description

This function return the UITreeItem at the specified index. (row)

Type: int= index

```
print(win.Find('tree_1').TopLevelItem(1).Text[0])  #will print the Text of the second␣
↪Item (row), first column
```

TakeTopLevelItem(int)

Description

This function removes and returns the UITreeItem at the specified index. (row)

Type: int= return item

```
win.Find('tree_1').TakeTopLevelItem(1)
```

InvisibleRootItem()

Description

This function is

Type: return item

Note: Not tested yet

win['mytree'].InvisibleRootItem()

HeaderItem()

Description

This function is

Type: return item

Note: Not tested yet

win['mytree'].HeaderItem()

IndexOfTopLevelItem(item)

Description

This function returns the index of the specified UITreeItem.

Type: return int

```
some_item = win.Find('tree_1').TopLevelItem(1)
print(win.Find('tree_1').IndexOfTopLevelItem(some_item))  #print 1
```

ItemAbove(item)

Description

This function returns the UITreeItem above the specified UITreeItem.

Type: item= UITreeItem returns UITreeItem

```
some_item = win.Find('tree_1').TopLevelItem(1)
item_above = win.Find('tree_1').ItemAbove(some_item)  #item_above is UITreeItem at index␣
↪0
```

ItemBelow(item)

Description

This function is

Type: item= UITreeItem returns UITreeItem

```
some_item = win.Find('tree_1').TopLevelItem(0)
item_below = win.Find('tree_1').ItemBelow(some_item)  #item_below is UITreeItem at index␣
↪1
```

ItemAt(point)

Description

This function is

Type: point= return item

Note: Not tested yet

win[‘mytree’].ItemAt(point)

Clear()

Description

This function empty all data from the Tree.

Type:

```
win.Find('tree_1').Clear()
```

VisualItemRect(item)

Description

This function returns the rectangle on the viewport occupied by the item

Type: returns {int x, int y, int width, int height}

```
some_item = win.Find('tree_1').TopLevelItem(1)
print(win.Find('tree_1').VisualItemRect(some_item))   #print {1: 20, 2: 20, 3: 208, 4:␣
↪20}
```

SetHeaderLabels(list)

Description

This function sets the labels header for multiple columns

Type: list of string

```
win.Find('tree_1').SetHeaderLabels(['header1', 'header2'])
```

SetHeaderItem(item)

Description

This function is

Type: item =

Note: Not tested yet

win[‘mytree’].SetHeaderItem(item)

InsertTopLevelItems(int, list)

Description

This function inserts a list of UITreeItems from a list at the specified index.

Type: int= index to insert items, list = list of UITreeItems

```
item1 = win.Find('tree_1').NewItem()
item1.Text[0] = 'name1'
item2 = win.Find('tree_1').NewItem()
item2.Text[0] = 'name2'
win.Find('tree_1').InsertTopLevelItems(0, [item1, item2])  #insert items at index 0
```

AddTopLevelItems(list)

Description

This function adds the list of TreeItems at the end of the Tree.

Type: list = list of TreeItem

```
item1 = win.Find('tree_1').NewItem()
item1.Text[0] = 'name1'
item2 = win.Find('tree_1').NewItem()
item2.Text[0] = 'name2'
win.Find('tree_1').AddTopLevelItems([item1, item2])
```

SelectedItems()

Description

This function is

Type: return list of all selected UITreeItems

Note: Not tested yet

win[‘mytree’].SelectedItems()

FindItems(string, flags, column)

Description

This function searches for a string with a dictionary or conditions in a specified column index.

flags:

- ‘MatchExactly’ : bool
- ‘MatchFixedString’ : bool
- ‘MatchContains’ : bool
- ‘MatchStartsWith’ : bool
- ‘MatchEndsWith’ : bool
- ‘MatchCaseSensitive’ : bool
- ‘MatchRegExp’ : bool
- ‘MatchWildcard’ : bool
- ‘MatchWrap’ : bool
- ‘MatchRecursive’ : bool

Type: string= text to find , flags= dict, column = int Returns list of UITreeItems

```
found_item = win.Find('tree_1').FindItems("*",
{
        'MatchExactly' : False,
        'MatchFixedString' : False,
        'MatchContains' : False,
```

(continues on next page)

(continued from previous page)

```
        'MatchStartsWith' : False,
        'MatchEndsWith' : False,
        'MatchCaseSensitive' : False,
        'MatchRegExp' : False,
        'MatchWildcard' : True,
        'MatchWrap' : False,
        'MatchRecursive' :True,
}, 0)
# print all items of column 0 matching conditions, * is used as a wildcard
```

SortItems(int, string)

Description

This function is sorting the TreeItems of the specified column index based on the specified ordering.

Check out the qt5 documentation for more details

order:

- 'AscendingOrder' : The items are sorted ascending e.g. starts with 'AAA' ends with 'ZZZ' in Latin-1 locales
- 'DescendingOrder' : The items are sorted descending e.g. starts with 'ZZZ' ends with 'AAA' in Latin-1 locales

Check out the qt5 documentation for more details

Type: int= column index, string= sorting option

```
win.Find('tree_1').SortItems(0, 'AscendingOrder')
```

ScrollToItem(item)

Description

This function is

Type: item=

Note: Not tested yet

win['mytree'].ScrollToItem(item)

ResetIndentation()

Description

This function is

Type: func

Note: Not tested yet

win['mytree'].ResetIndentation()

SortByColumn(int, string)

Description

This function Sorts the model by the values in the given column and order.

Check out the qt5 documentatatoin for more details

order:

- 'AscendingOrder' : The items are sorted ascending e.g. starts with 'AAA' ends with 'ZZZ' in Latin-1 locales
- 'DescendingOrder' : The items are sorted descending e.g. starts with 'ZZZ' ends with 'AAA' in Latin-1 locales

Type: int= column index, string= order

```
win.Find('tree_1').SortByColumn(0, 'AscendingOrder')
```

FrameWidth()

Description

This function is

Type: return int

Note: Not tested yet

win['mytree'].FrameWidth()

2.5.11 TreeItem

AddChild(item)

Description

This function is adding an item as a child to an existing TreeItem.

Type: func

```
    itm = win.Find('my_tree').NewItem()
    itm.Text[0] = "First cell"
itm2 = win.Find('my_tree').NewItem()
    itm2.Text[0] = "Child of itm"

win.Find('my_tree').AddTopLevelItem(itm)

    itm.AddChild(itm2)
```

InsertChild(int, item)

Description

This function is inserting an item as a child to an existing TreeItem to a specified index.

Type: func

```
parent = win.Find('tree_1').NewItem()
parent.Text[0] = 'Text A'
child = win.Find('tree_1').NewItem()
child.Text[0] = 'Text B'
win.Find('tree_1').AddTopLevelItem(parent)

parent.InsertChild(0, child)
```

RemoveChild(item)

Description

This function remove the child of the UITreeItem.

Type: func

```
parent.RemoveChild(child)
```

SortChildren(int, order)

Description

This function is sorting the Child of UITreeItem of the specified column index based on the specified ordering.

order:

- 'AscendingOrder' : The items are sorted ascending e.g. starts with 'AAA' ends with 'ZZZ' in Latin-1 locales
- 'DescendingOrder' : The items are sorted descending e.g. starts with 'ZZZ' ends with 'AAA' in Latin-1 locales

Type: int= column index, string= order

```
parent.SortChildren(0, 'AscendingOrder')
```

InsertChildren(int, list)

Description

This function inserts a list of UITreeItem as child of a parent UITreeItem at specified index.

Type: int= , list=

```
parent.InsertChildren(0, [child, child2])
```

AddChildren(list)

Description

This function adds a list of UITreeItem as child of a parent UITreeItem.

Type: list= [UITreeItem, . . .]

```
parent.AddChildren([child, child2])
```

IndexOfChild(item)

Description

This function returns the index of the specified UITreeItem child.

Type: return int

```
print(parent.IndexOfChild(child2))  #print 1 for second child
```

Clone()

Description

This function is

Type: return item

Note: Not tested yet

win['mytreeitem'].Clone()

TreeWidget()

Description

This function is

Type: return tree

Note: Not tested yet

win['mytreeitem'].TreeWidget()

Parent()

Description

This function returns the UITreeItem parent of the specified UITreeItem child.

Type: return item

```
print(child.Parent())
```

Child(int)

Description

This function returns the UITreeItem child at specified index of the UITreeItem parent.

Type: int= return item

```
print(parent.Child(0))  #the UITreeItem child at index 0
```

TakeChild(int)

Description

This function removes and returns the child UITreeItem at specified index.

Type: int=index return item

```
removed_child = parent.TakeChild(0)
```

ChildCount()

Description

This function returns the child count of the parent UITreeItem.

Type: return int

```
print(parent.ChildCount())
```

ColumnCount()

Description

This function returns the number of column of a UITreeItem containing data.

Type: return int

```
print(parent.ColumnCount())
```

2.5.12 Window

Show()

Description

This function is showing the window to the user.

Type: func

```
win.Show()
```

Hide()

Description

This function is hiding the window.

Type: func

```
win.Hide()
```

RecalcLayout()

Description

This function is

Type: func

Note: Not tested yet

win.RecalcLayout()

2.5.13 Dialog

Exec()

Description

This function is

Type: func

Note: Not tested yet

dialog.Exec()

IsRunning()

Description

This function is

Type: func

Note: Not tested yet

dialog.IsRunning()

Done()

Description

This function is

Type: func

Note: Not tested yet

dialog.Done()

RecalcLayout()

Description

This function is

Type: func

Note: Not tested yet

dialog.RecalcLayout()

Elements can be accessed by the window's FindWindow(id) function, or by assigning them to a variable for later usage, which is more efficient. The GetItems() function will return a dictionary of all child elements for ease of access.

```
win_itms = win.GetItems()
win_itms['ElementID'].func()
```

2.5.14 Timer

Start()

Description

This function starts the Timer element.

Type: func

Note: Not tested yet

ui.Timer({ 'ID': 'MyTimer', 'Interval': 1000 }) # 1000 millisecs

MyTimer.Start() dispatcher['On']['Timeout'] = OnTimer #this create a loop each 1000ms

---

Stop()

Description

This function stops the Timer element.

Type: func

Note: Not tested yet

MyTimer.Stop()

2.6 Event Handlers

Window objects will call user-defined event handler functions in response to various interaction events. Event handlers are managed using a window member called 'On'. This has sub-members for each GUI element with an ID, and those have members for each available event. To set up an event handler, define a function for it, then assign the function to the window's On.ID.

Event member as follows:

```
def OnClose(ev):
        dispatcher.ExitLoop()

win.On.myWindow.Close = OnClose
```

Alternatively, if your object's ID is stored in a string variable called 'buttonID', you could use:

```
win.On[buttonID].Clicked = OnButtonClicked
```

Important: Event handlers can be enabled or disabled for a given element by turning them on or off in the Events attribute:

```
ui.Slider({ 'ID': 'mySlider', 'Events': { 'SliderMoved': True } })
```

Some common events like Clicked or Close are enabled by default.

Many objects have specific events that can be handled:

2.6.1 Button

Clicked

Description

This event is triggered with a mouse click. (default=True)

Type: event

```
ui.Button({'ID':'buttonID', 'Text': "My Button"})

    def OnButtonClicked(ev):
    print(f"{win.Find('buttonID').Text} was clicked")

# assign event handlers
    win.On['buttonID'].Clicked = OnButtonClicked
```

Toggled

Description

This event is triggered if button is Toggled

Type: event

Note: Not tested yet

ui.Button({'ID':'buttonID', 'Text': "My Button"})

def OnButtonToggled(ev): print(f"{win.Find('buttonID').Text} was Toggled")

assign event handlers win.On['buttonID'].Toggled = OnButtonToggled

Pressed

Description

This event is triggered button is pressed. (default=False)

Type: event

```
ui.Button({'ID':'buttonID', 'Text': "My Button", 'Events': { 'Pressed': True }})

def OnButtonPressed(ev):
    print(f"{win.Find('buttonID').Text} was Pressed")

# assign event handlers
win.On['buttonID'].Pressed = OnButtonPressed
```

Released

Description

This event is triggered when button is released. (default=False)

Type: event

```
ui.Button({'ID':'buttonID', 'Text': "My Button", 'Events': { 'Released': True }})

def OnButtonReleased(ev):
    print(f"{win.Find('buttonID').Text} was Released")

# assign event handlers
win.On['buttonID'].Released = OnButtonReleased
```

2.6.2 CheckBox

Clicked

Description

This event is triggered with a mouse click (default=True)

Type: event

```
ui.CheckBox({'ID':'my_checkbox', 'Text': "My CheckBox" })

def OnCheckBoxClicked(ev):
    print(f"{win.Find('my_checkbox').Text} was Clicked")
```

(continues on next page)

(continued from previous page)

```
# assign event handlers
win.On['my_checkbox'].Clicked = OnCheckBoxClicked
```

Toggled

Description

This event is triggered

Type: event

Note: Not tested yet

win.On[‘my_checkbox’].Toggled = OnCheckboxToggle

Pressed

Description

This event is triggered when CheckBox is pressed (default=False)

Type: event

```
ui.CheckBox({'ID':'my_checkbox', 'Text': "My CheckBox", 'Events': { 'Pressed': True } })

    def OnCheckBoxPressed(ev):
    print(f"{win.Find('my_checkbox').Text} was Pressed")

# assign event handlers
win.On['my_checkbox'].Pressed = OnCheckBoxPressed
```

Released

Description

This event is triggered when CheckBox is released (default=False)

Type: event

```
ui.CheckBox({'ID':'my_checkbox', 'Text': "My CheckBox", 'Events': { 'Released': True } })

def OnCheckBoxReleased(ev):
    print(f"{win.Find('my_checkbox').Text} was Released")
```

(continues on next page)

(continued from previous page)

```
# assign event handlers
win.On['my_checkbox'].Released = OnCheckBoxReleased
```

2.6.3 ComboBox

CurrentIndexChanged

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_combobox'].CurrentIndexChanged = OnComboBoxCurrentIndexChanged

CurrentTextChanged

Description

This event is triggered each time the ComboBox current Text is changed

Type: event

```
ui.ComboBox({'ID':'combo_1', 'Text': "My ComboBox", 'Events': { 'CurrentTextChanged':␣
↪True} })

    def OnComboBoxCurrentTextChanged(ev):
    print(f"ComboBox CurrentTextChanged changed")

# assign event handlers
win.On['combo_1'].CurrentTextChanged = OnComboBoxCurrentTextChanged
```

TextEdited

Description

This event is triggered when Text is edited by user in a ComboBox item. `The ComboBox must be Editable`

Type: event

```
ui.ComboBox({'ID':'combo_1', 'Text': "My ComboBox", 'Editable': True, 'Events': {
↪'TextEdited': True} })

def OnComboBoxTextEdited(ev):
    print(f"ComboBox Text was Edited")

# assign event handlers
win.On['combo_1'].TextEdited = OnComboBoxTextEdited
```

EditTextChanged

Description

This event is triggered when modifications are made to a ComboBox item and ComboBox is changed. `The ComboBox must be Editable`

Type: event

Note: Not tested yet

win.On['my_combobox'].EditTextChanged = OnComboBoxEditTextChanged

---

EditingFinished

Description

This event is triggered `The ComboBox must be Editable`

Type: event

Note: Not tested yet

win.On['my_combobox'].EditingFinished = OnComboBoxEditingFinished

---

ReturnPressed

Description

This event is triggered when Return is pressed with the ComboBox item selected. Return will also add the modified ComboBox item to the list `The ComboBox must be Editable`

Type: event

```
ui.ComboBox({'ID':'combo_1', 'Text': "My ComboBox", 'Editable': True, 'Events': {
↪'ReturnPressed': True} })

    def OnComboBoxReturnPressed(ev):
    print(f"ReturnPressed on ComboBox")

# assign event handlers
win.On['combo_1'].ReturnPressed = OnComboBoxReturnPressed
```

Activated

Description

This event is triggered when activity is detected on the ComboBox.

Type: event

```
ui.ComboBox({'ID':'combo_1', 'Text': "My ComboBox", 'Events': { 'Activated': True } })

def OnComboBoxIActivated(ev):
    print(f"ComboBox was Activated")

# assign event handlers
win.On['combo_1'].Activated = OnComboBoxIActivated
```

2.6.4 SpinBox

ValueChanged

Description

This event is triggered when SpinBox value is changed. (default=True)

Type: event

```
ui.SpinBox({'ID':'spinbox_1'})

    def OnSpinBoxValueChanged(ev):
    print(f"Value Changed on SpinBox")

# assign event handlers
win.On['spinbox_1'].ValueChanged = OnSpinBoxValueChanged
```

EditingFinished

Description

This event is triggered when Edit are made to the SpinBox items. (default=False)

Type: event

```
ui.SpinBox({'ID':'spinbox_1', 'Events': { 'EditingFinished': True} })

def OnSpinBoxEditingFinished(ev):
    print(f"EditingFinished on SpinBox")

# assign event handlers
win.On['spinbox_1'].EditingFinished = OnSpinBoxEditingFinished
```

2.6.5 Slider

ValueChanged

Description

This event is triggered when Slider value is changed (default=True)

Type: event

```
ui.Slider({'ID':'slider_1'})

    def OnSliderValueChanged(ev):
    print(f"Slider value changed")

# assign event handlers
win.On['slider_1'].ValueChanged = OnSliderValueChanged
```

SliderMoved

Description

This event is triggered each time the slider is moved with mouse cursor. (default=False)

Type: event

```
ui.Slider({'ID':'slider_1', 'Events': {'SliderMoved': True } })

    def OnSliderSliderMoved(ev):
    print(f"Slider moved")
```

(continues on next page)

(continued from previous page)

```
# assign event handlers
win.On['slider_1'].SliderMoved = OnSliderSliderMoved
```

ActionTriggered

Description

This event is triggered when activity is detected on the Slider. (default=False)

Type: event

```
ui.Slider({'ID':'slider_1', 'Events': { 'ActionTriggered': True}  })

def OnSliderActionTriggered(ev):
    print(f"Action Triggered")

# assign event handlers
win.On['slider_1'].ActionTriggered = OnSliderActionTriggered
```

SliderPressed

Description

This event is triggered each time the slider is pressed, even if not moved. (default=False)

Type: event

```
ui.Slider({'ID':'slider_1', 'Events': {'SliderPressed': True}  })

def OnSliderSliderPressed(ev):
    print(f"Slider pressed")

# assign event handlers
win.On['slider_1'].SliderPressed = OnSliderSliderPressed
```

SliderReleased

Description

This event is triggered each time the slider is released, even if not moved. (default=False)

Type: event

```
ui.Slider({'ID':'slider_1', 'Events': { 'SliderReleased': True}  })

def OnSliderSliderReleased(ev):
    print(f"Slider released")

# assign event handlers
win.On['slider_1'].SliderReleased = OnSliderSliderReleased
```

RangeChanged

Description

This event is triggered is Minimum or Maximum slider value is change. (default=False)

Type: event

```
ui.Slider({'ID':'slider_1', 'Events': { 'RangeChanged': True}  })

    def OnSliderRangeChanged(ev):
    print("Range Changed")

# assign event handlers
win.On['slider_1'].RangeChanged = OnSliderRangeChanged

    win.Find('slider_1').Maximum = 4  #trigger RangeChanged
```

2.6.6 LineEdit

TextChanged

Description

This event is triggered each time Text is modified in the LineEdit element. (default=True)

Type: event

```
ui.LineEdit({'ID':'le_1' })

def OnLineEditTextChanged(ev):
    print(f"LineEdit text changed")

# assign event handlersdfg
win.On['le_1'].TextChanged = OnLineEditTextChanged
```

TextEdited

Description

This event is triggered each time Text is modified in the LineEdit element. (default=False)

Type: event

```
ui.LineEdit({'ID':'le_1', 'Events': {'TextEdited': True } })

def OnLineEditTextEdited(ev):
    print(f"LineEdit Text Edited")

# assign event handlersdfg
win.On['le_1'].TextEdited = OnLineEditTextEdited
```

EditingFinished

Description

This event is triggered if, after a modification to the Text, Return is pressed or Focus change to another Element. (default=False)

Type: event

```
ui.LineEdit({'ID':'le_1', 'Events': { 'EditingFinished': True } })

    def OnLineEditEditingFinished(ev):
    print(f"Editing Finished")

# assign event handlersdfg
win.On['le_1'].EditingFinished = OnLineEditEditingFinished
```

ReturnPressed

Description

This event is triggered when Return is pressed with LineEdit element selected. (default=False)

Type: event

```
ui.LineEdit({'ID':'le_1', 'Events': { 'ReturnPressed': True } })

def OnLineEditReturnPressed(ev):
    print(f"Return Pressed")
```

(continues on next page)

(continued from previous page)

```
# assign event handlersdfg
win.On['le_1'].ReturnPressed = OnLineEditReturnPressed
```

SelectionChanged

Description

This event is triggered

Type: event

Note: Not tested yet

win.On[‘my_le’].SelectionChanged = OnLineEditSelectionChanged

---

CursorPositionChanged

Description

This event is triggered each time the cursor change position in the LineEdit Element. (default=False) Inserting new character also changes the cursor position.

Type: event

```
ui.LineEdit({'ID':'le_1', 'Events': { 'CursorPositionChanged': True} })

def OnLineEditCursorPositionChanged(ev):
    print(f"Cursor Position Changed")

# assign event handlersdfg
win.On['le_1'].CursorPositionChanged = OnLineEditCursorPositionChanged
```

2.6.7 TextEdit

TextChanged

Description

This event is triggered each time Text is modified in the TextEdit element. (default=True)

Type: event

```
ui.TextEdit({'ID':'te_1' })

def OnLineEditTextChanged(ev):
    print(f"TextEdit text changed")

# assign event handlersdfg
win.On['te_1'].TextChanged = OnTextEditTextChanged
```

SelectionChanged

Description

This event is triggered

Type: event

Note: Not tested yet

win.On[‘my_te’].SelectionChanged = OnTextEditSelectionChanged

CursorPositionChanged

Description

This event is triggered each time the cursor change position in the TextEdit Element. (default=False) Inserting new character also changes the cursor position.

