Usage Guide

Using Blink to build a local web app has two basic steps:

1. Create a window and load all your HTML and JS.
2. Handle interaction between julia and your window.

1. Setting up a new Blink Window

Create a new window via `Window`, and load some html via `body!`.

```
julia> using Blink

julia> w = Window(async=false) # Ope
Blink.AtomShell.Window(...)

julia> body!(w, "Hello World", async
```

The main functions for setting content on a window are `content!(w, querySelector, html)` and `body!(w, html)`. `body!` is just shorthand for `content!(w, "body", html)`.

You can also load an external url via `loadurl`, which will replace the current content of the window:

```
loadurl(w, "http://julialang.org") #
```

Note the use of `async=false` in the examples above. By default, these functions return immediately, but setting `async=false` will

block until the function has completed. This is important if you are executing multiple statements in a row that depend on the previous statement having completed.

Loading stadalone HTML, CSS & JS files

You can load complete standalone files via the `load!` function. Blink will handle the file correctly based on its file type suffix:

```
load!(w, "ui/app.css")
load!(w, "ui/frameworks/jquery-3.3.1
```

You can also call the corresponding `importhtml!`, `loadcss!`, and `loadjs!` directly.

2. Setting up interaction between Julia and JS

This topic is covered in more detail in the Communication page.

Just as you can directly write to the DOM via `content!`, you can directly execute javscript via the `@js` macro.

```
julia> @js w Math.log(10)
2.302585092994046
```

To invoke julia code from javascript, you can pass a "message" to julia:

```
# Set up julia to handle the "press'
handle(w, "press") do args
  @show args
end
# Invoke the "press" message from ja
body!(w, """<button onclick='Blink.m
```

Previous: Blink.jl Documentation
Next: Communication