limekit docs

Release 1.0

Omega Msiska

Jan 29, 2024

CONTENTS

1 Let's get you started! 3

2 Showcase 5

3 Getting help 9

4 Support the project 11

4.1 Installation . . . 11
4.2 Widgets . . . 13
4.3 Widget Items . . . 39
4.4 Layout Managers . . . 42
4.5 App object . . . 45
4.6 Accessing Resources . . . 49
4.7 Batteries included . . . 49

Index 53

Version
1.0

Contact
omegamsiskah@gmail.com

Author
Omega Msiska

CHAPTER

ONE

LET'S GET YOU STARTED!

Limekit is a framework (wrapper for PySide6) for building desktop applications using the lua language without the need for HTML and CSS. The framework allows developers to maintain single lua codebase and create cross-platform apps that work on Windows, macOS, and Linux.

Note: The framework is being created in Python, but there's no need for you to learn Python at all.

Important: This is a test release deliberately lacking numerous features and properties.

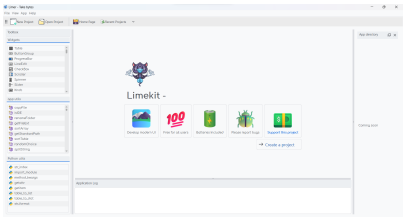
Presently, it only supports Windows and Linux, with MacOS support arriving soon.

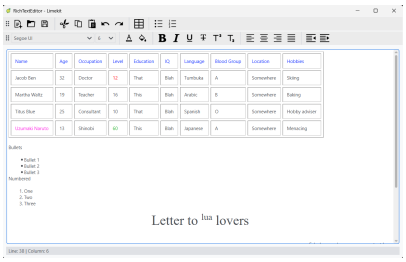
- *Part 1: Setup*: Guides you step-by-step to set up Python on your computer, which is crucial as the framework depends on the Python language.
- *Part 2: Widgets*: Focuses on interactivity. Think buttons, combo boxes, menus, check boxes, radio buttons and many more - they're all in the mix!
- *Part 3: Widget Items*: Covers how to interact with some special widgets such as the Tab, ToolBar, etc.
- *Part 4: Layout Managers*: Covers the different layouts available in the framework
- *Part 5: App object*: Shows you how to make the most of the utilities in the framework and all the extra goodies that come with it!
- *Part 6: User resources*: Shows you how to make the most of the utilities in the framework and all the extra goodies that come with it!
- *Part 7: Batteries included*: Covers all the other features provided by the framework, such as the sqlite3 database, using the system tray, displaying system notifications, threads, signals, and many more.

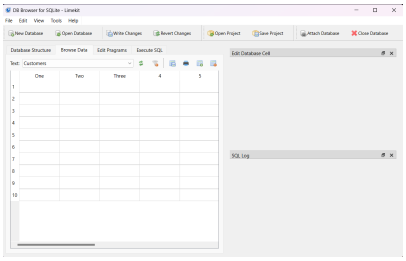
CHAPTER

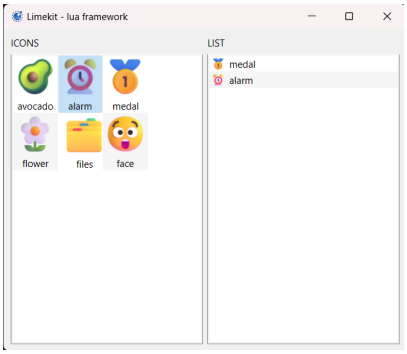
TWO

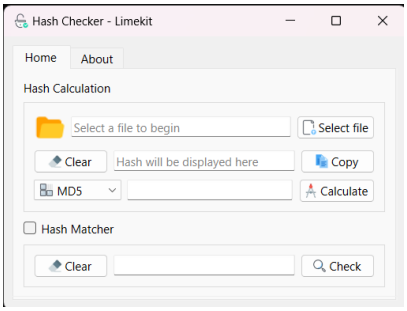
SHOWCASE

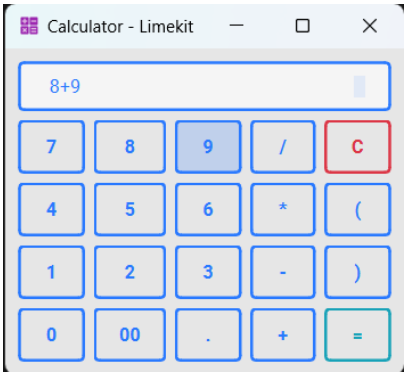


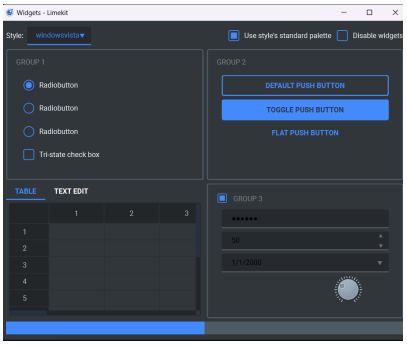


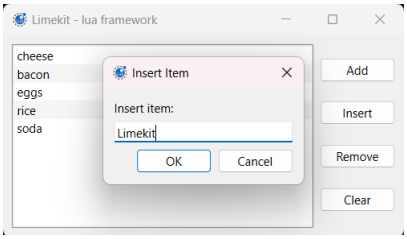


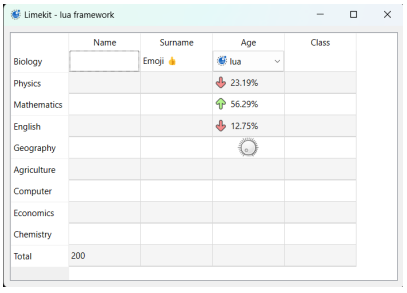


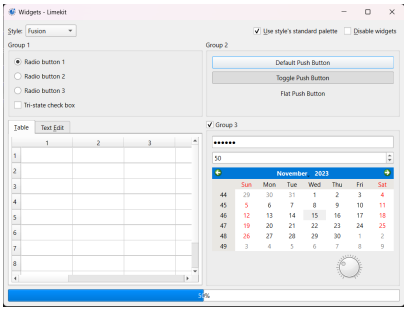

Hash Checker - Limekit
Home
About
Hash Calculation
Select a file to begin
Select file
Clear
Hash will be displayed here
Copy
MD5
Calculate
Hash Matcher
Clear
Check

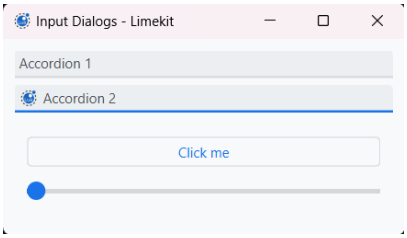

Calculator - Limekit
8+9
7
8
9
/
C
4
5
6
*
(
1
2
3
-
)
0
00
.
+
=

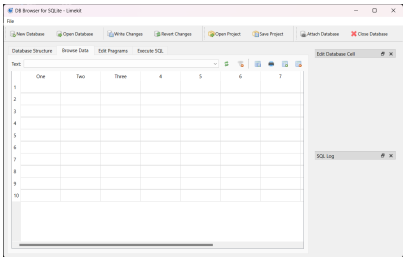

Widgets - Limekit
Style:
windowsvista
Use style's standard palette
Disable widgets
GROUP 1
Radiobutton
Radiobutton
Radiobutton
Tri-state check box
GROUP 2
DEFAULT PUSH BUTTON
TOGGLE PUSH BUTTON
FLAT PUSH BUTTON
TABLE
TEXT EDIT
GROUP 3


Limekit - lua framework
cheese
bacon
eggs
rice
soda
Insert Item
Insert item:
Limekit
OK
Cancel
Add
Insert
Remove
Clear


Limekit - lua framework
Name
Surname
Age
Class
Biology
Emoji
lua
Physics
23.19%
Mathematics
56.29%
English
12.75%
Geography
Agriculture
Computer
Economics
Chemistry
Total
200


Widgets - Limekit
Style: Fusion
Use style's standard palette
Disable widgets
Group 1
Radio button 1
Radio button 2
Radio button 3
Tri-state check box
Group 2
Default Push Button
Toggle Push Button
Flat Push Button
Table
Text Edit
Group 3
50
November 2023
Sun Mon Tue Wed Thu Fri Sat
44 29 30 31 1 2 3 4
45 5 6 7 8 9 10 11
46 12 13 14 15 16 17 18
47 19 20 21 22 23 24 25
48 26 27 28 29 30 1 2
49 3 4 5 6 7 8 9
51%


Input Dialogs - Limekit
Accordion 1
Accordion 2
Click me


DB Browser for SQLite - Limekit
File
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Edit Database Cell
SQL Log
One
Two
Three

CHAPTER

THREE

GETTING HELP

Having trouble?

Head over to our discord server Try asking in our r/limekit reddit community

Or contact me on omegamsiskah@gmail.com

CHAPTER

FOUR

SUPPORT THE PROJECT

Buy me a coffee to support the project. This will help me stay awake at night

visit buymeacoffee.com/omegamsiska

4.1 Installation

Content

- *What's Limekit?*
- *Installing Python*
- *Done installing. What's next?*

4.1.1 What's Limekit?

Limekit is framework for building desktop applications using the lua language without the need for HTML and CSS. On the other hand allowing developers to maintain one lua codebase and create cross-platform apps that work on `Windows`, `macOS`, and `Linux`.

~ It is being developed by company called `Take bytes`, with `Omega Msiska` as the lead developer on this project.

Key Features:

- **Modern UI**: Limekit allows developers to craft beautiful UI's with dark and light modes available.
- **Simplicity**: One of the notable features of Limekit is it's ability to create a working program in under 2 lines of code as shown below.

```
local window = Window{title='Limekit app', icon=images('app.png')}
window:show()
```

- This is basically enough for Limekit to run your program. Mind blowing right?

Guess what?

