Flare

Release 1.0

Tilman Oestereich, Andreas H. Kelch

Jun 19, 2023

FLARE

Web-App development framework for Python

CHAPTER

ONE

ABOUT

flare is an app development framework for Python-based web-apps running on top of Pyodide in the browser.

It has integrations to concepts with ViUR, an MVC-framework for the Google App Engine platform, but can also be used stand-alone.

Fire up the tiny Hello World live demo.

1.1 Getting started

1.1.1 System requirements

soon…

Serving own Pyodide

The script `bin/get-pyodide.py` downloads a minimal Pyodide with only `micropip` and `setuptools` from the Pyodide CDN. Pyodide can also be entirely built and configured on your own, for this check the documentation.

Depending on the location where you want to serve your app, some more configuration might be necessary regarding the WASM mimetype.

Google App Engine

To serve your own Pyodide via Google App Engine, add the following lines to your `app.yaml` file and modify them when needed, as Google App Engine doesn't recognize WASM files correctly.

```
handlers:
- url: /pyodide/(.*\.wasm)$
  static_files: pyodide/\1
  upload: pyodide/.*\.wasm$
  mime_type: application/wasm
- url: /pyodide
  static_dir: pyodide
```

Apache Webserver

For apache web-server, this `.htaccess` configuration helped to serve the app correctly.

```
RewriteEngine off
Options -ExecCGI +Indexes
IndexOrderDefault Descending Date

#Header always set Access-Control-Allow-Origin "*"
#Header always set Access-Control-Allow-Methods GET

<FilesMatch "\.py$">
    Options +Indexes -ExecCGI -Multiviews
    Order allow,deny
    Allow from all
    RemoveHandler .py
    AddType text/plain .py
</FilesMatch>

<FilesMatch "\.data$">
    Options +Indexes -ExecCGI -Multiviews
    Order allow,deny
    Allow from all
    RemoveHandler .data
    AddType application/octet-stream .data
</FilesMatch>

<FilesMatch "\.wasm$">
    Options +Indexes -ExecCGI -Multiviews
    Order allow,deny
    Allow from all
    RemoveHandler .wasm
    AddType application/wasm .wasm
</FilesMatch>
```

1.1.2 Setup and installation

Linux or WSL

soon...

Mac OS

soon...

1.1.3 Testproject

Setting up a new Python web-app with *flare* is fairly easy. This section describes several things and ways how *flare* can be used and configured.

HTML skeleton

Below is a shortened version of the code from *hello.html* delivered together with the *flare* repo. Such a skeleton must be individually created for an app written with *flare*.

Caution: Depending on where you put the html files, you need to change the source paths:

- <link rel="stylesheet" href="{path-to-flare-directory}/assets/css/style.css"/>
- <script src="{path-to-flare-directory}/assets/js/flare.js"></script>
- "path": "{path-to-flare-directory}/flare"

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="assets/css/style.css"/>

    <!-- (1) -->
    <script src="https://pyodide-cdn2.iodide.io/v0.16.1/full/pyodide.js"></script>
    <!-- <script src="pyodide/pyodide.js"></script> -->

    <!-- (2) -->
    <script src="assets/js/flare.js"></script>

    <script>
        window.addEventListener(
                "load",
                (event) => {
                    window.init = new flare({
                        prelude:                    // (3)
`
print("I'm before any fetch")
`,
                        fetch: {                    // (4)
                            "flare": {
                                "path": "flare"
                            }
                        },
                        kickoff:                    // (5)
`
from flare import *
html5.Body().appendChild('<a href="https://www.viur.dev">Hello World</a>')
flare.popup.Alert("Hello World")
`
                    });
                }
        );
    </script>
```

(continues on next page)

(continued from previous page)

```
</head>
<body class="is-loading"> <!-- (6) -->
</body>
</html>
```

Notable are the following sections:

1. This is the include for the used Pyodide version. When quickly setting up a project, the default CDN version of Pyodide can be used and is loaded from here. Indeed, it is also possible to serve Pyodide on your own. For this, the utility script `bin/get-pyodide.py` can be used. This script downloads a minimal version of Pyodide delivered from the CDN and stores it into a folder named `pyodide/`. In such a case, the CDN-include here must be removed, and replaced by the local include. `get-pyodide.py` patches some Pyodide-files to directly run from the URL `/pyodide`. You can override this setting by specifying a variable `window.languagePluginLoader` before including the `pyodide.js`.
2. *flare* serves a piece of JavaScript code that is necessary to pre-load flare itself and the Python application. For development, it was useful to directly fetch the py-files from the server and store them into a browser-internal filesystem when the Python interpreter from Pyodide can find it. This is done using the module in `init.js` and the configuration described next.
3. `prelude` is some Python code that is executed before any modules are fetched. It can be omitted, if not wanted.
4. `fetch` describes Python source modules that are being fetched before the application starts. This is very useful for development purposes. For every entry (which is the name of the Python package to be created), a further object describing the fetch `path` and an optional `optional` attribute is provided. Using the `path`-attribute, the *flare* init script looks for a file `files.json` which provides a listing of the files being fetched. This file is generated using `bin/gen-files-json.py` which is described below. A Pyodide package can also be pre-compiled from source files, but this is not described in detail here, yet.
5. `kickoff` is the Python code that is executed when all fetching is done and nothing failed. It is used as the entry point to start the web-app. In the *hello.html* file, it is just some "Hello World" stuff dumped out using flare.
6. The class `is-loading` is automatically removed when the kickoff code successfully executed. It can be used to show a loading animation or something similar.

Writing huger apps

When writing huger apps with multiple Python files, the above example doesn't satisfy. For this case, an HTML-file like above still serves as the entry point for the app, but requires a little more configuration.

Let's thing about the following minimal setup for a huger app:

- `/flare` is the flare repo serving as a library _ `/myapp` contains our app, which exists only of the files
  - `index.html` the app entry HTML
  - `__init__.py` the app source code
  - `files.json` which is the index file for the *flare* init script to find its sources

We only describe the files in `/myapp`:

index.html

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
```

(continues on next page)

(continued from previous page)

```
    <script src="https://pyodide-cdn2.iodide.io/v0.16.1/full/pyodide.js"></script>
    <script src="/flare/assets/js/flare.js"></script>
    <script>
        window.addEventListener(
                "load",
                (event) => {
                    window.init = new flare({
                        fetch: {
                            "flare": {
                                "path": "/flare/flare"
                            },
                            "myapp": {
                                "path": "."
                            }
                        }
                    }
                );
            }
        );
    </script>
</head>
<body class="is-loading">
</body>
</html>
```

init.py:

```
from flare import *

if __name__ == "myapp":
    html5.Body().appendChild('<a href="https://www.viur.dev">Hello World</a>')
    popup.Alert("Hello World")
```

files.json:

```
[
  "__init__.py"
]
```

The `files.json` was simply generated using the by `../flare/bin/gen-files-json.py`. Whenever a Python file is added, this must be done once. The `files.json` should also be added to version control, to make the app run out-of-the-box.

1.2 Reference Guide

1.2.1 Terminology

Originally, Flare was developed for ViUR. Because of these roots, some terms or objects may cause confusion without further explanation.

Lets start with a quick overview of some main components.

config.py
: This is a central location where you can store data or information that needs to be shared by the entire application. Some Examples are

  - paths
  - caches
  - configurations
  - versions

network.py
: The default format used for data exchange is json. Each query is made via the NetworkService class in network.py.

views
: Views are used to divide content within the application. There is always one view that is active. They are saved after their instantiation in a object and are only hooked into the DOM when this view is activated. A view can contain multiple widgets that replace the existing content when activated.

safeeval
: Executes a string containing Python code. The possible operations are strongly limited for security reasons. With safeeval *flare-if* can show and hide content without the need of coding, and *{{ expressions }}* can directly be interpreted inside of HTML-code.

icons
: The icon class can use any of the common image types. In most cases, you want to have icons that match the font color. In this case flare requires svg icons.

priorityqueues
: PriorityQueues are used to provide a plugin capability. For each type there is somewhere centrally an instance of such a PriorityQueue. In this the options are added with the insert function and prioritized with a numerical value and validation function. If now a suitable option is searched, the select function is called with parameters which are passed to the validation function. The first matching option in the prioritized list is then returned.

Only relevant if used with ViUR

bones
: Bones are in the ViUR ecosystem data field definitions. There are different types and hold besides the type information also display information like a description and tooltips.

moduleInfo
: Modules are the controllers of a ViUR application, and implement the application logic. In the case of relations, information about the target module may be required, which can be found in the module info.

1.2.2 Configuration

Flare is divided into different components, which have different complexity. In addition to a flare config, the forms and views components have their own config.

Flare config

Here are some default values configured.

- **flare.icon.svg.embedding.path**
  - defines the basepath for the used svg icons.
- **flare.icon.fallback.error**
  - defines the fallback icon name
- **flare.language.current**
  - sets the current active language

The views config will be merged on top.

Bind App

An app that uses flare often has its own config object. Flare provides a bindApp function that, in addition to setting the app instance in the configuation, also allows overriding the flare configuration. For example, you can change the default language in your app configuration and all Flare components will use that value instead of the default flare settings.

1.2.3 html5 (core library)

Any **flare** components are entirely established on top of the *html5*-library.

The *html5 library* is flare's core module and key feature, and manages access to the browser's DOM and its items, by implementing a Python object wrapper class for any HTML-element. Such an element is called *widget*. For example, `html5.Div()` is the widget representing a div-element, or `html5.A()` a widget representing an a-element. Widgets can be sub-classed into specialized components, which contain other widgets and components and interact together.

The document's body and head can directly be accessed by the static widgets `html5.Head()` and `html5.Body()`.

All these widgets are inheriting from an abstract widget wrapper called `html5.Widget`. `html5.Widget` is the overall superclass which contains most of the functions used when working with DOM elements. Therefore, all widgets are usually handled the same way, except leaf-type widgets, which may not contain any children.

First steps

When working with native html5-widgets, every widget must be created separately and stacked together in the desired order. This is well known from JavaScript's createElement-function.

Here's a little code sample.

```
from flare import html5

# Creating a new a-widget
a = html5.A()
a["href"] = "https://www.viur.dev"  # assign value to href-attribute
```

(continues on next page)

(continued from previous page)

```
a["target"] = "_blank"              # assign value to target-attribute
a.addClass("link")                  # Add style class "link" to element

# Append text node "Hello World" to the a-element
a.appendChild(html5.TextNode("Hello World"))

# Append the a-widget to the body-widget
html5.Body().appendChild(a)
```

Summarized:

- `html5.Xyz()` creates an instance of the desired widget. The notation is that the first letter is always in uppercase-order, the rest is hold in lowercase-order, therefore e.g. *`html5.Textarea()`* is used for a textarea.
- Attributes are accessible via the attribute indexing syntax, like `widget["attribute"]`. There are some special attributes like `style` or `data` that are providing a dict-like access, so `widget["style"]["border"] = "1px solid red"` is used.
- Stacking is performed with `widget.appendChild()`. There are also some additional functions for easier element stacking and child modification, these are - `widget.prependChild()` to prepend children, - `widget.insertBefore()` to insert a child before another child, - `widget.removeChild()` to remove a child.
- To access existing child widgets, use `widget.children(n)` to access the *n*-th child, or without *n* to retrieve a list of a children.

Parsing widgets from HTML-code

Above result can also be achieved much faster, by using the build-in *html5-parser and renderer*.

```
from flare import *

html5.Body().appendChild(
    "<a href='https://www.viur.dev' target='_blank' class='viur'>Hello World</a>"
)
```

That's quite simpler, right? This is a very handy feature for prototyping and to quickly integrate new HTML layouts.

`Widget.appendChild()` and other, corresponding functions, allow for an arbitrary number of elements to be added. HTML-code, widgets, text or even lists or tuples of those can be given, like so

```
ul = html5.Ul()
ul.appendChild("<li class='is-active'>lol</li>")
ul.prependChild(html5.Li(1337 * 42))
ul.appendChild("<li>me too</li>", html5.Li("and same as I"))
```

The HTML parser can also do more: When component classes (any class that inherits directly from html5.Widget, like html5.Div or so) are decorated with the *html5.tag*-decorator, these are automatically made available in the HTML-parser for recognition.