Type: event

```
ui.TextEdit({'ID':'te_1', 'Events': { 'CursorPositionChanged': True } })

def OnTextEditCursorPositionChanged(ev):
    print(f"Cursor Position Changed")

# assign event handlersdfg
win.On['te_1'].CursorPositionChanged = OnTextEditCursorPositionChanged
```

2.6.8 ColorPicker

ColorChanged

Description

This event is triggered when color is changed on ColorPicker element. (default=True)

Type: event

```
ui.ColorPicker({'ID':'colorpicker_1' })

def OnColorPickerColorChanged(ev):
    print(f"Color Changed")

# assign event handlersdfg
win.On['colorpicker_1'].ColorChanged = OnColorPickerColorChanged
```

2.6.9 TabBar

CurrentChanged

Description

This event is triggered each time Tab is changed. (default=True)

Type: event

```
ui.TabBar({'ID':'tabbar_1'})

def OnTabBarCurrentChanged(ev):
    print(f"Tab Changed")

# assign event handlersdfg
win.On['tabbar_1'].CurrentChanged = OnTabBarCurrentChanged
```

CloseRequested

Description

This event is triggered when a Tab is closed. (default=False) `TabsClosable must be set to True`

Type: event

```
ui.TabBar({'ID':'tabbar_1', 'TabsClosable': True, 'Events': {'CloseRequested': True} })
```

(continues on next page)

(continued from previous page)

```
    def OnTabBarCloseRequested(ev):
    print(f"Tab Close Requested")

# assign event handlersdfg
win.On['tabbar_1'].CloseRequested = OnTabBarCloseRequested
```

TabMoved

Description

This event is triggered when a Tab is moved. (default=False) `TabBar must be Movable`

Type: event

```
ui.TabBar({'ID':'tabbar_1', 'Movable': True, 'Events': {'TabMoved': True} })

    def OnTabBarTabMoved(ev):
    print(f"Tab Moved")

# assign event handlersdfg
win.On['tabbar_1'].TabMoved = OnTabBarTabMoved
```

TabBarClicked

Description

This event is triggered each time the TabBar is clicked. (default=False)

Type: event

```
ui.TabBar({'ID':'tabbar_1', 'Events': {'TabBarClicked': True} })

def OnTabBarClicked(ev):
    print(f"TabBar Clicked")

# assign event handlersdfg
win.On['tabbar_1'].TabBarClicked = OnTabBarClicked
```

TabBarDoubleClicked

Description

This event is triggered each time the TabBar is double clicked. (default=False)

Type: event

```
ui.TabBar({'ID':'tabbar_1', 'Events': {'TabBarDoubleClicked': True} })

def OnTabBarDoubleClicked(ev):
    print(f"TabBar Double Clicked")

# assign event handlersdfg
win.On['tabbar_1'].TabBarDoubleClicked = OnTabBarDoubleClicked
```

2.6.10 Tree

CurrentItemChanged

Description

This event is triggered when the TreeItem is changed. (default=True)

Type: event

```
ui.Tree({'ID':'tree_1'})

def OnCurrentItemChanged(ev):
        print('TreeItem was changed')

win.On['my_tree'].CurrentItemChanged = OnCurrentItemChanged
```

ItemClicked

Description

This event is triggered when a TreeItem is Clicked. (default=True)

Type: event

```
ui.Tree({'ID':'my_tree' })

def OnTreeItemClicked(ev):
        print('Item was clicked')

win.On['my_tree'].ItemClicked = OnTreeItemClicked
```

ItemPressed

Description

This event is triggered when a TreeItem is Clicked. (default=False)

Type: event

```
ui.Tree({'ID':'my_tree' })

def OnTreeItemPressed(ev):
        print('Item was pressed')

win.On['my_tree'].ItemPressed = OnTreeItemPressed
```

ItemActivated

Description

This event is triggered when a TreeItem is DoubleClicked or... (default=False)

Type: event

```
ui.Tree({'ID':'my_tree', 'Events' : { 'ItemDouItemActivatedbleClicked' : True } })

def OnTreeItemActivated(ev):
        print('Item was activated')

win.On['my_tree'].ItemActivated = OnTreeItemActivated
```

ItemDoubleClicked

Description

This event is triggered when a TreeItem is DoubleClicked. (default=False)

Type: event

```
ui.Tree({'ID':'my_tree', 'Events' : { 'ItemDoubleClicked' : True } })

def OnTreeItemDoubleClicked(ev):
        print('Item was double clicked')

win.On['my_tree'].ItemDoubleClicked = OnTreeItemDoubleClicked
```

ItemChanged

Description

This event is triggered

Type: event

Note: Not tested yet

win.On[‘my_tree’].ItemChanged = OnTreeItemChanged

ItemEntered

Description

This event is triggered when Enter is pressed on a TreeItem. (default=False)

Type: event

Note: Not tested yet

ui.Tree({‘ID’:’tree_1’, ‘Events’: {‘ItemEntered’: True} })

def OnItemEntered(ev): print(f”Item Entered”)

assign event handlers win.On[‘tree_1’].ItemEntered = OnItemEntered

---

ItemExpanded

Description

This event is triggered when a TreeItem with Child is Expanded. (default=False)

Type: event

```
    ui.Tree({'ID':'tree_1', 'Events': {'ItemExpanded': True} })

def OnItemExpanded(ev):
    print(f"Item Expanded")

# assign event handlers
win.On['tree_1'].ItemExpanded = OnItemExpanded
```

ItemCollapsed

Description

This event is triggered when a TreeItem with Child is Collapsed. (default=False)

Type: event

```
    ui.Tree({'ID':'tree_1', 'Events': {'ItemCollapsed': True} })

def OnItemExpanded(ev):
    print(f"Item Collapsed")

# assign event handlers
win.On['tree_1'].ItemCollapsed = OnItemCollapsed
```

CurrentItemChanged

Description

This event is triggered when the selected Tree item is changed. (default=True)

Type: event

Note: Not tested yet

ui.Tree({'ID':'tree_1'})

def OnCurrentItemChanged(ev): print(f"Current Item Changed")

assign event handlers win.On['tree_1'].CurrentItemChanged = OnCurrentItemChanged

---

ItemSelectionChanged

Description

This event is triggered when the selected Tree item is changed. (default=False)

Type: event

```
    ui.Tree({'ID':'tree_1', 'Events': {'ItemSelectionChanged': True}  })

def OnItemSelectionChanged(ev):
    print(f"Item Selection Changed")

# assign event handlers
win.On['tree_1'].ItemSelectionChanged = OnItemSelectionChanged
```

2.6.11 Window

Close

Description

This event is triggered when the Window close button is pressed. (default=True)

Type: event

```
# Event handlers
def OnWindowClose(ev):
    dispatcher.ExitLoop()

    win.On['my_window'].Close = OnWindowClose
```

Show

Description

This event is triggered when the Window is shown or returns from a hidden state.

Type: event

```
win = dispatcher.AddWindow({
    'ID': "my_window",
    'Geometry': [ 400,200,250,125 ],
    'WindowTitle': 'My Window',
    'Events': {'Show': True, 'Clicked': True, 'Close': True}
    },
    ui.VGroup([
            ui.Label({'ID':'label_1', 'Text': 'My label' })
    ])
)

def OnWindowShow(ev):
    print(f"Show")

    # assign event handlers
    win.On['my_window'].Show = OnWindowShow
```

Hide

Description

This event is triggered when the Window is hidden by another Window

Type: event

```
win = dispatcher.AddWindow({
    'ID': "my_window",
    'Geometry': [ 400,200,250,125 ],
    'WindowTitle': 'My Window',
    'Events': {'Hide': True, 'Clicked': True, 'Close': True}
    },
    ui.VGroup([
            ui.Label({'ID':'label_1', 'Text': 'My label' })
    ])
)

def OnWindowHide(ev):
    print(f"Hidden")

    # assign event handlers
    win.On['my_window'].Hide = OnWindowHide
```

Resize

Description

This event is triggered each time the Window is resized. (default=False)

Type: event

```
win = dispatcher.AddWindow({
    'ID': "my_window",
    'Events': {'Resize': True, 'Clicked': True, 'Close': True}
    },
    ui.VGroup([
            ui.Label({'ID':'label_1', 'Text': 'My label' })
    ])
)

    def OnWindowResize(ev):
    print(f"Window resized")

# assign event handler
    win.On['my_window'].Resize = OnWindowResize
```

MousePress

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].MousePress = OnWindowMousePress

MouseRelease

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].MouseRelease = OnWindowMouseRelease

MouseDoubleClick

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].MouseDoubleClick = OnWindowMouseDoubleClick

MouseMove

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].MouseMove = OnWindowMouseMove

Wheel

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].Wheel = OnWindowWheel

KeyPress

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].KeyPress = OnWindowKeyPress

KeyRelease

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].KeyRelease = OnWindowKeyRelease

FocusIn

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].FocusIn = OnWindowFocusIn

FocusOut

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].FocusOut = OnWindowFocusOut

ContextMenu

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].ContextMenu = OnWindowContextMenu

Enter

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].Enter = OnWindowEnter

Leave

Description

This event is triggered

Type: event

Note: Not tested yet

win.On['my_window'].Leave = OnWindowLeave

Event handler functions are called with a dictionary of related attributes such as who, what, when, sender, and modifiers.

Common events and some additional attributes they receive include:

- **MousePress:** Pos, GlobalPos, Button, Buttons
- **MouseRelease:** Pos, GlobalPos, Button, Buttons
- **MouseDoubleClick:** Pos, GlobalPos, Button, Buttons
- **MouseMove:** Pos, GlobalPos, Button, Buttons
- **Wheel:** Pos, GlobalPos, Buttons, Delta, PixelDelta, AngleDelta, Orientiation, Phase

- **KeyPress:** Key, Text, IsAutoRepeat, Count
- **KeyRelease:** Key, Text, IsAutoRepeat, Count
- **ContextMenu:** Pos, GlobalPos
- **Move:** Pos, OldPos
- **FocusIn:** Reason
- **FocusOut:** Reason

New in version Updated: as of 29 March 2022

2.7 API Introduction

In this package, you will find a brief introduction to the Scripting API for DaVinci Resolve Studio. Apart from this README.txt file, this package contains folders containing the basic import modules for scripting access (DaVinciResolve.py) and some representative examples.

From v16.2.0 onwards, the nodeIndex parameters accepted by SetLUT() and SetCDL() are 1-based instead of 0-based, i.e. 1 <= nodeIndex <= total number of nodes.

Overview

As with Blackmagic Design Fusion scripts, user scripts written in Lua and Python programming languages are supported. By default, scripts can be invoked from the Console window in the Fusion page, or via command line. This permission can be changed in Resolve Preferences, to be only from Console, or to be invoked from the local network. Please be aware of the security implications when allowing scripting access from outside of the Resolve application.

Prerequisites

DaVinci Resolve scripting requires one of the following to be installed (for all users):

- Lua 5.1
- Python 2.7 64-bit
- Python 3.6 64-bit

Using a script

DaVinci Resolve needs to be running for a script to be invoked.

For a Resolve script to be executed from an external folder, the script needs to know of the API location. You may need to set the these environment variables to allow for your Python installation to pick up the appropriate dependencies as shown below:

- Mac OS X: RESOLVE_SCRIPT_API="/Library/Application Support/Blackmagic Design/DaVinci Resolve/Developer/Scripting" RESOLVE_SCRIPT_LIB="/Applications/DaVinci Resolve/DaVinci Resolve.app/Contents/Libraries/Fusion/fusionscript.so" PYTHONPATH="$PYTHONPATH:$RESOLVE_SCRIPT_API/Modules/"
- Windows: RESOLVE_SCRIPT_API="%PROGRAMDATA%Blackmagic DesignDaVinci ResolveSupportDeveloperScripting" RESOLVE_SCRIPT_LIB="C:Program FilesBlackmagic DesignDaVinci Resolvefusionscript.dll" PYTHONPATH="%PYTHONPATH%;%RESOLVE_SCRIPT_API%Modules"
- Linux: RESOLVE_SCRIPT_API="/opt/resolve/Developer/Scripting" RESOLVE_SCRIPT_LIB="/opt/resolve/libs/Fusion/fusions PYTHONPATH="$PYTHONPATH:$RESOLVE_SCRIPT_API/Modules/" (Note: For standard ISO Linux installations, the path above may need to be modified to refer to /home/resolve instead of /opt/resolve)

As with Fusion scripts, Resolve scripts can also be invoked via the menu and the Console.

On startup, DaVinci Resolve scans the subfolders in the directories shown below and enumerates the scripts found in the Workspace application menu under Scripts. Place your script under Utility to be listed in all pages, under Comp or Tool to be available in the Fusion page or under folders for individual pages (Edit, Color or Deliver). Scripts under Deliver are additionally listed under render jobs. Placing your script here and invoking it from the menu is the easiest way to use scripts.

- Mac OS X:
  - All users: /Library/Application Support/Blackmagic Design/DaVinci Resolve/Fusion/Scripts
  - Specific user: /Users/<UserName>/Library/Application Support/Blackmagic Design/DaVinci Resolve/Fusion/Scripts
- Windows:
  - All users: %PROGRAMDATA%Blackmagic DesignDaVinci ResolveFusionScripts
  - Specific user: %APPDATA%RoamingBlackmagic DesignDaVinci ResolveSupportFusionScripts
- Linux:
  - All users: /opt/resolve/Fusion/Scripts (or /home/resolve/Fusion/Scripts/ depending on installation)
  - Specific user: $HOME/.local/share/DaVinciResolve/Fusion/Scripts

The interactive Console window allows for an easy way to execute simple scripting commands, to query or modify properties, and to test scripts. The console accepts commands in Python 2.7, Python 3.6 and Lua and evaluates and executes them immediately. For more information on how to use the Console, please refer to the DaVinci Resolve User Manual.

This example Python script creates a simple project:

```
#!/usr/bin/env python
import DaVinciResolveScript as dvr_script
resolve = dvr_script.scriptapp("Resolve")
fusion = resolve.Fusion()
projectManager = resolve.GetProjectManager()
projectManager.CreateProject("Hello World")
```

The resolve object is the fundamental starting point for scripting via Resolve. As a native object, it can be inspected for further scriptable properties - using table iteration and "getmetatable" in Lua and dir, help etc in Python (among other methods). A notable scriptable object above is fusion - it allows access to all existing Fusion scripting functionality.

Running DaVinci Resolve in headless mode

DaVinci Resolve can be launched in a headless mode without the user interface using the -nogui command line option. When DaVinci Resolve is launched using this option, the user interface is disabled. However, the various scripting APIs will continue to work as expected.

List and Dict Data Structures

Beside primitive data types, Resolve's Python API mainly uses list and dict data structures.

Lists are denoted by [...] and dicts are denoted by { ... } above.

As Lua does not support list and dict data structures, the Lua API implements "list" as a table with indices

- e.g. { [1] = listValue1, [2] = listValue2, ... }.

Similarly the Lua API implements "dict" as a table with the dictionary key as first element

- e.g. { [dictKey1] = dictValue1, [dictKey2] = dictValue2, ... }.

2.8 Basic Resolve API

Some commonly used API functions are described below (*).
As with the resolve object, each object is inspectable for properties and functions.

Important: You must import the necessary modules in your script to get the Resolve app.

```
from python_get_resolve import GetResolve
resolve = GetResolve()
```

2.8.1 Resolve

Fusion()

Description

Returns the Fusion object. Starting point for Fusion scripts.

Type: Fusion

GetMediaStorage()

Description

Returns the media storage object to query and act on media locations.

Type: MediaStorage

```
media_storage = resolve.GetMediaStorage()
```

GetProjectManager()

Description

Returns the project manager object for currently open database.

Type: ProjectManager

```
project_manager = resolve.GetProjectManager()
```

OpenPage(pageName)

Description

Switches to indicated page in DaVinci Resolve. Input can be one of:

- "media"
- "cut"
- "edit"
- "fusion"
- "color"
- "fairlight"
- "deliver"

Type: Bool

```
project_manager = resolve.OpenPage("edit")
```

GetCurrentPage()

Description

Returns the page currently displayed in the main window. Returned value can be one of:

- "media"
- "cut"
- "edit"
- "fusion"
- "color"
- "fairlight"
- "deliver"
- None

Type: String

```
current_page = resolve.GetCurrentPage()
```

GetProductName()

Description

Returns product name.

Type: String

```
product_name = resolve.GetProductName()
```

GetVersion()

Description

Returns list of product version fields in [major, minor, patch, build, suffix] format.

Type: [version fields]

```
version = resolve.GetVersion()
```

GetVersionString()

Description

Returns product version in "major.minor.patch[suffix].build" format.

Type: string

```
version = resolve.GetVersionString()
```

LoadLayoutPreset(presetName)

Description

Loads UI layout from saved preset named 'presetName'.

Type: Bool

```
resolve.LoadLayoutPreset('Custom Preset')
```

UpdateLayoutPreset(presetName)

Description

Overwrites preset named 'presetName' with current UI layout.

Type: Bool

```
resolve.UpdateLayoutPreset('Custom Preset')
```

ExportLayoutPreset(presetName, presetFilePath)

Description

Exports preset named 'presetName' to path 'presetFilePath'.

`presetName must exist to work`

Type: Bool

```
resolve.ExportLayoutPreset('Custom Preset', '/Users/admin/Desktop/Custom.preset')
```

DeleteLayoutPreset(presetName)

Description

Deletes preset named 'presetName'.

Type: Bool

```
resolve.DeleteLayoutPreset('Custom Preset')
```

SaveLayoutPreset(presetName)

Description

Saves current UI layout as a preset named 'presetName'.

Type: Bool

```
resolve.SaveLayoutPreset('Custom Preset')
```

ImportLayoutPreset(presetFilePath, presetName)

Description

Imports preset from path 'presetFilePath'. The optional argument 'presetName' specifies how the preset shall be named. If not specified, the preset is named based on the filename.

Type: Bool

```
resolve.ImportLayoutPreset('/Users/admin/Desktop/Custom.preset', 'Custom Preset')
```

Quit()

Description

Quits the Resolve App.

Type: None

```
resolve.Quit()
```

2.8.2 ProjectManager

Note: projectManager = resolve.GetProjectManager()

ArchiveProject(projectName,filePath,isArchiveSrcMedia=True,isArchiveRenderCache=True,isArchiveProxyMedia=

Description

Archives project to provided file path with the configuration as provided by the optional arguments

Note: Not tested yet

projectManager.ArchiveProject('Project', '/Users/admin/Desktop/Project.drp')

CreateProject(projectName)

Description

Creates and returns a project if projectName (string) is unique, and None if it is not.

Returns Project

```
projectManager.CreateProject('New Project')
```

DeleteProject(projectName)

Description

Delete project in the current folder if not currently loaded

Returns Bool

```
projectManager.DeleteProject('New Project')
```

LoadProject(projectName)

Description

Loads and returns the project with name = projectName (string) if there is a match found, and None if there is no matching Project.

Returns Project

```
projectManager.LoadProject('New Project')
```

GetCurrentProject()

Description

Returns the currently loaded Resolve project.

Returns Project

```
currentProject = projectManager.GetCurrentProject()
```

SaveProject()

Description

Saves the currently loaded project with its own name. Returns True if successful.

Returns Bool

```
projectManager.SaveProject()
```

CloseProject(project)

Description

Closes the specified project without saving.

Returns Bool

```
projectManager.CloseProject('Project01')
```

CreateFolder(folderName)

Description

Creates a folder if folderName (string) is unique.

Returns Bool

```
projectManager.CreateFolder('My Folder')
```

DeleteFolder(folderName)

Description

Deletes the specified folder if it exists. Returns True in case of success.

Returns Bool

```
projectManager.DeleteFolder('My Folder')
```

GetProjectListInCurrentFolder()

Description

Returns a list of project names in current folder.

Returns [project names...]

```
project_list = projectManager.GetProjectListInCurrentFolder()
```

GetFolderListInCurrentFolder()

Description

Returns a list of folder names in current folder.

Returns [folder names...]

```
folder_list = projectManager.GetFolderListInCurrentFolder()
```

GotoRootFolder()

Description

Opens root folder in database.

Returns Bool

```
projectManager.GotoRootFolder()
```

GotoParentFolder()

Description

Opens parent folder of current folder in database if current folder has parent.

Returns Bool

```
projectManager.GotoParentFolder()
```

GetCurrentFolder()

Description

Returns the current folder name.

Returns string

```
current_folder = projectManager.GetCurrentFolder()
```

OpenFolder(folderName)

Description

Opens folder under given name.

Returns Bool

```
projectManager.OpenFolder('My Folder')
```

ImportProject(filePath, projectName=None)

Description

Imports a project from the file path provided. Returns True if successful.

Returns Bool

```
projectManager.ImportProject('/Users/admin/Desktop/project.drp')
```

ExportProject(projectName, filePath, withStillsAndLUTs=True)

Description

Exports project to provided file path, including stills and LUTs if withStillsAndLUTs is True (enabled by default). Returns True in case of success.

Returns Bool

```
projectManager.ExportProject('my_project', '/Users/admin/Desktop/my_project.drp',␣
↪withStillsAndLUTs=True)
```

RestoreProject(filePath, projectName=None)

Description

Restores a project from the file path provided. Returns True if successful.

Returns Bool

```
projectManager.RestoreProject('/Users/admin/Desktop/my_project.drp')
```

GetCurrentDatabase()

Description

Returns a dictionary (with keys 'DbType', 'DbName' and optional 'IpAddress') corresponding to the current database connection

Returns {dbInfo}

```
current_db = projectManager.GetCurrentDatabase()
```

GetDatabaseList()

Description

Returns a list of dictionary items (with keys 'DbType', 'DbName' and optional 'IpAddress') corresponding to all the databases added to Resolve

Returns [{dbInfo}]

```
current_db = projectManager.GetCurrentDatabase()
```

SetCurrentDatabase({dbInfo})

Description

Switches current database connection to the database specified by the keys below, and closes any open project.

- **'DbType':** 'Disk' or 'PostgreSQL' (string)
- **'DbName':** database name (string)
- **'IpAddress':** IP address of the PostgreSQL server (string, optional key - defaults to '127.0.0.1')

Returns Bool

```
projectManager.SetCurrentDatabase({'DbType':'PostgreSQL','DbName':'my_db_name',
↪'IpAddress': '127.0.0.1'})
```

2.8.3 Project

Note: currentProject = projectManager.GetCurrentProject()

GetMediaPool()

Description

Returns the Media Pool object.

Returns MediaPool

```
media_pool = currentProject.GetMediaPool()
```

GetTimelineCount()

Description

Returns the number of timelines currently present in the project.

Returns int

```
timeline_count = currentProject.GetTimelineCount()
```

GetTimelineByIndex(idx)

Description

Returns timeline at the given index, 1 <= idx <= project.GetTimelineCount()

Returns Timeline

```
timeline = currentProject.GetTimelineByIndex(1)
```

GetCurrentTimeline()

Description

Returns the currently loaded timeline.

Returns Timeline

```
current_timeline = currentProject.GetCurrentTimeline()
```

SetCurrentTimeline(timeline)

Description

Sets given timeline as current timeline for the project. Returns True if successful.

Returns Bool

GetGallery()

Description

Returns the Gallery object.

Returns Gallery

```
gallery = currentProject.GetGallery()
```

GetName()

Description

Returns project name.

Returns string

```
project_name = currentProject.GetName()
```

SetName(projectName)

Description

Sets project name if given projectname (string) is unique.

Returns Bool

```
currentProject.SetName('New Name')
```

GetPresetList()

Description

Returns a list of presets and their information.

Returns [presets...]

```
preset_list = currentProject.GetPresetList()
```

SetPreset(presetName)

Description

Sets preset by given presetName (string) into project.

Returns Bool

AddRenderJob()

Description

Adds a render job based on current render settings to the render queue. Returns a unique job id (string) for the new render job.

Returns string

```
render_job_id = currentProject.AddRenderJob()
```

DeleteRenderJob(jobId)

Description

Deletes render job for input job id (string).

Returns Bool

```
render_job_id = currentProject.DeleteRenderJob('9dcfee97-7faf-4026-ac90-1a68480b5ca3')
```

DeleteAllRenderJobs()

Description

Deletes all render jobs in the queue.

Returns Bool

```
currentProject.DeleteAllRenderJobs()
```

GetRenderJobList()

Description

Returns a list of render jobs and their information.

Returns [render jobs...]

```
renderjob_list = currentProject.GetRenderJobList()
```

GetRenderPresetList()

Description

Returns a list of render presets and their information.

Returns [presets...]

```
renderpreset_list = currentProject.GetRenderPresetList()
```

StartRendering(jobId1, jobId2, ...)

Description

Starts rendering jobs indicated by the input job ids.

