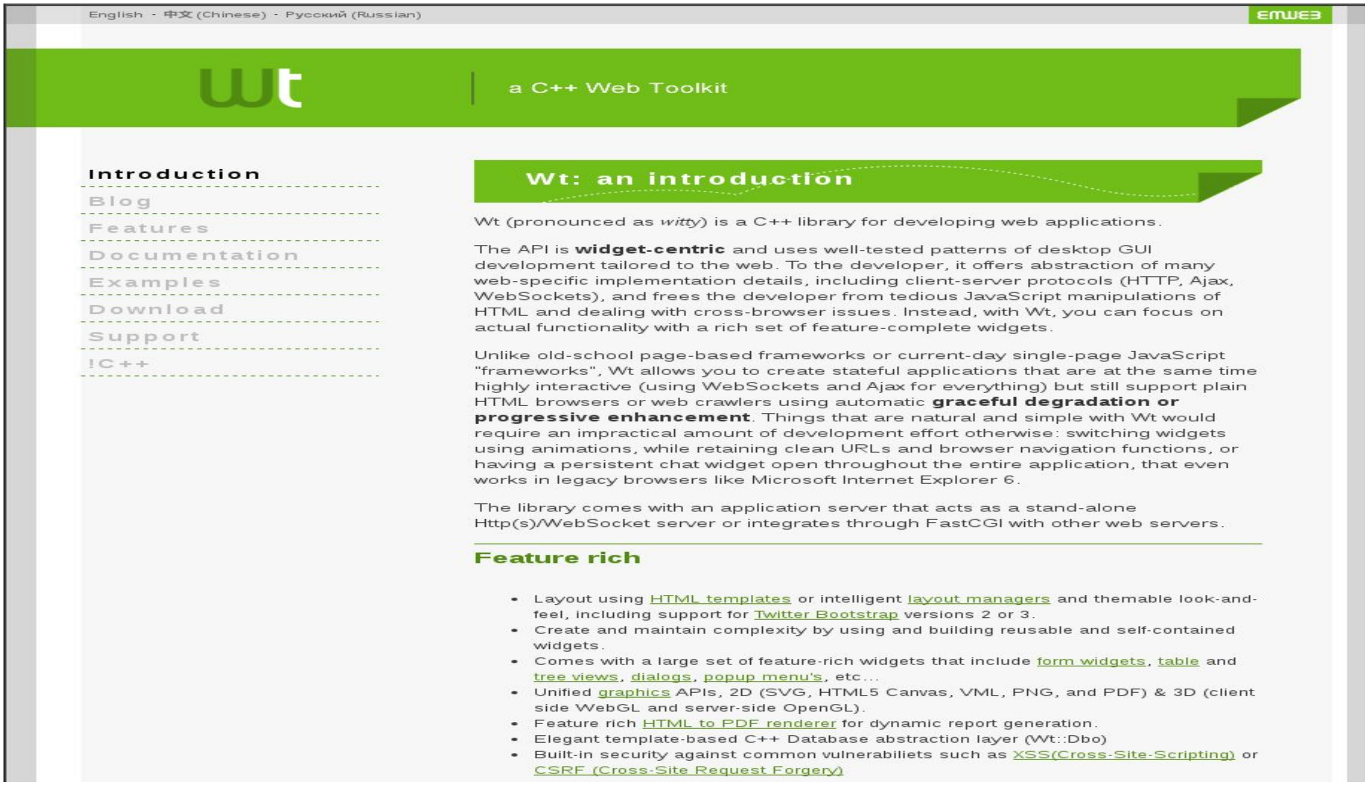
LuaWt

Lua bindings for a C++ Web Toolkit Library

Pavel Dolgov
@zer0main

- Why?
- What?
- Further development

Wt, a C++ Web Toolkit



Wt: an introduction

Wt (pronounced as *witty*) is a C++ library for developing web applications.

The API is **widget-centric** and uses well-tested patterns of desktop GUI development tailored to the web. To the developer, it offers abstraction of many web-specific implementation details, including client-server protocols (HTTP, Ajax, WebSockets), and frees the developer from tedious JavaScript manipulations of HTML and dealing with cross-browser issues. Instead, with Wt, you can focus on actual functionality with a rich set of feature-complete widgets.