- **No C, C++ or python knowledge**: You don't need to know any python, C or C++ to develop programs in `Limekit`, just lua
- **Cross-platform**: Run the same code base in Windows, Linux and macOS
- **Free**: The framework is free to use

4.1.2 Installing Python

The Limekit framework is built in Python 3.10, so **`you'll need to have Python 3.10 installed`** to use the framework. Follow the tutorial to get Python 3.10 installed on your OS.

Note: This guide is for those who haven't delved into Python before and are installing it for the very first time.

Windows

- Installing python on Windows is pretty straight forward. Simply visit python's website to download for your system

Once the installation is complete, open your terminal (Comamnd Prompt or PowerShell) and type the following command:

```
$ python
```

If you get a similar output as the one below, you are good to go!

```
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug  1 2022, 21:53:49) [MSC v.1932 64 bit (AMD64)]␣
↪on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Navigate to the bottom of the page to follow through the final stages

Linux

Note: The framework has only been tested on Ubuntu, but it's expected to function on other Linux distros as well.

Important: Before trying to install python3.10, try executing `python3.10` or `python3 --version` in the terminal to see if python is already installed, as most Linux distributions come with python installed

---

Head over to this website to learn how to install python3.10 on your Linux

4.1.3 Done installing. What's next?

Important: Read the below instructions before downlaoding anything

All required files should be downloaded from the `Releases` page (right hand side) in the github links provided

`Installation requires an active internet connection`

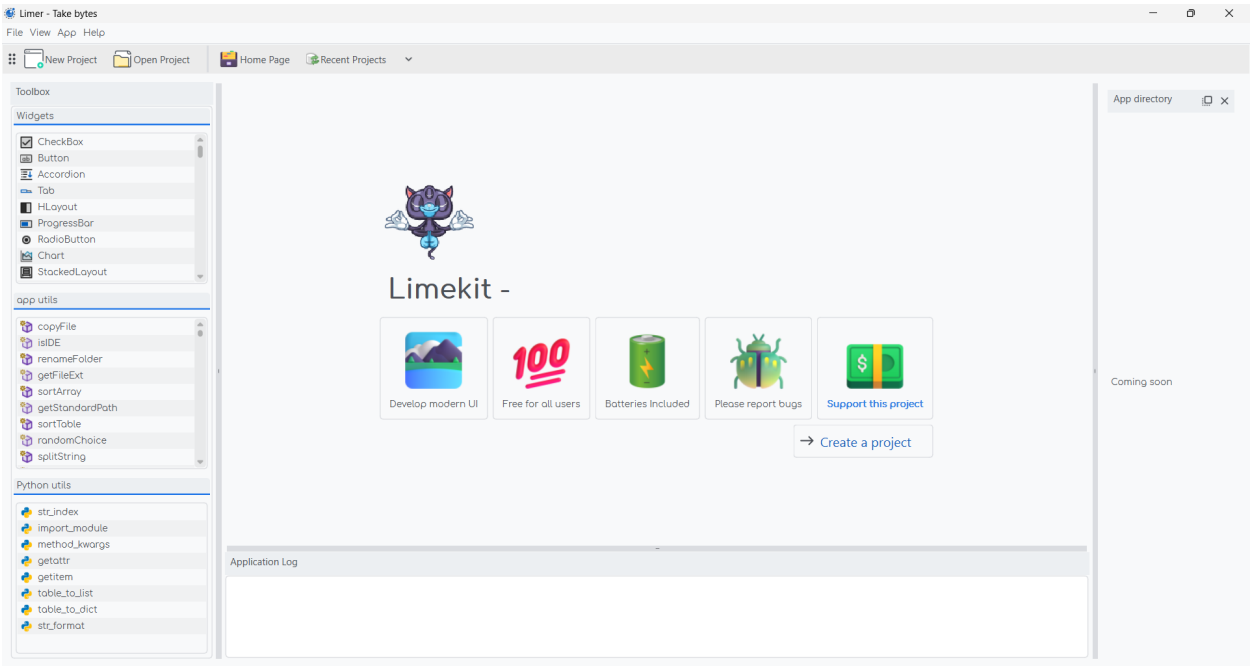
Head over to our github repo to download `Limer`.

There are `over 30 examples` prepared for your journey in Limekit, just click here to downlaod them

Note: `Limer` is the program that `only` runs your apps. It's not an IDE or an editor.

Download all the zip files from the `Releases` section and extract them. Inside the windows-only.zip or linuxmac-only.zip there should be a `READ ME.txt` file that explains everything.

If everything goes as planned, you'll be greeted by a screen similar to the one shown below.



You can now start learning how to use widgets.

4.2 Widgets

Widgets are graphical elements or components that make up the user interface of an application. They can be buttons, text boxes, labels, windows, checkboxes, sliders, and more. These widgets allow users to interact with the program by providing input, displaying information, or triggering specific actions. This framework provides a wide range of widgets that can be arranged to create a functional and visually appealing user interface.

Presently, there are 35 widgets at your disposal within the framework. Considering the framework is new, the count is subject to change. Anticipate additional widgets as the framework evolves.

Important: Every app requires a Window to function properly. In your `main.lua` file, create a window object like shown below.

```
local window = Window{title = 'Limekit', icon = images('app.png'), size={200,200}}
window:show()
```

This creates an interactive window that serves your layouts holding all of your widgets.

checkout using *Window*

Important: All widgets can be added to a layout using the *addChild* function

Hey, heads up!

Before you start, you might want to take a look at how to access your resources. Just head over *here*

For a comprehensive understanding of their usage, explore the available widgets in-depth:

4.2.1 Accordion

A powerful component designed to manage and display multiple tabs, each hosting distinct set of widgets or layouts but only one item can stay open at a time

```
local accordion = Accordion()
```

You can add a widget or a layout to the accordion using `addChild(widget, label, icon: optional)` or `addLayout(layout, label, icon: optional)`, respectively. Take a look at the different code snippet below

Adding a widget

```
local button = Button('Click me')
accordion:addChild(button, 'All Buttons')

-- or otherwise with an icon

local button = Button('Click me')
accordion:addChild(button, 'All Buttons', images('icon.png'))
```

Adding a layout

```
local layout = VLayout()
local button = Button('Click me')

layout:addChild(button)
accordion:addChild(layout, 'All Buttons')

-- or otherwise with an icon

local layout = VLayout()
local button = Button('Click me')
```

(continues on next page)

(continued from previous page)

```
layout:addChild(button)
accordion:addChild(layout, 'All Buttons')
```

checkout Layout Managers

Properties

addChild(*widget*, *label*, *icon: optional*)

Adds a widget in a tab to the accordion with given a label, and icon if necessary

addLayout(*layout*, *label*, *icon: optional*)

Adds a layout in a tab to the accordion with given a label, and icon if necessary

checkout *Using resources*

setToolTip(*text*)

Enable text that appears when a mouse hovers on the tab

getCurrentIndex()

Gets the current index of the visible tab

setCurrentIndex(*index*)

Sets the index of the tab to be visible

4.2.2 Button

The button, one of the mostly used widget in any Graphical User Interface (GUI). You can label it with text or even include icons to make it more visually informative. It's commonly used to trigger actions or functions within an application when clicked.

```
local button = Button(text)
```

This is done by calling a callback function after clicking on a button. You can specify such a callback function with the `setOnClick` method

```
1 button:setOnClick(function(sender)
2   -- some task here
3 end)
```

Properties

setOnClick(*callback*)

The function executed when the button is clicked

setIcon(*path*)

Sets an icon on the button

checkout *Using resources*

setIconSize(*width*, *height*)

Resizes the icon set in the button

setText(*text*)

Sets text on the button

getText()

Returns text on the button

setFlat()

Makes the button appear flat

setCheckable(*checkable: bool*)

Set whether or not the button can be checked/toggled

isChecked()

Return the check/toggle status of the button

setMenu(*menu*)

Sets menu to the button

setMargins(*left*, *top*, *right*, *bottom*)

Sets the margins of the button

checkout *working with menus*

4.2.3 ButtonGroup

This widgets helps you manage a group of *check boxes* or *radio buttons*, making sure that only one can be selected at a time. It simplifies handling of multiple buttons by letting you know which one's clicked or selected within the group.

```
local group = GroupBox()
```

You can add buttons (radio button and check box) to the widget using the `addButton(button)`

```
local maleButton = RadioButton('Male')
local femaleButton = RadioButton('Female')

group:addButton(maleButton)
group:addButton(femaleButton)
```

Important: Don not add the `ButtonGroup` to any layout. First, add the button to the `ButtonGroup`. After that, add the button to any layout you prefer, as shown below.

```
local boolean = RadioButton('True')

group:addButton(boolean)
layout:addChild(boolean)
```

Properties

addChild(*widget*, *label*, *icon: optional*)

Adds a widget in a tab to the accordion with a given label, and an icon if necessary

addLayout(*layout*, *label*, *icon: optional*)

Adds a layout in a tab to the accordion with a given label, and an icon if necessary

checkout *Using resources*

setToolTip(*text*)

Tooltips are brief informational messages that appear when the user hovers the mouse pointer over the tab

getCurrentIndex()

Gets the current index of the visible tab

setCurrentIndex(*index*)

Sets the index of the tab to be visible

4.2.4 Calendar

A powerful component designed to manage and display multiple tabs, each hosting distinct set of widgets or layouts.