Inheritance is normal

In most cases, both methods shown above are used together where necessary and useful. Especially when creating new components with a custom behavior inside your app, knowledge of both worlds is required.

To create new components, inheriting from existing widgets is usual. If we would like to add our link multiple times within our app, with additional click tracking, we can make it a separate component, like so:

```
import logging
from flare import *

class Link(html5.A):  # inherit Link from html5.A widget
    def __init__(self, url, *args, target="_blank", **kwargs):
        super().__init__()
        self.addClass("link")
        self["href"] = url
        self["target"] = "_blank"

        self.appendChild(*args, **kwargs)
        self.sinkEvent("onClick")

    def onClick(self, event):
        logging.info(f"The link to {self['href']} has been clicked")

html5.Body().appendChild(
    # Create a link with text
    Link("https://www.viur.dev", "ViUR Framework"),

    "<br>",

    # Create link with logo
    Link("https://www.python.org", """
        <img src="https://www.python.org/static/community_logos/python-powered-h-50x65.
↪png"
            title="Python Programming Language">
    """)
)
```

In this example, we just made our first custom component: The `Link`-class can be arbitrarily used.

Widget basics

Following sections describe the most widely used functions of the :class:`html5.Widget <flare.html5.Widget>`-class which are inherited by any widget or huger component in flare.

Constructor

All widgets share the same __init__-function, having the following signature:

```
def __init__(self, *args, appendTo=None, style=None, **kwargs)
```

- *args are any positional arguments that are passed to self.appendChild(). These can be either other widgets or strings containing HTML-code. Non-container widgets like html5.Br() or html5.Hr() don't allow anything passed to this parameter, and throw an Exception.
- appendTo can be set to another html5.Widget where the constructed widget automatically will be appended to. It substitutes an additional *appendChild()*-call to insert the constructed Widget to the parent.
- style allows to specify CSS-classes which are added to the constructed widget using
- **kwargs specifies any other parameters that are passed to appendChild(), like variables.

Insertion and removal

These methods manipulate the DOM and it's nodes

appendChild()

Appends another html5.Widget as child to the parent element:

```
self.appendChild("""<ul class='navlist'></ul>""")
self.nav.appendChild("""<li>Navigation Point 1</li>""")
```

prependChild()

Prepends a new child to the parent element

```
self.appendChild("""<ul class='navlist'></ul>""")
navpoint2 = self.nav.appendChild("""<li>Navigation Point 2</li>""")
navpoint2.prependChild(("""<li>Navigation Point 1</li>"""))
```

replaceChild()

Same as appendChild(), but removes the current children of the Widget first.

insertBefore()

Inserts a new child element before the target child element

```
self.appendChild("""<ul class='navlist'></ul>""")
navpoint = self.nav.appendChild("""<li>Navigation Point 1</li>""")
navpoint3 = self.nav.appendChild("""<li>Navigation Point 3</li>""")
navpoint2 = self.nav.insertBefore("""<li>Navigation Point 2</li>""", navpoint3)
```

If the child element that the new element is supposed to be inserted before does not exist, the new element is appended to the parent instead.

removeChild(), removeAllChildren()

Either removes one child from the parent element or any available children.

Visibility and usability

Widgets can be switched hidden or disabled. Form elements, for example, might be disabled when a specific condition isn't met. These functions here help to quickly change visibility and usability of widgets, including their child widgets which are switched recursively.

hide(), show()

Hides or shows a widget on demand.

To check whether a widget is hidden or not, evaluate `widget["hidden"]`. In the HTML-parser, this flag can be set using the `hidden` attribute, e.g. `<div hidden>You can't see me.</div>`.

enable(), disable()

Enable or disable the widget in the DOM. Useful for forms and similar UI applications.

To check whether a widget is disabled or not, evaluate `widget["disabled"]`. In the HTML-parser, this flag can be set using the `disabled` attribute, e.g. `<div disabled>I'm disabled</div>`.

class-attribute modification

These methods are helpful for adding CSS-classes quickly.

addClass()

Adds a class to the html5.Widget and checks to prevent adding the same class multiple times.

```
nav = self.appendChild("""<ul></ul>""")
nav.addClass('navlist')
```

Adding a class multiple times might be wanted and is valid. In this case, modify the widget's `class`-attribute directly by assigning a list to it.

removeClass()

Checks if the widget has that class and removes it

```
nav = self.appendChild("""<ul class='big-red-warning-border-color'></ul>""")
nav.removeClass('big-red-warning-border-color')
```

toggleClass()

Toggles a class *on* or *off*, depending on whether it has the specified class already or not.

hasClass()

Checks if the element has a given class or not. Returns True if class name is found and False otherwise.

```
nav = self.appendChild("""<ul class='big-red-warning-border-color'></ul>""")
if nav.hasClass('big-red-warning-border-color'):
    print("Help! There is a big red border around this element! Remove the class so we␣
↪can feel safe again")
```

HTML parser reference

The html5-library built into flare brings its own HTML-parser. Using this parser, any HTML-code can directly be turned into a flare DOM.

Additionally, some nice extensions regarding flare component and widget customization and conditional rendering is supported, as the HTML-renderer automatically creates the DOM from a parsed input and serves as some kind of template processor.

Data-based rendering

Using variables

Any variables provided via kwargs to *html5.fromHTML()* can be inserted in attributes or as TextNode-elements with their particular content when surrounded by {{ and }}. Inside this notation, full Python expression syntax is allowed, so that even calculations or concatenations can be done.

```
html5.Body().appendChild("""
    <div class="color-{{ l[1] + 40 }}">{{ d["world"] + "World" * 3 }} and {{ d }}</div>
""", l=[1,2,3], d={"world": "Hello"})
```

renders into

```
<div class="color-42">HelloWorldWorldWorld and {'world': 'Hello'}</div>
```

flare-if, flare-elif, flare-else

The attributes `flare-if`, `flare-elif` and `flare-else` can be used on all tags for conditional rendering.

This allows for any simple Python expression that evaluates to True or any computed non-boolean value representing True.

```
html5.Body().appendChild("""
    <div>begin</div>
    <div flare-if="i <= 10">i is just low</div>
    <div flare-elif="i <= 50 and j >=100">i and j have normal values</div>
    <div flare-elif="i > 50 and j >= 50">i and j have moderate values</div>
    <div flare-else>i and j are something different</div>
    <div>end</div>
""", i=50, j=151)
```

As variables, any arguments given to *html5.fromHTML()* (or related functions) as kwargs can be used.

html5.parseHTML()

```
def parseHTML(html: str, debug: bool=False) -> HtmlAst
```

Parses the provided HTML-code according to the tags registered by html5.registerTag() or components that use the *@tag*-decorator.

The function returns an abstract syntax tree representation (HtmlAst) of the HTML-code that can be rendered by *html5.fromHTML()*.

html5.fromHTML()

```
def fromHTML(html: [str, HtmlAst], appendTo: Widget=None, bindTo: Widget=None, debug:␣
↪bool=False, **kwargs) -> [Widget]
```

Renders HTML-code or compiled HTML-code (HtmlAst).

- appendTo: Defines the Widget where to append the generated widgets to
- bindTo: Defines the Widget where to bind widgets using the `[name]`-attribute to
- debug: Debugging output
- **kwargs: Any specified kwargs are available as *variables to any expressions*.

HTML-code can optionally be pre-compiled with *`html5.parseHTML()`*, and then executed multiple times (but with different variables) by fromHTML. This is useful when generating lists of same elements with only replaced variable data.

@html5.tag

Decorator to register a sub-class of `html5.Widget` either under its class-name, or an associated tag-name.

Examples:

```
from flare import html5

# register class Foo as <foo>-Tag
@html5.tag
class Foo(html5.Div):
    pass

# register class Bar as <baz>-Tag
@html5.tag("baz")
class Bar(html5.Div):
    pass
```

1.2.4 Ignite

Ignite is a CSS-framework written in LESS and serving as the base for all components used in flare. https://ignite.viur.dev/

In Flare, some simpler and more complex components are already implemented with appropriate CSS-classes.

Button

The Button can be used with `<flare-button>` tag und provides the possibility to add an icon before the Button text.

Input

The Input can be used with <flare-input> tag and provides the basis input element with ignite specific css classes.

Label

The Label can be used with <flare-label> tag and provides the basis label element with ignite specific css classes.

Switch

The switch is an on/off slide-control and can be used with <flare-switch> tag. The component stores the current state internally in a checkbox input field.

Check

The Check Component can be used with <flare-check> tag. Like the switch, the internal state is stored in a checkbox input field. Through this component the display of the checkbox can be customized via css.

Radio

The Radio Component can be used with <flare-radio> tag. The internal state is stored in a radio input field. Through this component the display of the checkbox can be customized via css.

Select

The Select can be used with <flare-select> tag and provides the basis select element with ignite specific css classes. In addition it adds per default a unselectable default option.

Textarea

The Textarea can be used with <flare-textarea> tag and provides the basis textarea element with ignite specific css classes.

Progress

The Progress can be used with <flare-progess> tag and provides the basis progress element with ignite specific css classes.

Item

The Item component can be used with the tag <flare-item> and provides a simple box component. It can contain an image as well as a title and a description

Table

The Table component can be used with the <flare-table> tag and provides the basis table element with ignite specific css classes. In additon this component provides the functions prepareRow and prepareCol to generate the table grid.

Popout

The Popout component can be used with the <flare-popout> tag. This component is a floating box and is often used as a tooltip or contextmenu. With the css classes "popout–sw", "popout–nw".. you can change the direction.

Popup

The are several types of popups windows. All popups are based on the base Popup Class. Each popup provides a close button, a header, a body and a footer. All Popups are automatically added to the <*body*>-tag.

Prompt

The Prompt is a simple Input box with a cancel and ok button. Use this to get some user Input.

Alert

The Alert is a simple Messagebox with an ok button. Use this for some Feedback.

Confirm

The Confirm is a Messagebox with a yes / no selection. Each button has its own callback so you can bump different actions based on the selection that was made

Textarea Dialog

This Popup basically does the same as the Prompt, but it uses a textarea field instead of an input field.

1.2.5 Network

The *network*-module contains some classes and functions that allow to communicate or work with other services.

Requesting data

The following classes are used to request data from another service.

HTTPRequest

HTTPRequest is a tiny wrapper around the Javascript object XMLHttpRequest. Only the OPENED (1) and DONE (4) statuses are used. In case of OPENED the payload is sent. If it is a post request, a possibly existing content type header is also set. Depending on the status, the success callback or the failure callback specified during instantiation is called.

```
HTTPRequest("GET", url, mySuccessFunction, myFailureFunction)
```

This tiny wrapper is used by the NetworkService, which encapsulates some ViUR-related request types

NetworkService

This function can be passed the following parameters in addition to the callback functions for success, failure and finished:

- module (str): Name of the target ViUR Module or None
- url (str): Path (relative to Module)
- params (dict): Dictionary of key-values paired url parameters
- modifies (bool): previously registered classes can be notified with a onDataChanged event
- secure (bool): for this ViUR request is an skey need, so fetch it before the request
- kickoff (bool): by default this value is true, but you can use it to wait before to start a request
- group (requestGroup): use this to bundle multiple requests and get at the end a final callback

This could be a simple request to test on a ViUR System if a user is logged in

```
NetworkService.request( "user", "view/self",
                    successHandler=iamAlreadyLoggedInFunction,
                    failureHandler=loginFunction)
```

Sometimes you need to do a bunch of requests with a callback at the end

```
agroup = requestGroup( allRequestsSuccessFunction )

for aKey in dbKeyListToDelete:
        NetworkService.request( amodule, "delete", { "key": aKey },
                        secure = True, #in case of deletion ViUR needs an skey
                        modifies = False, #avoids the onDataChanged event
                        group=agroup,
                        successHandler = singleItemSuccessFunction,
                        failureHandler = singleItemFailureFunction )
```

requestGroup

This class is used to execute several requests of the NetworkService one by one and finally call the callback specified during instantiation. In this case, be sure to set kickoff to False.