Returns Bool

```
currentProject.StartRendering('9dcfee97-7faf-4026-ac90-1a68480b5ca3', '4fcfea93-3faf-
↪4023-ac88-2a68480c5dd6')
```

StartRendering([jobIds...], isInteractiveMode=False)

Description

Starts rendering jobs indicated by the input job ids.

Returns Bool

Note: The optional "isInteractiveMode", when set, enables error feedback in the UI during rendering.

```
currentProject.StartRendering(['9dcfee97-7faf-4026-ac90-1a68480b5ca3', '4fcfea93-3faf-
↪4023-ac88-2a68480c5dd6'], isInteractiveMode = False)
```

StartRendering(isInteractiveMode=False)

Description

Starts rendering all queued render jobs.

Returns Bool

Note: The optional "isInteractiveMode", when set, enables error feedback in the UI during rendering.

```
currentProject.StartRendering(isInteractiveMode = True)
```

StopRendering()

Description

Stops any current render processes.

Returns None

```
currentProject.StopRendering()
```

IsRenderingInProgress()

Description

Returns True if rendering is in progress.

Returns Bool

```
if currentProject.IsRenderingInProgress():
    print('Render in progress')
else:
    print('Nothing is rendering')
```

LoadRenderPreset(presetName)

Description

Sets a preset as current preset for rendering if presetName (string) exists.

Returns Bool

```
currentProject.LoadRenderPreset('My preset')
```

SaveAsNewRenderPreset(presetName)

Description

Creates new render preset by given name if presetName(string) is unique.

Returns Bool

```
currentProject.SaveAsNewRenderPreset('My preset')
```

SetRenderSettings({settings})

Description

Sets given settings for rendering. Settings is a dict, with support for the keys:

Returns Bool

Refer to *"Looking up render settings" for more information* section for information for supported settings

```
currentProject.SetRenderSettings({"SelectAllFrames": True, "CustomName": 'My_Movie.mov',
↪"ExportVideo": True, "ExportAudio": True, "FormatWidth": 1920, "FormatHeight": 1080,
↪"FrameRate": 23.976})
```

GetRenderJobStatus(jobId)

Description

Returns a dict with job status and completion percentage of the job by given jobId (string).

Returns {status info}

```
job01_status = currentProject.GetRenderJobStatus('9dcfee97-7faf-4026-ac90-1a68480b5ca3')
```

GetSetting(settingName)

Description

Returns value of project setting (indicated by settingName, string). *Check the Project and Clip properties* section for information

Returns string

```
currentProject.GetSetting(settingName)
```

SetSetting(settingName, settingValue)

Description

Sets the project setting (indicated by settingName, string) to the value (settingValue, string). *Check the Project and Clip properties* section for information

Returns Bool

```
currentProject.SetSetting(settingName, settingValue)
```

GetRenderFormats()

Description

Returns a dict (format -> file extension) of available render formats.

Returns {render formats..}

```
render_formats = currentProject.GetRenderFormats()
'''
{'AVI': 'avi', 'BRAW': 'braw', 'Cineon': 'cin', 'DCP': 'dcp',
 'DPX': 'dpx', 'EXR': 'exr', 'IMF': 'imf', 'JPEG 2000': 'j2c',
  'MJ2': 'mj2', 'MKV': 'mkv', 'MP4': 'mp4', 'MTS': 'mts',
  'MXF OP-Atom': 'mxf', 'MXF OP1A': 'mxf_op1a', 'Panasonic AVC': 'pavc',
   'QuickTime': 'mov', 'TIFF': 'tif', 'Wave': 'wav'}
'''
```

GetRenderCodecs(renderFormat)

Description

Returns a dict (codec description -> codec name) of available codecs for given render format (string).

Returns {render codecs...}

```
render_codecs = currentProject.GetRenderCodecs('MP4')
#{'H.264': 'H264', 'H.265': 'H265'}
```

GetCurrentRenderFormatAndCodec()

Description

Returns a dict with currently selected format 'format' and render codec 'codec'.

Returns {format, codec}

```
render_format_codecs = currentProject.GetCurrentRenderFormatAndCodec()
#{'format': 'mov', 'codec': 'ProRes422'}
```

SetCurrentRenderFormatAndCodec(format, codec)

Description

Sets given render format (string) and render codec (string) as options for rendering.

Returns Bool

```
currentProject.SetCurrentRenderFormatAndCodec('mov', 'ProRes422')
```

GetCurrentRenderMode()

Description

Returns the render mode: 0 - Individual clips, 1 - Single clip.

Returns int

```
currentProject.GetCurrentRenderMode()
```

SetCurrentRenderMode(renderMode)

Description

Sets the render mode. Specify renderMode = 0 for Individual clips, 1 for Single clip.

Returns Bool

```
currentProject.SetCurrentRenderMode(1)
```

GetRenderResolutions(format, codec)

Description

Returns list of resolutions applicable for the given render format (string) and render codec (string).

Returns full list of resolutions if no argument is provided. Each element in the list is a dictionary with 2 keys "Width" and "Height".

Returns [{Resolution}]

```
resolution = currentProject.GetRenderResolutions('mov', 'ProRes422')
'''
[{'Width': 720, 'Height': 480}, {'Width': 720, 'Height': 486}, {'Width': 720, 'Height': 576},
{'Width': 1280, 'Height': 720}, {'Width': 1280, 'Height': 1080}, {'Width': 1920, 'Height': 1080},
```

(continues on next page)

(continued from previous page)

```
{'Width': 3840, 'Height': 2160}, {'Width': 7680, 'Height': 4320}, {'Width': 1828, 'Height': 1332}
↪,
{'Width': 1828, 'Height': 1556}, {'Width': 1998, 'Height': 1080}, {'Width': 2048, 'Height': 858},
{'Width': 2048, 'Height': 1080}, {'Width': 2048, 'Height': 1152}, {'Width': 2048, 'Height': 1556}
↪,
{'Width': 3072, 'Height': 2048}, {'Width': 3654, 'Height': 2664}, {'Width': 3656, 'Height': 3112}
↪,
{'Width': 3996, 'Height': 2160}, {'Width': 4096, 'Height': 1716}, {'Width': 4096, 'Height': 2160}
↪,
{'Width': 4096, 'Height': 3112}]
'''
```

RefreshLUTList()

Description

Refreshes LUT List

Returns Bool

```
currentProject.RefreshLUTList()
```

2.8.4 MediaStorage

Note: media_storage = resolve.GetMediaStorage()

GetMountedVolumeList()

Description

Returns list of folder paths corresponding to mounted volumes displayed in Resolve's Media Storage.

Returns [paths...]

```
mounted_volumes = media_storage.GetMountedVolumeList()
#['/Users/admin/Movies/DaVinci Resolve Studio', '/Volumes/Macintosh HD']
```

GetSubFolderList(folderPath)

Description

Returns list of folder paths in the given absolute folder path.

Returns [paths...]

```
subfolders = media_storage.GetSubFolderList('/Volumes/Macintosh HD')
#['/Volumes/Macintosh HD/Applications', '/Volumes/Macintosh HD/Library', '/Volumes/
↪Macintosh HD/System', '/Volumes/Macintosh HD/Users']
```

GetIsFolderStale()

Description

Returns true if folder is stale in collaboration mode, false otherwise

Returns [paths...]

Note: Not tested yet

media_storage.GetIsFolderStale()

GetFileList(folderPath)

Description

Returns list of media and file listings in the given absolute folder path. Note that media listings may be logically consolidated entries.

Returns [paths...]

```
file_list = media_storage.GetFileList('/Volumes/Macintosh HD/Users/admin/Movies/')
```

RevealInStorage(path)

Description

Expands and displays given file/folder path in Resolve's Media Storage.

Returns Bool

```
media_storage.RevealInStorage('/Volumes/Macintosh HD/Users/admin/Desktop')
```

AddItemListToMediaPool(item1, item2, ...)

Description

Adds specified file/folder paths from Media Storage into current Media Pool folder. Input is one or more file/folder paths. Returns a list of the MediaPoolItems created.

Returns [clips...]

```
clips_added = media_storage.AddItemListToMediaPool('/Volumes/Macintosh HD/Users/admin/
↪Desktop', '/Volumes/Macintosh HD/Users/admin/Movies')
#[<PyRemoteObject object at 0x7fa28007a7f8>, <PyRemoteObject object at 0x7fa28007a9d8>,
↪<PyRemoteObject object at 0x7fa28007ad98>]
```

AddItemListToMediaPool([items...])

Description

Adds specified file/folder paths from Media Storage into current Media Pool folder. Input is an array of file/folder paths. Returns a list of the MediaPoolItems created.

Returns [clips...]

```
clips_added = media_storage.AddItemListToMediaPool(['/Volumes/Macintosh HD/Users/admin/
↪Desktop', '/Volumes/Macintosh HD/Users/admin/Movies'])
#[<PyRemoteObject object at 0x7fa28007a7f8>, <PyRemoteObject object at 0x7fa28007a9d8>,
↪<PyRemoteObject object at 0x7fa28007ad98>]
```

AddClipMattesToMediaPool(MediaPoolItem, [paths], stereoEye)

Description

Adds specified media files as mattes for the specified MediaPoolItem. StereoEye is an optional argument for specifying which eye to add the matte to for stereo clips ("left" or "right"). Returns True if successful.

Returns Bool

Note: Not tested yet

media_storage.AddClipMattesToMediaPool(MediaPoolItem, ['/Volumes/Macintosh HD/Users/admin/Movies/lefteye.mov'], "left") media_storage.AddClipMattesToMediaPool(MediaPoolItem, ['/Volumes/Macintosh HD/Users/admin/Movies/righteye.mov'], "right")

AddTimelineMattesToMediaPool([paths])

Description

Adds specified media files as timeline mattes in current media pool folder. Returns a list of created MediaPoolItems.

Returns [MediaPoolItems]

Note: Not tested yet

media_storage.AddTimelineMattesToMediaPool([''/Volumes/Macintosh HD/Users/admin/Movies/lefteye.mov', '/Volumes/Macintosh HD/Users/admin/Movies/righteye.mov'])

2.8.5 MediaPool

Note: media_pool = currentProject.GetMediaPool()

GetRootFolder()

Description

Returns root Folder of Media Pool

Returns Folder

```
root_folder = media_pool.GetRootFolder()
#Folder (0x0x600017cec380) [App: 'Resolve' on 127.0.0.1, UUID: 3923b295-9579-4657-be42-
↪33b4f4594a93]
```

AddSubFolder(folder, name)

Description

Adds new subfolder under specified Folder object with the given name.

Returns Folder

```
root_folder = media_pool.GetRootFolder()
media_pool.AddSubFolder(root_folder, 'New_Sub01')
#New subfolder 'New_Sub01' at Root level in the MediaPool
```

RefreshFolders()

Description

Updates the folders in collaboration mode

Note: Not tested yet

root_folder = media_pool.GetRootFolder() media_pool.RefreshFolders()

CreateEmptyTimeline(name)

Description

Adds new timeline with given name.

Returns Timeline

```
media_pool.CreateEmptyTimeline('New Timeline')
```

AppendToTimeline(clip1, clip2, ...)

Description

Appends specified MediaPoolItem objects in the current timeline. Returns the list of appended timelineItems.

Returns [TimelineItem]

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
media_pool.AppendToTimeline(clip_list[0], clip_list[1])
#append the first 2 clips from root folder to the current timeline
```

AppendToTimeline([clips])

Description

Appends specified MediaPoolItem objects in the current timeline. Returns the list of appended timelineItems.

Returns [TimelineItem]

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
```

(continues on next page)

(continued from previous page)

```
media_pool.AppendToTimeline([clip_list[0], clip_list[1]])
#append the first 2 clips from root folder to the current timeline
```

AppendToTimeline([{clipInfo}, ...])

Description

Appends list of clipInfos specified as dict:

- "mediaPoolItem"
- "startFrame" (int)
- "endFrame" (int)
- (optional) "mediaType" (int; 1 - Video only, 2 - Audio only)

Returns the list of appended timelineItems.

Returns [TimelineItem]

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
media_pool.AppendToTimeline([{"mediaPoolItem":clip_list[0], "startFrame": 0, "endFrame":␣
↪10, "mediaType": 2}])
#appends the first clip from root folder, frames 0 to 10, only audio.
```

CreateTimelineFromClips(name, clip1, clip2,...)

Description

Creates new timeline with specified name, and appends the specified MediaPoolItem objects.

Returns Timeline

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
media_pool.CreateTimelineFromClips('My Amazing Timeline', clip_list[0], clip_list[1])
#create a timeline named 'My Amazing Timeline' with first 2 clips from root folder
```

CreateTimelineFromClips(name, [clips])

Description

Creates new timeline with specified name, and appends the specified MediaPoolItem objects.

Returns Timeline

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
media_pool.CreateTimelineFromClips('My Amazing Timeline', [clip_list[0], clip_list[1]])
#create a timeline named 'My Amazing Timeline' with first 2 clips from root folder
```

CreateTimelineFromClips(name, [{clipInfo}])

Description

Creates new timeline with specified name, appending the list of clipInfos specified as a dict of "mediaPoolItem", "startFrame" (int), "endFrame" (int).

Returns Timeline

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
media_pool.CreateTimelineFromClips('My Amazing Timeline', [{"mediaPoolItem":clip_list[0],
↪ "startFrame": 0, "endFrame": 10}, {"mediaPoolItem":clip_list[1], "startFrame": 0,
↪"endFrame": 10}])
#create a timeline named 'My Amazing Timeline' with first 2 clips, first 10 frames, from␣
↪root folder
```

ImportTimelineFromFile(filePath, {importOptions})

Description

Creates timeline based on parameters within given file and optional importOptions dict, with support for the keys:

- "timelineName": string, specifies the name of the timeline to be created
- "importSourceClips": Bool, specifies whether source clips should be imported, True by default
- "sourceClipsPath": string, specifies a filesystem path to search for source clips if the media is inaccessible in their original path and if "importSourceClips" is True
- "sourceClipsFolders": List of Media Pool folder objects to search for source clips if the media is not present in current folder and if "importSourceClips" is False
- "interlaceProcessing": Bool, specifies whether to enable interlace processing on the imported timeline being created. valid only for AAF import

Returns Timeline

```
root_folder = media_pool.GetRootFolder()
media_pool.ImportTimelineFromFile('/Users/admin/Desktop/exported_timeline.aaf', {
↪"timelineName": 'NewImport Timeline'})
```

DeleteTimelines([timeline])

Description

Deletes specified timelines in the media pool.

Returns Bool

```
first_timeline =  currentProject.GetTimelineByIndex(1)
second_timeline = currentProject.GetTimelineByIndex(2)

media_pool.DeleteTimelines([first_timeline, second_timeline])
```

GetCurrentFolder()

Description

Returns currently selected Folder.

Returns Folder

```
curent_folder = media_pool.GetCurrentFolder()
#Folder (0x0x600017cec380) [App: 'Resolve' on 127.0.0.1, UUID: 3723b295-9579-4657-be42-
↪33b4f4594a93]
```

SetCurrentFolder(Folder)

Description

Sets current folder by given Folder.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
folder_list = root_folder.GetSubFolderList()

media_pool.SetCurrentFolder(folder_list[0])
```

DeleteClips([clips])

Description

Deletes specified clips or timeline mattes in the media pool

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
media_pool.DeleteClips([clip_list[0], clip_list[1]])
#deletes the first 2 clips in the root folder
```

DeleteFolders([subfolders])

Description

Deletes specified subfolders in the media pool

Returns Bool

```
root_folder = media_pool.GetRootFolder()
folder_list = root_folder.GetSubFolderList()
media_pool.DeleteFolders([folder_list[0], folder_list[1]])
#deletes the first 2 subfolders of the root folder
```

MoveClips([clips], targetFolder)

Description

Moves specified clips to target folder.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
folder_list = root_folder.GetSubFolderList()
clip_list = root_folder.GetClipList()
media_pool.MoveClips([clip_list[0], clip_list[1]], folder_list[1])
#moves the first 2 clips from Root folder to second subfolder
```

MoveFolders([folders], targetFolder)

Description

Moves specified folders to target folder.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
folder_list = root_folder.GetSubFolderList()
subfolder02_content = folder_list[1].GetSubFolderList()
media_pool.MoveFolders([subfolder02_content[0], subfolder02_content[1]], root_folder)
#moves the first 2 folders of second's root subfolder to the root folder
```

GetClipMatteList(MediaPoolItem)

Description

Get mattes for specified MediaPoolItem, as a list of paths to the matte files.

Returns [paths]

```
root_folder = media_pool.GetRootFolder()
folder_list = root_folder.GetSubFolderList()
clip_list = root_folder.GetClipList()

matte_list = media_pool.GetClipMatteList(clip_list[0])
```

GetTimelineMatteList(Folder)

Description

Get mattes in specified Folder, as list of MediaPoolItems.

Returns [MediaPoolItems]

```
root_folder = media_pool.GetRootFolder()
matte_list_in_root = media_pool.GetTimelineMatteList(root_folder)
```

DeleteClipMattes(MediaPoolItem, [paths])

Description

Delete mattes based on their file paths, for specified MediaPoolItem. Returns True on success.

Returns Bool

Note: Not tested yet

root_folder = media_pool.GetRootFolder() first_timeline = currentProject.GetTimelineByIndex(1)

media_pool.DeleteClipMattes(first_timeline, ['/Users/admin/Desktop/matte.mov'])

RelinkClips([MediaPoolItem], folderPath)

Description

Update the folder location of specified media pool clips with the specified folder path.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()

media_pool.RelinkClips([clip_list[0], clip_list[1]], '/Users/admin/Movies/')
#relink first 2 clips of root mediaPool to the specified folder path
```

UnlinkClips([MediaPoolItem])

Description

Unlink specified media pool clips.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()

media_pool.UnlinkClips([clip_list[0]])
#unlink first clip of root mediaPool
```

ImportMedia([items...])

Description

Imports specified file/folder paths into current Media Pool folder. Input is an array of file/folder paths. Returns a list of the MediaPoolItems created.

Returns [MediaPoolItems]

```
media_pool.ImportMedia(['/Users/admin/Movies/sample1.mov', '/Users/admin/Movies/sample2.
↪mov'])
#import clips in current media pool folder
```

ImportMedia([{clipInfo}])

Description

Imports file path(s) into current Media Pool folder as specified in list of clipInfo dict. Returns a list of the MediaPoolItems created. Each clipInfo gets imported as one MediaPoolItem unless 'Show Individual Frames' is turned on.

Example: ImportMedia([{"FilePath":"file_%03d.dpx", "StartIndex":1, "EndIndex":100}]) would import clip "file_[001-100].dpx".

Returns [MediaPoolItems]

```
dpx_sequence1 = {"FilePath":"file_%03d.dpx", "StartIndex":1, "EndIndex":100}
media_pool.ImportMedia([dpx_sequence1])
#import dpx sequence file_[001-100].dpx in current media pool folder
```

ExportMetadata(fileName, [clips])

Description

Exports metadata of specified clips to 'fileName' in CSV format. If no clips are specified, all clips from media pool will be used.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
media_pool.ExportMetadata('/Users/admin/Desktop/metadata.csv', [clip_list[0], clip_
↪list[1]])
#export CSV file on Desktop containing first 2 clips metadata
```

2.8.6 Folder

Note: curent_folder = media_pool.GetCurrentFolder()

GetClipList()

Description

Returns a list of clips (items) within the folder.

Returns [clips...]

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
#[<PyRemoteObject object at 0x7fe25807a930>, <PyRemoteObject object at 0x7fe25807a948>,
↪<PyRemoteObject object at 0x7fe25807a960>]
```

GetName()

Description

Returns the media folder name.

Returns string

```
root_folder = media_pool.GetRootFolder()
root_folder.GetName()
#Master
```

GetSubFolderList()

Description

Returns a list of subfolders in the folder.

Returns [folders...]

```
root_folder = media_pool.GetRootFolder()
root_subfolders = root_folder.GetSubFolderList()
#[<PyRemoteObject object at 0x7fef3007a828>, <PyRemoteObject object at 0x7fef3007a840>,
↪<PyRemoteObject object at 0x7fef3007a858>, <PyRemoteObject object at 0x7fef3007a870>]
```

2.8.7 MediaPoolItem

GetName()

Description

Returns the clip name.

Returns string

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
first_clip_name = clip_list[0].GetName()
```

GetMetadata(metadataType=None)

Description

Returns the metadata value for the key 'metadataType'. If no argument is specified, a dict of all set metadata properties is returned.

Returns string|dict

Note: Not tested yet

root_folder = media_pool.GetRootFolder() clip_list = root_folder.GetClipList() first_clip_name = clip_list[0].GetMetadata()

SetMetadata(metadataType, metadataValue)

Description

Sets the given metadata to metadataValue (string). Returns True if successful.

Returns Bool

SetMetadata({metadata})

Description

Sets the item metadata with specified 'metadata' dict. Returns True if successful.

Returns Bool

GetMediaId()

Description

Returns the unique ID for the MediaPoolItem.

Returns string

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
first_clip_id = clip_list[0].GetMediaId()
#c0dc4522-ef60-4e3d-9adb-352eb868aaac
```

AddMarker(frameId, color, name, note, duration,customData)

Description

Creates a new marker at given frameId position and with given marker information. 'customData' is optional and helps to attach user specific data to the marker.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].AddMarker(20.0, "Green", "Marker Name", "Custom Notes", 10, 'secret_word')
#adds marker to the first clip of the mediapool
```

GetMarkers()

Description

Returns a dict (frameId -> {information}) of all markers and dicts with their information.

Example of output format: {96.0: {'color': 'Green', 'duration': 1.0, 'note': '', 'name': 'Marker 1', 'customData': ''}, ...} In the above example - there is one 'Green' marker at offset 96 (position of the marker)

Returns {markers...}

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
first_clip_markers = clip_list[0].GetMarkers()
#{20: {'color': 'Green', 'duration': 10, 'note': 'Custom Notes', 'name': 'Marker Name', 'customData
↪': 'secret_word'}}
```

GetMarkerByCustomData(customData)

Description

Returns marker {information} for the first matching marker with specified customData.

Returns {markers...}

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
first_clip_marker = clip_list[0].GetMarkerByCustomData('secret_word')
#{20: {'color': 'Green', 'duration': 10, 'note': 'Custom Notes', 'name': 'Marker Name', 'customData
↪': 'secret_word'}}
```

UpdateMarkerCustomData(frameId, customData)

Description

Updates customData (string) for the marker at given frameId position.

CustomData is not exposed via UI and is useful for scripting developer to attach any user specific data to markers.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].UpdateMarkerCustomData(20, 'New CustomData')
#{20: {'color': 'Green', 'duration': 10, 'note': 'Custom Notes', 'name': 'Marker Name', 'customData
↪': 'New CustomData'}}
```

GetMarkerCustomData(frameId)

Description

Returns customData string for the marker at given frameId position.

Returns string

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].GetMarkerCustomData(20)
#New CustomData
```

DeleteMarkersByColor(color)

Description

Delete all markers of the specified color from the media pool item. “All” as argument deletes all color markers.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].DeleteMarkersByColor('Green')
```

DeleteMarkerAtFrame(frameNum)

Description

Delete marker at frame number from the media pool item.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].DeleteMarkerAtFrame(20)
```

DeleteMarkerByCustomData(customData)

Description

Delete first matching marker with specified customData.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].DeleteMarkerByCustomData('New CustomData')
```

AddFlag(color)

Description

Adds a flag with given color (string).