```
local accoridon = Accordion()
```

You can add add a widget or a layout to the accordion using `addChild(widget, label, icon: optional)` or `addLayout(layout, label, icon: optional)`, respectively. Take a look at the different code snippet below

Adding a widget

```
local button = Button('Click me')
accordion:addChild(button, 'All Buttons')

-- or otherwise with an icon

local button = Button('Click me')
accordion:addChild(button, 'All Buttons', images('icon.png'))
```

Adding a layout

```
local layout = VLayout()
local button = Button('Click me')

layout:addChild(button)
accordion:addChild(layout, 'All Buttons')

-- or otherwise with an icon

local layout = VLayout()
local button = Button('Click me')

layout:addChild(button)
accordion:addChild(layout, 'All Buttons')
```

checkout Layout Managers

Properties

addChild(*widget*, *label*, *icon: optional*)

Adds a widget in a tab to the accordion with given a label, and icon if necessary

addLayout(*layout*, *label*, *icon: optional*)

Adds a layout in a tab to the accordion with given a label, and icon if necessary

checkout *Using resources*

setGridVisible(*visible: bool*)

Show or hide grid lines on the calendar

setToolTip(*text*)

Enable text that appears when a mouse hovers on the tab

getCurrentIndex()

Gets the current index of the visible tab

setCurrentIndex(*index*)

Sets the index of the tab to be visible

4.2.5 CheckBox

A small box that can be checked or unchecked by users. It's used to toggle something on or off in an application, like enabling or disabling a feature or selecting options.

```
local check = CheckBox(text)
```

Properties

setOnCheck(*callback*)

Executed when checked. Arguments passed include self and state.

setIcon(*path*)

Sets an icon on the check box

checkout *Using resources*

setIconSize(*width*, *height*)

Sets the icon size

setToolTip(*text*)

Tooltips are brief informational messages that appear when the user hovers the mouse pointer over the tab

getTooltip()

Returns the tooltip text

setToolTipDuration(*check: bool*)

Set how long the tooltip displays

getCheck()

Returns check status; true or false

setCheck(*check: bool*)

Sets the box to be checked or not

setText(*text*)

Sets the check box text

getText()

Gets the check box button

4.2.6 ComboBox

A widget that presents a dropdown list of predefined options for users to select from. It's commonly used to offer users a range of choices from a list of items or categories within an application. Users can select one option from the list.

```
local fruits = ComboBox(data: table)

-- or like

local fruits = ComboBox({'tomato', 'apple', 'pear', 'cherry'})
```

Properties

setOnItemSelected(*callback*)

Executed when an item is selected from the list.

getText()

Gets selected item text

addImageItem(*data: table*)

Add an item with an icon alongside it. Use the following format: `{'limekit', images('lua.png')}`

addImageItems(*data: table*)

Same as above method, only acceptng build data. Use the following format: `{{'limekit', images('lua.png')}, {'apple', images('icon.png')}, ...}`

addItem(*text*)

Adds a single item to the widget

setItems(*data: table*)

Sets data to the widget. Use the following format: `{'item 1',' item 2', 'item 3', ...}`

4.2.7 CommandButton

A button that combines a label and a command. It's designed to present descriptive text alongside an actionable command or link. This type of button is useful for displaying informative text or explanations along with a clickable action, providing more context to users about the function the button performs.

```
local date = CommandButton(text)
```

Properties

setOnClick(*callback*)

The function executed when the button is clicked

setText(*text*)

Sets text on the button

getText()

Returns text on the button

setDescription()

Sets the a description. This appears as text beneath the primary text

getDescription()

Returns the provided description

4.2.8 DatePicker

A widget in PyQt specifically designed for editing dates. It provides a user-friendly interface for selecting dates through a calendar or by manually inputting the date. It's handy for tasks requiring date selection or entry, ensuring users can easily input or modify dates in an application

```
local date = DatePicker()
```

Properties

setOnDatePick(*callback*)

The function executed when the button is clicked

Params
`self` and `date`

setDate(*self*, *year*, *month*, *day*, *hour: optional*, *minutes: optional*)

Sets the date

getDate()

Returns the date in year-month-day format

4.2.9 Dock

A versatile element used to create dockable panes or windows within an application's main window. These dock widgets can be moved, resized, and placed in various positions, such as docking them to the edges or floating them as separate windows. They're commonly used to provide additional panels or toolbars that users can arrange according to their preferences for a more customized user interface experience.

```
local dock = Dock()
```

Properties

setProperties(*properties*)

Sets rules on how the dock widget behaves. Available properties include `floatable`, `movable`, `closable` and `nil`

Parameters
properties – lua table

setMagneticAreas(*areas*)

Sets the areas the dock widget is allowed to float to. Available areas include `left`, `top`, `right`, `bottom`, `all` and `nil`

Parameters
areas – lua table

setTitle(*text*)

Sets text displayed on the dock widget

addChild(*child*)

Add a widget to the widget

setLayout(*layout*)

Add a layout to the widget

4.2.10 DoubleSpinner

A specialized input box designed for entering floating-point (decimal) numbers. It allows users to adjust and input numeric values with decimal precision using a spinning mechanism. You can set a range for the values, define the step size for increments or decrements, and control the number of decimal places displayed, making it ideal for precise numeric input.

```
local dock = DoubleSpinner()
```

Properties

setOnValueChange(*callback*)

The function executed when the value changes

Params
`self` and `value`

setRange(*start*, *end*)

Sets the `minimum` and `maximum` values for the widget

setValue(*value*)

Sets the value displayed on the widget

getValue()

Returns the current value

setPrefix(*prefix*)

Set the prefix of the spinner widget

setSuffix(*suffix*)

Set the suffix of the spinner widget

4.2.11 GifPlayer

A widget that allows you to view and play GIF (Graphics Interchange Format) files. It displays the animated or sequence of images contained within a GIF file, enabling you to watch the animation, which could be anything from a short loop to a longer sequence of frames. Typically, a GIF player provides controls to start, pause, stop, or navigate through the frames of the GIF, giving users control over the playback of the animation.

```
local gif = GifPlayer(path)
```

- GIF's play automatically.

Note: There's an issue with how GIFs are displayed in the Qt framework we're using. The output always ends up pixelated and in poor quality. Until the Qt developer addresses this problem, there's nothing we can do from our end.

Properties

`start()`
Starts playing the GIF

`stop()`
Stops playing the GIF

`pause()`
Pause the GIF animation loop

`nextFrame()`
Jumps to the next frame

`getCurrentFrame()`
Gets the current frame

`justToFrame()`
Navigates to a specified frame

`getState()`
Get state of the animation: running, `notrunning`, `paused` and `running`

4.2.12 GroupBox

A container that groups and organizes related widgets together. It provides a visual frame or box around the grouped widgets, along with an optional title, making it easier for users to understand the relationship between these widgets. It helps in structuring and presenting parts of a user interface, keeping elements organized and visually connected within an application

```
local group = GroupBox(title: optional)
```

Important: You can only set a layout to the widget and not add widgets directly

Note: using `setChecked(false)` disables all the widgets in the widget. However, this only applies once `setCheckable(true)` has been used.

Properties

setLayout(*layout*)
Set a primary layout for the widget

setBackgroundColor(*color*)
Sets background color for the widget

setTitle(*title*)
Sets the title for the widget

setToolTip(*text*)
Tooltips are brief informational messages that appear when the user hovers the mouse pointer over the tab

setToolTipDuration(*duration*)
Set how long the tooltip displays

getTooltip()
Returns the tooltip text

getCheck()
Returns check status; `true` or `false`

setCheck(*check: bool*)
Sets the box to be checked or not

getTitle()
Gets the title for the widget

4.2.13 HLine

A visual element used to separate or demarcate sections within a user interface. It's a simple line drawn horizontally across a window or widget, often used to visually divide different parts of the interface or to create a visual distinction between sections of content. This line is typically used for aesthetic purposes to improve the layout and readability of the user interface components.

```
local line = HLine()
```

Properties

Note: There are currently no properties available

4.2.14 Image

This widget allows developers to load and display images from files or resources, enabling users to view visual content as part of the application's design or functionality.

```
local image = Image(path)
```

Properties

setOnClick(*callback*)

Executed when the image is clicked.

setImage(*path*)

Sets the current image

setImageAlign(*align*)

Set how the image is aligned. Available options include `left`, `right`, `bottom`, `top`, `center`, `hcenter` and `vcenter`

resizeImage(*width*, *height*)

Resizes the images to the specified dimensions

4.2.15 Knob

A circular input control that allows users to interact by rotating it, usually to select numeric values within a defined range. It's commonly used for adjusting settings like volume, brightness, or any other parameter that can be controlled by rotating a knob. The circular design of the widget provides a visually intuitive way for users to modify values in an application

```
local knob = Knob()
```

Properties

setOnValueChanged(*callback*)

The function executed when the button is clicked

Params
`self` and `value`

setRange(*text*)

Sets the `minimum` and `maximum` values for the widget

setValue(*value*)

Sets current value for the widget

setMinValue(*value*)

Sets the minimum value for the widget

setMaxValue(*value*)

Sets the maximum value for the widget

getValue()

Get current value

setIndicators(*enable: bool*)

Set whether or not markers are visible on the widget

isIndicatorVisible()

Check if markers are visible or not

4.2.16 Label

A widget primarily used for showing text in a user interface. These widgets are often used to provide descriptions, titles, or informative messages within an application. They help in presenting static content that users can read or refer to as part of the interface design.

```
local label = Label(text)
```

Properties

setOnClick(*callback*)

Executed when the widget is clicked.

setText(*text*)

Sets text on the label

getText()

Returns the text displayed on the widget

setTextAlign(*align*)

Set how the text is aligned. Available options include `left`, `right`, `bottom`, `top`, `center`, `hcenter` and `vcenter`

setWordWrap(*enable: bool*)

Enable or disable word wrap

setBold(*enable: bool*)

Enable or disable bold text

setTextSize(*size*)

Sets text size

setCompanion(*widget*)

When the user hits the specific shortcut key shown on the label, the keyboard focus moves to the widget linked with that label.

The companion system works exclusively for `Labels` that have text where a single character has an `'&'` symbol before it. This '&' character is assigned as the shortcut key.

4.2.17 LineEdit

An input field where users can enter and edit a single line of text. It's commonly used for gathering textual information from users, such as usernames, passwords, search queries, or any other type of short text input.