Other useful functions

The following functions were often used in connection with data queries and were therefore placed here.

DeferredCall

This is a wrapper around the setTimeout JavascriptObject. After a delay time (default:25ms) the given function is called with the given parameters. This function is called outside the surrounding application flow! Two hidden parameters can be specified during initialization and will not be passed to the function:

- _delay: modifies the Timeout delay
- _callback: will be called after handling the deferred Funktion

```
DeferredCall(doSomeStuffLaterFunction,
        anArgumentForMyFunction,
        _delay=1000,
        _callback=sayHelloWennFinishedFunction)
```

1.2.6 Utils

soon...

1.2.7 Url handling

Flare Applications are SPA (Single Page Applications) the navigation is done via the #-hash part of the url. This part is treated by Flare like a normal url. The hash should have a form like this.

```
#/path/pathPart2../pathEnd?param1=value&param2=value
```

The following functions split the hash into the corresponding url components or reassemble them.

getUrlHashAsString

This function takes the hash of the url and splits it into args and kwargs. The return value is a tuple of the args string and the kwargs string. In most cases you want to use getUrlHashAsObject instead.

getUrlHashAsObject

Uses the return value of getUrlHashAsString and also creates a tuple consisting of args and kwargs. But now the first value is a list and the second is a dictionary.

setUrlHash

This function takes the objects from getUrlHashAsObject and reassembles them into a valid hash and finally sets the new url.

```
urlHash, urlParams = getUrlHashAsObject() #read hash
urlParams.update({"key":"newValue"}) #modify
setUrlHash(urlHash,urlParams) #write back
```

example

```
# current URL:
#  http://localhost:8080/app/app.html#/user/list?amount=99&status=10
urlHash, urlParams = getUrlHashAsObject() #read hash
print(urlHash,urlParams)
#['user','list'] {"amount":"99","status":"10"}
urlParams.update({"status":"5"}) # change query
setUrlHash(urlHash,urlParams) #write back to Url
# new URL:
#  http://localhost:8080/app/app.html#/user/list?amount=99&status=5
```

1.2.8 i18n

Flare provides the possibility to translate texts depending on the selected language. For each language a Python file with the language abbreviation is created in a folder called 'translations'. A dictionary with the following name format is then expected in the file:

```
lngDe
lngEn
lngNl
lngFr
...
```

The dictionary itself contains a mapping between a keyword and the translation in the corresponding language.

```
lngDe = {
        "List": "Liste",
        "Username": "Nutzername ist {name}"
        ...
}
```

To use these dictionaries they have to be initialized when starting the application.

```
from flare.i18n import buildTranslations,translate
buildTranslations("app") #the parameter is the name of the root folder
```

(continues on next page)

(continued from previous page)

```
#now you can use the translate function to get the translated text

print(translate('List'))
# "Liste"
```

translate

The Translate function can additionally have a fallback and any other parameters, which then replace marked positions in a template string.

```
print(translate('Username', {"name":"Alice"}))
# "Nutzername is Alice"
```

addTranslation

You can also update a translation at runtime. For this you have to specify the language, the keyword and the translation.

```
addTranslation("de","user","Nutzer")
```

more functions

the functions getLanguage and setLanguage(lang) allow to change and request the current language. After changing the language, it must be ensured that templates are rebuilt.

1.2.9 SVG Icons

Icons are dependent on css styling in flare. So icons in a text can have the same color as the surrounding text. This is possible by embedding svg icons. If the tag flare-svg-icon is used, the parameter value can be used to specify a path or name to an icon.

```
<flare-svg-icon value="/static/icons/my-icon.svg">
```

To keep the code in flare clear, only the icon name can be specified. If only the name is specified, the config variable conf["flare.icon.svg.embedding.path"] is used to compose the path of flare.

```
<flare-svg-icon value="my-icon">
```

It is also possible to define a fallback icon in case the icon cannot be loaded. If the title is set it will be transferred to the svg and in case the fallback icon is not set the first character of the text will be used as placeholder.

```
<flare-svg-icon value="my-icon" fallbackIcon="error" title="My Icon">
<!-- shows my-icon or on error the error icon -->

<flare-svg-icon value="my-icon" title="My Icon">
<!-- shows my-icon or the letter M -->
```

Icon

In practice icons can come in different file types. flare-icon can also handle other images and even use filebones directly. In case the icon is not an svg, it is not embedded, but included using img-tag. flare-icon can use the following image types:

- *.svg, *.jpg, *.png, *.gif, *.bmp, *.webp, *..jpeg

```
<flare-icon value="{{skel['image']}}">
<!--loads a filebone-->
```

The Svg-icon parameters fallbackIcon and title are also supported.

1.2.10 Views

Views allow switching between different widgets. A view must inherit from the View class abd can update multiple View-Widgets of the conf["app"]. Additionally, the dictOfWidgets must be filled in the constructor before the super call. The key of the dictionary must be present in the conf["app"] widget.

A view widget is the actual content that is then inserted into a widget in the main app. These view widgets must inherit from ViewWidget.

The currently active view is stored in a global state under conf["views_state"].

create a View

```
from flare.views.view import View, ViewWidget

class myView(View):

        def __init__(self):
                dictOfWidgets = {
                        "content"     : myViewContent
                        #each key muss exists as instancevariable in conf["app"]
                }
                super().__init__(dictOfWidgets)

class myViewContent(ViewWidget):

        def initWidget( self ):
                self.appendChild("Hello View")

        def onViewfocusedChanged( self, viewname, *args, **kwargs ):
                pass #here we can execute code, which muss be called wenn e View gets␣
↪focus
```

register a View

At this point, we have created a view. We have defined that the widget content from the main app should be replaced by the one from myViewContent. Now we need to register this view.

```
from flare.views.helpers import addView, removeView

addView(myView,"page1")
```

In this example, the view myView is registered under the name page1. A view can also be registered under multiple names. removeView removes the view again.

activate / switch a view

To activate a view the view instance of the state conf[“views_state”] must be updated. The status stores the name of the view that is currently displayed and can be updated as follows.

```
conf["views_state"].updateState("activeView", "page1")
```

Views with ViUR

In ViUR, modules have different views depending on the handler. generateView here encapsulates module name, actionname and data away in a params dictionary, which is then available in the view and can be loaded from the view in any ViewWidget.

```
#item is a adminInfo Entry

# generate a unique instancename, because a edit can be opened multiple times with same␣
↪parameters
instancename = "%s___%s" % (item[ "moduleName" ]+item[ "handler" ], str( time.time() ).
↪replace( ".", "_" ))

#create new viewInstance
viewInst = generateView( myView, item[ "moduleName" ], item[ "handler" ], data = item,␣
↪name=instancename )

#register this new view
conf[ "views_registered" ].update( { instancename: viewInst } )

# somewhere else in code, i.e in a Navigation
conf["views_state"].updateState("activeView", instancename)
```

1.2.11 ViUR

soon...

1.2.12 Safeeval

soon...

1.3 Tutorials

1.3.1 Hello World

In this tutorial, we will create a basic project that makes use of flare to create a simple web-app.

Project setup

In order to make flare accessible in your project, either download the flare master branch from github and extract it into a `flare` subdirectory in your project, or - if you are using git - clone it into a git submodule of your project by calling `git submodule add git@github.com:viur-framework/flare.git`.

Once this is done, you can create an `index.html` file that will make use of the now available flare assets.

The HTML

Basically all you need to do is add the flare CSS sheet and javascript file to your HTML file and you are good to go.

```
<link rel="stylesheet" href="flare/assets/css/style.css"/>
<script src="flare/assets/js/flare.js"></script>
```

A simple `index.html` file that uses flare might now look like this:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello World</title>
    <link rel="stylesheet" href="flare/assets/css/style.css"/>
    <script src="flare/assets/js/flare.js"></script>
    <script>
        window.addEventListener("load", () => {
            new flare({
                fetch: {
                    "flare": {
                        "path": "flare/flare"
                    }
                },
                kickoff:
`
from flare import *
flare.popup.Alert("Hello World")
```

(continues on next page)

(continued from previous page)

```
`
            });
        });
    </script>
</head>
<body class="is-loading">
</body>
</html>
```

Building from there

The `fetch` block is where the flare python modules are being loaded at application start. It is advisable to add your own python module structure fairly quickly.

1. Add a subdirectory `helloworld` next to your `index.html`.
2. Add a file `__init__.py`:

```
from . import helloworld
```

3. Add a file `helloworld.py`:

```
from flare import *


class HelloWorld(object):
    _message = None

    def __init__(self, message="Hello World"):
        self._message = message

    def show(self):
        popup.Alert(self._message)
```

4. Create a `files.json` file in your module directory and add the following content:

```
[
    "__init__.py",
    "helloworld.py"
]
```

5. Add a second block to the *fetch* in your index.html:

```
fetch: {
    "flare": {
        "path": "flare/flare"
    },
    "helloworld": {
        "path": "helloworld"
    }
},
```

6. Change your kickoff script to run the code in your module, instead:

```
from helloworld import *
helloworld.HelloWorld("Hello module world!").show()
```

To execute your hello world sample you can use the test webserver located in the `flare/tools/` folder. Just run `test-server.py` in your project directory and open `http://localhost:8080/index.html` in your browser.

1.3.2 Request JSON data

In this tutorial, we will use flares API to load some JSON data from an API and process it.

Project setup

Please refer to the "Hello World" tutorial on how to set up a basic project with flare.

Using HTTPRequest

Flare comes with a high level API to request data. The `flare.network` module contains a class `HTTPRequest`, whose constructor takes six parameters:

1. `method`: The HTTP method to use for the request (i.e. `GET`, `POST`, ...)
2. `url`: The URL to request
3. `callbackSuccess` (optional): A reference to the function which is to be called when the request succeeds (takes a response parameter)
4. `callbackFailure` (optional): A reference to the function which is to be called when the request fails (take the parameters responseText and status)
5. `payload` (optional): The body of the request, if one is to be sent
6. `content_type` (optional): A value for a `Content-Type` header (e.g. `application/json`)

Using this constructor immediately sends the request.

Handling the response

In order to parse a JSON response in a success callback, simply use the default `json` functionality:

```
def successCallback(result):
    data = json.loads(result)
```

This will simply turn the response into the appropriate native structure, based on what kind of JSON has been returned:

- Objects will be turned into `dict`
- Arrays will be turned into `list`
- `null` will be turned into `None`
- atomar values will be turned into their respective python counterpart

Example

As an example, we will request the current time of the time zone `Europe/Berlin` from a public API, then display it in a popup.

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Fetching data</title>
    <link rel="stylesheet" href="flare/assets/css/style.css"/>
    <script src="flare/assets/js/flare.js"></script>
    <script>
        window.addEventListener("load", () => {
            new flare({
                fetch: {
                    "flare": {
                        "path": "flare/flare"
                    }
                },
                kickoff:
`
import json
import logging
from flare import *
from flare.network import HTTPRequest

def _successCallback(result):
    data = json.loads(result)
    flare.popup.Alert(data["datetime"])

def _failureCallback(responseText, status):
    logging.error("Failure: %s %d", responseText, status)

HTTPRequest(
    "GET",
    "http://worldtimeapi.org/api/timezone/Europe/Berlin",
    _successCallback,
    _failureCallback
)
`
            });
        });
    </script>
</head>
<body class="is-loading">
</body>
</html>
```

1.3.3 Widgets

In this tutorial, we will introduce the widgets of flare.

Reusable UI components

Flare widgets are UI components that encapsulate one HTML element. They render their content using a template, other widgets or a mixture of both. When using templates, placeholders can be used to fill in dynamic data, elements can be qualified with a name to make them accessible, and events - such as a click - can have a handlers registered to.

Note that the template rendering will be performed only once, initially. After that, any changes to the content of the widget need to be performed programmatically. This should be kept in mind, especially when deciding where to draw the line between putting more complexity in the template and creating sub widgets.