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].AddFlag('Red')
```

GetFlagList()

Description

Returns a list of flag colors assigned to the item.

Returns [colors...]

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
first_clip_flags = clip_list[0].GetFlagList()
#['Red', 'Blue']
```

ClearFlags(color)

Description

Clears the flag of the given color if one exists. An "All" argument is supported and clears all flags.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].ClearFlags('All')
```

GetClipColor()

Description

Returns the item color as a string.

Returns string

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
first_clip_color = clip_list[0].GetClipColor()
```

SetClipColor(colorName)

Description

Sets the item color based on the colorName (string).

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].SetClipColor('Blue')
```

ClearClipColor()

Description

Clears the item color.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].ClearClipColor()
```

GetClipProperty(propertyName=None)

Description

Returns the property value for the key 'propertyName'.

If no argument is specified, a dict of all clip properties is returned. Check the section below for more information.

Returns string|dict

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
first_clip_filename = clip_list[0].GetClipProperty('File Name')
#sample.mov
first_clip_properties = clip_list[0].GetClipProperty(propertyName=None)
"""{'Alpha mode': 'None', 'Angle': '', 'Audio Bit Depth': '16', 'Audio Ch': '6', 'Audio Codec':
↪'Linear PCM',
'Audio Offset': '', 'Bit Depth': '10', 'Camera #': '', 'Clip Color': 'Blue', 'Clip Name': 'HD_24_
↪mono.mov',
'Comments': '', 'Data Level': 'Auto', 'Date Added': 'Sun Jan 30 2022 13:29:53', 'Date Created':
↪'Fri Mar 18 2016 16:47:44',
'Date Modified': 'Fri Mar 18 16:47:44 2016', 'Description': '', 'Drop frame': '0', 'Duration':
↪'00:00:30:00',
'Enable Deinterlacing': '0', 'End': '719', 'End TC': '01:00:30:00', 'FPS': 24.0, 'Field Dominance
↪': 'Auto',
```

(continues on next page)

(continued from previous page)

```
'File Name': 'sample.mov', 'File Path': '/Users/admin/Movies/sample.mov', 'Flags': '', 'Format':
↪'QuickTime',
'Frames': '720', 'Good Take': '', 'H-FLIP': 'Off', 'IDT': '', 'In': '', 'Input Color Space': 'Rec.709␣
↪(Scene)',
'Input LUT': '', 'Input Sizing Preset': 'None', 'Keyword': '', 'Noise Reduction': '', 'Offline␣
↪Reference': '',
'Out': '', 'PAR': 'Square', 'Proxy': 'None', 'Proxy Media Path': '', 'Reel Name': '', 'Resolution':
↪'1920x1080',
'Roll/Card': '', 'S3D Sync': '', 'Sample Rate': '48000', 'Scene': '', 'Sharpness': '', 'Shot': '',
↪'Slate TC': '01:00:00:00',
'Start': '0', 'Start KeyKode': '', 'Start TC': '01:00:00:00', 'Synced Audio': '', 'Take': '', 'Type':
↪'Video + Audio',
'Usage': '0', 'V-FLIP': 'Off', 'Video Codec': 'Apple ProRes 422', 'Super Scale': 1}"""
```

SetClipProperty(propertyName, propertyValue)

Description

Sets the given property to propertyValue (string).

Refer to section *"Looking up Project and Clip properties"* for information.

Returns Bool

Note: Not tested yet

root_folder = media_pool.GetRootFolder() clip_list = root_folder.GetClipList() clip_list[0].SetClipProperty("superScale", 3)

LinkProxyMedia(proxyMediaFilePath)

Description

Links proxy media located at path specified by arg 'proxyMediaFilePath' with the current clip. 'proxyMediaFilePath' should be absolute clip path.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].LinkProxyMedia('/Users/admin/Desktop/proxy.mov')
```

UnlinkProxyMedia()

Description

Unlinks any proxy media associated with clip.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].UnlinkProxyMedia()
```

ReplaceClip(filePath)

Description

Replaces the underlying asset and metadata of MediaPoolItem with the specified absolute clip path.

Returns Bool

```
root_folder = media_pool.GetRootFolder()
clip_list = root_folder.GetClipList()
clip_list[0].ReplaceClip('/Users/admin/Movies/New_Media.mov')
```

2.8.8 Timeline

Note: current_timeline = currentProject.GetCurrentTimeline()

GetName()

Description

Returns the timeline name.

Returns string

```
timeline_name = current_timeline.GetName()
```

SetName(timelineName)

Description

Sets the timeline name if timelineName (string) is unique. Returns True if successful.

Returns Bool

```
current_timeline.SetName('Better Timeline name')
```

GetStartFrame()

Description

Returns the frame number at the start of timeline.

Returns int

```
start_frame = current_timeline.GetStartFrame()
#86400 for a timeline starting at 01:00:00:00
```

GetEndFrame()

Description

Returns the frame number at the end of timeline.

Returns int

```
end_frame = current_timeline.GetEndFrame()
```

SetStartTimecode(timecode)

Description

Set the start timecode of the timeline to the string 'timecode'. Returns true when the change is successful, false otherwise.

Returns bool

Note: Not tested yet

current_timeline.SetStartTimecode('09:58:30:00')

GetStartTimecode()

Description

Returns the start timecode for the timeline.

Returns String

Note: Not tested yet

current_timecode = current_timeline.GetStartTimecode()

GetTrackCount(trackType)

Description

Returns the number of tracks for the given track type (“audio”, “video” or “subtitle”).

Returns int

```
video_track_count = current_timeline.GetTrackCount("video")
```

GetItemListInTrack(trackType, index)

Description

Returns a list of timeline items on that track (based on trackType and index). 1 <= index <= GetTrackCount(trackType).

Returns [items...]

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
#returns a list of video items from the current video track V1
#[<PyRemoteObject object at 0x7fdbb807a978>, <PyRemoteObject object at 0x7fdbb807a990>]
```

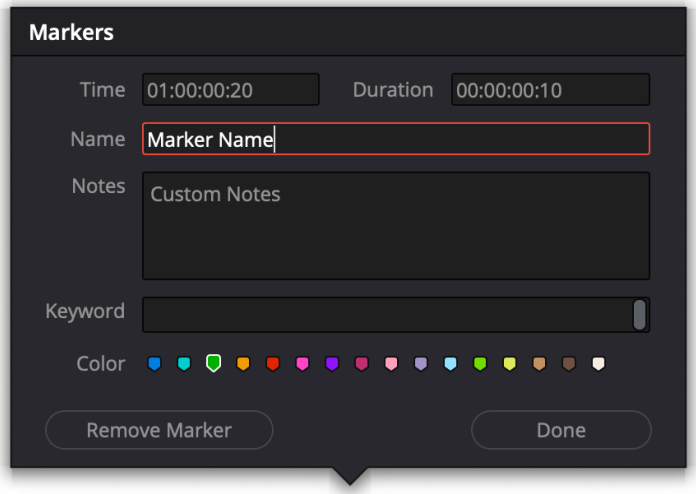
AddMarker(frameId, color, name, note, duration, customData)

Description

Creates a new marker at given frameId position and with given marker information. ‘customData’ is optional and helps to attach user specific data to the marker.

Returns Bool

```
current_timeline.AddMarker(20.0, "Green", "Marker Name", "Custom Notes", 10, 'secret_word
↪')
```



GetMarkers()

Description

Returns a dict (frameId -> {information}) of all markers and dicts with their information.

Example: a value of {96.0: {'color': 'Green', 'duration': 1.0, 'note': '', 'name': 'Marker 1', 'customData': ''}, ...} indicates a single green marker at timeline offset 96

Returns {markers...}

```
markers = current_timeline.GetMarkers()
#{20: {'color': 'Green', 'duration': 10, 'note': 'Custom Notes', 'name': 'Marker Name', 'customData
↪': 'secret_word'}}
```

GetMarkerByCustomData(customData)

Description

Returns marker {information} for the first matching marker with specified customData.

Returns {markers...}

```
marker = current_timeline.GetMarkerByCustomData('secret_word')
#{20: {'color': 'Green', 'duration': 10, 'note': 'Custom Notes', 'name': 'Marker Name', 'customData
↪': 'secret_word'}}
```

UpdateMarkerCustomData(frameId, customData)

Description

Updates customData (string) for the marker at given frameId position. CustomData is not exposed via UI and is useful for scripting developer to attach any user specific data to markers.

Returns Bool

```
current_timeline.UpdateMarkerCustomData(20, 'New CustomData')
#{20: {'color': 'Green', 'duration': 10, 'note': 'Custom Notes', 'name': 'Marker Name', 'customData
↪': 'New CustomData'}}
```

GetMarkerCustomData(frameId)

Description

Returns customData string for the marker at given frameId position.

Returns string

```
current_timeline.GetMarkerCustomData(20)
#New CustomData
```

DeleteMarkersByColor(color)

Description

Deletes all timeline markers of the specified color. An "All" argument is supported and deletes all timeline markers.

Returns Bool

```
current_timeline.DeleteMarkersByColor('Green')
```

DeleteMarkerAtFrame(frameNum)

Description

Deletes the timeline marker at the given frame number.

Returns Bool

```
current_timeline.DeleteMarkerAtFrame(20)
```

DeleteMarkerByCustomData(customData)

Description

Delete first matching marker with specified customData.

Returns Bool

```
current_timeline.DeleteMarkerByCustomData('New CustomData')
```

ApplyGradeFromDRX(path, gradeMode, item1, item2, ...)

Description

Loads a still from given file path (string) and applies grade to Timeline Items with gradeMode (int): 0 - "No keyframes", 1 - "Source Timecode aligned", 2 - "Start Frames aligned".

Returns Bool

Note: Not tested yet

#

ApplyGradeFromDRX(path, gradeMode, [items])

Description

Loads a still from given file path (string) and applies grade to Timeline Items with gradeMode (int): 0 - "No keyframes", 1 - "Source Timecode aligned", 2 - "Start Frames aligned".

Returns Bool

Note: Not tested yet

#

GetCurrentTimecode()

Description

Returns a string timecode representation for the current playhead position, while on Cut, Edit, Color, Fairlight and Deliver pages.

Returns string

```
current_timecode = current_timeline.GetCurrentTimecode()
#01:00:00:00
```

SetCurrentTimecode(timecode)

Description

Sets current playhead position from input timecode for Cut, Edit, Color, Fairlight and Deliver pages.

Returns Bool

```
current_timeline.SetCurrentTimecode('01:00:10:00')
```

GetCurrentVideoItem()

Description

Returns the current video timeline item.

Returns item

```
current_video_item = current_timeline.GetCurrentVideoItem()
#Timeline item (0x0x600017de9320) [App: 'Resolve' on 127.0.0.1, UUID: 3723b295-9579-4657-
↪be42-33b4f4594a93]
```

GetCurrentClipThumbnailImage()

Description

Returns a dict (keys “width”, “height”, “format” and “data”) with data containing raw thumbnail image data (RGB 8-bit image data encoded in base64 format) for current media in the Color Page. An example of how to retrieve and interpret thumbnails is provided in 6_get_current_media_thumbnail.py in the Examples folder.

`CurrentPage must be the Color Page`

Returns {thumbnailData}

```
current_thumbnail = current_timeline.GetCurrentClipThumbnailImage()
thumbnail_format = current_thumbnail['format']
#RGB 8 bit
```

GetTrackName(trackType, trackIndex)

Description

Returns the track name for track indicated by trackType (“audio”, “video” or “subtitle”) and index. 1 <= trackIndex <= GetTrackCount(trackType).

Returns string

```
V1_track_name = current_timeline.GetTrackName("video", 1)
#Video 1
```

SetTrackName(trackType, trackIndex, name)

Description

Sets the track name (string) for track indicated by trackType (“audio”, “video” or “subtitle”) and index. 1 <= trackIndex <= GetTrackCount(trackType).

Returns Bool

```
current_timeline.SetTrackName("video", 1, 'Best track V1')
```

DuplicateTimeline(timelineName)

Description

Duplicates the timeline and returns the created timeline, with the (optional) timelineName, on success.

Returns timeline

```
current_timeline.DuplicateTimeline('New Timeline Copy')
```

CreateCompoundClip([timelineItems], {clipInfo})

Description

Creates a compound clip of input timeline items with an optional clipInfo map: {"startTimecode" : "00:00:00:00", "name" : "Compound Clip 1"}. It returns the created timeline item.

Returns timelineItem

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
clip_info = {"startTimecode" : "00:00:00:00", "name" : "Compound Clip 1"}

current_timeline.CreateCompoundClip([timeline_items[0], timeline_items[1]], clip_info)
#new compound clip from first 2 items in current timeline
```

CreateFusionClip([timelineItems])

Description

Creates a Fusion clip of input timeline items. It returns the created timeline item.

Returns timelineItem

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)

current_timeline.CreateFusionClip([timeline_items[0], timeline_items[1]])
#new fusion clip from first 2 items in current timeline
```

ImportIntoTimeline(filePath, {importOptions})

Description

Imports timeline items from an AAF file and optional importOptions dict into the timeline, with support for the keys:

- "autoImportSourceClipsIntoMediaPool": Bool, specifies if source clips should be imported into media pool, True by default
- "ignoreFileExtensionsWhenMatching": Bool, specifies if file extensions should be ignored when matching, False by default
- "linkToSourceCameraFiles": Bool, specifies if link to source camera files should be enabled, False by default
- "useSizingInfo": Bool, specifies if sizing information should be used, False by default
- "importMultiChannelAudioTracksAsLinkedGroups": Bool, specifies if multi-channel audio tracks should be imported as linked groups, False by default
- "insertAdditionalTracks": Bool, specifies if additional tracks should be inserted, True by default
- "insertWithOffset": string, specifies insert with offset value in timecode format - defaults to "00:00:00:00", applicable if "insertAdditionalTracks" is False

- "sourceClipsPath": string, specifies a filesystem path to search for source clips if the media is inaccessible in their original path and if "ignoreFileExtensionsWhenMatching" is True
- "sourceClipsFolders": string, list of Media Pool folder objects to search for source clips if the media is not present in current folder

Returns Bool

```
aaf_file = '/Users/admin/Desktop/exported_timeline.aaf'
import_options = {"autoImportSourceClipsIntoMediaPool": True,
↪"ignoreFileExtensionsWhenMatching": True}
current_timeline.ImportIntoTimeline(aaf_file, import_options)
```

Export(fileName, exportType, exportSubtype)

Description

Exports timeline to 'fileName' as per input exportType & exportSubtype format.

Returns Bool

Refer to section *"Looking up timeline exports properties"* for information on the parameters.

```
current_timeline.Export('/Users/admin/Desktop/exported_timeline.aaf', resolve.EXPORT_AAF,
↪ resolve.EXPORT_AAF_NEW)
```

GetSetting(settingName)

Description

Returns value of timeline setting (indicated by settingName : string).

Refer to section *"Looking up Project and Clip properties"* for information.

Returns string

```
timeline_settings = current_timeline.GetSetting()
timeline_playback_fps = current_timeline.GetSetting('timelinePlaybackFrameRate')
#23.976
```

SetSetting(settingName, settingValue)

Description

Sets timeline setting (indicated by settingName : string) to the value (settingValue : string).

Refer to section *"Looking up Project and Clip properties"* for information.

Returns Bool

```
success = current_timeline.SetSetting('timelineOutputResolutionWidth', '1920')
#True if the Settings was modified properly
#WARNING: The 'Use Project Settings' option might prevent some parameters change.
```

InsertGeneratorIntoTimeline(generatorName)

Description

Inserts a generator (indicated by generatorName : string) into the timeline.

Generator List:

- '10 Step'
- '100mV Steps'
- 'BT.2111 Color Bar HLG Narrow'
- 'BT.2111 Color Bar PQ Full'
- 'BT.2111 Color Bar PQ Narrow'
- 'EBU Color Bar'
- 'Four Color Gradient'
- 'Grey Scale'
- 'SMPTE Color Bar'
- 'Solid Color'
- 'Window'
- 'YCbCr Ramp'

Returns TimelineItem

```
current_timeline.InsertGeneratorIntoTimeline('SMPTE Color Bar')
```

InsertFusionGeneratorIntoTimeline(generatorName)

Description

Inserts a Fusion generator (indicated by generatorName : string) into the timeline.

Generator List:

- 'Contours'
- 'Noise Gradient'
- 'Paper'
- 'Texture Background'

Returns TimelineItem

Note: Not tested yet

current_timeline.InsertFusionGeneratorIntoTimeline('Noise Gradient')

InsertFusionCompositionIntoTimeline()

Description

Inserts a Fusion composition into the timeline.

Returns TimelineItem

Note: Not tested yet

current_timeline.InsertFusionCompositionIntoTimeline()

InsertOFXGeneratorIntoTimeline(generatorName)

Description

Inserts an OFX generator (indicated by generatorName : string) into the timeline.

Returns TimelineItem

Note: Not tested yet

current_timeline.InsertOFXGeneratorIntoTimeline('OFX Generator')

InsertTitleIntoTimeline(titleName)

Description

Inserts a title (indicated by titleName : string) into the timeline.

Title List:

- 'Left Lower Third'
- 'Middle Lower Third'
- 'Right Lower Third'
- 'Scroll'
- 'Text'

Returns TimelineItem

```
current_timeline.InsertTitleIntoTimeline('Text')
```

InsertFusionTitleIntoTimeline(titleName)

Description

Inserts a Fusion title (indicated by titleName : string) into the timeline.

Fusion Title List:

- 'Background Reveal'
- 'Background Reveal Lower Third'
- 'Call Out'
- ...

Returns TimelineItem

Note: Not tested yet

current_timeline.InsertFusionTitleIntoTimeline('Background Reveal')

GrabStill()

Description

Grabs still from the current video clip. Returns a GalleryStill object.

Returns galleryStill

```
current_timeline.GrabStill()
```

GrabAllStills(stillFrameSource)

Description

Grabs stills from all the clips of the timeline at 'stillFrameSource' (1 - First frame, 2 - Middle frame). Returns the list of GalleryStill objects.

Returns [galleryStill]

```
current_timeline.GrabAllStills(2)
#grab a still of each clips from middle
```

2.8.9 TimelineItem

GetName()

Description

Returns the item name.

Returns string

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
first_timeline_item_name = timeline_items[0].GetName()
```

GetDuration()

Description

Returns the item duration.

Returns int

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
first_timeline_item_duration = timeline_items[0].GetDuration()
```

GetEnd()

Description

Returns the end frame position on the timeline.

Returns int

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
first_timeline_item_end = timeline_items[0].GetEnd()
```

GetFusionCompCount()

Description

Returns number of Fusion compositions associated with the timeline item.

Returns int

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
timeline_items[0].GetFusionCompCount()
```

GetFusionCompByIndex(compIndex)

Description

Returns the Fusion composition object based on given index. 1 <= compIndex <= timelineItem.GetFusionCompCount()

Returns fusionComp

GetFusionCompNameList()

Description

Returns a list of Fusion composition names associated with the timeline item.

Returns [names...]

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
timeline_items[0].GetFusionCompNameList()
```

GetFusionCompByName(compName)

Description

Returns the Fusion composition object based on given name.

Returns fusionComp

Note: Not tested yet

timeline_items = current_timeline.GetItemListInTrack("video", 1) timeline_items[0].GetFusionCompByName()

GetLeftOffset()

Description

Returns the maximum extension by frame for clip from left side.

Returns int

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
first_clip_left_offset = timeline_items[0].GetLeftOffset()
```

GetRightOffset()

Description

Returns the maximum extension by frame for clip from right side.

Returns int

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
first_clip_right_offset = timeline_items[0].GetRightOffset()
```

GetStart()

Description

Returns the start frame position on the timeline.

Returns int

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
first_clip_start_frame = timeline_items[0].GetStart()
#86400 if timeline starts at 01:00:00:00
```

SetProperty(propertyKey, propertyValue)

Description

Sets the value of property "propertyKey" to value "propertyValue"

Refer to *"Looking up Timeline item properties" for more information*

Returns Bool

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
timeline_items[0].SetProperty('ZoomX', 2.0)
```

GetProperty(propertyKey)

Description

returns the value of the specified key if no key is specified, the method returns a dictionary(python) or table(lua) for all supported keys

Returns int/[key:value]

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
first_clip_pan = timeline_items[0].GetProperty('Pan')
#0.0
first_clip_properties = timeline_items[0].GetProperty()
{'Pan': 0.0, 'Tilt': 0.0, 'ZoomX': 1.0, 'ZoomY': 1.0, 'ZoomGang': True, 'RotationAngle':␣
↪0.0, 'AnchorPointX': 0.0,
'AnchorPointY': 0.0, 'Pitch': 0.0, 'Yaw': 0.0, 'FlipX': False, 'FlipY': False, 'CropLeft
↪': 0.0, 'CropRight': 0.0,
'CropTop': 0.0, 'CropBottom': 0.0, 'CropSoftness': 0.0, 'CropRetain': False,
↪'DynamicZoomEase': 0, 'CompositeMode': 0,
'Opacity': 100.0, 'Distortion': 0.0, 'RetimeProcess': 0, 'MotionEstimation': 0, 'Scaling
↪': 0, 'ResizeFilter': 0}
```

AddMarker(frameId, color, name, note, duration,customData)

Description

Creates a new marker at given frameId position and with given marker information. 'customData' is optional and helps to attach user specific data to the marker.

Returns Bool

Note: Not tested yet

timeline_items = current_timeline.GetItemListInTrack("video", 1) success = timeline_items[0].AddMarker(20, "Green", "Marker Name", "Custom Notes", 10, 'secret_word') #adds marker to the first clip of the timeline

GetMarkers()

Description

Returns a dict (frameId -> {information}) of all markers and dicts with their information.

Returns {markers...}

Example: a value of {96.0: {'color': 'Green', 'duration': 1.0, 'note': '', 'name': 'Marker 1', 'customData': ''}, ...} indicates a single green marker at clip offset 96

Note: Not tested yet

timeline_items = current_timeline.GetItemListInTrack("video", 1) markers = timeline_items[0].GetMarkers()

GetMarkerByCustomData(customData)

Description

Returns marker {information} for the first matching marker with specified customData.

Returns {markers...}

Note: Not tested yet

timeline_items = current_timeline.GetItemListInTrack("video", 1) marker = timeline_items[0].GetMarkerByCustomData('secret_word')

UpdateMarkerCustomData(frameId, customData)

Description

Updates customData (string) for the marker at given frameId position. CustomData is not exposed via UI and is useful for scripting developer to attach any user specific data to markers.

Returns Bool

Note: Not tested yet

timeline_items = current_timeline.GetItemListInTrack("video", 1) marker = timeline_items[0].UpdateMarkerCustomData(20.0, 'new custom data')

GetMarkerCustomData(frameId)

Description

Returns customData string for the marker at given frameId position.

Returns string

Note: Not tested yet

timeline_items = current_timeline.GetItemListInTrack("video", 1) marker = timeline_items[0].GetMarkerCustomData(20.0)

DeleteMarkersByColor(color)

Description

Delete all markers of the specified color from the timeline item. "All" as argument deletes all color markers.

Returns Bool

Note: Not tested yet

timeline_items = current_timeline.GetItemListInTrack("video", 1) timeline_items[0].DeleteMarkersByColor('Red')

DeleteMarkerAtFrame(frameNum)

Description

Delete marker at frame number from the timeline item.

Returns Bool

Note: Not tested yet

timeline_items = current_timeline.GetItemListInTrack("video", 1) timeline_items[0].DeleteMarkerAtFrame(20)

DeleteMarkerByCustomData(customData)

Description

Delete first matching marker with specified customData.