```
local input = LineEdit(text: optional)
```

Properties

setOnTextChange(*callback*)

The function executed text inside changes

setOnReturnPress(*callback*)

The function executed when `Enter` key has been pressed

setOnTextSelection(*callback*)

The function executed when text inside the widget get's selected

Params
`self` and `text`

setInputMode(*callback*)

Sets how text input is handled. Available options `normal`, `hideinput`, `passwordonedit` and `password`

setText(*text*)

Sets text in the widget

getText()

Returns the text input

setHint(*text*)

Sets hint/placeholder text in the widget

getSelectedText()

Returns selected text

checkTextSelected()

Returns if widget has any text selected

setReadOnly(*enable: bool*)

Sets widget to read-only or allow input

redo()

Redo text input

undo()

Undo text input

setMaxLength()

Sets the maximum input length in the widget

selectAll()

Selects all text

getStartSelection()

Returns the start cursor position for selected text

getEndSelection()

Returns the end cursor position for selected text

getSelectionLength()

Returns count for text selected

4.2.18 ListBox

A widget used to display a list of items. It allows users to view a collection of items arranged in a list format within an application. Each item in the list can contain text and/or icons. This widget provides functionalities to add, remove, select, and manipulate items in the list, making it a versatile tool for presenting and managing lists of information or options in an interface.

```
local list = ListBox(data: optional)
```

Properties

setOnItemSelect(*callback*)

The function when an item is selected

Params
`self`, `text` and `index`

setItemViewMode(*callback*)

Sets how items are displayed in the widget: Available options: `icons` and `list`

setItems(*data: table*)

Sets data to the widget. Use the following format: `{'item 1', 'item 2', 'item 3', ...}`

addImageItem(*data: table*)

Add an item with an icon alongside it. Use the following format: `{'limekit', images('lua.png')}`

addImageItems(*data: table*)

Same as above method, only acceptng build data. Use the following format: `{{'limekit', images('lua.png')}, {'apple', images('icon.png')}, ...}`

addItem(*text*)

Adds a single item to the widget

removeItem(*row*)

Adds a single item to the widget

getCurrentRow(*text*)

Returns row of selected item

getText()

Returns selected item's text

getinsertItemAtText(*row*, *text*)

Inserts an item at a specified row

setAltRowColors()

Sets alternating colors to the rows

setIconSizes(*width*, *height*)

Sets icons to a specified sizes

getTextAt(*row*)

Gets text at a specified row

getItemsCount()

Returns total items available

clear()

Removes all item in the list

setAllowDragDrop(*enable: bool*)

Enables or disables dragging and dropping of items

setDragEnabled(*enable: bool*)

Enables or disables drag

4.2.19 Menu

A customizable pop-up menu that appears in response to user actions, such as clicking a button. It provides a list of options or actions for users to choose from, typically displayed as a drop-down menu. Developers can populate a menu with various menu items, submenus and separators to create a hierarchy of actions or choices. `Menu` is commonly used with other widgets, like buttons or toolbars or the system tray, to offer users a set of actions or options in a neatly organized menu format, improving the usability and functionality of the application.

```
local menu = Menu()
```

Properties

buildFromTemplate(*template: table*)

A powerful features that allows developing complex menus using tables.

```
local menubar = MenuBar();
menubar:buildFromTemplate({{
      label = '&File',
      submenu = {{
          label = 'New File',
          name = 'new_file',
          shortcut = "Ctrl+N",
          icon = images('newfile.png'),
          click = createFileFunc}}
    },
    {
      label = 'Help',
      submenu = {{
        label = 'About'}}
    }})
```

4.2.20 MenuBar

This widget provides a horizontal bar typically placed at the top of the application window, containing various menus. Each menu can hold menu items, which when clicked, can trigger specific actions or open sub-menus. The MenuBar widget is an essential component for organizing and providing access to different functionalities or commands within an application.

```
local menubar = MenuBar()
```

checkout MenuItems

Note: It is highly recommended to use the `buildFromTemplate(template)` as this saves you a lot of time from individualy creating `MenuItem` objects. This powerful features uses tables

Properties

buildFromTemplate(*template: table*)

A powerful features that allows developing complex menus using tables.

```
local menubar = MenuBar();
menubar:buildFromTemplate({{
      label = '&File',
      submenu = {{
          label = 'New File',
          name = 'new_file',
          shortcut = "Ctrl+N",
          icon = images('newfile.png'),
          click = createFileFunc}}
    },
    {
      label = 'Help',
      submenu = {{
        label = 'About'}}
    }})
```

- The preceding code example illustrates the capabilities of the `buildFromTemplate` function, showcasing its power in creating a sample menubar using tables

getChild(*name*)

Returns the `MenuItem` widget assigned with the name. This only applies when `name` was used on a `submenu` inside `buildFromTemplate`.

4.2.21 Modal

A window that pops up to interact with users, often for specific tasks or information gathering. It's a specialized window used to prompt users for input, display information, or perform actions that require user interaction.

```
local modal = Modal(window, title)
modal:show()
```

Note: You can only set one primary layout for each modal

Properties

setOnShown(*callback*)

Executed whenever the window is shown or displayed on the screen.

setOnClose(*callback*)

Executed when the window is closing.

Params

`self` and `event`: use `ignore()` and `accept()` on `event`; `event.ignore()` or `event.accept()`

setOnResize(*callback*)

Executed whenever the window is being resized.

minimize()

Minimizes or iconifies a window to the system taskbar or dock. It shrinks the window and places an icon representing the window in the taskbar or dock, allowing users to easily restore the window later

setMinSize(*width*, *height*)

Sets the minimum size

setMaxHeight(*height*)

Sets the maximum height

setMinHeight(*height*)

Sets the minimum height

setMaxWidth(*width*)

Sets the maximum width

setMinWidth(*width*)

Sets the minimum width

setMaxSize(*width*, *height*)

Sets the maximum size

setTitle(*title*)

Sets the title for the window

setMainWidget(*widget*)

Sets any widget as the central widget, causing it to take up the available space.

setSize(*width*, *height*)

Sets the size of the window

setLayout(*layout*)

Sets a primary layout for the window

setIcon(*path*)

Sets the icon for the window

setFixedSize(*width*, *height*)

Sets a fixed size to restrict resizing the window

show()

Shows the window

dismiss()

Closes the modal

4.2.22 ProgressBar

A visual widget that shows the progress of a task or an operation. It appears as a bar that fills up gradually, representing the completion status of a process.

```
local progress = ProgressBar()
```

Properties

setRange(*start*, *end*)

Sets the progress range

setValue(*value*)

Sets the current value

getValue()

Returns the current value

setOrientation(*orientation*)

Sets the orientation: `horizontal` or `vertical`

4.2.23 RadioButton

This functions as a button that allows toggling between being selected (checked) or deselected (unchecked). Generally, these buttons offer users a choice among several options where only one can be selected at a time. Within a set of radio buttons, selecting a new option automatically deselects the previously chosen one.

Properties

```
local check = RadioButton(text)
```

setOnCheck(*callback*)

Executed when checked. Arguments passed include self and state.

setIcon(*path*)

Sets an icon on the radio button

checkout *Using resources*

setIconSize(*width*, *height*)

Sets the icon size

setToolTip(*text*)

Tooltips are brief informational messages that appear when the user hovers the mouse pointer over the tab

setToolTipDuration(*duration*)

Set how long the tooltip displays

getToolTip()

Returns the tooltip text

getCheck()

Returns check status; `true` or `false`

setCheck(*check: bool*)

Sets the box to be checked or not

setText(*text*)

Sets the radio button text

getText()

Gets the radio button button

4.2.24 Scroller

4.2.25 Slider

A widget that lets users select a value from a range by sliding a handle or by clicking at specific points along a track allowing them to set a value according to the position of the handle.

```
local slider = Slider()
```

Properties

setRange(*start*, *end*)

Sets the slider's range

setValue(*value*)

Sets the current value

getValue()

Returns the current value

setOrientation(*orientation*)

Sets the orientation: `horizontal` or `vertical`

setTickPosition(*position*)

Sets the position of tick marks on a slider

`Positions`: `none`, `above`, `left`, `below`, `right` and `bothsides`

4.2.26 SlidingStackedWidget

4.2.27 Splitter

4.2.28 Spinner

A widget that allows users to input integer values within a specified range. It provides a convenient interface for users to select numbers by either typing directly into the box or by using up and down arrow buttons to increase or decrease the value

```
local dock = Spinner()
```

Properties

setOnValueChange(*callback*)

The function executed when the value changes

Params
`self` and `value`

setRange(*start*, *end*)

Sets the `minimum` and `maximum` values for the widget

setValue(*value*)

Sets the value displayed on the widget

getValue()

Returns the current value

setPrefix(*prefix*)

Set the prefix of the spinner widget

setSuffix(*suffix*)

Set the suffix of the spinner widget

4.2.29 Tab

A container widget that organizes multiple pages or panels of content into separate tabs, allowing users to switch between different sections within a single window.

```
local tab = Tab()
```

Hey!

Check out *TabItem* for more

Properties

setOnTabClose(*callback*)

Executed when the tab is being closed

addTab(tabitem, title, icon: optional):

Adds a `TabItem` to the widget

setTabsCloseable(*closeable: bool*)

Sets whether tabs can be closed

setTabsPosition(*position*)

Sets the side where the tabs appear: `left`, `top`, `right`, `bottom`

setToolTip(*index*, *text*)

Sets tooltip text to a particular tab index

getCurrentIndex()

Returns the current tab index

removeTab(*index*)

Removes a tab on a specified index

4.2.30 Table

A widget where users can view, edit, and interact with data presented in a grid-like format. It allows developers to populate the table with data, set headers for rows and columns, and enable users to edit cell contents. This widget offers functionalities to manage and manipulate tabular data, including sorting, selecting cells, adding or removing rows/columns dynamically. It's a powerful tool for displaying structured data or creating spreadsheet-like interfaces.