Unlike old-school page-based frameworks or current-day single-page JavaScript "frameworks", Wt allows you to create stateful applications that are at the same time highly interactive (using WebSockets and Ajax for everything) but still support plain HTML browsers or web crawlers using automatic **graceful degradation or progressive enhancement**. Things that are natural and simple with Wt would require an impractical amount of development effort otherwise: switching widgets using animations, while retaining clean URLs and browser navigation functions, or having a persistent chat widget open throughout the entire application, that even works in legacy browsers like Microsoft Internet Explorer 6.

The library comes with an application server that acts as a stand-alone Http(s)/WebSocket server or integrates through FastCGI with other web servers.

Feature rich

- Layout using HTML templates or intelligent layout managers and themable look-and-feel, including support for Twitter Bootstrap versions 2 or 3.
- Create and maintain complexity by using and building reusable and self-contained widgets.
- Comes with a large set of feature-rich widgets that include form widgets, table and tree views, dialogs, popup menu's, etc...
- Unified graphics APIs, 2D (SVG, HTML5 Canvas, VML, PNG, and PDF) & 3D (client side WebGL and server-side OpenGL).
- Feature rich HTML to PDF renderer for dynamic report generation.
- Elegant template-based C++ Database abstraction layer (Wt::Dbo)
- Built-in security against common vulnerabiliets such as XSS(Cross-Site-Scripting) or CSRF (Cross-Site Request Forgery)

A Witty game: Hangman

Ready to play ?

English words (18957 words)

New game

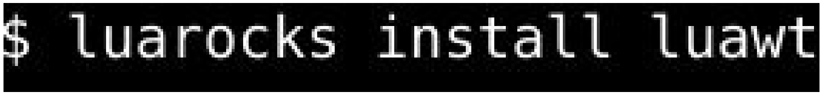
Gaming Grounds Highscores

Current version - 0.0.1

~70 widgets, about 2000 methods…

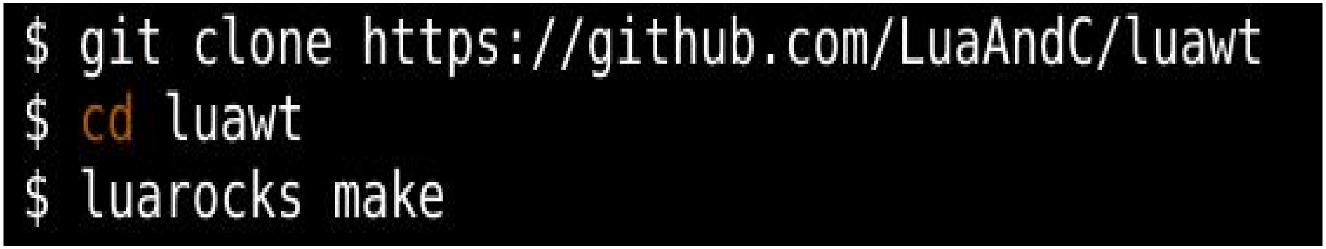
Wt versions: >= 3.3.0

How to install luawt



```
$ luarocks install luawt
```

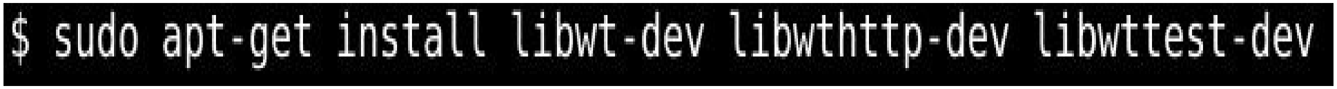
OR



```
$ git clone https://github.com/LuaAndC/luawt
$ cd luawt
$ luarocks make
```

Dependencies

- Wt >= 3.3.0
- Lua >= 5.1



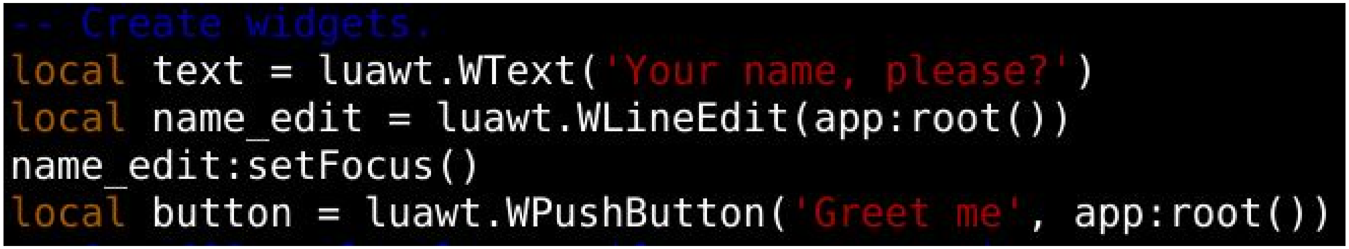
```
$ sudo apt-get install libwt-dev libwthttp-dev libwttest-dev
```