Simple Widget

To get acquainted with widgets, we will create a simple one that just contains some static HTML.

```
from flare import html5


class SimpleWidget(html5.Div):
    def __init__(self):
        super().__init__(
            # language=HTML
            """
            <span style="color: red;">I am a </span><span style="color: blue;">Widget!</
↪span>
            """
        )
```

As we can see, our SimpleWidget contains just two basic HTML ``s, wrapped in a `` (as denoted by the derivation from `html5.Div`. That means, that our widget fundamentally is a , and can be treated as such. For example, it can now simply be put into the DOM somewhere. Try it out by just appending it to the <body>:

```
html5.Body().appendChild(SimpleWidget())
```

Placeholders

Static widgets like the one we just created do have their uses, but most of the time, you will want to display some dynamic data as well. For this, the template provided in the super constructor call can be enriched with placeholders, which are then replaced with the given data when the template is being rendered.

```
class ParametrizedWidget(html5.Div):
    def __init__(self, number: int):
        super().__init__(
            # language=HTML
            """
            My parameter is <span style="color: red;">{{number}}</span>
            """, number=number
        )
```

(continues on next page)

(continued from previous page)

```
html5.Body().appendChild(ParametrizedWidget(5))
```

Now our widget expects a "number" parameter, which is then passed into the template. However, you need to tell the template, which parameter is to be supplied with what value, which is a bit awkward. A much better approach is to expect a `dict` in your constructor signature, and unpack it when passing it to the super constructor.

```
class ParametrizedWidget(html5.Div):
    def __init__(self, parameters: dict):
        super().__init__(
            # language=HTML
            """
            My parameter is <span style="color: red;">{{number}}</span>
            """, **parameters
        )


html5.Body().appendChild(ParametrizedWidget({"number": 5}))
```

Note that the rendering of the template happens only once. If the parameters change after that, there is no built in reactivity; you have to handle these cases yourself. Let's look into that now.

Placeholders

What we're gonna build now is a widget that reacts to an event by increasing a number and displaying it. We will create a counter widget that displays a number, with a button that increases the number whenever it is clicked.

```
class CounterWidget(html5.Div):
    value = 0

    def __init__(self):
        super().__init__(
            # language=HTML
            """
            Counter: <span [name]="valueDisplay">{{value}}</span> <button @click=
↪"increase">Increase!</button>
            """, value=self.value
        )

    def increase(self):
        self.value += 1
        self.valueDisplay.replaceChild(self.value)
```

First, let's take a look at the `[name]` attribute. This attribute registers the element on which it is defined as a field of your widget class. As a result of that, we can simply access the that contains the number in the `increase` method with the field name given by the `[name]` attribute value, which in this case is `valueDisplay`.

Next, we are registering the `increase` method as a handler on the click event of the button, by using the `@click` attribute on the <button> element.

What the `increase` method then does, is increase `value` by one. But since, as stated, this will not do anything by itself, it also updates the content of the called `valueDisplay`, by replacing its content with the new value.

Conditional elements

When working with widgets, you often want to exclude elements from being rendered in certain conditions. As an example, your widget might combine a picture and a text, but you want to support the case that just one of both is provided.

You do not need to manipulate the DOM manually after rendering to achieve this. You can use the `flare-if` attribute instead. If the expression you provide as its value is not truthy, the element on which it is placed is excluded from rendering.

Let's build that widget that combines a picture and text.

```
class ImageAndTextWidget(html5.Div):
    def __init__(self, parameters: dict):
        super().__init__(
            # language=HTML
            """
            <div flare-if="pictureUrl" style="text-align: center;">
                <img src="{{pictureUrl}}"/>
            </div>
            <p flare-if="text">{{text}}</p>
            """, **parameters
        )


html5.Body().appendChild(ImageAndTextWidget(
    {
        "pictureUrl": "https://upload.wikimedia.org/wikipedia/commons/thumb/4/46/A_
↪kitten_on_the_lawn_%28Pixabay%29.jpg/640px-A_kitten_on_the_lawn_%28Pixabay%29.jpg",
        "text": "Look! Kitty likes the flowers!"
    }
))
```

Running this, you get a picture of a kitten with some text below. If you now play around with it, you will notice that by omitting the `pictureUrl`, you do not get a broken image, but the entire that contains the image is gone. The same goes for the <p> if you provide no text.

1.3.4 Views

In this tutorial, we will introduce flare's concept of views.

Building blocks

There are two basic concepts in flare in regards to views: The view itself, and the view widgets.

In essence, a view has a name, and it consists of a collection of view widgets, with the information on where in the DOM to actually display them. A view widget is a special kind of widget based on a html5 , that is being hooked into and removed from the appropriate place in the DOM. It also gets a notification whenever view switching occurs.

Views rely on the concept of a central "app" class. Since the view mostly just contains a dictionary on which view widget to place where, it expects to find an app class which has the target elements as fields. The view will then use the field name as key in its view widget dictionary.

Example

As an example, we are going to create simple flip flop views: Two views, each containing a button to show the other view.

For this, we create a new file `views.py` where we put all the following code. We start off with the view widgets of the two views:

```
from flare import html5, bindApp
from flare.button import Button
from flare.config import conf, updateConf
from flare.views.view import View, ViewWidget
from flare.views.helpers import addView, removeView, updateDefaultView


class FlipViewContent(ViewWidget):
    def initWidget(self):
        self.appendChild(Button("Flip!", self.switch))
        self.appendChild(" - Flop!")

    def onViewfocusedChanged(self, viewname, *args, **kwargs):
        pass

    def switch(self):
        conf["views_state"].updateState("activeView", "flop")


class FlopViewContent(ViewWidget):
    def initWidget(self):
        self.appendChild("Flip! - ")
        self.appendChild(Button("Flop!", self.switch))

    def onViewfocusedChanged(self, viewname, *args, **kwargs):
        pass

    def switch(self):
        conf["views_state"].updateState("activeView", "flip")
```

These two view widgets are virtually identical. They both contain a button that calls their `switch` method, which triggers the switch over to the other view, by changing `activeView` to the name of the other view. Next, we define the view classes themselves.

```
class FlipView(View):
    def __init__(self):
        super().__init__({
            "content": FlipViewContent
        })


class FlopView(View):
    def __init__(self):
        super().__init__({
            "content": FlopViewContent
        })
```

Again, these two views are virtually identical. All they do is contain the information on where to put their respective content. In this case, they both only have one view widget, and they both bind it to the same place: An element named `content`. In order for this resolution to work, there needs to be an app class, which has a field named `content` which points to the element where the view shall be rendered. Let's build one.

```
class App(html5.Div):

    def __init__(self):
        super(App, self).__init__()
        html5.Body().appendChild(self)
        bindApp(self, conf)
```

As you can see, we derive our app class from a div, hook it into the DOM, and call `bindApp` to register it in the configuration, so that the view system can access it. Now we add the `content` field to it, and make sure that it is properly connected to the DOM:

```
class App(html5.Div):
    content = html5.Div()

    def __init__(self):
        super(App, self).__init__()
        html5.Body().appendChild(self)
        bindApp(self, conf)
        self.appendChild(self.content)
```

Only one final step remains: Registering the two views, setting the flip view to active, and actually running the app.

```
class App(html5.Div):
    content = html5.Div()

    def __init__(self):
        super(App, self).__init__()
        html5.Body().appendChild(self)
        bindApp(self, conf)
        self.appendChild(self.content)
        addView(FlipView, "flip")
        addView(FlopView, "flop")
        conf["views_state"].updateState("activeView", "flip")


app = App()
```

As seen with the `addView` calls, we register the two view classes, giving them the names "flip" and "flop". These names are then used to switch the `activeView`, as we already did earlier in the `switch` methods of the view widgets.

That's it. Make flare load your `views.py` by adding the following lines of code to you `__init__.py` file:

```
from . import views

views.App()
```

Now and you can have fun with flipping and flopping the two views.

1.3.5 Translation

soon...

1.3.6 Update url

soon...

1.4 API Reference

This page contains auto-generated API reference documentation[1].

1.4.1 flare

Flare is an application development framework for writing software frontends in pure Python.

Subpackages

flare.html5

Submodules

flare.html5.core

HTML5 Widget abstraction library.

- Provides a Widget-abstraction for each HTML-element
- Routing of attribute getter/setter and Jquery-style helpers
- Fully-integrated HTML-parser for quick Widget prototyping

Module Contents

Classes

TextNode	Represents a piece of text inside the DOM.
_WidgetClassWrapper	Built-in mutable sequence.
_WidgetDataWrapper	dict() -> new empty dictionary
_WidgetStyleWrapper	dict() -> new empty dictionary
Widget	
_attrLabel	
_attrCharset	

continues on next page

[1] Created with sphinx-autoapi

Table 1 – continued from previous page

_attrCite
_attrDatetime
_attrForm
_attrAlt
_attrAutofocus
_attrDisabled
_attrChecked
_attrIndeterminate
_attrName
_attrValue
_attrAutocomplete
_attrRequired
_attrMultiple
_attrSize
_attrFor
_attrInputs
_attrFormhead
_attrHref
_attrTarget
_attrType
_attrMedia
_attrDimensions
_attrUsemap
_attrMultimedia
_attrRel
_attrSrc

continues on next page

Table 1 – continued from previous page

A
Area
Audio
Bdo
Blockquote
BodyCls
Canvas
Command
_Del
Dialog
Abbr
Address
Article
Aside
B
Bdi
Br
Caption
Cite
Code
Datalist
Dfn
Div
Em
Embed
Figcaption

continues on next page

Table 1 – continued from previous page

Figure
Footer
Header
H1
H2
H3
H4
H5
H6
Hr
I
Kdb
Legend
Mark
Noscript
P
Rq
Rt
Ruby
S
Samp
Section
Small
Strong
Sub
Summery

continues on next page

Table 1 – continued from previous page

Sup
U
Var
Wbr
Button
Fieldset
Form
Input
Label
Optgroup
Option
Output
Select
Textarea
HeadCls
Iframe
Img
Ins
Keygen
Link
Ul
Ol
Li
Dl
Dt
Dd

continues on next page

Table 1 – continued from previous page

Map
Menu
Meta
Meter
Nav
Object
Param
Progress
Q
Script
Source
Span
Details
Summary
Style
Tr
Td
Th
Thead
Tbody
ColWrapper
RowWrapper
Table
Time
Track
Video

continues on next page

Table 1 – continued from previous page

Template	
HtmlAst	Abstract syntax tree element used by parseHTML().

Functions

domCreateAttribute(tag[, ns])	Creates a new HTML/SVG/... attribute.
domCreateElement(tag[, ns])	Creates a new HTML/SVG/... tag.
domCreateTextNode([txt])	
domGetElementById(idTag)	
domElementFromPoint(x, y)	
domGetElementsByTagName(tag)	
domParseString(string[, mimetype])	Parses the given string with the optionally given mimetype using JavaScript's DOM parser.
domConvertEncodedText(txt)	Convert HTML-encoded text (containing HTML entities) into its decoded string representation.
Body()	
Head()	
unescape(val[, maxLength])	Unquotes several HTML-quoted characters in a string.
doesEventHitWidgetOrParents(event, widget)	Test if event 'event' hits widget 'widget' (or *any* of its parents).
doesEventHitWidgetOrChildren(event, widget)	Test if event 'event' hits widget 'widget' (or *any* of its children).
textToHtml(node, text)	Generates html nodes from text by splitting text into content and into line breaks html5.Br.
parseInt(s[, ret])	Parses a value as int.
parseFloat(s[, ret])	Parses a value as float.
getKey(event)	Returns the Key Identifier of the given event.
isArrowLeft(event)	
isArrowUp(event)	
isArrowRight(event)	
isArrowDown(event)	
isEscape(event)	
isReturn(event)	
isControl(event)	

continues on next page

Table 2 – continued from previous page

isShift(event)	
isMeta(event)	
registerTag(tagName, widgetClass[, override])	
tag(arg)	Decorator to register a sub-class of html5.Widget either under its class-name or an associated tag-name.
_buildTags([debug])	Generates a dictionary of all to the html5-library known tags and their associated objects and attributes.
parseHTML(→ HtmlAst)	Parses the provided HTML-code according to the tags registered by html5.registerTag() or components that used the html5.tag-decorator.
fromHTML(→ [Widget])	Parses the provided HTML code according to the objects defined in the html5-library.