```
local table = Table(rows, columns) -- optional parameters; specfies number of rows and␣
↪columns
```

Important: Row and column indexing starts at `0`

Properties

setOnCellEditFinished(*callback*)

Executed when user finishes editing a particular cell

setOnCellClicked(*callback*)

Executed when a cell was clicked

setOnCellDoubleClicked(*callback*)

Executed when a cell gets double clicked

setOnCellSelection(*callback*)

Executed when a cell is selected

Params
self, row, column

addData(*row*, *column*, *text*)

Adds text to row and column

setImageData(*image*, *text*, *row*, *column*)

Sets an image and text on specified row and column

setColumnHeaders(*columns: table*)

Sets columns for the widget,ie, `{'Name', 'Age', 'Location', ...}`

setRowHeaders(*headers: table*)

Same as above property

getCurrentColumn()

Returns the current column

getCurrentRow()

Returns the current row

setMaxColumns(*columns*)

Sets the maximum columns for the widget

setMaxRows(*rows*)

Sets the maximum rows for the widget

setColumnHeaderToolTip(*column: number*)

Sets the tooltip for a particular header index

getColumnHeaderText(*column: number*)

Returns the column header text

getColumnsCount()

Returns total columns in the widget

getRowsCount()

Returns total rows in the widget

setGridVisible(*visibility*)

Sets grid-lines visibility for the widget

setRowLabelsVisible(*visibility*)

Sets the visibility for row labels; `1,2,3,4,5` on the left hand side

setCellChild(*row*, *column*, *child*)

Sets a widget on a particular row and column

setAutoColumnResize()

Sets columns to automatically adjust to content

setAutoRowResize()

Sets rows to automatically adjust to content

setColumnFitsContent(*column*)

Manually adjust a particular column

deleteRow(*row*)

Deletes a particular row

setCellsEditable(*editable: bool*)

Enables or disables cell editing

setAltRowColors(*altcolors: bool*)

Sets alternating colors

setColumnSorting(*enable: bool*)

Eanbles or disables column header sorting

clear()

Clears all content, including the headers

clearContent()

Clears only values in the cells, excluding the headers and row labels

findDataItem(*text*)

Searches for particular text in the widget

insertColumnAt(*column*)

Inserts a new column on a specified column index

insertRowAt(*row*)

Inserts a new row on a specified row index

removeColumnAt(*column*)

Removes a column at a particular column index

removeRowAt(*row*)

Removes a row at a particular row index

getItemAt(*row*, *column*)

Returns a `TableItem` at a particular row and column

getSelectedCells()

Returns all selected cells

getSelectedCell()

Returns selected cell

setSpan(*row*, *column*, *rowSpan*, *columnSpan*)

Merges cells toegther, allowing them to span multiple rows and columns within the table. Useful for creating cells that cover a large area within the table layout

4.2.31 TextField

An input field where users can enter and edit a single line of text. It's commonly used for gathering textual information from users, such as usernames, passwords, search queries, or any other type of short text input.

```
local input = TextField(text: optional)
```

Note: The next release will include a lot of text formating features

Properties

setOnTextChange(*callback*)

The function executed text inside changes

Params
`self` and `text`

setText(*text*)

Sets text in the widget

getText()

Returns the text input

setHint(*text*)

Sets hint/placeholder text in the widget

selectAll()

Seleects all text

setReadOnly(*enable: bool*)

Sets widget to read-only or allow input

redo()

Redo text input

undo()

Undo text input

setMaxLength()

Sets the maximum input length in the widget

appendText(*text*)

Appends text to the widget

4.2.32 TimePicker

4.2.33 ToolBar

A movable panel that contains various interactive elements like buttons, tool buttons, text fields, or other widgets. It's commonly used to provide quick access to frequently used actions or tools within an application.

```
local toolbar = Toolbar()
```

Hey!

Check out *ToolbarButton* for more

Properties

setIconStyle(*style*)

Sets the toolbar icon style; `icononly`, `textonly`, `textbesideicon`, `textundericon`, `followstyle`

setIconSize(*width*, *height*)

Adjusts the icon size for all `ToolbarButton`

addButton(*data*, *startNode*, *endNode*)

Adds a `ToolbarButton` to the widget

4.2.34 VLine

A visual element used to separate or demarcate sections within a user interface. It's a simple line drawn vertically across a window or widget, often used to visually divide different parts of the interface or to create a visual distinction between sections of content. This line is typically used for aesthetic purposes to improve the layout and readability of the user interface components.

```
local line = VLine()
```

Properties

Note: There are currently no properties available

4.2.35 Window

This is the fundamental widgets that serves as the main window of any application. It provies a framework for building the main user interface of an application, typically containing menu bars, toolbars, and a layout where other widgets are placed.

```
local window = Window{title = 'Limekit', icon = images('app.png'), size={200,200}}
window:show()
```

Important: You should first set a layout to hold all of your widgets.

Properties

setOnShown(*callback*)

Executed whenever the window is shown or displayed on the screen.

setOnClose(*callback*)

Executed when the window is closing.

setOnResize(*callback*)

Executed whenever the window is being resized.

maximize()

Used to enlarge a window to fill the entire screen. It allows the window to occupy the maximum available space on the screen

minimize()

Minimizes or iconifies a window to the system taskbar or dock. It shrinks the window and places an icon representing the window in the taskbar or dock, allowing users to easily restore the window later

setMinSize(*width*, *height*)

Sets the minimum size

setMaxHeight(*height*)

Sets the maximum height

setMinHeight(*height*)

Sets the minimum height

setMaxWidth(*width*)

Sets the maximum width

setMinWidth(*width*)

Sets the minimum width

setMaxSize(*width*, *height*)

Sets the maximum size

setCustomCursor(*path*)

Sets a custom cursor icon from path for the window

setTitle(*title*)

Sets the title for the window

setMainWidget(*widget*)

Sets any widget as the central widget, causing it to take up the available space.

setSize(*width*, *height*)

Sets the size of the window

setLayout(*layout*)

Sets a primary layout for the window

addDock(*dock*, *area: optional*)

Adds a dock to the window.

Areas available: `left`, `right`, `top`, `bottom`, `allareas` and `nodock`

checkout *Docks*

setIcon(*path*)

Sets the icon for the window

addToolbar(*toolbar*)

Adds a toolbar to the window

checkout *Tool bars*

setMenubar(*menubar*)

Sets a menubar for the window

checkout *Menu bars*

Note: There can only be one menubar per window

---

center()

Centers the window

setFixedSize(*width*, *height*)

Sets a fixed size to restrict resizing the window

show()

Shows the window

4.3 Widget Items

Due to the complexity of some widgets, the mojority of them require some special classes to make them work. This turotial will cover them in detail.

4.3.1 MenuItem

A powerful component designed to manage and display multiple tabs, each hosting distinct set of widgets or layouts but only one item can stay open at a time

```
local accordion = Accordion()
```

You can add a widget or a layout to the accordion using `addChild(widget, label, icon: optional)` or `addLayout(layout, label, icon: optional)`, respectively. Take a look at the different code snippet below

Adding a widget

```
local button = Button('Click me')
accordion:addChild(button, 'All Buttons')

-- or otherwise with an icon

local button = Button('Click me')
accordion:addChild(button, 'All Buttons', images('icon.png'))
```

Adding a layout

```
local layout = VLayout()
local button = Button('Click me')

layout:addChild(button)
accordion:addChild(layout, 'All Buttons')

-- or otherwise with an icon

local layout = VLayout()
local button = Button('Click me')

layout:addChild(button)
accordion:addChild(layout, 'All Buttons')
```

checkout Layout Managers

Properties

addChild(*widget*, *label*, *icon: optional*)

Adds a widget in a tab to the accordion with given a label, and icon if necessary

addLayout(*layout*, *label*, *icon: optional*)

Adds a layout in a tab to the accordion with given a label, and icon if necessary

checkout *Using resources*

setToolTip(*text*)

Enable text that appears when a mouse hovers on the tab

getCurrentIndex()

Gets the current index of the visible tab

setCurrentIndex(*index*)

Sets the index of the tab to be visible

4.3.2 TabItem

A widget added to a `Tab`.

```
local widgets = TabItem()
```

Properties

setLayout(*layout*)
: Adds a layout to the widget

4.3.3 ToolbarButton

A widget/button that can only be used in a `Toolbar`

```
-- setting the title to - (dash) treats it as a separator
local homepage = ToolbarButton(title: optional)
```

Properties

setOnClick(*callback*)
: The function executed when the button is clicked

setText(*text*)
: Sets the text on the button

setIcon(*path*)
: Sets an icon on the button

setToolTip(*tooltip*)
: Sets the tooltip for the button

setMenu(*menu*)
: Sets a menu for the button

setCheckable(*checkable: bool*)
: Allows the button to be checkable or to be toggled

isChecked()
: Checks if the button is checked/toggles

setChecked(*check: bool*)
: Checks/toggles the button

setVisibility(*visible: bool*)
: Sets the visibility for the button

4.4 Layout Managers

Layouts are structures used to arrange and manage the positioning of widgets within a graphical user interface. They define the organization and alignment of various elements such as buttons, input fields, labels, and other widgets within a window. Layouts help ensure that the components within an application are positioned correctly and adjust dynamically when the window is resized or modified, aiding in the creation of visually consistent and responsive user interfaces. Some common layouts in layouts in Limekit are listed below, each serving different purposes in organizing the visual elements of an application.

4.4.1 HLayout

A layout that helps organize widgets in a horizontal arrangement within a window or interface. It stacks elements side by side and from left to right, making it easy to place widgets horizontally one after another. This layout ensures that when you add widgets, they align horizontally and adjust their positions automatically if the window is resized. It's useful for creating interfaces where elements are stacked horizontally, like a list of items or a column of buttons.

```
local layout = HLayout()
```

Properties

addChild(*widget*, *stretch: optional*)

The **optional** stretch parameter in the `addChild` method determines how much space a widget occupies relative to other widgets in the layout.

For instance, if you add three widgets to a horizontal layout and set the stretch factor to 0 for the first widget, 1 for the second widget, and 2 for the third widget, the third widget will take up more horizontal space compared to the first and second widgets. The space distribution is proportional to the stretch factors assigned to each widget.