How to use

A Web application in luawt is just a Lua program.

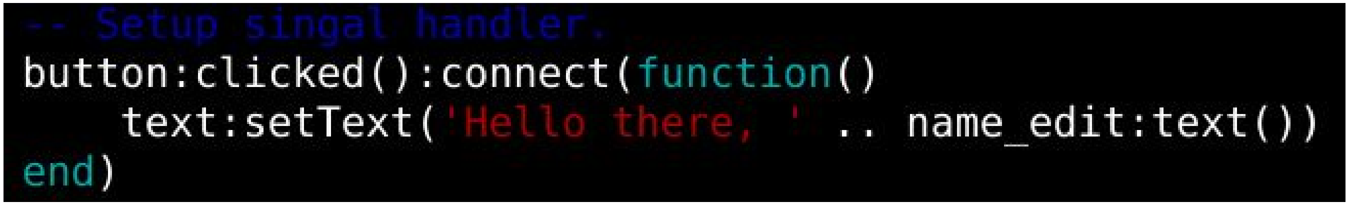
```lua
-- This code is executed in constructor of new application.
-- It setups widgets of a start page shown to a user.
local code = [[
    -- app is an instance of WApplication
    -- env is an instance of WEnvironment
    local app, env = ...

    -- To create new widgets, we need luawt module.
    local luawt = require 'luawt'

    -- Create widgets.
    local textarea = luawt.WTextArea(app:root())
    local button = luawt.WPushButton(app:root())
    button:setText("Click me")

    -- Setup signal handler: uppercase text in the textarea.
    button:clicked():connect(function()
        textarea:setText(textarea:text():upper())
    end)
]]

local luawt = require 'luawt'

-- Start WServer with the code above.
local server = luawt.WServer({
    code = code,
    ip = '127.0.0.1',
    port = 8080,
})
server:start()
server:waitForShutdown()
```

Widgets



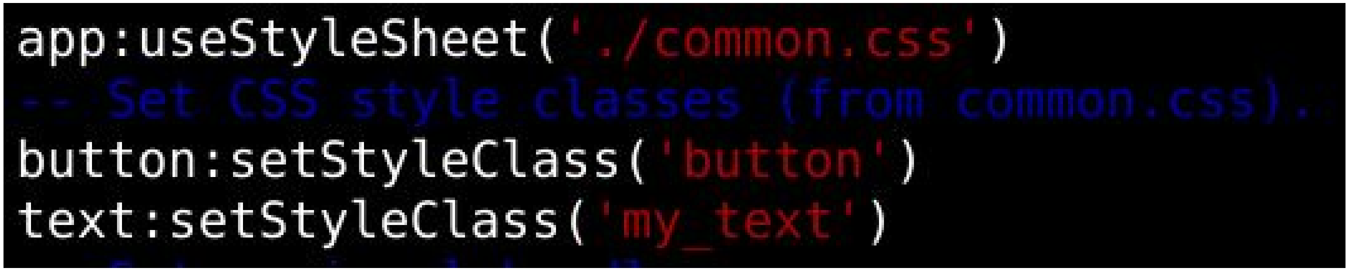
```lua
-- Create widgets.
local text = luawt.WText('Your name, please?')
local name_edit = luawt.WLineEdit(app:root())
name_edit:setFocus()
local button = luawt.WPushButton('Greet me', app:root())
```