Attributes

htmlExpressionEvaluator
document
__domParser
_body
_head
__tags
__reVarReplacer

flare.html5.core.**htmlExpressionEvaluator**

flare.html5.core.**document**

flare.html5.core.**domCreateAttribute**(*tag*, *ns=None*)

Creates a new HTML/SVG/... attribute.

Parameters

ns – the namespace. Default: HTML. Possible values: HTML, SVG, XBL, XUL

flare.html5.core.**domCreateElement**(*tag*, *ns=None*)

Creates a new HTML/SVG/... tag.

Parameters

ns – the namespace. Default: HTML. Possible values: HTML, SVG, XBL, XUL

flare.html5.core.**domCreateTextNode**(*txt=''*)

flare.html5.core.**domGetElementById**(*idTag*)

flare.html5.core.**domElementFromPoint**(*x*, *y*)

flare.html5.core.**domGetElementsByTagName**(*tag*)

flare.html5.core.**__domParser**

flare.html5.core.**domParseString**(*string*, *mimetype='text/html'*)

Parses the given string with the optionally given mimetype using JavaScript's DOM parser. :param string: Any XML/HTML string processable by DOMParser. :param mimetype: The mimetype to use.

Returns
Returns the parsed document DOM.

flare.html5.core.**domConvertEncodedText**(*txt*)

Convert HTML-encoded text (containing HTML entities) into its decoded string representation.

The reason for this function is the handling of HTML entities, which is not properly supported by native JavaScript.

We use the browser's DOM parser to do this, according to https://stackoverflow.com/questions/3700326/decode-amp-back-to-in-javascript

Parameters
txt – The encoded text.

Returns
The decoded text.

class flare.html5.core.**TextNode**(*txt=None*, **args*, ***kwargs*)

Bases: object

Represents a piece of text inside the DOM.

This is the *only* object not deriving from "Widget", as it does not support any of its properties.

_setText(*txt*)

_getText()

__str__()

Return str(self).

onAttach()

onDetach()

_setDisabled(*disabled*)

_getDisabled()

children()

class flare.html5.core.**_WidgetClassWrapper**(*targetWidget*)

Bases: list

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

set(*value*)

_updateElem()

append(*p_object*)

Append object to the end of the list.

clear()

Remove all items from list.

remove(*value*)

Remove first occurrence of value.

Raises ValueError if the value is not present.

extend(*iterable*)

Extend list by appending elements from the iterable.

insert(*index*, *p_object*)

Insert object before index.

pop(*index=None*)

Remove and return item at index (default last).

Raises IndexError if list is empty or index is out of range.

class flare.html5.core.**_WidgetDataWrapper**(*targetWidget*)

Bases: dict

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via:
d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs**
in the keyword argument list. For example: dict(one=1, two=2)

__setitem__(*key*, *value*)

Set self[key] to value.

update(*E=None*, ***F*)

D.update([E,]**F) -> None. Update D from dict/iterable E and F. If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

class flare.html5.core.**_WidgetStyleWrapper**(*targetWidget*)

Bases: dict

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via:
d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs**
in the keyword argument list. For example: dict(one=1, two=2)

__setitem__(*key*, *value*)

Set self[key] to value.

update(*E=None*, ***F*)

D.update([E,]**F) -> None. Update D from dict/iterable E and F. If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

class flare.html5.core.**Widget**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: object

_namespace

_tagName

_leafTag = False

style = []

sinkEvent(**args*)

unsinkEvent(**args*)

addEventListener(*event*, *callback*)

Adds an event listener callback to an event on a Widget.

Parameters

- **event** – The event string, e.g. "click" or "mouseover"
- **callback** – The callback function to be called on the given event. This callback function can either accept no parameters, receive the pure Event-object from JavaScript as one parameter, or receive both the pure Event-object from JavaScript and the Widget-instance where the event was triggered on.

removeEventListener(*event*, *callback*)

Removes an event listener callback from a Widget.

The event listener must be previously added by Widget.addEventListener().

Parameters

- **event** – The event string, e.g. "click" or "mouseover"
- **callback** – The callback function to be removed

disable()

Disables an element, in case it is not already disabled.

On disabled elements, events are not triggered anymore.

enable()

Enables an element, in case it is not already enabled.

_getTargetfuncName(*key*, *type*)

__getitem__(*key*)

__setitem__(*key*, *value*)

__str__()

Return str(self).

__iter__()

_getData()

Custom data attributes are intended to store custom data private to the page or application, for which there are no more appropriate attributes or elements.

Parameters
name –

Returns

_getTranslate()

Specifies whether an elements attribute values and contents of its children are to be translated when the page is localized, or whether to leave them unchanged.

Returns
True | False

_setTranslate(*val*)

Specifies whether an elements attribute values and contents of its children are to be translated when the page is localized, or whether to leave them unchanged.

Parameters
val – True | False

_getTitle()

Advisory information associated with the element.

Returns
str

_setTitle(*val*)

Advisory information associated with the element.

Parameters
val – str

_getTabindex()

Specifies whether the element represents an element that is is focusable (that is, an element which is part of the sequence of focusable elements in the document), and the relative order of the element in the sequence of focusable elements in the document.

Returns
number

_setTabindex(*val*)

Specifies whether the element represents an element that is is focusable (that is, an element which is part of the sequence of focusable elements in the document), and the relative order of the element in the sequence of focusable elements in the document.

Parameters
val – number

_getSpellcheck()

Specifies whether the element represents an element whose contents are subject to spell checking and grammar checking.

Returns
True | False

_setSpellcheck(*val*)

Specifies whether the element represents an element whose contents are subject to spell checking and grammar checking.

Parameters
val – True | False

_getLang()

Specifies the primary language for the contents of the element and for any of the elements attributes that contain text.

Returns
language tag e.g. de|en|fr|es|it|ru|

_setLang(*val*)

Specifies the primary language for the contents of the element and for any of the elements attributes that contain text.

Parameters
val – language tag

_getHidden()

Specifies that the element represents an element that is not yet, or is no longer, relevant.

Returns
True | False

_setHidden(*val*)

Specifies that the element represents an element that is not yet, or is no longer, relevant.

Parameters
val – True | False

_getDisabled()

_setDisabled(*disable*)

_getDropzone()

Specifies what types of content can be dropped on the element, and instructs the UA about which actions to take with content when it is dropped on the element.

Returns
"copy" | "move" | "link"

_setDropzone(*val*)

Specifies what types of content can be dropped on the element, and instructs the UA about which actions to take with content when it is dropped on the element.

Parameters
val – "copy" | "move" | "link"

_getDraggable()

Specifies whether the element is draggable.

Returns
True | False | "auto"

_setDraggable(*val*)

Specifies whether the element is draggable.

Parameters
val – True | False | "auto"

_getDir()

Specifies the elements text directionality.

Returns
ltr | rtl | auto

_setDir(*val*)

Specifies the elements text directionality.

Parameters
val – ltr | rtl | auto

_getContextmenu()

The value of the id attribute on the menu with which to associate the element as a context menu.

Returns

_setContextmenu(*val*)

The value of the id attribute on the menu with which to associate the element as a context menu.

Parameters
val –

_getContenteditable()

Specifies whether the contents of the element are editable.

Returns
True | False

_setContenteditable(*val*)

Specifies whether the contents of the element are editable.

Parameters
val – True | False

_getAccesskey()

A key label or list of key labels with which to associate the element; each key label represents a keyboard shortcut which UAs can use to activate the element or give focus to the element.

Parameters
self –

Returns

_setAccesskey(*val*)

A key label or list of key labels with which to associate the element; each key label represents a keyboard shortcut which UAs can use to activate the element or give focus to the element.

Parameters

- **self** –
- **val** –

_getId()

Specifies a unique id for an element.

Parameters
self –

Returns

`_setId`(*val*)

Specifies a unique id for an element.

Parameters

- `self` –
- `val` –

`_getClass`()

The class attribute specifies one or more classnames for an element.

Returns

`_setClass`(*value*)

The class attribute specifies one or more classnames for an element.

Parameters

- `self` –
- `value` –

@raise ValueError:

`_getStyle`()

The style attribute specifies an inline style for an element.

Parameters
`self` –

Returns

`_getRole`()

Specifies a role for an element.

@param self: @return:

`_setRole`(*val*)

Specifies a role for an element.

@param self: @param val:

`hide`()

Hide element, if shown.

Returns

`show`()

Show element, if hidden.

Returns

`isHidden`()

Checks if a widget is hidden.

Returns
True if hidden, False otherwise.

isVisible()

Checks if a widget is visible.

Returns
True if visible, False otherwise.

onBind(*widget*, *name*)

Event function that is called on the widget when it is bound to another widget with a name.

This is only done by the HTML parser, a manual binding by the user is not triggered.

onAttach()

onDetach()

__collectChildren(**args*, ***kwargs*)

Internal function for collecting children from args.

This is used by appendChild(), prependChild(), insertChild() etc.

insertBefore(*insert*, *child*, ***kwargs*)

insertAfter(*insert*, *child*, ***kwargs*)

prependChild(**args*, ***kwargs*)

appendChild(**args*, ***kwargs*)

replaceChild(**args*, ***kwargs*)

removeChild(*child*)

removeAllChildren()

Removes all child widgets of the current widget.

isParentOf(*widget*)

Checks if an object is the parent of widget.

Parameters
widget (Widget) – The widget to check for.

Returns
True, if widget is a child of the object, else False.

isChildOf(*widget*)

Checks if an object is the child of widget.

Parameters
widget (Widget) – The widget to check for.

Returns
True, if object is a child of widget, else False.

hasClass(*className*)

Determine whether the current widget is assigned the given class.

Parameters
className (*str*) – The class name to search for.

addClass(**args*)

Adds a class or a list of classes to the current widget.

If the widget already has the class, it is ignored.

Parameters
args (*list of str | list of list of str*) – A list of class names. This can also be a list.

removeClass(**args*)

Removes a class or a list of classes from the current widget.

Parameters
args (*list of str | list of list of str*) – A list of class names. This can also be a list.

toggleClass(*on*, *off=None*)

Toggles the class on.

If the widget contains a class on, it is toggled by off. off can either be a class name that is substituted, or nothing.

Parameters

- **on** (*str*) – Classname to test for. If on does not exist, but off, off is replaced by on.
- **off** (*str*) – Classname to replace if on existed.

Returns
Returns True, if on was switched, else False.

Return type
bool

onBlur(*event*)

onChange(*event*)

onContextMenu(*event*)

onFocus(*event*)

onFocusIn(*event*)

onFocusOut(*event*)

onFormChange(*event*)

onFormInput(*event*)

onInput(*event*)

onInvalid(*event*)

onReset(*event*)

onSelect(*event*)

onSubmit(*event*)

onKeyDown(*event*)

onKeyPress(*event*)

onKeyUp(*event*)

onClick(*event*, *wdg=None*)

onDblClick(*event*)

onDrag(*event*)

onDragEnd(*event*)

onDragEnter(*event*)

onDragLeave(*event*)

onDragOver(*event*)

onDragStart(*event*)

onDrop(*event*)

onMouseDown(*event*)

onMouseMove(*event*)

onMouseOut(*event*)

onMouseOver(*event*)

onMouseUp(*event*)

onMouseWheel(*event*)

onScroll(*event*)

onTouchStart(*event*)

onTouchEnd(*event*)

onTouchMove(*event*)

onTouchCancel(*event*)

focus()

blur()

parent()

children(*n=None*)

Access children of widget.

If `n` is ommitted, it returns a list of all child-widgets; Else, it returns the N'th child, or None if its out of bounds.

Parameters
`n` (*int*) – Optional offset of child widget to return.