```
local layout = HLayout()

local button1 = Button("Button 1")
local button2 = Button("Button 2")
local button3 = Button("Button 3")

layout:addChild(button1)     -- default stretch factor is 0 anyway
layout:addChild(button2, 1)  -- stretch factor 1
layout:addChild(button3, 2)  -- stretch factor 2
```

Note: You don't have to specify a stretch value unless needed. By default, it's always set to `0`. Feel free to omit it if you don't require a specific stretch for your layout.

setContentAlignment(*alignment*)

This property allows you to specify the alignment of the widgets within a layout. This method sets the alignment of the entire layout's content within its allocated space. Available alignment flags: `leading`, `left`, `tight`, `trailing`, `hcenter`, `justify`, `absolute`, `horizontal_mask`, `top`, `bottom`, `vcenter`, `center`, `baseline` and `horizontal_mask`

addLayout(*layout*)

This allows you to add an entire layout as a single element within another layout. This is helpful when you want to nest layouts to create more complex UI structures.

addStretch(*stretch*)

This property is used to add stretchable space within a layout. This stretchable space pushes the widgets towards the beginning or end of the layout, depending on where the stretch is added.

For instance, if you have a horizontal layout and you add addStretch before and after adding widgets, it will push the widgets to the center, creating space before and after them that expands or contracts based on the available space.

```
local layout = HLayout()

local button1 = Button("Button 1")
local button2 = Button("Button 2")
local button3 = Button("Button 3")

layout:addStretch(1)  -- Adds stretchable space before buttons
layout:addChild(button1)
layout:addChild(button2)
layout:addChild(button3)
layout:addStretch(1)  -- Adds stretchable space after buttons
```

setMargins(*left*, *top*, *right*, *bottom*)

Sets the margins of the layout

4.4.2 VLayout

A layout that helps organize widgets in a vertical arrangement within a window or interface. It stacks elements on top of each other from top to bottom, making it easy to place widgets vertically one after another. This layout ensures that when you add widgets, they align vertically and adjust their positions automatically if the window is resized. It's useful for creating interfaces where elements are stacked vertically, like a list of items or a column of buttons.

```
local layout = VLayout()
```

Properties

addChild(*widget*, *stretch: optional*)

The **optional** stretch parameter in the `addChild` method determines how much space a widget occupies relative to other widgets in the layout.

For instance, if you add three widgets to a vertical layout and set the stretch factor to 0 for the first widget, 1 for the second widget, and 2 for the third widget, the third widget will take up more vertical space compared to the first and second widgets. The space distribution is proportional to the stretch factors assigned to each widget.

```
local layout = VLayout()

local button1 = Button("Button 1")
local button2 = Button("Button 2")
local button3 = Button("Button 3")

layout:addChild(button1)     -- default stretch factor is 0 anyway
layout:addChild(button2, 1)  -- stretch factor 1
layout:addChild(button3, 2)  -- stretch factor 2
```

Note: You don't have to specify a stretch value unless needed. By default, it's always set to `0`. Feel free to omit it if you don't require a specific stretch for your layout.

`setContentAlignment`(*alignment*)

This property allows you to specify the alignment of the widgets within a layout. This method sets the alignment of the entire layout's content within its allocated space. Available alignment flags: `leading`, `left`, `tight`, `trailing`, `hcenter`, `justify`, `absolute`, `horizontal_mask`, `top`, `bottom`, `vcenter`, `center`, `baseline` and `vertical_mask`

`addLayout`(*layout*)

This allows you to add an entire layout as a single element within another layout. This is helpful when you want to nest layouts to create more complex UI structures.

`addStretch`(*stretch*)

This property is used to add stretchable space within a layout. This stretchable space pushes the widgets towards the beginning or end of the layout, depending on where the stretch is added.

For instance, if you have a horizontal layout and you add addStretch before and after adding widgets, it will push the widgets to the center, creating space before and after them that expands or contracts based on the available space.

```
local layout = HLayout()

local button1 = Button("Button 1")
local button2 = Button("Button 2")
local button3 = Button("Button 3")

layout:addStretch(1)  -- Adds stretchable space before buttons
layout:addChild(button1)
layout:addChild(button2)
layout:addChild(button3)
layout:addStretch(1)  -- Adds stretchable space after buttons
```

`setMargins`(*left*, *top*, *right*, *bottom*)

Sets the margins of the layout

4.4.3 FormLayout

Layout manager designed for creating forms or input layouts in a user interface. It arranges widgets in a two-column format: one column for labels and another for input fields or other widgets. This layout is commonly used when you have a series of labels and corresponding input fields, such as in a settings page or a data entry form.

```
local layout = FormLayout()
```

Note: Additonal properties shall be provided on our next release.

Properties

addChild(*label*, *widget*)

Adds a widget to a layout with a given label

addLayout(label, layout):

Adds a layout to the widget with a given label

4.4.4 GridLayout

This layout is like a grid or table where you can neatly arrange widgets in rows and columns. It's a layout manager that helps you organize various elements within a window or interface. You can specify where each widget should go in the grid, and the layout ensures they are positioned accordingly. It's great for creating structured interfaces where widgets need to be aligned in rows and columns, similar to organizing information in a table.

```
local layout = GridLayout()
```

Properties

addChild(*widget*, *row*, *column*, *rowSpan: optional*, *columnSpan: optional*)

Adds a widget to the layout on a specified row (`x position`) and column (`y position`). The `rowSpan` and `columnSpan` are optional but allow a widget to span multiple rows or columns.

addLayout(*layout*, *row*, *column*, *rowSpan: optional*, *columnSpan: optional*)

Similar to the above property, now only for layouts.

getChildAt(*row*, *column*)

Retrieves a widget at a specified row and column

setColumnStretch(*column*, *stretch*)

This method allows you to control how the columns of the layout expand or contract when the parent widget is resized.

setMargins(*left*, *top*, *right*, *bottom*)

Sets the margins of the layout

4.5 App object

Is a collection of different functions (tools) that your program can use for various task. These tasks include reading files, gathering info about the computer's CPU, checking how much memory is available, choosing files and many more.

```
-- an example of how to access various functions from the app object
local processor = app.getProcessorName()
```

isIDE()

Returns whether the app is being run in `Limer` or as executable

joinPaths(...)

Joins multiple paths into one

randomChoice(*table*)

Randomly selects a string from a table

getStandardPath(*location*)

A set of standard locations on a user's system where different types of files or resources are commonly stored. These locations are predefined and provide a consistent way to access specific directories such as user's documents, desktop, applications, and more across different platform. Available arguments for `location` parameter include: `desktop`, `documents`, `fonts`, `applications`, `music`, `movies`, `pictures`, `temp`, `home`, `applocaldata`, `cache`, `genericdata`, `runtime`, `config`, `download`, `genericcache`, `genericconfig`, `appdata`, `appconfig`, `publicshare`, `templates`

```
-- how to get the location for desktop
local desktop = app.getStandardPath('desktop')
```

splitString(*string*, *delimeter*)

Split the string by the delimite

intRange(*from*, *to*)

Returns a table with a ranger of integers

sleep(*seconds*)

Sleep for some time. Use `1, 2, 3, ...` and not `1000, 2000` to represent time

weightedGraph(*data*, *startNode*, *endNode*)

Perfoms a weighted graph algorithm on data.

```
-- Based on the provided data, calculates shortest path from point a to point c
local graph = app.weightedGraph({{'point a', 'point b', 20}, {'point a', 'point c',␣
↪10},
   {'point b', 'point c', 50}}, 'point a', 'point c')
```

getStyles()

Returns platform-dependent UI styles that can be applies to the whole application.

setStyle(*style*)

Set the style to the whole application. Obtained from the above function

makeHash(*hashType*, *string*)

Generate a hash from the `string` based on the hash-type provided: Available hash types: `md5`, `sha1`, `sha224`, `sha256`, `sha384`, `sha512`, `sha3_224`, `sha3_256`, `sha3_384`, `sha3_512`

hexToRGB(*hex*)

Converts a hex to RGB values

readFileLines(*file*)

Reads the file lines for a particular file

bytesToReadableSize(*bytes*)

Converts bytes to readable size, ie, `2 kb, 10 GB`

toBase64(*string*)

Converts string to base64 encoding

fromBase64(*b64*)

Converts base64 string to readable string

setFont(*file*, *textSize*)

Sets the font and text size for the whole application

extractZip(*zip*, *destination*)

Extracts the content of a zip file to some destination

isFolder(*path*)

Checks if given path is a folder or not

exists(*path*)

Checks if given path is empty or not

isFolderEmpty(*path*)

Checks if a given path is an exmpty dir or not

isFileEmpty(*file*)

Checks if a given file is exmpty or not

getFileSize(*file*)

Returns file size

getFileExt(*file*)

Returns only the file extension for a file path

copyFile(*source*, *destination*)

Copies a file from source to destination

readFile(*file*)

Reads a file and returns its content

writeFile(*file*, *content*)

Write content to a file

appendFile(*file*, *content*)

Does not overwrite, only appeands content to the file

quit()

Quits the application

setClipboardText(*text*)

Sets text to the clipboard

getClipboardText()

Returns text from the clipboard

listFolder(*path*)

Returns a list of files in a folder

renameFile(*file*, *newName*)

Renames a file

renameFolder(*path*, *newName*)

Renames a folder

createFolder(*path*)

Creates a new folder

playSound(*file*)

Plays any audio format

getProcesses()

Returns a list for running processes

killProcess(*pid*)

Kills/terminates a running process by a pid (Process Identifier)

getUsers()

Returns available users on a system

getCPUCount()

Returns the number of CPUs available

getBatteryInfo()

Returns available battery info

getDiskPartitions()

Returns available partitions in a system

getDiskInfo(*path*)

Returns disk info. `path` can obtained from above method, ie, `C:\\`, `D:\\`, `E:\\`

getBootTime()

Returns the system's boot time

getMachineType()

Returns the machine type, ie, `AMD64`

getNetworkNodeName()

Returns the network node name

getProcessorName()

Returns the processor info

getPlatformName()

Returns the platform name, ie, `Windows-10-10.0.22621-SP0`

getSystemRelease()

Returns the OS's release

getOSName()

Returns the OS's name, ie, `Windows`

getOSVersion()

Returns available users on a system

4.6 Accessing Resources

There are three folders for app resources: `images`, `script`, and `misc`. To access resources from each folder, functions with names matching the folders are used.

Note: The `misc` folder contains miscellaneous content, from lua modules, to audio, to csv files.

It is automatically added to `package.path`

Important: Put your own `scripts` and `images` in the `scripts` and `images` folders, respectively, and store third-party lua modules in the `misc` folder and not in the `scripts` folder.

Use the function `image(path)` to access images. Apply this same concept to access resources in the other folders.