Returns Bool

Note: Not tested yet

timeline_items = current_timeline.GetItemListInTrack("video", 1) timeline_items[0].DeleteMarkerByCustomData('new custom data')

AddFlag(color)

Description

Adds a flag with given color (string).

Returns Bool

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
timeline_items[0].AddFlag('Red')
```

GetFlagList()

Description

Returns a list of flag colors assigned to the item.

Returns [colors...]

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
timeline_items[0].GetFlagList()
```

ClearFlags(color)

Description

Clear flags of the specified color. An "All" argument is supported to clear all flags.

Returns Bool

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
timeline_items[0].ClearFlags('Red')
```

GetClipColor()

Description

Returns the item color as a string.

Returns string

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
timeline_items[0].GetClipColor()
```

SetClipColor(colorName)

Description

Sets the item color based on the colorName (string).

Returns Bool

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
timeline_items[0].SetClipColor('Green')
```

ClearClipColor()

Description

Clears the item color.

Returns Bool

```
timeline_items = current_timeline.GetItemListInTrack("video", 1)
timeline_items[0].ClearClipColor()
```

AddFusionComp()

Description

Adds a new Fusion composition associated with the timeline item.

Returns fusionComp

Note: Not tested yet

timeline_items[0].AddFusionComp()

ImportFusionComp(path)

Description

Imports a Fusion composition from given file path by creating and adding a new composition for the item.

Returns fusionComp

Note: Not tested yet

timeline_items[0].ImportFusionComp('/Users/admin/Movies/comp.mov')

ExportFusionComp(path, compIndex)

Description

Exports the Fusion composition based on given index to the path provided.

Returns Bool

Note: Not tested yet

timeline_items[0].ExportFusionComp('/Users/admin/Movies/comp.mov', 1)

DeleteFusionCompByName(compName)

Description

Deletes the named Fusion composition.

Returns Bool

Note: Not tested yet

timeline_items[0].DeleteFusionCompByName('compName')

LoadFusionCompByName(compName)

Description

Loads the named Fusion composition as the active composition.

Returns fusionComp

Note: Not tested yet

timeline_items[0].LoadFusionCompByName('compName')

RenameFusionCompByName(oldName, newName)

Description

Renames the Fusion composition identified by oldName.

Returns Bool

Note: Not tested yet

timeline_items[0].RenameFusionCompByName('oldName', 'newName')

AddVersion(versionName, versionType)

Description

Adds a new color version for a video clipbased on versionType (0 - local, 1 - remote).

Returns Bool

```
timeline_items[0].AddVersion('My version', 0)
```

GetCurrentVersion()

Description

Returns the current version of the video clip. The returned value will have the keys versionName and versionType(0 - local, 1 - remote).

Returns {versionName... }

```
timeline_items[0].GetCurrentVersion()
#{'versionName': 'My version', 'versionType': 0}
```

DeleteVersionByName(versionName, versionType)

Description

Deletes a color version by name and versionType (0 - local, 1 - remote).

Returns Bool

Note: Not tested yet

timeline_items[0].DeleteVersionByName('My version', 0)

LoadVersionByName(versionName, versionType)

Description

Loads a named color version as the active version. versionType: 0 - local, 1 - remote.

Returns Bool

```
timeline_items[0].LoadVersionByName('My version', 0)
```

RenameVersionByName(oldName, newName, versionType)

Description

Renames the color version identified by oldName and versionType (0 - local, 1 - remote).

Returns Bool

```
timeline_items[0].RenameVersionByName('My version', 'My version2', 0)
```

GetVersionNameList(versionType)

Description

Returns a list of all color versions for the given versionType (0 - local, 1 - remote).

Returns [names...]

```
version_list = timeline_items[0].GetVersionNameList(0)
#['Version 1', 'My version', 'My version2', 'My version3']
```

GetMediaPoolItem()

Description

Returns the media pool item corresponding to the timeline item if one exists.

Returns MediaPoolItem

```
mediapool_item = timeline_items[0].GetMediaPoolItem()
#Media pool item (0x0x600023d8be90) [App: 'Resolve' on 127.0.0.1, UUID: 3723b295-9579-
↪4657-be42-33b4f4594a93]
```

GetStereoConvergenceValues()

Description

Returns a dict (offset -> value) of keyframe offsets and respective convergence values.

Returns {keyframes… }

```
timeline_items[0].GetStereoConvergenceValues()
#{92: 0.0}
```

GetStereoLeftFloatingWindowParams()

Description

For the LEFT eye -> returns a dict (offset -> dict) of keyframe offsets and respective floating window params. Value at particular offset includes the left, right, top and bottom floating window values.

Returns {keyframes… }

```
timeline_items[0].GetStereoLeftFloatingWindowParams()
#{92: {'left': 0.0, 'right': 0.0, 'top': 0.0, 'bottom': 0.0}}
```

GetStereoRightFloatingWindowParams()

Description

For the RIGHT eye -> returns a dict (offset -> dict) of keyframe offsets and respective floating window params. Value at particular offset includes the left, right, top and bottom floating window values.

Returns {keyframes...}

```
timeline_items[0].GetStereoRightFloatingWindowParams()
```

SetLUT(nodeIndex, lutPath)

Description

Sets LUT on the node mapping the node index provided, 1 <= nodeIndex <= total number of nodes. The lutPath can be an absolute path, or a relative path (based off custom LUT paths or the master LUT path). The operation is successful for valid lut paths that Resolve has already discovered (see Project.RefreshLUTList).

Returns Bool

Note: Not tested yet

lut_applied = timeline_items[0].SetLUT(1, '/Users/admin/Movies/lut.3dl')

SetCDL([CDL map])

Description

Keys of map are: "NodeIndex", "Slope", "Offset", "Power", "Saturation", where 1 <= NodeIndex <= total number of nodes.

Example python code - SetCDL({"NodeIndex" : "1", "Slope" : "0.5 0.4 0.2", "Offset" : "0.4 0.3 0.2", "Power" : "0.6 0.7 0.8", "Saturation" : "0.65"})

Returns Bool

Note: Not tested yet

cdl_applied = timeline_items[0].SetCDL({"NodeIndex" : "1", "Slope" : "0.5 0.4 0.2", "Offset" : "0.4 0.3 0.2", "Power" : "0.6 0.7 0.8", "Saturation" : "0.65"})

AddTake(mediaPoolItem, startFrame, endFrame)

Description

Adds mediaPoolItem as a new take. Initializes a take selector for the timeline item if needed. By default, the full clip extents is added. startFrame (int) and endFrame (int) are optional arguments used to specify the extents.

Returns Bool

```
mediapool_item = timeline_items[0].GetMediaPoolItem()
timeline_items[0].AddTake(mediapool_item)
```

GetSelectedTakeIndex()

Description

Returns the index of the currently selected take, or 0 if the clip is not a take selector.

Returns int

```
timeline_items[0].GetSelectedTakeIndex()
```

GetTakesCount()

Description

Returns the number of takes in take selector, or 0 if the clip is not a take selector.

Returns int

```
timeline_items[0].GetTakesCount()
```

GetTakeByIndex(idx)

Description

Returns a dict (keys “startFrame”, “endFrame” and “mediaPoolItem”) with take info for specified index.

Returns {takeInfo...}

```
timeline_items[0].GetTakeByIndex(1)
#{'mediaPoolItem': <PyRemoteObject object at 0x7fc9f007a960>, 'startFrame': 0, 'endFrame':␣
↪704}
```

DeleteTakeByIndex(idx)

Description

Deletes a take by index, 1 <= idx <= number of takes.

Returns Bool

```
timeline_items[0].DeleteTakeByIndex(1)
```

SelectTakeByIndex(idx)

Description

Selects a take by index, 1 <= idx <= number of takes.

Returns Bool

```
timeline_items[0].SelectTakeByIndex(1)
```

FinalizeTake()

Description

Finalizes take selection.

Returns Bool

```
timeline_items[0].FinalizeTake()
```

CopyGrades([tgtTimelineItems])

Description

Copies the current grade to all the items in tgtTimelineItems list. Returns True on success and False if any error occured.

Returns Bool

```
timeline_items[0].CopyGrades([timeline_items[1], timeline_items[2]])
#copy grade from first item to second and third timeline item
```

UpdateSidecar()

Description

Updates sidecar file for BRAW clips or RMD file for R3D clips.

Returns Bool

Note: Not tested yet

timeline_items[0].UpdateSidecar()

2.8.10 Gallery

Note: gallery = currentProject.GetGallery()

GetAlbumName(galleryStillAlbum)

Description

Returns the name of the GalleryStillAlbum object 'galleryStillAlbum'.

Returns string

```
current_still_album = gallery.GetCurrentStillAlbum()
album_name = gallery.GetAlbumName(current_still_album)
#Stills 1
```

SetAlbumName(galleryStillAlbum, albumName)

Description

Sets the name of the GalleryStillAlbum object 'galleryStillAlbum' to 'albumName'.

Returns Bool

```
still_albums = gallery.GetGalleryStillAlbums()
gallery.SetAlbumName(still_albums[0], 'Old Stills')
#rename the first StillAlbum
```

GetCurrentStillAlbum()

Description

Returns current album as a GalleryStillAlbum object.

Returns galleryStillAlbum

```
current_still_album = gallery.GetCurrentStillAlbum()
#GalleryStillAlbum (0x0x600023ec4b00) [App: 'Resolve' on 127.0.0.1, UUID: 3723b295-9579-
↪4657-be42-33b4f4594a93]
```

SetCurrentStillAlbum(galleryStillAlbum)

Description

Sets current album to GalleryStillAlbum object 'galleryStillAlbum'.

Returns Bool

```
still_albums = gallery.GetGalleryStillAlbums()
gallery.SetCurrentStillAlbum(still_albums[1])
#change the StillAlbum to the second
```

GetGalleryStillAlbums()

Description

Returns the gallery albums as a list of GalleryStillAlbum objects.

Returns [galleryStillAlbum]

```
still_albums = gallery.GetGalleryStillAlbums()
#[<PyRemoteObject object at 0x7fccf807a990>, <PyRemoteObject object at 0x7fccf807a9a8>]
```

2.8.11 GalleryStillAlbum

GetStills()

Description

Returns the list of GalleryStill objects in the album.

Returns [galleryStill]

```
current_still_album = gallery.GetCurrentStillAlbum()
stills = current_still_album.GetStills()
#[<PyRemoteObject object at 0x7fa8d801a990>]
```

GetLabel(galleryStill)

Description

Returns the label of the galleryStill.

Returns string

```
current_still_album = gallery.GetCurrentStillAlbum()
stills = current_still_album.GetStills()

current_still_album.GetLabel(stills[0])
#Label ot the first still in the current galleryStill
```

SetLabel(galleryStill, label)

Description

Sets the new 'label' to GalleryStill object 'galleryStill'.

Returns Bool

```
current_still_album = gallery.GetCurrentStillAlbum()
stills = current_still_album.GetStills()

current_still_album.SetLabel(stills[0], 'Best label')
#change label ot the first still in the current galleryStill
```

ExportStills([galleryStill], folderPath, filePrefix, format)

Description

Exports list of GalleryStill objects '[galleryStill]' to directory 'folderPath', with filename prefix 'filePrefix', using file format 'format' (supported formats: dpx, cin, tif, jpg, png, ppm, bmp, xpm).

Returns Bool

```
current_still_album = gallery.GetCurrentStillAlbum()
stills = current_still_album.GetStills()
```

(continues on next page)

(continued from previous page)

```
current_still_album.ExportStills([stills[0], stills[1]], '/Users/admin/Desktop/',
↪'stills_ProjectA_', 'png')
#export first 2 stills to folder
```

DeleteStills([galleryStill])

Description

Deletes specified list of GalleryStill objects '[galleryStill]'.

```
current_still_album = gallery.GetCurrentStillAlbum()
stills = current_still_album.GetStills()

current_still_album.DeleteStills([stills[0], stills[1]])
#delete first and second still in the current galleryStill
```

2.8.12 GalleryStill

Description

This class does not provide any API functions but the object type is used by functions in other classes.

Returns Bool

2.9 Timeline item properties

This section covers additional notes for the function "TimelineItem:SetProperty" and "TimelineItem:GetProperty". These functions are used to get and set properties mentioned.

The supported keys with their accepted values are:

- "Pan" : floating point values from -4.0*width to 4.0*width
- "Tilt" : floating point values from -4.0*height to 4.0*height
- "ZoomX" : floating point values from 0.0 to 100.0
- "ZoomY" : floating point values from 0.0 to 100.0
- "ZoomGang" : a boolean value
- "RotationAngle" : floating point values from -360.0 to 360.0
- "AnchorPointX" : floating point values from -4.0*width to 4.0*width
- "AnchorPointY" : floating point values from -4.0*height to 4.0*height
- "Pitch" : floating point values from -1.5 to 1.5
- "Yaw" : floating point values from -1.5 to 1.5

- "FlipX" : boolean value for flipping horizonally
- "FlipY" : boolean value for flipping vertically
- "CropLeft" : floating point values from 0.0 to width
- "CropRight" : floating point values from 0.0 to width
- "CropTop" : floating point values from 0.0 to height
- "CropBottom" : floating point values from 0.0 to height
- "CropSoftness" : floating point values from -100.0 to 100.0
- "CropRetain" : boolean value for "Retain Image Position" checkbox
- "DynamicZoomEase" : A value from the following constants
  - DYNAMIC_ZOOM_EASE_LINEAR = 0
  - DYNAMIC_ZOOM_EASE_IN
  - DYNAMIC_ZOOM_EASE_OUT
  - DYNAMIC_ZOOM_EASE_IN_AND_OUT
- "CompositeMode" : A value from the following constants
  - COMPOSITE_NORMAL = 0
  - COMPOSITE_ADD
  - COMPOSITE_SUBTRACT
  - COMPOSITE_DIFF
  - COMPOSITE_MULTIPLY
  - COMPOSITE_SCREEN
  - COMPOSITE_OVERLAY
  - COMPOSITE_HARDLIGHT
  - COMPOSITE_SOFTLIGHT
  - COMPOSITE_DARKEN
  - COMPOSITE_LIGHTEN
  - COMPOSITE_COLOR_DODGE
  - COMPOSITE_COLOR_BURN
  - COMPOSITE_EXCLUSION
  - COMPOSITE_HUE
  - COMPOSITE_SATURATE
  - COMPOSITE_COLORIZE
  - COMPOSITE_LUMA_MASK
  - COMPOSITE_DIVIDE
  - COMPOSITE_LINEAR_DODGE
  - COMPOSITE_LINEAR_BURN
  - COMPOSITE_LINEAR_LIGHT

  - COMPOSITE_VIVID_LIGHT
  - COMPOSITE_PIN_LIGHT
  - COMPOSITE_HARD_MIX
  - COMPOSITE_LIGHTER_COLOR
  - COMPOSITE_DARKER_COLOR
  - COMPOSITE_FOREGROUND
  - COMPOSITE_ALPHA
  - COMPOSITE_INVERTED_ALPHA
  - COMPOSITE_LUM
  - COMPOSITE_INVERTED_LUM
- "Opacity" : floating point value from 0.0 to 100.0
- "Distortion" : floating point value from -1.0 to 1.0
- "RetimeProcess" : A value from the following constants
  - RETIME_USE_PROJECT = 0
  - RETIME_NEAREST
  - RETIME_FRAME_BLEND
  - RETIME_OPTICAL_FLOW
- "MotionEstimation" : A value from the following constants
  - MOTION_EST_USE_PROJECT = 0
  - MOTION_EST_STANDARD_FASTER
  - MOTION_EST_STANDARD_BETTER
  - MOTION_EST_ENHANCED_FASTER
  - MOTION_EST_ENHANCED_BETTER
  - MOTION_EST_SPEED_WRAP
- "Scaling" : A value from the following constants
  - SCALE_USE_PROJECT = 0
  - SCALE_CROP
  - SCALE_FIT
  - SCALE_FILL
  - SCALE_STRETCH
- "ResizeFilter" : A value from the following constants
  - RESIZE_FILTER_USE_PROJECT = 0
  - RESIZE_FILTER_SHARPER
  - RESIZE_FILTER_SMOOTHER
  - RESIZE_FILTER_BICUBIC
  - RESIZE_FILTER_BILINEAR

- RESIZE_FILTER_BESSEL
- RESIZE_FILTER_BOX
- RESIZE_FILTER_CATMULL_ROM
- RESIZE_FILTER_CUBIC
- RESIZE_FILTER_GAUSSIAN
- RESIZE_FILTER_LANCZOS
- RESIZE_FILTER_MITCHELL
- RESIZE_FILTER_NEAREST_NEIGHBOR
- RESIZE_FILTER_QUADRATIC
- RESIZE_FILTER_SINC
- RESIZE_FILTER_LINEAR

Values beyond the range will be clipped width and height are same as the UI max limits

The arguments can be passed as a key and value pair or they can be grouped together into a dictionary (for python) or table (for lua) and passed as a single argument.

Getting the values for the keys that uses constants will return the number which is in the constant

2.10 Project and Clip properties

This section covers additional notes for the functions "Project:GetSetting", "Project:SetSetting", "Timeline:GetSetting", "Timeline:SetSetting", "MediaPoolItem:GetClipProperty" and "MediaPoolItem:SetClipProperty". These functions are used to get and set properties otherwise available to the user through the Project Settings and the Clip Attributes dialogs.

The functions follow a key-value pair format, where each property is identified by a key (the settingName or propertyName parameter) and possesses a value (typically a text value). Keys and values are designed to be easily correlated with parameter names and values in the Resolve UI. Explicitly enumerated values for some parameters are listed below.

Some properties may be read only - these include intrinsic clip properties like date created or sample rate, and properties that can be disabled in specific application contexts (e.g. custom colorspaces in an ACES workflow, or output sizing parameters when behavior is set to match timeline)

2.10.1 Getting values

Invoke "Project:GetSetting", "Timeline:GetSetting" or "MediaPoolItem:GetClipProperty" with the appropriate property key. To get a snapshot of all queryable properties (keys and values), you can call "Project:GetSetting", "Timeline:GetSetting" or "MediaPoolItem:GetClipProperty" without parameters (or with a NoneType or a blank property key). Using specific keys to query individual properties will be faster. Note that getting a property using an invalid key will return a trivial result.

2.10.2 Setting values

Invoke "Project:SetSetting", "Timeline:SetSetting" or "MediaPoolItem:SetClipProperty" with the appropriate property key and a valid value. When setting a parameter, please check the return value to ensure the success of the operation. You can troubleshoot the validity of keys and values by setting the desired result from the UI and checking property snapshots before and after the change.

The following Project properties have specifically enumerated values: "superScale" - the property value is an enumerated integer between 0 and 3 with these meanings: 0=Auto, 1=no scaling, and 2, 3 and 4 represent the Super Scale multipliers 2x, 3x and 4x. Affects: • x = Project:GetSetting('superScale') and Project:SetSetting('superScale', x)

"timelineFrameRate" - the property value is one of the frame rates available to the user in project settings under "Timeline fra
by appending the frame rate with "DF", e.g. "29.97 DF" will enable drop frame and "29.97" will disable drop frame

Affects: x = Project:GetSetting('timelineFrameRate') and Project:SetSetting('timelineFrameRate', x)

The following Clip properties have specifically enumerated values: "superScale" - the property value is an enumerated integer between 1 and 3 with these meanings: 1=no scaling, and 2, 3 and 4 represent the Super Scale multipliers 2x, 3x and 4x. Affects: x = MediaPoolItem:GetClipProperty('Super Scale') and MediaPoolItem:SetClipProperty('Super Scale', x)

2.11 Timeline export properties

This section covers the parameters for the argument Export(fileName, exportType, exportSubtype).

exportType can be one of the following constants:

- resolve.EXPORT_AAF
- resolve.EXPORT_DRT
- resolve.EXPORT_EDL
- resolve.EXPORT_FCP_7_XML
- resolve.EXPORT_FCPXML_1_3
- resolve.EXPORT_FCPXML_1_4
- resolve.EXPORT_FCPXML_1_5
- resolve.EXPORT_FCPXML_1_6
- resolve.EXPORT_FCPXML_1_7
- resolve.EXPORT_FCPXML_1_8
- resolve.EXPORT_FCPXML_1_9
- resolve.EXPORT_FCPXML_1_10
- resolve.EXPORT_HDR_10_PROFILE_A
- resolve.EXPORT_HDR_10_PROFILE_B
- resolve.EXPORT_TEXT_CSV
- resolve.EXPORT_TEXT_TAB
- resolve.EXPORT_DOLBY_VISION_VER_2_9
- resolve.EXPORT_DOLBY_VISION_VER_4_0

exportSubtype can be one of the following enums:

- resolve.EXPORT_NONE
- resolve.EXPORT_AAF_NEW
- resolve.EXPORT_AAF_EXISTING
- resolve.EXPORT_CDL
- resolve.EXPORT_SDL
- resolve.EXPORT_MISSING_CLIPS

Please note that exportSubType is a required parameter for resolve.EXPORT_AAF and resolve.EXPORT_EDL. For rest of the exportType, exportSubtype is ignored.

When exportType is resolve.EXPORT_AAF, valid exportSubtype values are resolve.EXPORT_AAF_NEW and resolve.EXPORT_AAF_EXISTING. When exportType is resolve.EXPORT_EDL, valid exportSubtype values are resolve.EXPORT_CDL, resolve.EXPORT_SDL, resolve.EXPORT_MISSING_CLIPS and resolve.EXPORT_NONE.

Note: Replace 'resolve.' when using the constants above, if a different Resolve class instance name is used.

2.12 Render Settings

This section covers the supported settings for the method SetRenderSettings({settings})

The parameter setting is a dictionary containing the following keys:

- "SelectAllFrames": Bool (when set True, the settings MarkIn and MarkOut are ignored)
- "MarkIn": int
- "MarkOut": int
- "TargetDir": string
- "CustomName": string
- "UniqueFilenameStyle": 0 - Prefix, 1 - Suffix.
- "ExportVideo": Bool
- "ExportAudio": Bool
- "FormatWidth": int
- "FormatHeight": int
- "FrameRate": float (examples: 23.976, 24)
- "PixelAspectRatio": string (for SD resolution: "16_9" or "4_3") (other resolutions: "square" or "cinemascope")
- "VideoQuality" possible values for current codec (if applicable):
  - 0 (int) - will set quality to automatic
  - [1 -> MAX] (int) - will set input bit rate
  - ["Least", "Low", "Medium", "High", "Best"] (String) - will set input quality level
- "AudioCodec": string (example: "aac")
- "AudioBitDepth": int

- "AudioSampleRate": int
- "ColorSpaceTag" : string (example: "Same as Project", "AstroDesign")
- "GammaTag" : string (example: "Same as Project", "ACEScct")
- "ExportAlpha": Bool
- "EncodingProfile": string (example: "Main10"). Can only be set for H.264 and H.265.
- "MultiPassEncode": Bool. Can only be set for H.264.
- "AlphaMode": 0 - Premultipled, 1 - Straight. Can only be set if "ExportAlpha" is true.
- "NetworkOptimization": Bool. Only supported by QuickTime and MP4 formats.

2.13 Resolve API Readme

Important: For the latest version of this README.TXT, please refer to the local file included with your Davinci Resolve software. (Help > Documentation > Developer)

A HTML formatted version was also created

New in version Updated: as of 29 March 2022

2.13.1 Overview

As with Blackmagic Design Fusion scripts, user scripts written in Lua and Python programming languages are supported. By default, scripts can be invoked from the Console window in the Fusion page, or via command line. This permission can be changed in Resolve Preferences, to be only from Console, or to be invoked from the local network. Please be aware of the security implications when allowing scripting access from outside of the Resolve application.

2.13.2 Prerequisites

DaVinci Resolve scripting requires one of the following to be installed (for all users):

- Lua 5.1
- Python 2.7 64-bit
- Python 3.6 64-bit

2.13.3 Using a script

DaVinci Resolve needs to be running for a script to be invoked.

For a Resolve script to be executed from an external folder, the script needs to know of the API location. You may need to set the these environment variables to allow for your Python installation to pick up the appropriate dependencies as shown below:

- Mac OS X: RESOLVE_SCRIPT_API="/Library/Application Support/Blackmagic Design/DaVinci Resolve/Developer/Scripting" RESOLVE_SCRIPT_LIB="/Applications/DaVinci Resolve/DaVinci Resolve.app/Contents/Libraries/Fusion/fusionscript.so" PYTHONPATH="$PYTHONPATH:$RESOLVE_SCRIPT_API/Modules/"

- Windows: RESOLVE_SCRIPT_API="%PROGRAMDATA%Blackmagic DesignDaVinci ResolveSupport-DeveloperScripting" RESOLVE_SCRIPT_LIB="C:Program FilesBlackmagic DesignDaVinci Resolvefusion-script.dll" PYTHONPATH="%PYTHONPATH%;%RESOLVE_SCRIPT_API%Modules"
- Linux: RESOLVE_SCRIPT_API="/opt/resolve/Developer/Scripting" RESOLVE_SCRIPT_LIB="/opt/resolve/libs/Fusion/fusions PYTHONPATH="$PYTHONPATH:$RESOLVE_SCRIPT_API/Modules/" (Note: For standard ISO Linux installations, the path above may need to be modified to refer to /home/resolve instead of /opt/resolve)

As with Fusion scripts, Resolve scripts can also be invoked via the menu and the Console.

On startup, DaVinci Resolve scans the subfolders in the directories shown below and enumerates the scripts found in the Workspace application menu under Scripts. Place your script under Utility to be listed in all pages, under Comp or Tool to be available in the Fusion page or under folders for individual pages (Edit, Color or Deliver). Scripts under Deliver are additionally listed under render jobs. Placing your script here and invoking it from the menu is the easiest way to use scripts.

- Mac OS X:
  - All users: /Library/Application Support/Blackmagic Design/DaVinci Resolve/Fusion/Scripts
  - Specific user: /Users/<UserName>/Library/Application Support/Blackmagic Design/DaVinci Resolve/Fusion/Scripts
- Windows:
  - All users: %PROGRAMDATA%Blackmagic DesignDaVinci ResolveFusionScripts
  - Specific user: %APPDATA%RoamingBlackmagic DesignDaVinci ResolveSupportFusionScripts
- Linux:
  - All users: /opt/resolve/Fusion/Scripts (or /home/resolve/Fusion/Scripts/ depending on installation)
  - Specific user: $HOME/.local/share/DaVinciResolve/Fusion/Scripts

The interactive Console window allows for an easy way to execute simple scripting commands, to query or modify properties, and to test scripts. The console accepts commands in Python 2.7, Python 3.6 and Lua and evaluates and executes them immediately. For more information on how to use the Console, please refer to the DaVinci Resolve User Manual.

This example Python script creates a simple project:

```
#!/usr/bin/env python
import DaVinciResolveScript as dvr_script
resolve = dvr_script.scriptapp("Resolve")
fusion = resolve.Fusion()
projectManager = resolve.GetProjectManager()
projectManager.CreateProject("Hello World")
```

The resolve object is the fundamental starting point for scripting via Resolve. As a native object, it can be inspected for further scriptable properties - using table iteration and "getmetatable" in Lua and dir, help etc in Python (among other methods). A notable scriptable object above is fusion - it allows access to all existing Fusion scripting functionality.

2.13.4 Running DaVinci Resolve in headless mode

DaVinci Resolve can be launched in a headless mode without the user interface using the -nogui command line option. When DaVinci Resolve is launched using this option, the user interface is disabled. However, the various scripting APIs will continue to work as expected.

2.13.5 Basic Resolve API

Some commonly used API functions are described below (*). As with the resolve object, each object is inspectable for properties and functions.

Resolve

- Fusion() –> Fusion # Returns the Fusion object. Starting point for Fusion scripts.
- GetMediaStorage() –> MediaStorage # Returns the media storage object to query and act on media locations.
- GetProjectManager() –> ProjectManager # Returns the project manager object for currently open database.
- OpenPage(pageName) –> Bool # Switches to indicated page in DaVinci Resolve. Input can be one of (“media”, “cut”, “edit”, “fusion”, “color”, “fairlight”, “deliver”).
- GetCurrentPage() –> String # Returns the page currently displayed in the main window. Returned value can be one of (“media”, “cut”, “edit”, “fusion”, “color”, “fairlight”, “deliver”, None).
- GetProductName() –> string # Returns product name.
- GetVersion() –> [version fields] # Returns list of product version fields in [major, minor, patch, build, suffix] format.
- GetVersionString() –> string # Returns product version in “major.minor.patch[suffix].build” format.
- LoadLayoutPreset(presetName) –> Bool # Loads UI layout from saved preset named ‘presetName’.
- UpdateLayoutPreset(presetName) –> Bool # Overwrites preset named ‘presetName’ with current UI layout.
- ExportLayoutPreset(presetName, presetFilePath) –> Bool # Exports preset named ‘presetName’ to path ‘presetFilePath’.
- DeleteLayoutPreset(presetName) –> Bool # Deletes preset named ‘presetName’.
- SaveLayoutPreset(presetName) –> Bool # Saves current UI layout as a preset named ‘presetName’.
- ImportLayoutPreset(presetFilePath, presetName) –> Bool # Imports preset from path ‘presetFilePath’. The optional argument ‘presetName’ specifies how the preset shall be named. If not specified, the preset is named based on the filename.
- Quit() –> None # Quits the Resolve App.

ProjectManager

- ArchiveProject(projectName,filePath,isArchiveSrcMedia=True,isArchiveRenderCache=True,isArchiveProxyMedia=False) –> Bool # Archives project to provided file path with the configuration as provided by the optional arguments
- CreateProject(projectName) –> Project # Creates and returns a project if projectName (string) is unique, and None if it is not.
- DeleteProject(projectName) –> Bool # Delete project in the current folder if not currently loaded
- LoadProject(projectName) –> Project # Loads and returns the project with name = projectName (string) if there is a match found, and None if there is no matching Project.
- GetCurrentProject() –> Project # Returns the currently loaded Resolve project.

- SaveProject() –> Bool # Saves the currently loaded project with its own name. Returns True if successful.
- CloseProject(project) –> Bool # Closes the specified project without saving.
- CreateFolder(folderName) –> Bool # Creates a folder if folderName (string) is unique.
- DeleteFolder(folderName) –> Bool # Deletes the specified folder if it exists. Returns True in case of success.
- GetProjectListInCurrentFolder() –> [project names...] # Returns a list of project names in current folder.
- GetFolderListInCurrentFolder() –> [folder names...] # Returns a list of folder names in current folder.
- GotoRootFolder() –> Bool # Opens root folder in database.
- GotoParentFolder() –> Bool # Opens parent folder of current folder in database if current folder has parent.
- GetCurrentFolder() –> string # Returns the current folder name.
- OpenFolder(folderName) –> Bool # Opens folder under given name.
- ImportProject(filePath, projectName=None) –> Bool # Imports a project from the file path provided with given project name, if any. Returns True if successful.
- ExportProject(projectName, filePath, withStillsAndLUTs=True) –> Bool # Exports project to provided file path, including stills and LUTs if withStillsAndLUTs is True (enabled by default). Returns True in case of success.
- RestoreProject(filePath, projectName=None) –> Bool # Restores a project from the file path provided with given project name, if any. Returns True if successful.
- GetCurrentDatabase() –> {dbInfo} # Returns a dictionary (with keys 'DbType', 'DbName' and optional 'IpAddress') corresponding to the current database connection
- GetDatabaseList() –> [{dbInfo}] # Returns a list of dictionary items (with keys 'DbType', 'DbName' and optional 'IpAddress') corresponding to all the databases added to Resolve
- SetCurrentDatabase({dbInfo}) –> Bool # Switches current database connection to the database specified by the keys below, and closes any open project.
  - 'DbType': 'Disk' or 'PostgreSQL' (string)
  - 'DbName': database name (string)
  - 'IpAddress': IP address of the PostgreSQL server (string, optional key - defaults to '127.0.0.1')

Project

- GetMediaPool() –> MediaPool # Returns the Media Pool object.
- GetTimelineCount() –> int # Returns the number of timelines currently present in the project.
- GetTimelineByIndex(idx) –> Timeline # Returns timeline at the given index, 1 <= idx <= project.GetTimelineCount()
- GetCurrentTimeline() –> Timeline # Returns the currently loaded timeline.
- SetCurrentTimeline(timeline) –> Bool # Sets given timeline as current timeline for the project. Returns True if successful.
- GetGallery() –> Gallery # Returns the Gallery object.
- GetName() –> string # Returns project name.
- SetName(projectName) –> Bool # Sets project name if given projectname (string) is unique.
- GetPresetList() –> [presets...] # Returns a list of presets and their information.
- SetPreset(presetName) –> Bool # Sets preset by given presetName (string) into project.

- AddRenderJob() –> string # Adds a render job based on current render settings to the render queue. Returns a unique job id (string) for the new render job.
- DeleteRenderJob(jobId) –> Bool # Deletes render job for input job id (string).
- DeleteAllRenderJobs() –> Bool # Deletes all render jobs in the queue.
- GetRenderJobList() –> [render jobs...] # Returns a list of render jobs and their information.
- GetRenderPresetList() –> [presets...] # Returns a list of render presets and their information.
- StartRendering(jobId1, jobId2, ...) –> Bool # Starts rendering jobs indicated by the input job ids.
- StartRendering([jobIds...], isInteractiveMode=False) –> Bool # Starts rendering jobs indicated by the input job ids.
  - The optional "isInteractiveMode", when set, enables error feedback in the UI during rendering.
- StartRendering(isInteractiveMode=False) –> Bool # Starts rendering all queued render jobs.
  - The optional "isInteractiveMode", when set, enables error feedback in the UI during rendering.
- StopRendering() –> None # Stops any current render processes.
- IsRenderingInProgress() –> Bool # Returns True if rendering is in progress.
- LoadRenderPreset(presetName) –> Bool # Sets a preset as current preset for rendering if presetName (string) exists.
- SaveAsNewRenderPreset(presetName) –> Bool # Creates new render preset by given name if presetName(string) is unique.
- SetRenderSettings({settings}) –> Bool # Sets given settings for rendering. Settings is a dict, with support for the keys:
  - Refer to "Looking up render settings" section for information for supported settings
- GetRenderJobStatus(jobId) –> {status info} # Returns a dict with job status and completion percentage of the job by given jobId (string).
- GetSetting(settingName) –> string # Returns value of project setting (indicated by settingName, string). Check the section below for more information.
- SetSetting(settingName, settingValue) –> Bool # Sets the project setting (indicated by settingName, string) to the value (settingValue, string). Check the section below for more information.
- GetRenderFormats() –> {render formats..} # Returns a dict (format -> file extension) of available render formats.
- GetRenderCodecs(renderFormat) –> {render codecs...} # Returns a dict (codec description -> codec name) of available codecs for given render format (string).
- GetCurrentRenderFormatAndCodec() –> {format, codec} # Returns a dict with currently selected format 'format' and render codec 'codec'.
- SetCurrentRenderFormatAndCodec(format, codec) –> Bool # Sets given render format (string) and render codec (string) as options for rendering.
- GetCurrentRenderMode() –> int # Returns the render mode: 0 - Individual clips, 1 - Single clip.
- SetCurrentRenderMode(renderMode) –> Bool # Sets the render mode. Specify renderMode = 0 for Individual clips, 1 for Single clip.
- GetRenderResolutions(format, codec) –> [{Resolution}] # Returns list of resolutions applicable for the given render format (string) and render codec (string). Returns full list of resolutions if no argument is provided. Each element in the list is a dictionary with 2 keys "Width" and "Height".
- RefreshLUTList() –> Bool # Refreshes LUT List

MediaStorage

- GetMountedVolumeList() –> [paths...] # Returns list of folder paths corresponding to mounted volumes displayed in Resolve's Media Storage.
- GetSubFolderList(folderPath) –> [paths...] # Returns list of folder paths in the given absolute folder path.
- GetFileList(folderPath) –> [paths...] # Returns list of media and file listings in the given absolute folder path. Note that media listings may be logically consolidated entries.
- RevealInStorage(path) –> Bool # Expands and displays given file/folder path in Resolve's Media Storage.
- AddItemListToMediaPool(item1, item2, ...) –> [clips...] # Adds specified file/folder paths from Media Storage into current Media Pool folder. Input is one or more file/folder paths. Returns a list of the MediaPoolItems created.
- AddItemListToMediaPool([items...]) –> [clips...] # Adds specified file/folder paths from Media Storage into current Media Pool folder. Input is an array of file/folder paths. Returns a list of the MediaPoolItems created.
- AddClipMattesToMediaPool(MediaPoolItem, [paths], stereoEye) –> Bool # Adds specified media files as mattes for the specified MediaPoolItem. StereoEye is an optional argument for specifying which eye to add the matte to for stereo clips ("left" or "right"). Returns True if successful.
- AddTimelineMattesToMediaPool([paths]) –> [MediaPoolItems] # Adds specified media files as timeline mattes in current media pool folder. Returns a list of created MediaPoolItems.

MediaPool

- GetRootFolder() –> Folder # Returns root Folder of Media Pool
- AddSubFolder(folder, name) –> Folder # Adds new subfolder under specified Folder object with the given name.
- RefreshFolders() –> Bool # Updates the folders in collaboration mode
- CreateEmptyTimeline(name) –> Timeline # Adds new timeline with given name.
- AppendToTimeline(clip1, clip2, ...) –> [TimelineItem] # Appends specified MediaPoolItem objects in the current timeline. Returns the list of appended timelineItems.
- AppendToTimeline([clips]) –> [TimelineItem] # Appends specified MediaPoolItem objects in the current timeline. Returns the list of appended timelineItems.
- AppendToTimeline([{clipInfo}, ...]) –> [TimelineItem] # Appends list of clipInfos specified as dict of "mediaPoolItem", "startFrame" (int), "endFrame" (int), (optional) "mediaType" (int; 1 - Video only, 2 - Audio only). Returns the list of appended timelineItems.
- CreateTimelineFromClips(name, clip1, clip2,...) –> Timeline # Creates new timeline with specified name, and appends the specified MediaPoolItem objects.
- CreateTimelineFromClips(name, [clips]) –> Timeline # Creates new timeline with specified name, and appends the specified MediaPoolItem objects.
- CreateTimelineFromClips(name, [{clipInfo}]) –> Timeline # Creates new timeline with specified name, appending the list of clipInfos specified as a dict of "mediaPoolItem", "startFrame" (int), "endFrame" (int).
- ImportTimelineFromFile(filePath, {importOptions}) –> Timeline # Creates timeline based on parameters within given file and optional importOptions dict, with support for the keys:
  - "timelineName": string, specifies the name of the timeline to be created
  - "importSourceClips": Bool, specifies whether source clips should be imported, True by default
  - "sourceClipsPath": string, specifies a filesystem path to search for source clips if the media is inaccessible in their original path and if "importSourceClips" is True

  - "sourceClipsFolders": List of Media Pool folder objects to search for source clips if the media is not present in current folder and if "importSourceClips" is False
  - "interlaceProcessing": Bool, specifies whether to enable interlace processing on the imported timeline being created. valid only for AAF import
- DeleteTimelines([timeline]) –> Bool # Deletes specified timelines in the media pool.
- GetCurrentFolder() –> Folder # Returns currently selected Folder.
- SetCurrentFolder(Folder) –> Bool # Sets current folder by given Folder.
- DeleteClips([clips]) –> Bool # Deletes specified clips or timeline mattes in the media pool
- DeleteFolders([subfolders]) –> Bool # Deletes specified subfolders in the media pool
- MoveClips([clips], targetFolder) –> Bool # Moves specified clips to target folder.
- MoveFolders([folders], targetFolder) –> Bool # Moves specified folders to target folder.
- GetClipMatteList(MediaPoolItem) –> [paths] # Get mattes for specified MediaPoolItem, as a list of paths to the matte files.
- GetTimelineMatteList(Folder) –> [MediaPoolItems] # Get mattes in specified Folder, as list of MediaPoolItems.
- DeleteClipMattes(MediaPoolItem, [paths]) –> Bool # Delete mattes based on their file paths, for specified MediaPoolItem. Returns True on success.
- RelinkClips([MediaPoolItem], folderPath) –> Bool # Update the folder location of specified media pool clips with the specified folder path.
- UnlinkClips([MediaPoolItem]) –> Bool # Unlink specified media pool clips.
- ImportMedia([items...]) –> [MediaPoolItems] # Imports specified file/folder paths into current Media Pool folder. Input is an array of file/folder paths. Returns a list of the MediaPoolItems created.
- ImportMedia([{clipInfo}]) –> [MediaPoolItems] # Imports file path(s) into current Media Pool folder as specified in list of clipInfo dict. Returns a list of the MediaPoolItems created.
  - Each clipInfo gets imported as one MediaPoolItem unless 'Show Individual Frames' is turned on.
  - Example: ImportMedia([{"FilePath":"file_%03d.dpx", "StartIndex":1, "EndIndex":100}]) would import clip "file_[001-100].dpx".
- ExportMetadata(fileName, [clips]) –> Bool # Exports metadata of specified clips to 'fileName' in CSV format.
  - If no clips are specified, all clips from media pool will be used.

Folder

- GetClipList() –> [clips...] # Returns a list of clips (items) within the folder.
- GetName() –> string # Returns the media folder name.
- GetSubFolderList() –> [folders...] # Returns a list of subfolders in the folder.
- GetIsFolderStale() –> bool # Returns true if folder is stale in collaboration mode, false otherwise

MediaPoolItem

- GetName() –> string # Returns the clip name.
- GetMetadata(metadataType=None) –> string|dict # Returns the metadata value for the key 'metadataType'.
  - If no argument is specified, a dict of all set metadata properties is returned.
- SetMetadata(metadataType, metadataValue) –> Bool # Sets the given metadata to metadataValue (string). Returns True if successful.

- SetMetadata({metadata}) –> Bool # Sets the item metadata with specified 'metadata' dict. Returns True if successful.
- GetMediaId() –> string # Returns the unique ID for the MediaPoolItem.
- AddMarker(frameId, color, name, note, duration,customData) –> Bool
  - Creates a new marker at given frameId position and with given marker information. 'customData' is optional and helps to attach user specific data to the marker.
- GetMarkers() –> {markers...} # Returns a dict (frameId -> {information}) of all markers and dicts with their information.
  - Example of output format: {96.0: {'color': 'Green', 'duration': 1.0, 'note': '', 'name': 'Marker 1', 'customData': ''}, ...}
  - In the above example - there is one 'Green' marker at offset 96 (position of the marker)
- GetMarkerByCustomData(customData) –> {markers...} # Returns marker {information} for the first matching marker with specified customData.
- UpdateMarkerCustomData(frameId, customData) –> Bool # Updates customData (string) for the marker at given frameId position. CustomData is not exposed via UI and is useful for scripting developer to attach any user specific data to markers.
- GetMarkerCustomData(frameId) –> string # Returns customData string for the marker at given frameId position.
- DeleteMarkersByColor(color) –> Bool # Delete all markers of the specified color from the media pool item. "All" as argument deletes all color markers.
- DeleteMarkerAtFrame(frameNum) –> Bool # Delete marker at frame number from the media pool item.
- DeleteMarkerByCustomData(customData) –> Bool # Delete first matching marker with specified customData.
- AddFlag(color) –> Bool # Adds a flag with given color (string).
- GetFlagList() –> [colors...] # Returns a list of flag colors assigned to the item.
- ClearFlags(color) –> Bool # Clears the flag of the given color if one exists. An "All" argument is supported and clears all flags.
- GetClipColor() –> string # Returns the item color as a string.
- SetClipColor(colorName) –> Bool # Sets the item color based on the colorName (string).
- ClearClipColor() –> Bool # Clears the item color.
- GetClipProperty(propertyName=None) –> string|dict # Returns the property value for the key 'propertyName'.
  - If no argument is specified, a dict of all clip properties is returned. Check the section below for more information.
- SetClipProperty(propertyName, propertyValue) –> Bool # Sets the given property to propertyValue (string). Check the section below for more information.
- LinkProxyMedia(proxyMediaFilePath) –> Bool # Links proxy media located at path specified by arg 'proxyMediaFilePath' with the current clip. 'proxyMediaFilePath' should be absolute clip path.
- UnlinkProxyMedia() –> Bool # Unlinks any proxy media associated with clip.
- ReplaceClip(filePath) –> Bool # Replaces the underlying asset and metadata of MediaPoolItem with the specified absolute clip path.

Timeline

- GetName() –> string # Returns the timeline name.

- SetName(timelineName) –> Bool # Sets the timeline name if timelineName (string) is unique. Returns True if successful.
- GetStartFrame() –> int # Returns the frame number at the start of timeline.
- GetEndFrame() –> int # Returns the frame number at the end of timeline.
- SetStartTimecode(timecode) –> Bool # Set the start timecode of the timeline to the string 'timecode'. Returns true when the change is successful, false otherwise.
- GetStartTimecode() –> string # Returns the start timecode for the timeline.
- GetTrackCount(trackType) –> int # Returns the number of tracks for the given track type ("audio", "video" or "subtitle").
- GetItemListInTrack(trackType, index) –> [items...] # Returns a list of timeline items on that track (based on trackType and index). 1 <= index <= GetTrackCount(trackType).
- AddMarker(frameId, color, name, note, duration, customData) –> Bool
  - Creates a new marker at given frameId position and with given marker information. 'customData' is optional and helps to attach user specific data to the marker.
- GetMarkers() –> {markers...} # Returns a dict (frameId -> {information}) of all markers and dicts with their information.
  - Example: a value of {96.0: {'color': 'Green', 'duration': 1.0, 'note': '', 'name': 'Marker 1', 'customData': ''}, ... } indicates a single green marker at timeline offset 96
- GetMarkerByCustomData(customData) –> {markers...} # Returns marker {information} for the first matching marker with specified customData.
- UpdateMarkerCustomData(frameId, customData) –> Bool # Updates customData (string) for the marker at given frameId position. CustomData is not exposed via UI and is useful for scripting developer to attach any user specific data to markers.
- GetMarkerCustomData(frameId) –> string # Returns customData string for the marker at given frameId position.
- DeleteMarkersByColor(color) –> Bool # Deletes all timeline markers of the specified color. An "All" argument is supported and deletes all timeline markers.
- DeleteMarkerAtFrame(frameNum) –> Bool # Deletes the timeline marker at the given frame number.
- DeleteMarkerByCustomData(customData) –> Bool # Delete first matching marker with specified customData.
- ApplyGradeFromDRX(path, gradeMode, item1, item2, ...)–> Bool # Loads a still from given file path (string) and applies grade to Timeline Items with gradeMode (int): 0 - "No keyframes", 1 - "Source Timecode aligned", 2 - "Start Frames aligned".
- ApplyGradeFromDRX(path, gradeMode, [items]) –> Bool # Loads a still from given file path (string) and applies grade to Timeline Items with gradeMode (int): 0 - "No keyframes", 1 - "Source Timecode aligned", 2 - "Start Frames aligned".
- GetCurrentTimecode() –> string # Returns a string timecode representation for the current playhead position, while on Cut, Edit, Color, Fairlight and Deliver pages.
- SetCurrentTimecode(timecode) –> Bool # Sets current playhead position from input timecode for Cut, Edit, Color, Fairlight and Deliver pages.
- GetCurrentVideoItem() –> item # Returns the current video timeline item.
- GetCurrentClipThumbnailImage() –> {thumbnailData} # Returns a dict (keys "width", "height", "format" and "data") with data containing raw thumbnail image data (RGB 8-bit image data encoded in base64 format) for current media in the Color Page.

  - An example of how to retrieve and interpret thumbnails is provided in 6_get_current_media_thumbnail.py in the Examples folder.

- GetTrackName(trackType, trackIndex) –> string # Returns the track name for track indicated by trackType (“audio”, “video” or “subtitle”) and index. 1 <= trackIndex <= GetTrackCount(trackType).
- SetTrackName(trackType, trackIndex, name) –> Bool # Sets the track name (string) for track indicated by trackType (“audio”, “video” or “subtitle”) and index. 1 <= trackIndex <= GetTrackCount(trackType).
- DuplicateTimeline(timelineName) –> timeline # Duplicates the timeline and returns the created timeline, with the (optional) timelineName, on success.
- CreateCompoundClip([timelineItems], {clipInfo}) –> timelineItem # Creates a compound clip of input timeline items with an optional clipInfo map: {“startTimecode” : “00:00:00:00”, “name” : “Compound Clip 1”}. It returns the created timeline item.
- CreateFusionClip([timelineItems]) –> timelineItem # Creates a Fusion clip of input timeline items. It returns the created timeline item.
- ImportIntoTimeline(filePath, {importOptions}) –> Bool # Imports timeline items from an AAF file and optional importOptions dict into the timeline, with support for the keys:
  - “autoImportSourceClipsIntoMediaPool”: Bool, specifies if source clips should be imported into media pool, True by default
  - “ignoreFileExtensionsWhenMatching”: Bool, specifies if file extensions should be ignored when matching, False by default
  - “linkToSourceCameraFiles”: Bool, specifies if link to source camera files should be enabled, False by default
  - “useSizingInfo”: Bool, specifies if sizing information should be used, False by default
  - “importMultiChannelAudioTracksAsLinkedGroups”: Bool, specifies if multi-channel audio tracks should be imported as linked groups, False by default
  - “insertAdditionalTracks”: Bool, specifies if additional tracks should be inserted, True by default
  - “insertWithOffset”: string, specifies insert with offset value in timecode format - defaults to “00:00:00:00”, applicable if “insertAdditionalTracks” is False
  - “sourceClipsPath”: string, specifies a filesystem path to search for source clips if the media is inaccessible in their original path and if “ignoreFileExtensionsWhenMatching” is True
  - “sourceClipsFolders”: string, list of Media Pool folder objects to search for source clips if the media is not present in current folder
- Export(fileName, exportType, exportSubtype) –> Bool # Exports timeline to ‘fileName’ as per input exportType & exportSubtype format.
  - Refer to section “Looking up timeline exports properties” for information on the parameters.
- GetSetting(settingName) –> string # Returns value of timeline setting (indicated by settingName : string). Check the section below for more information.
- SetSetting(settingName, settingValue) –> Bool # Sets timeline setting (indicated by settingName : string) to the value (settingValue : string). Check the section below for more information.
- InsertGeneratorIntoTimeline(generatorName) –> TimelineItem # Inserts a generator (indicated by generatorName : string) into the timeline.