Returns
Returns all children or only the requested one.

Return type
list | *Widget* | None

sortChildren(*key*, *reversed=False*)

Sorts our direct children. They are rearranged on DOM level.

Key must be a function accepting one widget as parameter and must return the key used to sort these widgets.

fromHTML(*html*, *appendTo=None*, *bindTo=None*, *replace=False*, *vars=None*, ***kwargs*)

Parses html and constructs its elements as part of self.

Parameters

- **html** – HTML code.
- **appendTo** – The entity where the HTML code is constructed below. This defaults to self in usual case.
- **bindTo** – The entity where the named objects are bound to. This defaults to self in usual case.
- **replace** – Clear entire content of appendTo before appending.
- **vars** – Deprecated; Same as kwargs.
- ****kwargs** – Additional variables provided as a dict for {{placeholders}} inside the HTML

Returns

class flare.html5.core.**_attrLabel**

Bases: object

_getLabel()

_setLabel(*val*)

class flare.html5.core.**_attrCharset**

Bases: object

_getCharset()

_setCharset(*val*)

class flare.html5.core.**_attrCite**

Bases: object

_getCite()

_setCite(*val*)

class flare.html5.core.**_attrDatetime**

Bases: object

_getDatetime()

_setDatetime(*val*)

class flare.html5.core.**_attrForm**

Bases: object

_getForm()

_setForm(*val*)

class flare.html5.core._attrAlt

Bases: object

_getAlt()

_setAlt(*val*)

class flare.html5.core._attrAutofocus

Bases: object

_getAutofocus()

_setAutofocus(*val*)

class flare.html5.core._attrDisabled

Bases: object

class flare.html5.core._attrChecked

Bases: object

_getChecked()

_setChecked(*val*)

class flare.html5.core._attrIndeterminate

Bases: object

_getIndeterminate()

_setIndeterminate(*val*)

class flare.html5.core._attrName

Bases: object

_getName()

_setName(*val*)

class flare.html5.core._attrValue

Bases: object

_getValue()

_setValue(*val*)

class flare.html5.core._attrAutocomplete

Bases: object

_getAutocomplete()

_setAutocomplete(*val*)

class flare.html5.core._attrRequired

Bases: object

_getRequired()

_setRequired(*val*)

class flare.html5.core._attrMultiple

Bases: object

_getMultiple()

_setMultiple(*val*)

class flare.html5.core._attrSize

Bases: object

_getSize()

_setSize(*val*)

class flare.html5.core._attrFor

Bases: object

_getFor()

_setFor(*val*)

class flare.html5.core._attrInputs

Bases: *_attrRequired*

_getMaxlength()

_setMaxlength(*val*)

_getPlaceholder()

_setPlaceholder(*val*)

_getReadonly()

_setReadonly(*val*)

class flare.html5.core._attrFormhead

Bases: object

_getFormaction()

_setFormaction(*val*)

_getFormenctype()

_setFormenctype(*val*)

_getFormmethod()

_setFormmethod(*val*)

_getFormtarget()

_setFormtarget(*val*)

_getFormnovalidate()

_setFormnovalidate(*val*)

class flare.html5.core._attrHref

Bases: object

_getHref()

Url of a Page.

Parameters
self –

_setHref(*val*)

Url of a Page.

Parameters
val – URL

_getHreflang()

_setHreflang(*val*)

class flare.html5.core._attrTarget

Bases: object

_getTarget()

_setTarget(*val*)

class flare.html5.core._attrType

Bases: object

_getType()

_setType(*val*)

class flare.html5.core._attrMedia

Bases: *_attrType*

_getMedia()

_setMedia(*val*)

class flare.html5.core._attrDimensions

Bases: object

_getWidth()

_setWidth(*val*)

_getHeight()

_setHeight(*val*)

class flare.html5.core._attrUsemap

Bases: object

_getUsemap()

_setUsemap(*val*)

class flare.html5.core._attrMultimedia

Bases: object

_getAutoplay()

_setAutoplay(*val*)

_getPlaysinline()

_setPlaysinline(*val*)

_getControls()

_setControls(*val*)

_getLoop()

_setLoop(*val*)

_getMuted()

_setMuted(*val*)

_getPreload()

_setPreload(*val*)

class flare.html5.core.**_attrRel**

Bases: object

_getRel()

_setRel(*val*)

class flare.html5.core.**_attrSrc**

Bases: object

_getSrc()

_setSrc(*val*)

class flare.html5.core.**A**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrHref*, *_attrTarget*, *_attrMedia*, *_attrRel*, *_attrName*

_tagName = 'a'

_getDownload()

The download attribute specifies the path to a download.

Returns
filename

_setDownload(*val*)

The download attribute specifies the path to a download.

Parameters
val – filename

class flare.html5.core.**Area**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *A*, *_attrAlt*

_tagName = 'area'

_leafTag = True

_getCoords()

_setCoords(*val*)

_getShape()

_setShape(*val*)

class flare.html5.core.**Audio**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrSrc*, *_attrMultimedia*

_tagName = 'audio'

class flare.html5.core.**Bdo**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'bdo'

class flare.html5.core.**Blockquote**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'blockquote'

_getBlockquote()

_setBlockquote(*val*)

class flare.html5.core.**BodyCls**(**args*, ***kwargs*)

Bases: *Widget*

flare.html5.core._body

flare.html5.core.**Body**()

class flare.html5.core.**Canvas**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDimensions*

_tagName = 'canvas'

class flare.html5.core.**Command**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrLabel*, *_attrType*, *_attrDisabled*, *_attrChecked*

_tagName = 'command'

_getIcon()

_setIcon(*val*)

_getRadiogroup()

_setRadiogroup(*val*)

class flare.html5.core.**_Del**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrCite*, *_attrDatetime*

_tagName = '_del'

class flare.html5.core.**Dialog**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'dialog'

_getOpen()

_setOpen(*val*)

class flare.html5.core.**Abbr**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'abbr'

class flare.html5.core.**Address**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'address'

class flare.html5.core.**Article**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'article'

class flare.html5.core.**Aside**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'aside'

class flare.html5.core.**B**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'b'

class flare.html5.core.**Bdi**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'bdi'

class flare.html5.core.**Br**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'br'

_leafTag = True

class flare.html5.core.**Caption**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'caption'

class flare.html5.core.**Cite**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'cite'

class flare.html5.core.**Code**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'code'

class flare.html5.core.Datalist(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'datalist'

class flare.html5.core.Dfn(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'dfn'

class flare.html5.core.Div(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'div'

class flare.html5.core.Em(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'em'

class flare.html5.core.Embed(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrSrc, _attrType, _attrDimensions

_tagName = 'embed'

_leafTag = True

class flare.html5.core.Figcaption(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'figcaption'

class flare.html5.core.Figure(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'figure'

class flare.html5.core.Footer(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'footer'

class flare.html5.core.Header(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'header'

class flare.html5.core.H1(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'h1'

class flare.html5.core.H2(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'h2'

class flare.html5.core.**H3**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'h3'

class flare.html5.core.**H4**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'h4'

class flare.html5.core.**H5**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'h5'

class flare.html5.core.**H6**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'h6'

class flare.html5.core.**Hr**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'hr'

_leafTag = True

class flare.html5.core.**I**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'i'

class flare.html5.core.**Kdb**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'kdb'

class flare.html5.core.**Legend**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'legend'

class flare.html5.core.**Mark**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'mark'

class flare.html5.core.**Noscript**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'noscript'

class flare.html5.core.**P**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'p'

class flare.html5.core.**Rq**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'rq'

class flare.html5.core.Rt(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'rt'

class flare.html5.core.Ruby(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'ruby'

class flare.html5.core.S(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 's'

class flare.html5.core.Samp(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'samp'

class flare.html5.core.Section(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'section'

class flare.html5.core.Small(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'small'

class flare.html5.core.Strong(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'strong'

class flare.html5.core.Sub(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'sub'

class flare.html5.core.Summery(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'summery'

class flare.html5.core.Sup(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'sup'

class flare.html5.core.U(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'u'

class flare.html5.core.Var(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'var'

class flare.html5.core.Wbr(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'wbr'

class flare.html5.core.Button(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrDisabled, _attrType, _attrForm, _attrAutofocus, _attrName, _attrValue, _attrFormhead

_tagName = 'button'

class flare.html5.core.Fieldset(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrDisabled, _attrForm, _attrName

_tagName = 'fieldset'

class flare.html5.core.Form(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrDisabled, _attrName, _attrTarget, _attrAutocomplete

_tagName = 'form'

_getNovalidate()

_setNovalidate(val)

_getAction()

_setAction(val)

_getMethod()

_setMethod(val)

_getEnctype()

_setEnctype(val)

_getAccept_attrCharset()

_setAccept_attrCharset(val)

class flare.html5.core.Input(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrDisabled, _attrType, _attrForm, _attrAlt, _attrAutofocus, _attrChecked, _attrIndeterminate, _attrName, _attrDimensions, _attrValue, _attrFormhead, _attrAutocomplete, _attrInputs, _attrMultiple, _attrSize, _attrSrc

_tagName = 'input'

_leafTag = True

_getAccept()

_setAccept(val)

_getList()

_setList(val)

_getMax()

_setMax(*val*)

_getMin()

_setMin(*val*)

_getPattern()

_setPattern(*val*)

_getStep()

_setStep(*val*)

class flare.html5.core.**Label**(**args*, *forElem=None*, ***kwargs*)

Bases: *Widget*, *_attrForm*, *_attrFor*

_tagName = 'label'

autoIdCounter = 0

class flare.html5.core.**Optgroup**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrLabel*

_tagName = 'optgroup'

class flare.html5.core.**Option**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrLabel*, *_attrValue*

_tagName = 'option'

_getSelected()

_setSelected(*val*)

class flare.html5.core.**Output**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrForm*, *_attrName*, *_attrFor*

_tagName = 'output'

class flare.html5.core.**Select**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrForm*, *_attrAutofocus*, *_attrName*, *_attrRequired*, *_attrMultiple*, *_attrSize*

_tagName = 'select'

_getSelectedIndex()

_getOptions()

class flare.html5.core.**Textarea**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrForm*, *_attrAutofocus*, *_attrName*, *_attrInputs*, *_attrValue*

_tagName = 'textarea'

_getCols()

_setCols(*val*)

_getRows()

_setRows(*val*)

_getWrap()

_setWrap(*val*)

class flare.html5.core.HeadCls(**args*, ***kwargs*)

Bases: *Widget*

flare.html5.core._head

flare.html5.core.Head()

class flare.html5.core.Iframe(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrSrc*, *_attrName*, *_attrDimensions*

_tagName = 'iframe'

_getSandbox()

_setSandbox(*val*)

_getSrcdoc()

_setSrcdoc(*val*)

_getSeamless()

_setSeamless(*val*)

class flare.html5.core.Img(*src=None*, **args*, ***kwargs*)

Bases: *Widget*, *_attrSrc*, *_attrDimensions*, *_attrUsemap*, *_attrAlt*

_tagName = 'img'

_leafTag = True

_getCrossorigin()

_setCrossorigin(*val*)

_getIsmap()

_setIsmap(*val*)

class flare.html5.core.Ins(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrCite*, *_attrDatetime*

_tagName = 'ins'

class flare.html5.core.Keygen(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Form*, *_attrAutofocus*, *_attrDisabled*

_tagName = 'keygen'

_getChallenge()

_setChallenge(*val*)

_getKeytype()

_setKeytype(*val*)

class flare.html5.core.**Link**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrHref*, *_attrMedia*, *_attrRel*

_tagName = 'link'

_leafTag = True

_getSizes()

_setSizes(*val*)

class flare.html5.core.**Ul**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'ul'

class flare.html5.core.**Ol**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'ol'

class flare.html5.core.**Li**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'li'

class flare.html5.core.**Dl**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'dl'

class flare.html5.core.**Dt**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'dt'

class flare.html5.core.**Dd**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'dd'

class flare.html5.core.**Map**(**args*, *forElem=None*, ***kwargs*)