Event handling



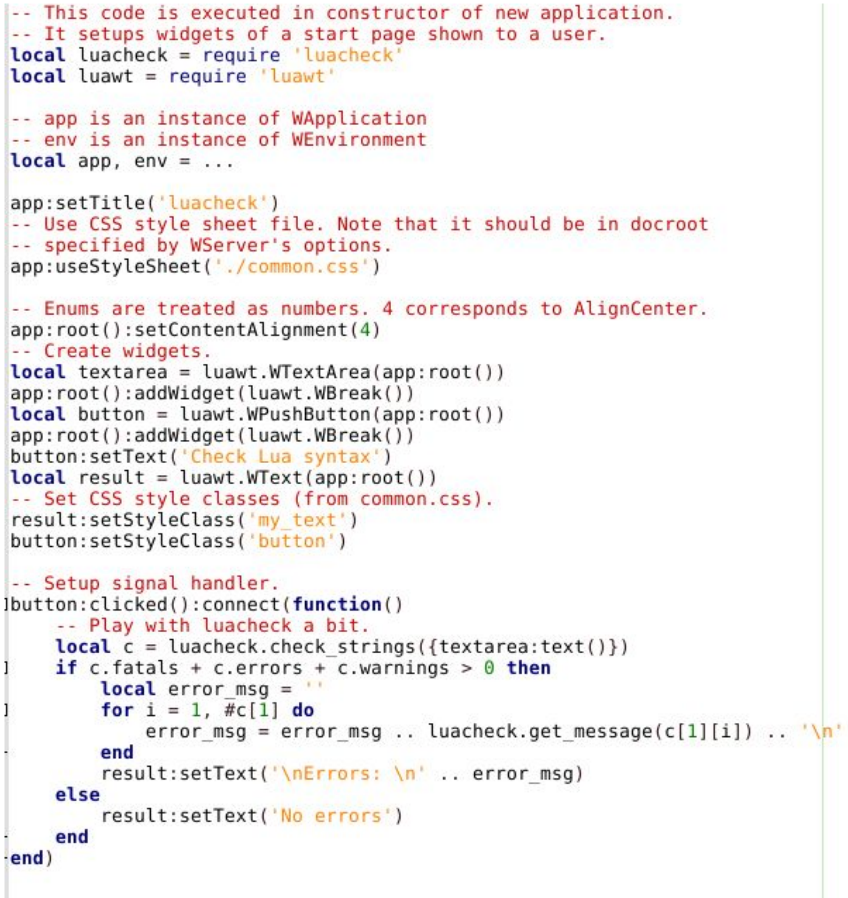
```
-- Setup singal handler.
button:clicked():connect(function()
    text:setText('Hello there, ' .. name_edit:text())
end)
```

Style issues



```
app:useStyleSheet('./common.css')
-- Set CSS style classes (from common.css).
button:setStyleClass('button')
text:setStyleClass('my_text')
```