- InsertFusionGeneratorIntoTimeline(generatorName) –> TimelineItem # Inserts a Fusion generator (indicated by generatorName : string) into the timeline.
- InsertFusionCompositionIntoTimeline() –> TimelineItem # Inserts a Fusion composition into the timeline.

- InsertOFXGeneratorIntoTimeline(generatorName) –> TimelineItem # Inserts an OFX generator (indicated by generatorName : string) into the timeline.
- InsertTitleIntoTimeline(titleName) –> TimelineItem # Inserts a title (indicated by titleName : string) into the timeline.
- InsertFusionTitleIntoTimeline(titleName) –> TimelineItem # Inserts a Fusion title (indicated by titleName : string) into the timeline.
- GrabStill() –> galleryStill # Grabs still from the current video clip. Returns a GalleryStill object.
- GrabAllStills(stillFrameSource) –> [galleryStill] # Grabs stills from all the clips of the timeline at 'stillFrameSource' (1 - First frame, 2 - Middle frame). Returns the list of GalleryStill objects.

TimelineItem

- GetName() –> string # Returns the item name.
- GetDuration() –> int # Returns the item duration.
- GetEnd() –> int # Returns the end frame position on the timeline.
- GetFusionCompCount() –> int # Returns number of Fusion compositions associated with the timeline item.
- GetFusionCompByIndex(compIndex) –> fusionComp # Returns the Fusion composition object based on given index. 1 <= compIndex <= timelineItem.GetFusionCompCount()
- GetFusionCompNameList() –> [names...] # Returns a list of Fusion composition names associated with the timeline item.
- GetFusionCompByName(compName) –> fusionComp # Returns the Fusion composition object based on given name.
- GetLeftOffset() –> int # Returns the maximum extension by frame for clip from left side.
- GetRightOffset() –> int # Returns the maximum extension by frame for clip from right side.
- GetStart() –> int # Returns the start frame position on the timeline.
- SetProperty(propertyKey, propertyValue) –> Bool # Sets the value of property "propertyKey" to value "propertyValue"
  - Refer to "Looking up Timeline item properties" for more information
- **GetProperty(propertyKey) –> int/[key:value] # returns the value of the specified key** # if no key is specified, the method returns a dictionary(python) or table(lua) for all supported keys
- AddMarker(frameId, color, name, note, duration, customData) –> Bool
  - Creates a new marker at given frameId position and with given marker information. 'customData' is optional and helps to attach user specific data to the marker.
- GetMarkers() –> {markers...} # Returns a dict (frameId -> {information}) of all markers and dicts with their information.
  - Example: a value of {96.0: {'color': 'Green', 'duration': 1.0, 'note': '', 'name': 'Marker 1', 'customData': ''}, ...} indicates a single green marker at clip offset 96
- GetMarkerByCustomData(customData) –> {markers...} # Returns marker {information} for the first matching marker with specified customData.
- UpdateMarkerCustomData(frameId, customData) –> Bool # Updates customData (string) for the marker at given frameId position. CustomData is not exposed via UI and is useful for scripting developer to attach any user specific data to markers.
- GetMarkerCustomData(frameId) –> string # Returns customData string for the marker at given frameId position.

- DeleteMarkersByColor(color) –> Bool # Delete all markers of the specified color from the timeline item. "All" as argument deletes all color markers.
- DeleteMarkerAtFrame(frameNum) –> Bool # Delete marker at frame number from the timeline item.
- DeleteMarkerByCustomData(customData) –> Bool # Delete first matching marker with specified customData.
- AddFlag(color) –> Bool # Adds a flag with given color (string).
- GetFlagList() –> [colors...] # Returns a list of flag colors assigned to the item.
- ClearFlags(color) –> Bool # Clear flags of the specified color. An "All" argument is supported to clear all flags.
- GetClipColor() –> string # Returns the item color as a string.
- SetClipColor(colorName) –> Bool # Sets the item color based on the colorName (string).
- ClearClipColor() –> Bool # Clears the item color.
- AddFusionComp() –> fusionComp # Adds a new Fusion composition associated with the timeline item.
- ImportFusionComp(path) –> fusionComp # Imports a Fusion composition from given file path by creating and adding a new composition for the item.
- ExportFusionComp(path, compIndex) –> Bool # Exports the Fusion composition based on given index to the path provided.
- DeleteFusionCompByName(compName) –> Bool # Deletes the named Fusion composition.
- LoadFusionCompByName(compName) –> fusionComp # Loads the named Fusion composition as the active composition.
- RenameFusionCompByName(oldName, newName) –> Bool # Renames the Fusion composition identified by oldName.
- AddVersion(versionName, versionType) –> Bool # Adds a new color version for a video clipbased on versionType (0 - local, 1 - remote).
- GetCurrentVersion() –> {versionName...} # Returns the current version of the video clip. The returned value will have the keys versionName and versionType(0 - local, 1 - remote).
- DeleteVersionByName(versionName, versionType) –> Bool # Deletes a color version by name and versionType (0 - local, 1 - remote).
- LoadVersionByName(versionName, versionType) –> Bool # Loads a named color version as the active version. versionType: 0 - local, 1 - remote.
- RenameVersionByName(oldName, newName, versionType)–> Bool # Renames the color version identified by oldName and versionType (0 - local, 1 - remote).
- GetVersionNameList(versionType) –> [names...] # Returns a list of all color versions for the given versionType (0 - local, 1 - remote).
- GetMediaPoolItem() –> MediaPoolItem # Returns the media pool item corresponding to the timeline item if one exists.
- GetStereoConvergenceValues() –> {keyframes...} # Returns a dict (offset -> value) of keyframe offsets and respective convergence values.
- GetStereoLeftFloatingWindowParams() –> {keyframes...} # For the LEFT eye -> returns a dict (offset -> dict) of keyframe offsets and respective floating window params. Value at particular offset includes the left, right, top and bottom floating window values.
- GetStereoRightFloatingWindowParams() –> {keyframes...} # For the RIGHT eye -> returns a dict (offset -> dict) of keyframe offsets and respective floating window params. Value at particular offset includes the left, right, top and bottom floating window values.

- SetLUT(nodeIndex, lutPath) –> Bool # Sets LUT on the node mapping the node index provided, 1 <= nodeIndex <= total number of nodes.
  - The lutPath can be an absolute path, or a relative path (based off custom LUT paths or the master LUT path).
  - The operation is successful for valid lut paths that Resolve has already discovered (see Project.RefreshLUTList).
- SetCDL([CDL map]) –> Bool # Keys of map are: "NodeIndex", "Slope", "Offset", "Power", "Saturation", where 1 <= NodeIndex <= total number of nodes.
  - Example python code - SetCDL({"NodeIndex" : "1", "Slope" : "0.5 0.4 0.2", "Offset" : "0.4 0.3 0.2", "Power" : "0.6 0.7 0.8", "Saturation" : "0.65"})
- AddTake(mediaPoolItem, startFrame, endFrame) –> Bool # Adds mediaPoolItem as a new take. Initializes a take selector for the timeline item if needed. By default, the full clip extents is added. startFrame (int) and endFrame (int) are optional arguments used to specify the extents.
- GetSelectedTakeIndex() –> int # Returns the index of the currently selected take, or 0 if the clip is not a take selector.
- GetTakesCount() –> int # Returns the number of takes in take selector, or 0 if the clip is not a take selector.
- GetTakeByIndex(idx) –> {takeInfo...} # Returns a dict (keys "startFrame", "endFrame" and "mediaPoolItem") with take info for specified index.
- DeleteTakeByIndex(idx) –> Bool # Deletes a take by index, 1 <= idx <= number of takes.
- SelectTakeByIndex(idx) –> Bool # Selects a take by index, 1 <= idx <= number of takes.
- FinalizeTake() –> Bool # Finalizes take selection.
- CopyGrades([tgtTimelineItems]) –> Bool # Copies the current grade to all the items in tgtTimelineItems list. Returns True on success and False if any error occured.
- UpdateSidecar() –> Bool # Updates sidecar file for BRAW clips or RMD file for R3D clips.

Gallery

- GetAlbumName(galleryStillAlbum) –> string # Returns the name of the GalleryStillAlbum object 'galleryStillAlbum'.
- SetAlbumName(galleryStillAlbum, albumName) –> Bool # Sets the name of the GalleryStillAlbum object 'galleryStillAlbum' to 'albumName'.
- GetCurrentStillAlbum() –> galleryStillAlbum # Returns current album as a GalleryStillAlbum object.
- SetCurrentStillAlbum(galleryStillAlbum) –> Bool # Sets current album to GalleryStillAlbum object 'galleryStillAlbum'.
- GetGalleryStillAlbums() –> [galleryStillAlbum] # Returns the gallery albums as a list of GalleryStillAlbum objects.

GalleryStillAlbum

- GetStills() –> [galleryStill] # Returns the list of GalleryStill objects in the album.
- GetLabel(galleryStill) –> string # Returns the label of the galleryStill.
- SetLabel(galleryStill, label) –> Bool # Sets the new 'label' to GalleryStill object 'galleryStill'.
- ExportStills([galleryStill], folderPath, filePrefix, format) –> Bool # Exports list of GalleryStill objects '[galleryStill]' to directory 'folderPath', with filename prefix 'filePrefix', using file format 'format' (supported formats: dpx, cin, tif, jpg, png, ppm, bmp, xpm).

- DeleteStills([galleryStill]) –> Bool # Deletes specified list of GalleryStill objects '[galleryStill]'.

GalleryStill

- This class does not provide any API functions but the object type is used by functions in other classes.

2.13.6 List and Dict Data Structures

Beside primitive data types, Resolve's Python API mainly uses list and dict data structures.

Lists are denoted by [...] and dicts are denoted by { ... } above.

As Lua does not support list and dict data structures, the Lua API implements "list" as a table with indices

- e.g. { [1] = listValue1, [2] = listValue2, ... }.

Similarly the Lua API implements "dict" as a table with the dictionary key as first element

- e.g. { [dictKey1] = dictValue1, [dictKey2] = dictValue2, ... }.

2.13.7 Looking up Project and Clip properties

This section covers additional notes for the functions "Project:GetSetting", "Project:SetSetting", "Timeline:GetSetting", "Timeline:SetSetting", "MediaPoolItem:GetClipProperty" and "MediaPoolItem:SetClipProperty". These functions are used to get and set properties otherwise available to the user through the Project Settings and the Clip Attributes dialogs.

The functions follow a key-value pair format, where each property is identified by a key (the settingName or propertyName parameter) and possesses a value (typically a text value). Keys and values are designed to be easily correlated with parameter names and values in the Resolve UI. Explicitly enumerated values for some parameters are listed below.

Some properties may be read only - these include intrinsic clip properties like date created or sample rate, and properties that can be disabled in specific application contexts (e.g. custom colorspaces in an ACES workflow, or output sizing parameters when behavior is set to match timeline)

Getting values: Invoke "Project:GetSetting", "Timeline:GetSetting" or "MediaPoolItem:GetClipProperty" with the appropriate property key. To get a snapshot of all queryable properties (keys and values), you can call "Project:GetSetting", "Timeline:GetSetting" or "MediaPoolItem:GetClipProperty" without parameters (or with a NoneType or a blank property key). Using specific keys to query individual properties will be faster. Note that getting a property using an invalid key will return a trivial result.

Setting values: Invoke "Project:SetSetting", "Timeline:SetSetting" or "MediaPoolItem:SetClipProperty" with the appropriate property key and a valid value. When setting a parameter, please check the return value to ensure the success of the operation. You can troubleshoot the validity of keys and values by setting the desired result from the UI and checking property snapshots before and after the change.

The following Project properties have specifically enumerated values: "superScale" - the property value is an enumerated integer between 0 and 3 with these meanings: 0=Auto, 1=no scaling, and 2, 3 and 4 represent the Super Scale multipliers 2x, 3x and 4x. Affects: • x = Project:GetSetting('superScale') and Project:SetSetting('superScale', x)

"timelineFrameRate" - the property value is one of the frame rates available to the user in project settings under "Timeline fra
by appending the frame rate with "DF", e.g. "29.97 DF" will enable drop frame and "29.97" will disable drop frame

Affects: • x = Project:GetSetting('timelineFrameRate') and Project:SetSetting('timelineFrameRate', x)

The following Clip properties have specifically enumerated values: "superScale" - the property value is an enumerated integer between 1 and 3 with these meanings: 1=no scaling, and 2, 3 and 4 represent the Super Scale multipliers 2x, 3x and 4x. Affects: • x = MediaPoolItem:GetClipProperty('Super Scale') and MediaPoolItem:SetClipProperty('Super Scale', x)

2.13.8 Looking up Render Settings

This section covers the supported settings for the method SetRenderSettings({settings})

The parameter setting is a dictionary containing the following keys:

- "SelectAllFrames": Bool (when set True, the settings MarkIn and MarkOut are ignored)
- "MarkIn": int
- "MarkOut": int
- "TargetDir": string
- "CustomName": string
- "UniqueFilenameStyle": 0 - Prefix, 1 - Suffix.
- "ExportVideo": Bool
- "ExportAudio": Bool
- "FormatWidth": int
- "FormatHeight": int
- "FrameRate": float (examples: 23.976, 24)
- "PixelAspectRatio": string (for SD resolution: "16_9" or "4_3") (other resolutions: "square" or "cinemascope")
- "VideoQuality" possible values for current codec (if applicable):
  - 0 (int) - will set quality to automatic
  - [1 -> MAX] (int) - will set input bit rate
  - ["Least", "Low", "Medium", "High", "Best"] (String) - will set input quality level
- "AudioCodec": string (example: "aac")
- "AudioBitDepth": int
- "AudioSampleRate": int
- "ColorSpaceTag" : string (example: "Same as Project", "AstroDesign")
- "GammaTag" : string (example: "Same as Project", "ACEScct")
- "ExportAlpha": Bool
- "EncodingProfile": string (example: "Main10"). Can only be set for H.264 and H.265.
- "MultiPassEncode": Bool. Can only be set for H.264.
- "AlphaMode": 0 - Premultipled, 1 - Straight. Can only be set if "ExportAlpha" is true.
- "NetworkOptimization": Bool. Only supported by QuickTime and MP4 formats.

2.13.9 Looking up timeline export properties

This section covers the parameters for the argument Export(fileName, exportType, exportSubtype).

exportType can be one of the following constants:

- resolve.EXPORT_AAF
- resolve.EXPORT_DRT
- resolve.EXPORT_EDL
- resolve.EXPORT_FCP_7_XML
- resolve.EXPORT_FCPXML_1_3
- resolve.EXPORT_FCPXML_1_4
- resolve.EXPORT_FCPXML_1_5
- resolve.EXPORT_FCPXML_1_6
- resolve.EXPORT_FCPXML_1_7
- resolve.EXPORT_FCPXML_1_8
- resolve.EXPORT_FCPXML_1_9
- resolve.EXPORT_FCPXML_1_10
- resolve.EXPORT_HDR_10_PROFILE_A
- resolve.EXPORT_HDR_10_PROFILE_B
- resolve.EXPORT_TEXT_CSV
- resolve.EXPORT_TEXT_TAB
- resolve.EXPORT_DOLBY_VISION_VER_2_9
- resolve.EXPORT_DOLBY_VISION_VER_4_0

exportSubtype can be one of the following enums:

- resolve.EXPORT_NONE
- resolve.EXPORT_AAF_NEW
- resolve.EXPORT_AAF_EXISTING
- resolve.EXPORT_CDL
- resolve.EXPORT_SDL
- resolve.EXPORT_MISSING_CLIPS

Please note that exportSubType is a required parameter for resolve.EXPORT_AAF and resolve.EXPORT_EDL. For rest of the exportType, exportSubtype is ignored.

When exportType is resolve.EXPORT_AAF, valid exportSubtype values are resolve.EXPORT_AAF_NEW and resolve.EXPORT_AAF_EXISTING. When exportType is resolve.EXPORT_EDL, valid exportSubtype values are resolve.EXPORT_CDL, resolve.EXPORT_SDL, resolve.EXPORT_MISSING_CLIPS and resolve.EXPORT_NONE.

Note: Replace 'resolve.' when using the constants above, if a different Resolve class instance name is used.

2.13.10 Looking up Timeline item properties

This section covers additional notes for the function "TimelineItem:SetProperty" and "TimelineItem:GetProperty". These functions are used to get and set properties mentioned.

The supported keys with their accepted values are:

- "Pan" : floating point values from -4.0*width to 4.0*width
- "Tilt" : floating point values from -4.0*height to 4.0*height
- "ZoomX" : floating point values from 0.0 to 100.0
- "ZoomY" : floating point values from 0.0 to 100.0
- "ZoomGang" : a boolean value
- "RotationAngle" : floating point values from -360.0 to 360.0
- "AnchorPointX" : floating point values from -4.0*width to 4.0*width
- "AnchorPointY" : floating point values from -4.0*height to 4.0*height
- "Pitch" : floating point values from -1.5 to 1.5
- "Yaw" : floating point values from -1.5 to 1.5
- "FlipX" : boolean value for flipping horizonally
- "FlipY" : boolean value for flipping vertically
- "CropLeft" : floating point values from 0.0 to width
- "CropRight" : floating point values from 0.0 to width
- "CropTop" : floating point values from 0.0 to height
- "CropBottom" : floating point values from 0.0 to height
- "CropSoftness" : floating point values from -100.0 to 100.0
- "CropRetain" : boolean value for "Retain Image Position" checkbox
- "DynamicZoomEase" : A value from the following constants
  - DYNAMIC_ZOOM_EASE_LINEAR = 0
  - DYNAMIC_ZOOM_EASE_IN
  - DYNAMIC_ZOOM_EASE_OUT
  - DYNAMIC_ZOOM_EASE_IN_AND_OUT
- "CompositeMode" : A value from the following constants
  - COMPOSITE_NORMAL = 0
  - COMPOSITE_ADD
  - COMPOSITE_SUBTRACT
  - COMPOSITE_DIFF
  - COMPOSITE_MULTIPLY
  - COMPOSITE_SCREEN
  - COMPOSITE_OVERLAY
  - COMPOSITE_HARDLIGHT

  - COMPOSITE_SOFTLIGHT
  - COMPOSITE_DARKEN
  - COMPOSITE_LIGHTEN
  - COMPOSITE_COLOR_DODGE
  - COMPOSITE_COLOR_BURN
  - COMPOSITE_EXCLUSION
  - COMPOSITE_HUE
  - COMPOSITE_SATURATE
  - COMPOSITE_COLORIZE
  - COMPOSITE_LUMA_MASK
  - COMPOSITE_DIVIDE
  - COMPOSITE_LINEAR_DODGE
  - COMPOSITE_LINEAR_BURN
  - COMPOSITE_LINEAR_LIGHT
  - COMPOSITE_VIVID_LIGHT
  - COMPOSITE_PIN_LIGHT
  - COMPOSITE_HARD_MIX
  - COMPOSITE_LIGHTER_COLOR
  - COMPOSITE_DARKER_COLOR
  - COMPOSITE_FOREGROUND
  - COMPOSITE_ALPHA
  - COMPOSITE_INVERTED_ALPHA
  - COMPOSITE_LUM
  - COMPOSITE_INVERTED_LUM
- "Opacity" : floating point value from 0.0 to 100.0
- "Distortion" : floating point value from -1.0 to 1.0
- "RetimeProcess" : A value from the following constants
  - RETIME_USE_PROJECT = 0
  - RETIME_NEAREST
  - RETIME_FRAME_BLEND
  - RETIME_OPTICAL_FLOW
- "MotionEstimation" : A value from the following constants
  - MOTION_EST_USE_PROJECT = 0
  - MOTION_EST_STANDARD_FASTER
  - MOTION_EST_STANDARD_BETTER
  - MOTION_EST_ENHANCED_FASTER

  - MOTION_EST_ENHANCED_BETTER
  - MOTION_EST_SPEED_WRAP
- "Scaling" : A value from the following constants
  - SCALE_USE_PROJECT = 0
  - SCALE_CROP
  - SCALE_FIT
  - SCALE_FILL
  - SCALE_STRETCH
- "ResizeFilter" : A value from the following constants
  - RESIZE_FILTER_USE_PROJECT = 0
  - RESIZE_FILTER_SHARPER
  - RESIZE_FILTER_SMOOTHER
  - RESIZE_FILTER_BICUBIC
  - RESIZE_FILTER_BILINEAR
  - RESIZE_FILTER_BESSEL
  - RESIZE_FILTER_BOX
  - RESIZE_FILTER_CATMULL_ROM
  - RESIZE_FILTER_CUBIC
  - RESIZE_FILTER_GAUSSIAN
  - RESIZE_FILTER_LANCZOS
  - RESIZE_FILTER_MITCHELL
  - RESIZE_FILTER_NEAREST_NEIGHBOR
  - RESIZE_FILTER_QUADRATIC
  - RESIZE_FILTER_SINC
  - RESIZE_FILTER_LINEAR

Values beyond the range will be clipped width and height are same as the UI max limits

The arguments can be passed as a key and value pair or they can be grouped together into a dictionary (for python) or table (for lua) and passed as a single argument.

Getting the values for the keys that uses constants will return the number which is in the constant

2.13.11 Deprecated Resolve API Functions

The following API functions are deprecated.

- ProjectManager
  - GetProjectsInCurrentFolder() –> {project names…} # Returns a dict of project names in current folder.
  - GetFoldersInCurrentFolder() –> {folder names…} # Returns a dict of folder names in current folder* .
- Project
  - GetPresets() –> {presets…} # Returns a dict of presets and their information.
  - GetRenderJobs() –> {render jobs…} # Returns a dict of render jobs and their information.
  - GetRenderPresets() –> {presets…} # Returns a dict of render presets and their information.
- MediaStorage
  - GetMountedVolumes() –> {paths…} # Returns a dict of folder paths corresponding to mounted volumes displayed in Resolve's Media Storage.
  - GetSubFolders(folderPath) –> {paths…} # Returns a dict of folder paths in the given absolute folder path.
  - GetFiles(folderPath) –> {paths…} # Returns a dict of media and file listings in the given absolute folder path. Note that media listings may be logically consolidated entries.
  - AddItemsToMediaPool(item1, item2, …) –> {clips…} # Adds specified file/folder paths from Media Storage into current Media Pool folder. Input is one or more file/folder paths. Returns a dict of the MediaPoolItems created.
  - AddItemsToMediaPool([items…]) –> {clips…} # Adds specified file/folder paths from Media Storage into current Media Pool folder. Input is an array of file/folder paths. Returns a dict of the MediaPoolItems created.
- Folder
  - GetClips() –> {clips…} # Returns a dict of clips (items) within the folder.
  - GetSubFolders() –> {folders…} # Returns a dict of subfolders in the folder.
- MediaPoolItem
  - GetFlags() –> {colors…} # Returns a dict of flag colors assigned to the item* .
- Timeline
  - GetItemsInTrack(trackType, index) –> {items…} # Returns a dict of Timeline items on the video or audio track (based on trackType) at specified
- TimelineItem
  - GetFusionCompNames() –> {names…} # Returns a dict of Fusion composition names associated with the timeline item.
  - GetFlags() –> {colors…} # Returns a dict of flag colors assigned to the item.
  - GetVersionNames(versionType) –> {names…} # Returns a dict of version names by provided versionType: 0 - local, 1 - remote.

2.13.12 Unsupported Resolve API Functions

The following API (functions and paraameters) are no longer supported. Use job IDs instead of indices.

Project

- StartRendering(index1, index2, ...) –> Bool # Please use unique job ids (string) instead of indices.
- StartRendering([idxs...]) –> Bool # Please use unique job ids (string) instead of indices.
- DeleteRenderJobByIndex(idx) –> Bool # Please use unique job ids (string) instead of indices.
- GetRenderJobStatus(idx) –> {status info} # Please use unique job ids (string) instead of indices.
- GetSetting and SetSetting –> {} # settingName videoMonitorUseRec601For422SDI is now replaced with videoMonitorUseMatrixOverrideFor422SDI and videoMonitorMatrixOverrideFor422SDI.
- # settingName perfProxyMediaOn is now replaced with perfProxyMediaMode which takes values 0 - disabled, 1 - when available, 2 - when source not available.

2.14 Workflow Integration Readme

Important: For the latest version of this README.TXT, please refer to the local file included with your Davinci Resolve software. (Help > Documentation > Developer)

New in version Updated: as of 25 August, 2020

In this package, you will find a brief introduction to the Workflow Integration Plugins support for DaVinci Resolve Studio. Apart from this README.txt file, this package contains following folders:

- Examples: containing some representative sample plugin, and a sample script.
- Scripts: containing some sample workflow scripts to interact with Resolve.

2.14.1 Overview

DaVinci Resolve Studio now supports Workflow Integration Plugins to be loaded and communicate with Resolve. Resolve can run one or more Workflow Integration Plugins at the same time. Users can write their own Workflow Integration Plugin (an Electron app) which could be loaded into DaVinci Resolve Studio. To interact with Resolve, Resolve's JavaScript APIs can be used from the plugin.

Alternatively, a Python or Lua script can be invoked, with the option of a user interface built with Resolve's built-in Qt-based UIManager, or with an external GUI manager. See the "Sample Workflow Integration Script" section below for details.

2.14.2 Sample Workflow Integration Plugin

A sample Workflow Integration Plugin is available in the "Examples/SamplePlugin" directory. In order for Resolve to register this plugin, this directory needs to be copied to 'Workflow Integration Plugins' root directory (mentioned in below section). Once a plugin is registered, plugin can be loaded from UI sub-menu under 'Workspace->Workflow Integrations'. This will load the plugin and show the plugin HTML page in a separate window.

Sample plugin helps to understand how a plugin should be structured and how it works with Resolve. Please refer to the directory/file structure, manifest file info, plugin loading, JavaScript API usage examples, etc. This sample plugin and scripts demonstrates few basic scriptable JavaScript API usages to interact with Resolve.

2.14.3 Loading Workflow Integration Plugin

On startup, DaVinci Resolve Studio scans the Workflow Integration Plugins root directory and enumerates all plugin modules. For each valid plugin module, it creates a UI sub-menu entry under 'Workspace->Workflow Integrations' menu. DaVinci Resolve Studio reads the basic details of the plugin from its manifest.xml file during load time. Once plugin is loaded, user can click on the 'Workflow Integrations' sub-menu to load the corresponding plugin.

2.14.4 Workflow Integration Plugin directory structure