Bases: *Label*, *_attrType*

_tagName = 'map'

class flare.html5.core.**Menu**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'menu'

class flare.html5.core.**Meta**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrName*, *_attrCharset*

_tagName = 'meta'

_leafTag = True

_getContent()

_setContent(*val*)

class flare.html5.core.**Meter**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Form*, *_attrValue*

_tagName = 'meter'

_getHigh()

_setHigh(*val*)

_getLow()

_setLow(*val*)

_getMax()

_setMax(*val*)

_getMin()

_setMin(*val*)

_getOptimum()

_setOptimum(*val*)

class flare.html5.core.**Nav**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'nav'

class flare.html5.core.**Object**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Form*, *_attrType*, *_attrName*, *_attrDimensions*, *_attrUsemap*

_tagName = 'object'

class flare.html5.core.**Param**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrName*, *_attrValue*

_tagName = 'param'

_leafTag = True

class flare.html5.core.**Progress**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrValue*

_tagName = 'progress'

_getMax()

_setMax(*val*)

class flare.html5.core.**Q**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrCite*

_tagName = 'q'

class flare.html5.core.Script(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrSrc, _attrCharset

_tagName = 'script'

_getAsync()

_setAsync(val)

_getDefer()

_setDefer(val)

class flare.html5.core.Source(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrMedia, _attrSrc

_tagName = 'source'

_leafTag = True

class flare.html5.core.Span(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'span'

class flare.html5.core.Details(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'details'

_getOpen()

_setOpen(val)

class flare.html5.core.Summary(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'summary'

class flare.html5.core.Style(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrMedia

_tagName = 'style'

_getScoped()

_setScoped(val)

class flare.html5.core.Tr(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'tr'

_getRowspan()

_setRowspan(span)

class flare.html5.core.**Td**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'td'

_getColspan()

_setColspan(*span*)

_getRowspan()

_setRowspan(*span*)

class flare.html5.core.**Th**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Td*

_tagName = 'th'

class flare.html5.core.**Thead**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'thead'

class flare.html5.core.**Tbody**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'tbody'

class flare.html5.core.**ColWrapper**(*parentElem*, **args*, ***kwargs*)

Bases: object

__getitem__(*item*)

__setitem__(*key*, *value*)

class flare.html5.core.**RowWrapper**(*parentElem*, **args*, ***kwargs*)

Bases: object

__getitem__(*item*)

class flare.html5.core.**Table**(**args*, ***kwargs*)

Bases: *Widget*

_tagName = 'table'

prepareRow(*row*)

prepareCol(*row*, *col*)

prepareGrid(*rows*, *cols*)

clear()

_getCell()

getRowCount()

class flare.html5.core.**Time**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDatetime*

_tagName = 'time'

class flare.html5.core.Track(*args, forElem=None, **kwargs)

Bases: Label, _attrSrc

_tagName = 'track'

_leafTag = True

_getKind()

_setKind(val)

_getSrclang()

_setSrclang(val)

_getDefault()

_setDefault(val)

class flare.html5.core.Video(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrSrc, _attrDimensions, _attrMultimedia

_tagName = 'video'

_getPoster()

_setPoster(val)

class flare.html5.core.Template(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'template'

flare.html5.core.unescape(val, maxLength=0)

Unquotes several HTML-quoted characters in a string.

Parameters

- **val** (*str*) – The value to be unescaped.
- **maxLength** (*int*) – Cut-off after maxLength characters. A value of 0 means "unlimited". (default)

Returns

The unquoted string.

Return type

str

flare.html5.core.doesEventHitWidgetOrParents(event, widget)

Test if event 'event' hits widget 'widget' (or *any* of its parents).

flare.html5.core.doesEventHitWidgetOrChildren(event, widget)

Test if event 'event' hits widget 'widget' (or *any* of its children).

flare.html5.core.textToHtml(node, text)

Generates html nodes from text by splitting text into content and into line breaks html5.Br.

Parameters

- **node** – The node where the nodes are appended to.
- **text** – The text to be inserted.

flare.html5.core.**parseInt**(*s*, *ret=0*)

Parses a value as int.

flare.html5.core.**parseFloat**(*s*, *ret=0.0*)

Parses a value as float.

flare.html5.core.**getKey**(*event*)

Returns the Key Identifier of the given event.

Available Codes: https://www.w3.org/TR/2006/WD-DOM-Level-3-Events-20060413/keyset.html#KeySet-Set

flare.html5.core.**isArrowLeft**(*event*)

flare.html5.core.**isArrowUp**(*event*)

flare.html5.core.**isArrowRight**(*event*)

flare.html5.core.**isArrowDown**(*event*)

flare.html5.core.**isEscape**(*event*)

flare.html5.core.**isReturn**(*event*)

flare.html5.core.**isControl**(*event*)

flare.html5.core.**isShift**(*event*)

flare.html5.core.**isMeta**(*event*)

flare.html5.core.**__tags**

flare.html5.core.**__reVarReplacer**

flare.html5.core.**registerTag**(*tagName*, *widgetClass*, *override=True*)

flare.html5.core.**tag**(*arg*)

Decorator to register a sub-class of html5.Widget either under its class-name or an associated tag-name.

Examples

```python # register class Foo as <foo>-Tag @html5.tag class Foo(html5.Div):
    pass
# register class Bar as <baz>-Tag @html5.tag("baz") class Bar(html5.Div):
    pass
```

flare.html5.core.**_buildTags**(*debug=False*)

Generates a dictionary of all to the html5-library known tags and their associated objects and attributes.

class flare.html5.core.**HtmlAst**

Bases: list

Abstract syntax tree element used by parseHTML().

flare.html5.core.**parseHTML**(*html: str*, *debug: bool = False*) → *HtmlAst*

Parses the provided HTML-code according to the tags registered by html5.registerTag() or components that used the html5.tag-decorator.

flare.html5.core.**fromHTML**(*html: [str,* HtmlAst*]*, *appendTo:* Widget *= None*, *bindTo:* Widget *= None*, *debug: bool = False*, ***kwargs*) → [*Widget*]

Parses the provided HTML code according to the objects defined in the html5-library.

html can also be pre-compiled by *parseHTML()* so that it executes faster.

Constructs all objects as DOM nodes. The first level is chained into appendTo. If no appendTo is provided, appendTo will be set to html5.Body().

If bindTo is provided, objects are bound to this widget.

```python from vi import html5

div = html5.Div() html5.parse.fromHTML('''

**<div>Yeah!**
<a href="hello world" [name]="myLink" class="trullman bernd" disabled> hah ala malla" bababtschga" <img src="/static/images/icon_home.svg" style="background-color: red;"/>st <em>ah</em>ralla <i>malla tralla</i> da </a>lala

</div>''', div)

div.myLink.appendChild("appended!") ```

## flare.html5.svg

SVG abstraction layer integrations for HTML5.
```

Module Contents

Classes

_attrSvgViewBox
_attrSvgDimensions
_attrSvgPoints
_attrSvgTransform
_attrSvgXlink
_attrSvgStyles
SvgWidget
Svg
SvgCircle
SvgEllipse
SvgG
SvgImage
SvgLine
SvgPath
SvgPolygon
SvgPolyline
SvgRect
SvgText

class flare.html5.svg.**_attrSvgViewBox**

Bases: object

_getViewbox()

_setViewbox(*val*)

_getPreserveaspectratio()

_setPreserveaspectratio(*val*)

class flare.html5.svg._attrSvgDimensions

Bases: object

_getWidth()

_setWidth(*val*)

_getHeight()

_setHeight(*val*)

_getX()

_setX(*val*)

_getY()

_setY(*val*)

_getR()

_setR(*val*)

_getRx()

_setRx(*val*)

_getRy()

_setRy(*val*)

_getCx()

_setCx(*val*)

_getCy()

_setCy(*val*)

class flare.html5.svg._attrSvgPoints

Bases: object

_getPoints()

_setPoints(*val*)

_getX1()

_setX1(*val*)

_getY1()

_setY1(*val*)

_getX2()

_setX2(*val*)

_getY2()

_setY2(*val*)

class flare.html5.svg._attrSvgTransform
Bases: object

_getTransform()

_setTransform(*val*)

class flare.html5.svg._attrSvgXlink
Bases: object

_getXlinkhref()

_setXlinkhref(*val*)

class flare.html5.svg._attrSvgStyles
Bases: object

_getFill()

_setFill(*val*)

_getStroke()

_setStroke(*val*)

class flare.html5.svg.SvgWidget(**args*, *appendTo=None*, *style=None*, ***kwargs*)
Bases: *flare.html5.core.Widget*

_namespace = 'SVG'

class flare.html5.svg.Svg(**args*, *appendTo=None*, *style=None*, ***kwargs*)
Bases: *SvgWidget*, *_attrSvgViewBox*, *_attrSvgDimensions*, *_attrSvgTransform*

_tagName = 'svg'

_getVersion()

_setVersion(*val*)

_getXmlns()

_setXmlns(*val*)

class flare.html5.svg.SvgCircle(**args*, *appendTo=None*, *style=None*, ***kwargs*)
Bases: *SvgWidget*, *_attrSvgTransform*, *_attrSvgDimensions*

_tagName = 'circle'

class flare.html5.svg.SvgEllipse(**args*, *appendTo=None*, *style=None*, ***kwargs*)
Bases: *SvgWidget*, *_attrSvgTransform*, *_attrSvgDimensions*

_tagName = 'ellipse'

class flare.html5.svg.SvgG(**args*, *appendTo=None*, *style=None*, ***kwargs*)
Bases: *SvgWidget*, *_attrSvgTransform*, *_attrSvgStyles*

_tagName = 'g'

_getSvgTransform()

_setSvgTransform(*val*)

class flare.html5.svg.**SvgImage**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *SvgWidget*, *_attrSvgViewBox*, *_attrSvgDimensions*, *_attrSvgTransform*, *_attrSvgXlink*

_tagName = 'image'

class flare.html5.svg.**SvgLine**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *SvgWidget*, *_attrSvgTransform*, *_attrSvgPoints*

_tagName = 'line'

class flare.html5.svg.**SvgPath**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *SvgWidget*, *_attrSvgTransform*

_tagName = 'path'

_getD()

_setD(*val*)

_getPathLength()

_setPathLength(*val*)

class flare.html5.svg.**SvgPolygon**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *SvgWidget*, *_attrSvgTransform*, *_attrSvgPoints*

_tagName = 'polygon'

class flare.html5.svg.**SvgPolyline**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *SvgWidget*, *_attrSvgTransform*, *_attrSvgPoints*

_tagName = 'polyline'

class flare.html5.svg.**SvgRect**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *SvgWidget*, *_attrSvgDimensions*, *_attrSvgTransform*, *_attrSvgStyles*

_tagName = 'rect'

class flare.html5.svg.**SvgText**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *SvgWidget*, *_attrSvgDimensions*, *_attrSvgTransform*, *_attrSvgStyles*

_tagName = 'text'

Package Contents

Classes

TextNode	Represents a piece of text inside the DOM.
_WidgetClassWrapper	Built-in mutable sequence.
_WidgetDataWrapper	dict() -> new empty dictionary
_WidgetStyleWrapper	dict() -> new empty dictionary
Widget	

continues on next page

Table 3 – continued from previous page

_attrLabel
_attrCharset
_attrCite
_attrDatetime
_attrForm
_attrAlt
_attrAutofocus
_attrDisabled
_attrChecked
_attrIndeterminate
_attrName
_attrValue
_attrAutocomplete
_attrRequired
_attrMultiple
_attrSize
_attrFor
_attrInputs
_attrFormhead
_attrHref
_attrTarget
_attrType
_attrMedia
_attrDimensions
_attrUsemap
_attrMultimedia

continues on next page

Table 3 – continued from previous page

_attrRel
_attrSrc
A
Area
Audio
Bdo
Blockquote
BodyCls
Canvas
Command
_Del
Dialog
Abbr
Address
Article
Aside
B
Bdi
Br
Caption
Cite
Code
Datalist
Dfn
Div
Em

continues on next page

Table 3 – continued from previous page

Embed
Figcaption
Figure
Footer
Header
H1
H2
H3
H4
H5
H6
Hr
I
Kdb
Legend
Mark
Noscript
P
Rq
Rt
Ruby
S
Samp
Section
Small
Strong

continues on next page

Table 3 – continued from previous page

Sub
Summery
Sup
U
Var
Wbr
Button
Fieldset
Form
Input
Label
Optgroup
Option
Output
Select
Textarea
HeadCls
Iframe
Img
Ins
Keygen
Link
Ul
Ol
Li
Dl

continues on next page

Table 3 – continued from previous page

Dt
Dd
Map
Menu
Meta
Meter
Nav
Object
Param
Progress
Q
Script
Source
Span
Details
Summary
Style
Tr
Td
Th
Thead
Tbody
ColWrapper
RowWrapper
Table
Time

continues on next page

Table 3 – continued from previous page

Track	
Video	
Template	
HtmlAst	Abstract syntax tree element used by parseHTML().