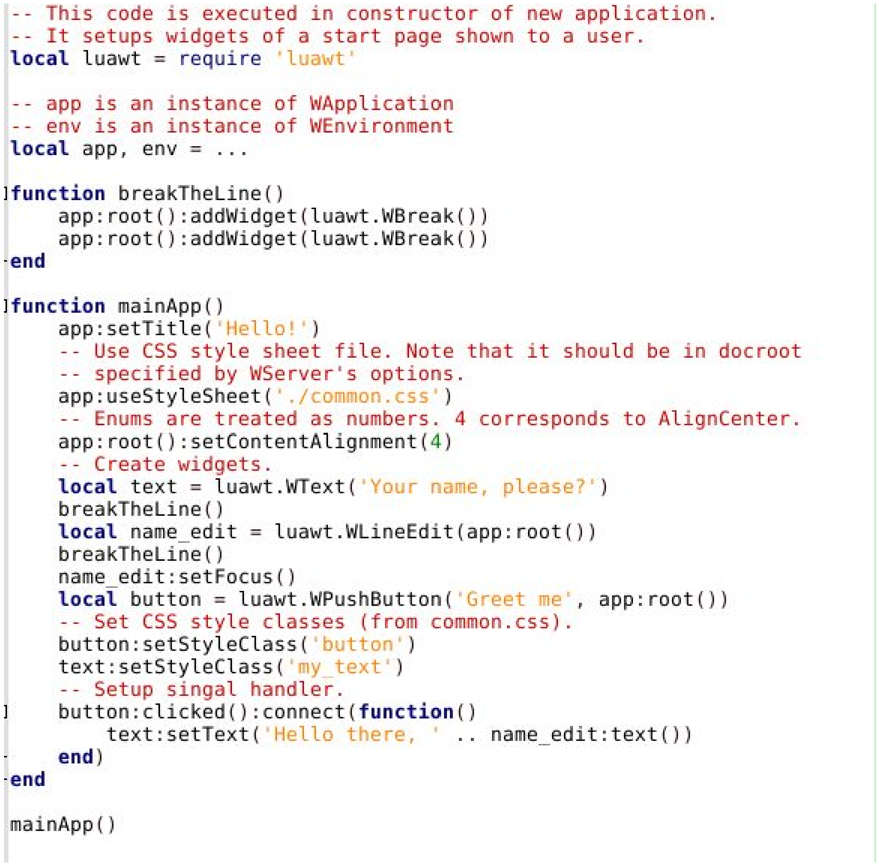
Examples: luacheck



```lua
-- This code is executed in constructor of new application.
-- It setups widgets of a start page shown to a user.
local luacheck = require 'luacheck'
local luawt = require 'luawt'

-- app is an instance of WApplication
-- env is an instance of WEnvironment
local app, env = ...

app:setTitle('luacheck')
-- Use CSS style sheet file. Note that it should be in docroot
-- specified by WServer's options.
app:useStyleSheet('./common.css')

-- Enums are treated as numbers. 4 corresponds to AlignCenter.
app:root():setContentAlignment(4)
-- Create widgets.
local textarea = luawt.WTextArea(app:root())
app:root():addWidget(luawt.WBreak())
local button = luawt.WPushButton(app:root())
app:root():addWidget(luawt.WBreak())
button:setText('Check Lua syntax')
local result = luawt.WText(app:root())
-- Set CSS style classes (from common.css).
result:setStyleClass('my_text')
button:setStyleClass('button')

-- Setup signal handler.
button:clicked():connect(function()
    -- Play with luacheck a bit.
    local c = luacheck.check_strings({textarea:text()})
    if c.fatals + c.errors + c.warnings > 0 then
        local error_msg = ''
        for i = 1, #c[1] do
            error_msg = error_msg .. luacheck.get_message(c[1][i]) .. '\n'
        end
        result:setText('\nErrors: \n' .. error_msg)
    else
        result:setText('No errors')
    end
end)
```

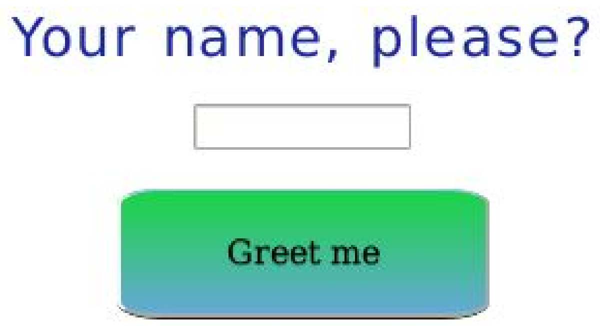
Examples: luacheck

Examples: hello

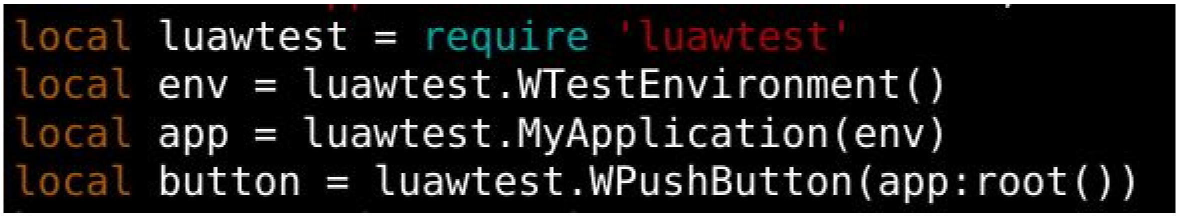


```lua
-- This code is executed in constructor of new application.
-- It setups widgets of a start page shown to a user.
local luawt = require 'luawt'

-- app is an instance of WApplication
-- env is an instance of WEnvironment
local app, env = ...

function breakTheLine()
    app:root():addWidget(luawt.WBreak())
    app:root():addWidget(luawt.WBreak())
end

function mainApp()
    app:setTitle('Hello!')
    -- Use CSS style sheet file. Note that it should be in docroot
    -- specified by WServer's options.
    app:useStyleSheet('./common.css')
    -- Enums are treated as numbers. 4 corresponds to AlignCenter.
    app:root():setContentAlignment(4)
    -- Create widgets.
    local text = luawt.WText('Your name, please?')
    breakTheLine()
    local name_edit = luawt.WLineEdit(app:root())
    breakTheLine()
    name_edit:setFocus()
    local button = luawt.WPushButton('Greet me', app:root())
    -- Set CSS style classes (from common.css).
    button:setStyleClass('button')
    text:setStyleClass('my_text')
    -- Setup singal handler.
    button:clicked():connect(function()
        text:setText('Hello there, ' .. name_edit:text())
    end)
end

mainApp()
```