```
local button = Button('Next')
button:setIcon(images('next_icon.png')) -- this points to the icon inside the images␣
↪folder

-- accessing subfolders

local button = Button('Play')
button:setIcon(images('folder1/folder2/folder3/play.png'))
```

Note: Do not use use `../` or `./` to access folders.

4.7 Batteries included

Content

- *1. Theming*
- *2. Dialogs*
- *3. System Tray and System Notification*

This comprehensive tutorial covers the rest of the available functionalities in the framework. These include accessing SQLite3 databases, sending system notifications, utilizing the system tray, theming, multitasking, handling signals, and much more.

Let's go through them in detail

4.7.1 1. Theming

Themes are what make modern design in Limekit possible.

There are currently 3 categories of themes in Limekit; `darklight`, `material`, `darkstyle` and `misc`. Use `getThemes()` to get themes for a particular theme category, and `setTheme(theme)` to set the theme.

```
local theme = Theme('darklight')
local allThemes = theme:getThemes()

theme:setTheme(allThemes[1])
```

4.7.2 2. Dialogs

- Limekit offers various dialogs for tasks such as opening files, saving files, receiving input, and more, all of which will be discussed.

File and Folder dialogs

Note: `filters` allows you to specify the type of files that will be displayed in the dialog's file type dropdown filter.

example: `{['Image Files'] = {'.jpg', '.png'}, Audio = {'.mp3', '.flac'}, Archieves = {'.zip', '.tar', '.rar'}}`

`dir` is used to specify the initial directory that the file dialog will open when it's launched. Pass an empty string to ignore

---

```
-- allows opening of files
app.openFileDialog(window, title, dir, filters)
```

```
app.saveFileDialog(window, title, dir, filters)
```

```
app.folderPickerDialog(window, title, dir)
```

Input Dialogs

```
app.textInputDialog(window, title, label)
```

```
-- can ignore content
app.multilineInputDialog(window, title, label, content)
```

```
-- items: table
-- can ignore startIndex
app.comboBoxInputDialog(window, title, label, items, startIndex)
```

```
-- step: increment by
-- can ignore step
app.integerInputDialog(window, title, label, startValue, minValue, maxValu, step)
```

```
-- can ignore step
app.doubleInputDialog(window, title, label, value, minValue, maxValue, step)
```

Alerts

```
-- returns true or false
-- for an alert like this, you don't really need the result
local result = app.alert(window, title, message)
```

```
-- Contains the 'do not show this message again' button
app.infoMessageDialog(window, title, message)
```

```
app.aboutAlertDialog(window, title, message)
```

```
app.criticalAlertDialog(window, title, message)
```

```
app.infoAlertDialog(window, title, message)
```

```
-- returns true or false
local result = app.questionAlertDialog(window, title, message)
```

```
app.warningAlertDialog(window, title, message)
```

Other

```
-- type: hex or RGB
local color = app.colorPickerDialog(window, type)
```

4.7.3 3. System Tray and System Notification

Learn how to create a system tray icon or send system notifications

System Tray

```
local tray = SysTray(icon)
```

Properties

setIcon(*icon*)

Sets the icon

setToolTip(*text*)

Sets the icon

setMenu(*menu*)

Sets menu

checkout *Menus*

setVisibility(*text*)

Sets the visibility

System Notifications

Allows you to send notification from your app.