```
com.<company>.<plugin_name>/
        package.js
        main.js
        index.html
        manifest.xml
        node_modules/
                <Node.js modules>
        js/
                <supporting js files>
        css/
                <css files containing styling info>
        img/
                <image files>
```

2.14.5 Workflow Integration Plugins root directory

User should place their Workflow Integration Plugin under the following directory:

Note: Mac OS X: "/Library/Application Support/Blackmagic Design/DaVinci Resolve/Workflow Integration Plugins/"

Windows: "%PROGRAMDATA%Blackmagic DesignDaVinci ResolveSupportWorkflow Integration Plugins"

2.14.6 Supported platforms

- Plugins: Windows, Mac OS X (not supported on Linux currently)
- Scripts: Windows, Mac OS X, Linux

2.14.7 Using scriptable JavaScript API

Scriptable JavaScript API execution happens under HTML environment like any typical website. Once HTML page is loaded it can execute scriptable JavaScript API as needed (like clicking on a button, etc.)

This example JavaScript snippet creates a simple project in DaVinci Resolve Studio:

```
const WorkflowIntegration = require('./WorkflowIntegration.node');
isInitialized = WorkflowIntegration.Initialize('com.blackmagicdesign.resolve.sampleplugin
↪');
if (isInitialized) {
```

(continues on next page)

(continued from previous page)

```
    resolve = WorkflowIntegration.GetResolve();
    resolve.GetProjectManager().CreateProject("Hello World");
}
```

The resolve object is the fundamental starting point for scripting via Resolve. As a native object, it can be inspected for further scriptable properties and functions in JavaScript.

2.14.8 WorkflowIntegration module API

To interact with Resolve you need to use 'WorkflowIntegration.node' Node.js module file in your plugin app. Below are the WorkflowIntegration (module) JavaScript API functions to communicate with Resolve.

WorkflowIntegration

- Initialize(<pluginId>) –> Bool # Returns true if initialization is successful, false otherwise. <pluginId> is the unique plugin id string configured in the manifest.xml file.
- GetResolve() –> Resolve # Returns Resolve object.
- RegisterCallback(callbackName, callbackFunc) –> Bool
  - Returns true if input callback name/function is registered successfully, false otherwise.
  - 'callbackName' should be a valid supported callback string name (refer to the below section 'Supported callbacks').
  - 'callbackFunc' should be a valid JavaScript function without any arguments.
- DeregisterCallback(callbackName) –> Bool # Returns true if input callback name is deregistered successfully, false otherwise.
- CleanUp() –> Bool # Returns true if cleanup is successful, false otherwise. This should be called during plugin app quit.
- SetAPITimeout(valueInSecs) –> Bool
  - By default, apis dont timeout. In order to enable timeout, set a non-zero positive integer value in the arg 'valueInSecs'.
  - Setting it to 0 will disable timeout. This function will return true if the timeout is set/reset successfully.

2.14.9 Supported callbacks

- 'RenderStart'
- 'RenderStop'

Please note that there is no console based support for JavaScript API.

2.14.10 Sample Workflow Integration Script

A sample Workflow Integration Python script is also available in the "Examples" directory. In order for Resolve to register this script, it needs to be copied to the 'Workflow Integration Plugins' root directory (mentioned in the above section).

Once a script is registered, it can be also loaded from the 'Workspace' menu, under 'Workflow Integrations'. This will invoke the script and show the sample UIManager window.

Workflow Integration scripts work similarly to other scripts in Resolve, and use the same scripting API. This example script provides a basic introduction into creating a popup Workflow application using a UIManager window, with simple layout of text fields and buttons, and event handlers to dispatch functions for integration with the user's facility. Alternatively, third-party UI managers such PyQt may be used instead, or no GUI at all.

When launched by Resolve, plugin scripts are automatically provided with 'resolve' and 'project' variables for immediate and easy access to Resolve's scripting API. Additional third-party modules may be imported for access to asset-management systems as desired.

2.14.11 UIManager Introduction

There are two main objects needed to manage a window, the UIManager that handles layout, and the UIDispatcher that manages interaction events, accessed as follows:

```
ui = fusion.UIManager()
dispatcher = bmd.UIDispatcher(ui)
```

Windows are created with the the UIDispatcher, passing a dictionary of attributes like ID and Text, with GUI elements in nested layouts all created with the UIManager.

2.14.12 UIDispatcher Functions

The UIDispatcher object has a few important functions to manage processing of events. The most important are:

- AddWindow(props, children): Accepts a dictionary of properties and a list of children, returns a Window object
- AddDialog(props, children): Accepts a dictionary of properties and a list of children, returns a Dialog object
- int RunLoop(): Call when your window is ready to receive user clicks and other events
- ExitLoop(int): Terminates the event processing, and returns any supplied exit code from RunLoop()

Common usage is to create your window and set up any event handlers, including a Close handler for the window that calls ExitLoop(), then Show() your window and call RunLoop() to wait for user interaction:

```
ui = fusion.UIManager
dispatcher = bmd.UIDispatcher(ui)

win = dispatcher.AddWindow({ 'ID': 'myWindow' }, [ ui.Label({ 'Text': 'Hello World!' })␣
↪])

def OnClose(ev):
        dispatcher.ExitLoop()

win.On.myWindow.Close = OnClose
```

(continues on next page)

(continued from previous page)

```
win.Show()
dispatcher.RunLoop()
```

AddWindow() will also accept a single child without needing a list, or a single dictionary containing both proprties and child elements, for ease of use.

As well as constructing new child elements and layouts, the UIManager also offers a few useful functions:

- FindWindow(ID): Returns an element with matching ID
- FindWindows(ID): Returns a list of all elements with matching ID
- QueueEvent(element, event, info): Calls the element's event handler for 'event', passing it the dictionary 'info'

2.14.13 UIManager Elements

The element's ID is used to find, manage, and dispatch events for that element. GUI elements also support a set of common attributes including Enabled, Hidden, Visible, Font, WindowTitle, BackgroundColor, Geometry, ToolTip, StatusTip, StyleSheet, WindowOpacity, MinimumSize, MaximumSize, and FixedSize. Some other common GUI elements and their main attributes include:

Element	Attributes
Label	Text, Alignment, FrameStyle, WordWrap, Indent, Margin
Button	Text, Down, Checkable, Checked, Icon, IconSize, Flat
CheckBox	Text, Down, Checkable, Checked, Tristate, CheckState
ComboBox:	ItemText, Editable, CurrentIndex, CurrentText, Count
SpinBox:	Value, Minimum, Maximum, SingleStep, Prefix, Suffix, Alignment, ReadOnly, Wrapping
Slider:	Value, Minimum, Maximum, SingleStep, PageStep, Orientation, Tracking, SliderPosition
LineEdit:	Text, PlaceholderText, Font, MaxLength, ReadOnly, Modified, ClearButtonEnabled
TextEdit:	Text, PlaceholderText, HTML, Font, Alignment, ReadOnly, TextColor, TextBackgroundColor, TabStopWidth, Lexer, LexerColors
ColorPicker:	Text, Color, Tracking, DoAlpha
Font:	Family, StyleName, PointSize, PixelSize, Bold, Italic, Underline, Overline, StrikeOut, Kerning, Weight, Stretch, MonoSpaced
Icon:	File
TabBar:	CurrentIndex, TabsClosable, Expanding, AutoHide, Movable, DrawBase, UsesScrollButtons, DocumentMode, ChangeCurrentOnDrag
Tree:	ColumnCount, SortingEnabled, ItemsExpandable, ExpandsOnDoubleClick, AutoExpandDelay, HeaderHidden, IconSize, RootIsDecorated,
	Animated, AllColumnsShowFocus, WordWrap, TreePosition, SelectionBehavior, SelectionMode, UniformRowHeights, Indentation,
	VerticalScrollMode, HorizontalScrollMode, AutoScroll, AutoScrollMargin, TabKeyNavigation, AlternatingRowColors,
	FrameStyle, LineWidth, MidLineWidth, FrameRect, FrameShape, FrameShadow
TreeItem:	Selected, Hidden, Expanded, Disabled, FirstColumnSpanned, Flags, ChildIndicatorPolicy

Some elements also have property arrays, indexed by item or column (zero-based), e.g. newItem.Text[2] = 'Third column text'

Combo: ItemText[] TabBar: TabText[], TabToolTip[], TabWhatsThis[], TabTextColor[] Tree: ColumnWidth[] Treeitem: Text[], StatusTip[], ToolTip[], WhatsThis[], SizeHint[], TextAlignment[], CheckState[], BackgroundColor[], TextColor[], Icon[], Font[]

Some elements like Label and Button will automatically recognise and render basic HTML in their Text attributes, and TextEdit is capable of displaying and returning HTML too. Element attributes can be specified when creating the element, or can be read or changed later:

```
win.Find('myButton').Text = "Processing..."
```

2.14.14 Elements Functions

Most elements have functions that can be called from them as well:

- Show()
- Hide()
- Raise()
- Lower()
- Close() Returns boolean
- Find(ID) Returns child element with matching ID
- GetChildren() Returns list
- AddChild(element)
- RemoveChild(element)
- SetParent(element)
- Move(point)
- Resize(size)
- Size() Returns size
- Pos() Returns position
- HasFocus() Returns boolean
- SetFocus(reason) Accepts string "MouseFocusReason", "TabFocusReason", "ActiveWindowFocusReason", "OtherFocusreason", etc
- FocusWidget() Returns element
- IsActiveWindow() Returns boolean
- SetTabOrder(element)
- Update()
- Repaint()
- SetPaletteColor(r,g,b)
- QueueEvent(name, info) Accepts event name string and dictionary of event attributes
- GetItems() Returns dictionary of all child elements

Some elements have extra functions of their own:

Element	Functions
Label	SetSelection(int, int), bool HasSelection(), string SelectedText(), int SelectionStart()
Button	Click(), Toggle(), AnimateClick()
CheckBox	Click(), Toggle(), AnimateClick()
ComboBox	AddItem(string), InsertItem(string), AddItems(list), InsertItems(int, list), InsertSeparator(int), RemoveItem(int), Clear(),
	SetEditText(string), ClearEditText(), Count(), ShowPopup(), HidePopup()
SpinBox	SetRange(int, int), StepBy(int), StepUp(), StepDown(), SelectAll(), Clear()
Slider	SetRange(int, int), TriggerAction(string)
LineEdit	SetSelection(int, int), bool HasSelectedText(), string SelectedText(), int SelectionStart(), SelectAll(), Clear(), Cut(),
	Copy(), Paste(), Undo(), Redo(), Deselect(), Insert(string), Backspace(), Del(), Home(bool), End(bool),CursorPositionAt(point)
TextEdit	InsertPlainText(string), InsertHTML(string), Append(string), SelectAll(), Clear(), Cut(), Copy(), Paste(), Undo(), Redo(),
	ScrollToAnchor(string), ZoomIn(int), ZoomOut(int), EnsureCursorVisible(), MoveCursor(moveOperation, moveMode), bool CanPaste()
	string AnchorAt(point), bool Find(string, findFlags)
TabBar	int AddTab(strubg), int InsertTab(string), int Count(), RemoveTab(int), MoveTab(int, int)
Tree	AddTopLevelItem(item), InsertTopLevelItem(item), SetHeaderLabel(string), int CurrentColumn(), int SortColumn(),
	int TopLevelItemCount(), item CurrentItem(), item TopLevelItem(int), item TakeTopLevelItem(int), item InvisibleRootItem(),
	item HeaderItem(), int IndexOfTopLevelItem(item), item ItemAbove(item), item ItemBelow(item), item ItemAt(point),
	Clear(), rect VisualItemRect(item), SetHeaderLabels(list), SetHeaderItem(item), InsertTopLevelItems(list),
	AddTopLevelItems(list), list SelectedItems(), list FindItems(string, flags), SortItems(int, order), ScrollToItem(item),
	ResetIndentation(), SortByColumn(int, order), int FrameWidth()
TreeItem	AddChild(item), InsertChild(item), RemoveChild(iitem), SortChildren(int, order), InsertChildren(int, list),AddChildren(list),
	int IndexOfChild(item), item Clone(), tree TreeWidget(), item Parent(), item Child(int), item TakeChild(int),
	int ChildCount(), int ColumnCount()
Window	Show(), Hide(), RecalcLayout()
Dialog	Exec(), IsRunning(), Done(), RecalcLayout()

Elements can be accessed by the window's FindWindow(id) function, or by assigning them to a variable for later usage, which is more efficient. The GetItems() function will return a dictionary of all child elements for ease of access.

2.14.15 UIManager Layout

Additionally, elements can be nested to define layout, using the HGroup and VGroup elements. As with Window and other elements, tou can pass a single dictionary or list with all properties and children, or separate them into a dict of properties and list of children, for convenience:

```
winLayout = ui.VGroup([
        ui.Label({ 'Text': "A 2x2 grid of buttons", 'Weight': 1 }),

        ui.HGroup({ 'Weight': 5 }, [
                ui.Button({ 'ID': "myButton1",  'Text': "Go" }),
                ui.Button({ 'ID': "myButton2",  'Text': "Stop" }),
                ]),
        ui.VGap(2),
        ui.HGroup({ 'Weight': 5 }, [
                ui.Button({ 'ID': "myButtonA",  'Text': "Begin" }),
                ui.Button({ 'ID': "myButtonB",  'Text': "End" }),
                ]),
        ]),
win = dispatcher.AddWindow({ 'ID': "myWindow" }, winLayout)
```

HGap and VGap elements can included for finer spacing control. Note also the Weight attribute, which can be applied to most elements to control how they adjust their relative sizes. A Weight of 0 will use the element's minimum size.

2.14.16 Event Handlers

Window objects will call user-defined event handler functions in response to various interaction events. Event handlers are managed using a window member called 'On'. This has sub-members for each GUI element with an ID, and those have members for each available event. To set up an event handler, define a function for it, then assign the function to the window's On.ID.Event member as follows:

```
def OnClose(ev):
        dispatcher.ExitLoop()

win.On.myWindow.Close = OnClose
```

Alternatively, if your object's ID is stored in a string variable called 'buttonID', you could use:

```
win.On[buttonID].Clicked = OnButtonClicked
```

Many objects have specific events that can be handled:

- Button: Clicked, Toggled, Pressed, Released
- CheckBox: Clicked, Toggled, Pressed, Released
- ComboBox: CurrentIndexChanged, CurrentTextChanged, TextEdited, EditTextChanged, EditingFinished, ReturnPressed, Activated
- SpinBox: ValueChanged, EditingFinished
- Slider: ValueChanged, SliderMoved, ActionTriggered, SliderPressed, SliderReleased, RangeChanged
- LineEdit: TextChanged, TextEdited, EditingFinished, ReturnPressed, SelectionChanged, CursorPositionChanged
- TextEdit: TextChanged, SelectionChanged, CursorPositionChanged

- ColorPicker: ColorChanged
- TabBar: CurrentChanged, CloseRequested, TabMoved, TabBarClicked, TabBarDoubleClicked
- Tree:
  - CurrentItemChanged, ItemClicked, ItemPressed, ItemActivated, ItemDoubleClicked, ItemChanged, ItemEntered,
  - ItemExpanded, ItemCollapsed, CurrentItemChanged, ItemSelectionChanged
- Window:
  - Close, Show, Hide, Resize, MousePress, MouseRelease, MouseDoubleClick, MouseMove, Wheel, KeyPress, KeyRelease,
  - FocusIn, FocusOut, ContextMenu, Enter, Leave

Event handler functions are called with a dictionary of related attributes such as who, what, when, sender, and modifiers. Common events and some additional attributes they receive include:

- MousePress: Pos, GlobalPos, Button, Buttons
- MouseRelease: Pos, GlobalPos, Button, Buttons
- MouseDoubleClick: Pos, GlobalPos, Button, Buttons
- MouseMove: Pos, GlobalPos, Button, Buttons
- Wheel: Pos, GlobalPos, Buttons, Delta, PixelDelta, AngleDelta, Orientiation, Phase
- KeyPress: Key, Text, IsAutoRepeat, Count
- KeyRelease: Key, Text, IsAutoRepeat, Count
- ContextMenu: Pos, GlobalPos
- Move: Pos, OldPos
- FocusIn: Reason
- FocusOut: Reason

Event handlers can be enabled or disabled for a given element by turning them on or off in the Events attribute:

```
ui.Slider({ 'ID': 'mySlider', 'Events': { 'SliderMoved': true } })
```

Some common events like Clicked or Close are enabled by default.

2.14.17 Basic Resolve API

Please refer to the *Basic Resolve API section* for the list of the functions that Resolve offers for scripted control. For plugin scripts, the 'resolve' and 'project' variables are automatically set up for you, and may be used to access any part of Resolve's API.

2.14.18 Further Information

This document provides a basic introduction only, and does not list all available UIManager elements and attributes. As UIManager is based on Qt, you can refer to the Qt documentation at https://doc.qt.io/qt-5/qwidget.html for more information on element types and their attributes. There are also many third-party examples and discussions available on user forums for DaVinci Resolve and Fusion Studio.