Examples: hello

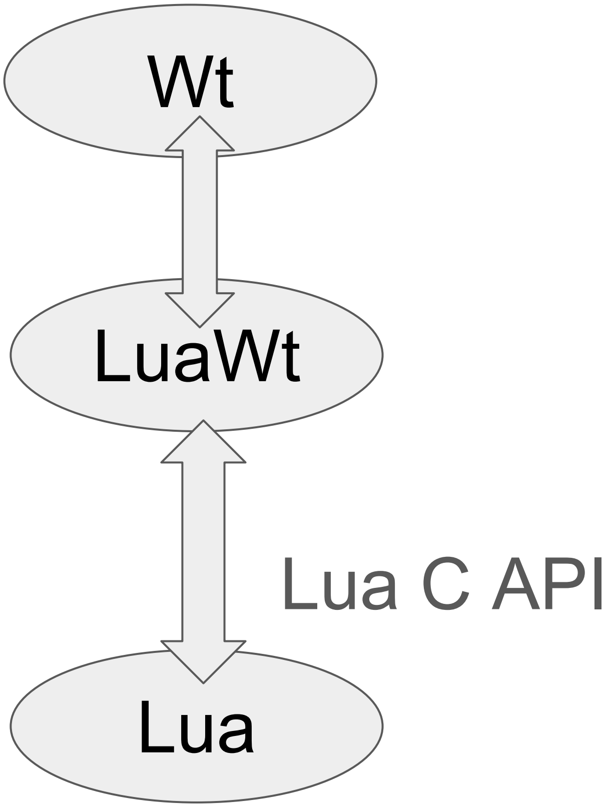


Luawtest

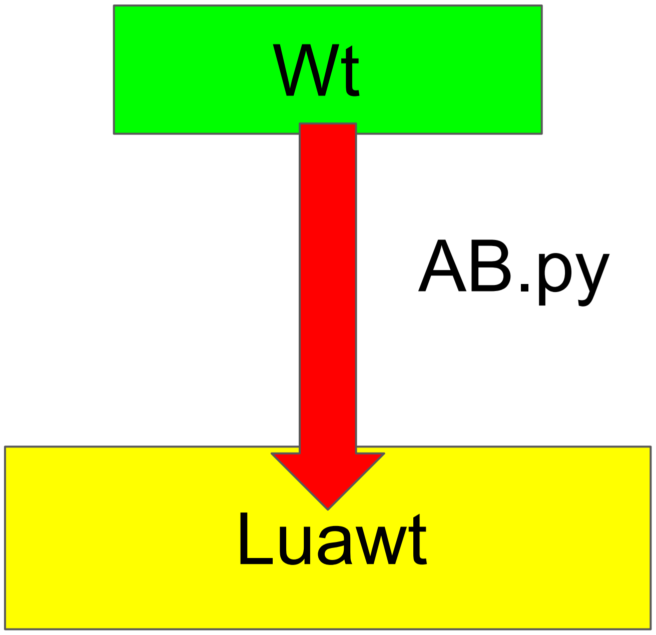


```lua
local luawtest = require 'luawtest'
local env = luawtest.WTestEnvironment()
local app = luawtest.MyApplication(env)
local button = luawtest.WPushButton(app:root())
```