```
local tray = SysNotification(icon)
```

Properties

setOnClick(*callback*)

Executed when the notification is clicked

setMessage{title, message, icon: optional, duration: optional}

Sets the required attributes of a system notification

`duration`, ie, `2000, 3000, 10000` and not `1,2,3,4,5` for seconds

INDEX

A

addButton()
    built-in function, 37
addChild()
    built-in function, 15, 17, 18, 40, 43, 45
addData()
    built-in function, 34
addDock()
    built-in function, 39
addImageItem()
    built-in function, 19, 27
addImageItems()
    built-in function, 19, 27
addItem()
    built-in function, 19, 27
addLayout()
    built-in function, 15, 17, 18, 40, 42, 44, 45
addStretch()
    built-in function, 42, 44
addToolbar()
    built-in function, 39
appendFile()
    built-in function, 47
appendText()
    built-in function, 37

B

buildFromTemplate()
    built-in function, 28, 29
built-in function
    addButton(), 37
    addChild(), 15, 17, 18, 40, 43, 45
    addData(), 34
    addDock(), 39
    addImageItem(), 19, 27
    addImageItems(), 19, 27
    addItem(), 19, 27
    addLayout(), 15, 17, 18, 40, 42, 44, 45
    addStretch(), 42, 44
    addToolbar(), 39
    appendFile(), 47
    appendText(), 37
buildFromTemplate(), 28, 29
bytesToReadableSize(), 46
center(), 39
checkTextSelected(), 26
clear(), 27, 35
clearContent(), 35
copyFile(), 47
createFolder(), 47
deleteRow(), 35
dismiss(), 30
exists(), 47
extractZip(), 47
findDataItem(), 35
fromBase64(), 46
getBatteryInfo(), 48
getBootTime(), 48
getCheck(), 18, 23, 31
getChild(), 29
getChildAt(), 45
getClipboardText(), 47
getColumnHeaderText(), 35
getColumnsCount(), 35
getCPUCount(), 48
getCurrentColumn(), 34
getCurrentFrame(), 22
getCurrentIndex(), 15, 17, 18, 33, 40
getCurrentRow(), 27, 34
getDate(), 20
getDescription(), 20
getDiskInfo(), 48
getDiskPartitions(), 48
getEndSelection(), 26
getFileExt(), 47
getFileSize(), 47
getinsertItemAtText(), 27
getItemAt(), 36
getItemsCount(), 27
getMachineType(), 48
getNetworkNodeName(), 48
getOSName(), 48
getOSVersion(), 48
getPlatformName(), 48

getProcesses(), 48
getProcessorName(), 48
getRowsCount(), 35
getSelectedCell(), 36
getSelectedCells(), 36
getSelectedText(), 26
getSelectionLength(), 26
getStandardPath(), 46
getStartSelection(), 26
getState(), 22
getStyles(), 46
getSystemRelease(), 48
getText(), 16, 19, 20, 26, 27, 32, 36
getTextAt(), 27
getTitle(), 23
getToolTip(), 31
getTooltip(), 18, 23
getUsers(), 48
getValue(), 31, 32
hexToRGB(), 46
insertColumnAt(), 35
insertRowAt(), 35
intRange(), 46
isChecked(), 16
isFileEmpty(), 47
isFolder(), 47
isFolderEmpty(), 47
isIDE(), 45
isIndicatorVisible(), 24
joinPaths(), 45
justToFrame(), 22
killProcess(), 48
listFolder(), 47
makeHash(), 46
maximize(), 38
minimize(), 30, 38
nextFrame(), 22
pause(), 22
playSound(), 47
quit(), 47
randomChoice(), 45
readFile(), 47
readFileLines(), 46
redo(), 26, 36
removeColumnAt(), 36
removeItem(), 27
removeRowAt(), 36
removeTab(), 33
renameFile(), 47
renameFolder(), 47
resizeImage(), 24
selectAll(), 26, 36
setAllowDragDrop(), 28
setAltRowColors(), 27, 35
setAutoColumnResize(), 35
setAutoRowResize(), 35
setBackgroundColor(), 23
setBold(), 25
setCellChild(), 35
setCellsEditable(), 35
setCheck(), 18, 23, 31
setCheckable(), 16, 41
setChecked(), 41
setClipboardText(), 47
setColumnFitsContent(), 35
setColumnHeaders(), 34
setColumnHeaderToolTip(), 35
setColumnSorting(), 35
setColumnStretch(), 45
setCompanion(), 25
setContentAlignment(), 42, 44
setCurrentIndex(), 15, 17, 18, 40
setCustomCursor(), 38
setDate(), 20
setDescription(), 20
setDragEnabled(), 28
setFixedSize(), 30, 39
setFlat(), 16
setFont(), 46
setGridVisible(), 18
setHint(), 26, 36
setIcon(), 18, 30, 31, 39, 51
setIconSize(), 15, 18, 31
setIconSizes(), 27
setIconStyle(), 37
setImage(), 24
setImageAlign(), 24
setImageData(), 34
setIndicators(), 24
setInputMode(), 26
setItems(), 19, 27
setItemViewMode(), 27
setLayout(), 21, 30, 39, 41
setMagneticAreas(), 21
setMainWidget(), 30, 39
setMargins(), 16, 43–45
setMaxColumns(), 34
setMaxHeight(), 30, 38
setMaxLength(), 26, 37
setMaxRows(), 34
setMaxSize(), 30, 38
setMaxValue(), 24
setMaxWidth(), 30, 38
setMenu(), 16, 51
setMenubar(), 39
setMinHeight(), 30, 38
setMinSize(), 30, 38
setMinValue(), 24

setMinWidth(), 30, 38
setOnCellClicked(), 34
setOnCellDoubleClicked(), 34
setOnCellEditFinished(), 34
setOnCellSelection(), 34
setOnCheck(), 31
setOnClick(), 24, 25, 52
setOnClose(), 30, 38
setOnItemSelect(), 27
setOnItemSelected(), 19
setOnResize(), 30, 38
setOnReturnPress(), 26
setOnShown(), 30, 38
setOnTabClose(), 33
setOnTextChange(), 36
setOnTextSelection(), 26
setOnValueChanged(), 24
setOrientation(), 31, 32
setPrefix(), 21, 33
setRange(), 21, 24, 33
setReadOnly(), 26, 36
setRowHeaders(), 34
setRowLabelsVisible(), 35
setSize(), 30, 39
setSpan(), 36
setStyle(), 46
setSuffix(), 21, 33
setTabsCloseable(), 33
setTabsPosition(), 33
setText(), 15, 18, 20, 26, 32, 36
setTextAlign(), 25
setTextSize(), 25
setTickPosition(), 32
setTitle(), 21, 23, 30, 39
setToolTip(), 15, 17, 18, 23, 31, 33, 40, 41, 51
setToolTipDuration(), 18, 23, 31
setValue(), 21, 24, 31, 33
setVisibility(), 52
setWordWrap(), 25
show(), 30, 39
sleep(), 46
splitString(), 46
start(), 22
stop(), 22
toBase64(), 46
undo(), 26, 37
weightedGraph(), 46
writeFile(), 47
bytesToReadableSize()
built-in function, 46

C

center()
built-in function, 39
checkTextSelected()
built-in function, 26
clear()
built-in function, 27, 35
clearContent()
built-in function, 35
copyFile()
built-in function, 47
createFolder()
built-in function, 47

D

deleteRow()
built-in function, 35
dismiss()
built-in function, 30

E

exists()
built-in function, 47
extractZip()
built-in function, 47

F

findDataItem()
built-in function, 35
fromBase64()
built-in function, 46

G

getBatteryInfo()
built-in function, 48
getBootTime()
built-in function, 48
getCheck()
built-in function, 18, 23, 31
getChild()
built-in function, 29
getChildAt()
built-in function, 45
getClipboardText()
built-in function, 47
getColumnHeaderText()
built-in function, 35
getColumnsCount()
built-in function, 35
getCPUCount()
built-in function, 48
getCurrentColumn()
built-in function, 34
getCurrentFrame()
built-in function, 22
getCurrentIndex()
built-in function, 15, 17, 18, 33, 40

getCurrentRow()
    built-in function, 27, 34
getDate()
    built-in function, 20
getDescription()
    built-in function, 20
getDiskInfo()
    built-in function, 48
getDiskPartitions()
    built-in function, 48
getEndSelection()
    built-in function, 26
getFileExt()
    built-in function, 47
getFileSize()
    built-in function, 47
getinsertItemAtText()
    built-in function, 27
getItemAt()
    built-in function, 36
getItemsCount()
    built-in function, 27
getMachineType()
    built-in function, 48
getNetworkNodeName()
    built-in function, 48
getOSName()
    built-in function, 48
getOSVersion()
    built-in function, 48
getPlatformName()
    built-in function, 48
getProcesses()
    built-in function, 48
getProcessorName()
    built-in function, 48
getRowsCount()
    built-in function, 35
getSelectedCell()
    built-in function, 36
getSelectedCells()
    built-in function, 36
getSelectedText()
    built-in function, 26
getSelectionLength()
    built-in function, 26
getStandardPath()
    built-in function, 46
getStartSelection()
    built-in function, 26
getState()
    built-in function, 22
getStyles()
    built-in function, 46
getSystemRelease()
    built-in function, 48
getText()
    built-in function, 16, 19, 20, 26, 27, 32, 36
getTextAt()
    built-in function, 27
getTitle()
    built-in function, 23
getToolTip()
    built-in function, 31
getTooltip()
    built-in function, 18, 23
getUsers()
    built-in function, 48
getValue()
    built-in function, 31, 32

H

hexToRGB()
    built-in function, 46

I

insertColumnAt()
    built-in function, 35
insertRowAt()
    built-in function, 35
intRange()
    built-in function, 46
isChecked()
    built-in function, 16
isFileEmpty()
    built-in function, 47
isFolder()
    built-in function, 47
isFolderEmpty()
    built-in function, 47
isIDE()
    built-in function, 45
isIndicatorVisible()
    built-in function, 24

J

joinPaths()
    built-in function, 45
justToFrame()
    built-in function, 22

K

killProcess()
    built-in function, 48

L

listFolder()

built-in function, 47

M

makeHash()
built-in function, 46
maximize()
built-in function, 38
minimize()
built-in function, 30, 38

N

nextFrame()
built-in function, 22

P

pause()
built-in function, 22
playSound()
built-in function, 47

Q

quit()
built-in function, 47

R

randomChoice()
built-in function, 45
readFile()
built-in function, 47
readFileLines()
built-in function, 46
redo()
built-in function, 26, 36
removeColumnAt()
built-in function, 36
removeItem()
built-in function, 27
removeRowAt()
built-in function, 36
removeTab()
built-in function, 33
renameFile()
built-in function, 47
renameFolder()
built-in function, 47
resizeImage()
built-in function, 24

S

selectAll()
built-in function, 26, 36
setAllowDragDrop()
built-in function, 28
setAltRowColors()
built-in function, 27, 35
setAutoColumnResize()
built-in function, 35
setAutoRowResize()
built-in function, 35
setBackgroundColor()
built-in function, 23
setBold()
built-in function, 25
setCellChild()
built-in function, 35
setCellsEditable()
built-in function, 35
setCheck()
built-in function, 18, 23, 31
setCheckable()
built-in function, 16, 41
setChecked()
built-in function, 41
setClipboardText()
built-in function, 47
setColumnFitsContent()
built-in function, 35
setColumnHeaders()
built-in function, 34
setColumnHeaderToolTip()
built-in function, 35
setColumnSorting()
built-in function, 35
setColumnStretch()
built-in function, 45
setCompanion()
built-in function, 25
setContentAlignment()
built-in function, 42, 44
setCurrentIndex()
built-in function, 15, 17, 18, 40
setCustomCursor()
built-in function, 38
setDate()
built-in function, 20
setDescription()
built-in function, 20
setDragEnabled()
built-in function, 28
setFixedSize()
built-in function, 30, 39
setFlat()
built-in function, 16
setFont()
built-in function, 46
setGridVisible()
built-in function, 18

setHint()
    built-in function, 26, 36
setIcon()
    built-in function, 18, 30, 31, 39, 51
setIconSize()
    built-in function, 15, 18, 31
setIconSizes()
    built-in function, 27
setIconStyle()
    built-in function, 37
setImage()
    built-in function, 24
setImageAlign()
    built-in function, 24
setImageData()
    built-in function, 34
setIndicators()
    built-in function, 24
setInputMode()
    built-in function, 26
setItems()
    built-in function, 19, 27
setItemViewMode()
    built-in function, 27
setLayout()
    built-in function, 21, 30, 39, 41
setMagneticAreas()
    built-in function, 21
setMainWidget()
    built-in function, 30, 39
setMargins()
    built-in function, 16, 43–45
setMaxColumns()
    built-in function, 34
setMaxHeight()
    built-in function, 30, 38
setMaxLength()
    built-in function, 26, 37
setMaxRows()
    built-in function, 34
setMaxSize()
    built-in function, 30, 38
setMaxValue()
    built-in function, 24
setMaxWidth()
    built-in function, 30, 38
setMenu()
    built-in function, 16, 51
setMenubar()
    built-in function, 39
setMinHeight()
    built-in function, 30, 38
setMinSize()
    built-in function, 30, 38
setMinValue()
    built-in function, 24
setMinWidth()
    built-in function, 30, 38
setOnCellClicked()
    built-in function, 34
setOnCellDoubleClicked()
    built-in function, 34
setOnCellEditFinished()
    built-in function, 34
setOnCellSelection()
    built-in function, 34
setOnCheck()
    built-in function, 31
setOnClick()
    built-in function, 24, 25, 52
setOnClose()
    built-in function, 30, 38
setOnItemSelect()
    built-in function, 27
setOnItemSelected()
    built-in function, 19
setOnResize()
    built-in function, 30, 38
setOnReturnPress()
    built-in function, 26
setOnShown()
    built-in function, 30, 38
setOnTabClose()
    built-in function, 33
setOnTextChange()
    built-in function, 36
setOnTextSelection()
    built-in function, 26
setOnValueChanged()
    built-in function, 24
setOrientation()
    built-in function, 31, 32
setPrefix()
    built-in function, 21, 33
setRange()
    built-in function, 21, 24, 33
setReadOnly()
    built-in function, 26, 36
setRowHeaders()
    built-in function, 34
setRowLabelsVisible()
    built-in function, 35
setSize()
    built-in function, 30, 39
setSpan()
    built-in function, 36
setStyle()
    built-in function, 46

setSuffix()
    built-in function, 21, 33
setTabsCloseable()
    built-in function, 33
setTabsPosition()
    built-in function, 33
setText()
    built-in function, 15, 18, 20, 26, 32, 36
setTextAlign()
    built-in function, 25
setTextSize()
    built-in function, 25
setTickPosition()
    built-in function, 32
setTitle()
    built-in function, 21, 23, 30, 39
setToolTip()
    built-in function, 15, 17, 18, 23, 31, 33, 40, 41, 51
setToolTipDuration()
    built-in function, 18, 23, 31
setValue()
    built-in function, 21, 24, 31, 33
setVisibility()
    built-in function, 52
setWordWrap()
    built-in function, 25
show()
    built-in function, 30, 39
sleep()
    built-in function, 46
splitString()
    built-in function, 46
start()
    built-in function, 22
stop()
    built-in function, 22

T

toBase64()
    built-in function, 46

U

undo()
    built-in function, 26, 37

W

weightedGraph()
    built-in function, 46
writeFile()
    built-in function, 47