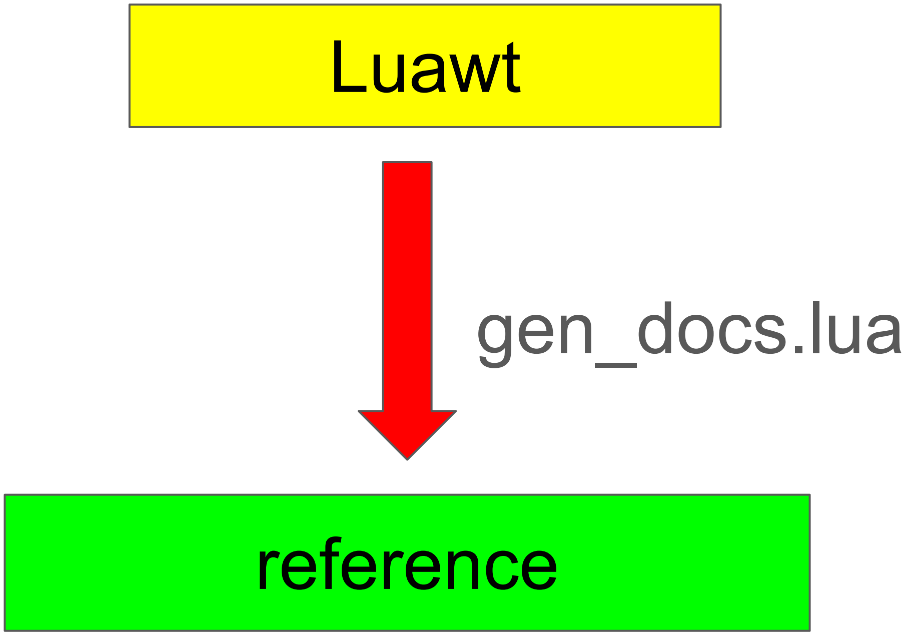
Implementation



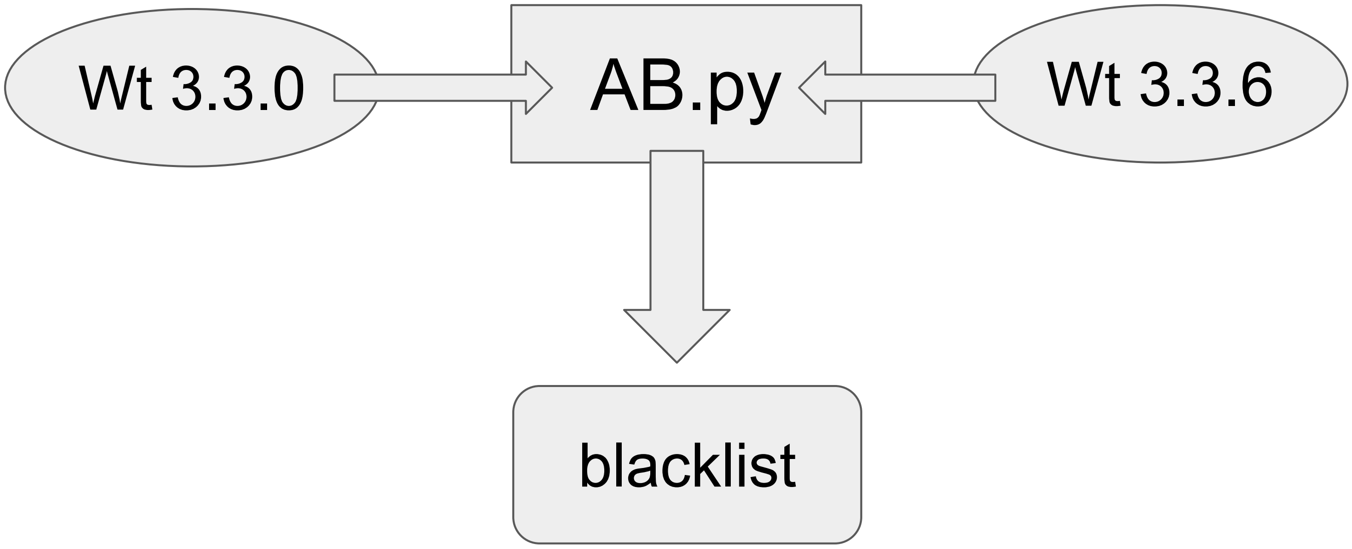
Tools: automate_bindings.py



Tools: gen_docs.lua



Compatibility with different Wt versions



Further development

- Tests
- Comprehensive documentation
- More widgets
- ….

https://github.com/LuaAndC/luawt/issues

Questions?

https://github.com/LuaAndC/luawt

https://github.com/zer0main

@zer0main