Functions

domCreateAttribute(tag[, ns])	Creates a new HTML/SVG/... attribute.
domCreateElement(tag[, ns])	Creates a new HTML/SVG/... tag.
domCreateTextNode([txt])	
domGetElementById(idTag)	
domElementFromPoint(x, y)	
domGetElementsByTagName(tag)	
domParseString(string[, mimetype])	Parses the given string with the optionally given mimetype using JavaScript's DOM parser.
domConvertEncodedText(txt)	Convert HTML-encoded text (containing HTML entities) into its decoded string representation.
Body()	
Head()	
unescape(val[, maxLength])	Unquotes several HTML-quoted characters in a string.
doesEventHitWidgetOrParents(event, widget)	Test if event 'event' hits widget 'widget' (or *any* of its parents).
doesEventHitWidgetOrChildren(event, widget)	Test if event 'event' hits widget 'widget' (or *any* of its children).
textToHtml(node, text)	Generates html nodes from text by splitting text into content and into line breaks html5.Br.
parseInt(s[, ret])	Parses a value as int.
parseFloat(s[, ret])	Parses a value as float.
getKey(event)	Returns the Key Identifier of the given event.
isArrowLeft(event)	
isArrowUp(event)	
isArrowRight(event)	
isArrowDown(event)	
isEscape(event)	

continues on next page

Table 4 – continued from previous page

isReturn(event)	
isControl(event)	
isShift(event)	
isMeta(event)	
registerTag(tagName, widgetClass[, override])	
tag(arg)	Decorator to register a sub-class of html5.Widget either under its class-name or an associated tag-name.
_buildTags([debug])	Generates a dictionary of all to the html5-library known tags and their associated objects and attributes.
parseHTML(→ HtmlAst)	Parses the provided HTML-code according to the tags registered by html5.registerTag() or components that used the html5.tag-decorator.
fromHTML(→ [Widget])	Parses the provided HTML code according to the objects defined in the html5-library.

Attributes

htmlExpressionEvaluator
document
__domParser
_body
_head
__tags
__reVarReplacer

flare.html5.**htmlExpressionEvaluator**

flare.html5.**document**

flare.html5.**domCreateAttribute**(*tag*, *ns=None*)

Creates a new HTML/SVG/… attribute.

Parameters

ns – the namespace. Default: HTML. Possible values: HTML, SVG, XBL, XUL

flare.html5.**domCreateElement**(*tag*, *ns=None*)

Creates a new HTML/SVG/… tag.

Parameters

ns – the namespace. Default: HTML. Possible values: HTML, SVG, XBL, XUL

flare.html5.**domCreateTextNode**(*txt=''*)

flare.html5.**domGetElementById**(*idTag*)

flare.html5.**domElementFromPoint**(*x*, *y*)

flare.html5.**domGetElementsByTagName**(*tag*)

flare.html5.**__domParser**

flare.html5.**domParseString**(*string*, *mimetype='text/html'*)

Parses the given string with the optionally given mimetype using JavaScript's DOM parser. :param string: Any XML/HTML string processable by DOMParser. :param mimetype: The mimetype to use.

Returns
Returns the parsed document DOM.

flare.html5.**domConvertEncodedText**(*txt*)

Convert HTML-encoded text (containing HTML entities) into its decoded string representation.

The reason for this function is the handling of HTML entities, which is not properly supported by native JavaScript.

We use the browser's DOM parser to do this, according to https://stackoverflow.com/questions/3700326/decode-amp-back-to-in-javascript

Parameters
txt – The encoded text.

Returns
The decoded text.

class flare.html5.**TextNode**(*txt=None*, **args*, ***kwargs*)

Bases: object

Represents a piece of text inside the DOM.

This is the *only* object not deriving from "Widget", as it does not support any of its properties.

_setText(*txt*)

_getText()

__str__()
Return str(self).

onAttach()

onDetach()

_setDisabled(*disabled*)

_getDisabled()

children()

class flare.html5.**_WidgetClassWrapper**(*targetWidget*)

Bases: list

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

set(*value*)

_updateElem()

append(*p_object*)

Append object to the end of the list.

clear()

Remove all items from list.

remove(*value*)

Remove first occurrence of value.

Raises ValueError if the value is not present.

extend(*iterable*)

Extend list by appending elements from the iterable.

insert(*index*, *p_object*)

Insert object before index.

pop(*index=None*)

Remove and return item at index (default last).

Raises IndexError if list is empty or index is out of range.

class flare.html5.**_WidgetDataWrapper**(*targetWidget*)

Bases: dict

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via:
d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs**
in the keyword argument list. For example: dict(one=1, two=2)

__setitem__(*key*, *value*)

Set self[key] to value.

update(*E=None*, ***F*)

D.update([E,]**F) -> None. Update D from dict/iterable E and F. If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

class flare.html5.**_WidgetStyleWrapper**(*targetWidget*)

Bases: dict

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via:
d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs**
in the keyword argument list. For example: dict(one=1, two=2)

__setitem__(*key*, *value*)

Set self[key] to value.

update(*E=None*, ***F*)

D.update([E,]**F) -> None. Update D from dict/iterable E and F. If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

class flare.html5.**Widget**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: object

_namespace

_tagName

_leafTag = False

style = []

sinkEvent(**args*)

unsinkEvent(**args*)

addEventListener(*event*, *callback*)

Adds an event listener callback to an event on a Widget.

Parameters

- **event** – The event string, e.g. "click" or "mouseover"
- **callback** – The callback function to be called on the given event. This callback function can either accept no parameters, receive the pure Event-object from JavaScript as one parameter, or receive both the pure Event-object from JavaScript and the Widget-instance where the event was triggered on.

removeEventListener(*event*, *callback*)

Removes an event listener callback from a Widget.

The event listener must be previously added by Widget.addEventListener().

Parameters

- **event** – The event string, e.g. "click" or "mouseover"
- **callback** – The callback function to be removed

disable()

Disables an element, in case it is not already disabled.

On disabled elements, events are not triggered anymore.

enable()

Enables an element, in case it is not already enabled.

_getTargetfuncName(*key*, *type*)

__getitem__(*key*)

__setitem__(*key*, *value*)

__str__()

Return str(self).

__iter__()

_getData()

Custom data attributes are intended to store custom data private to the page or application, for which there are no more appropriate attributes or elements.

Parameters
name –

Returns

_getTranslate()

Specifies whether an elements attribute values and contents of its children are to be translated when the page is localized, or whether to leave them unchanged.

Returns
True | False

_setTranslate(*val*)

Specifies whether an elements attribute values and contents of its children are to be translated when the page is localized, or whether to leave them unchanged.

Parameters
val – True | False

_getTitle()

Advisory information associated with the element.

Returns
str

_setTitle(*val*)

Advisory information associated with the element.

Parameters
val – str

_getTabindex()

Specifies whether the element represents an element that is is focusable (that is, an element which is part of the sequence of focusable elements in the document), and the relative order of the element in the sequence of focusable elements in the document.

Returns
number

_setTabindex(*val*)

Specifies whether the element represents an element that is is focusable (that is, an element which is part of the sequence of focusable elements in the document), and the relative order of the element in the sequence of focusable elements in the document.

Parameters
val – number

`_getSpellcheck()`

Specifies whether the element represents an element whose contents are subject to spell checking and grammar checking.

Returns
True | False

`_setSpellcheck(val)`

Specifies whether the element represents an element whose contents are subject to spell checking and grammar checking.

Parameters
val – True | False

`_getLang()`

Specifies the primary language for the contents of the element and for any of the elements attributes that contain text.

Returns
language tag e.g. de|en|fr|es|it|ru|

`_setLang(val)`

Specifies the primary language for the contents of the element and for any of the elements attributes that contain text.

Parameters
val – language tag

`_getHidden()`

Specifies that the element represents an element that is not yet, or is no longer, relevant.

Returns
True | False

`_setHidden(val)`

Specifies that the element represents an element that is not yet, or is no longer, relevant.

Parameters
val – True | False

`_getDisabled()`

`_setDisabled(disable)`

`_getDropzone()`

Specifies what types of content can be dropped on the element, and instructs the UA about which actions to take with content when it is dropped on the element.

Returns
“copy” | “move” | “link”

`_setDropzone(val)`

Specifies what types of content can be dropped on the element, and instructs the UA about which actions to take with content when it is dropped on the element.

Parameters
val – “copy” | “move” | “link”

`_getDraggable()`

Specifies whether the element is draggable.

Returns
True | False | "auto"

`_setDraggable(val)`

Specifies whether the element is draggable.

Parameters
val – True | False | "auto"

`_getDir()`

Specifies the elements text directionality.

Returns
ltr | rtl | auto

`_setDir(val)`

Specifies the elements text directionality.

Parameters
val – ltr | rtl | auto

`_getContextmenu()`

The value of the id attribute on the menu with which to associate the element as a context menu.

Returns

`_setContextmenu(val)`

The value of the id attribute on the menu with which to associate the element as a context menu.

Parameters
val –

`_getContenteditable()`

Specifies whether the contents of the element are editable.

Returns
True | False

`_setContenteditable(val)`

Specifies whether the contents of the element are editable.

Parameters
val – True | False

`_getAccesskey()`

A key label or list of key labels with which to associate the element; each key label represents a keyboard shortcut which UAs can use to activate the element or give focus to the element.

Parameters
self –

Returns

`_setAccesskey(val)`

A key label or list of key labels with which to associate the element; each key label represents a keyboard shortcut which UAs can use to activate the element or give focus to the element.

Parameters

- **self** –
- **val** –

_getId()

Specifies a unique id for an element.

Parameters
self –

Returns

_setId(*val*)

Specifies a unique id for an element.

Parameters

- **self** –
- **val** –

_getClass()

The class attribute specifies one or more classnames for an element.

Returns

_setClass(*value*)

The class attribute specifies one or more classnames for an element.

Parameters

- **self** –
- **value** –

@raise ValueError:

_getStyle()

The style attribute specifies an inline style for an element.

Parameters
self –

Returns

_getRole()

Specifies a role for an element.

@param self: @return:

_setRole(*val*)

Specifies a role for an element.

@param self: @param val:

hide()

Hide element, if shown.

Returns

show()

Show element, if hidden.

Returns

isHidden()

Checks if a widget is hidden.

Returns
True if hidden, False otherwise.

isVisible()

Checks if a widget is visible.

Returns
True if visible, False otherwise.

onBind(*widget*, *name*)

Event function that is called on the widget when it is bound to another widget with a name.

This is only done by the HTML parser, a manual binding by the user is not triggered.

onAttach()

onDetach()

__collectChildren(**args*, ***kwargs*)

Internal function for collecting children from args.

This is used by appendChild(), prependChild(), insertChild() etc.

insertBefore(*insert*, *child*, ***kwargs*)

insertAfter(*insert*, *child*, ***kwargs*)

prependChild(**args*, ***kwargs*)

appendChild(**args*, ***kwargs*)

replaceChild(**args*, ***kwargs*)

removeChild(*child*)

removeAllChildren()

Removes all child widgets of the current widget.

isParentOf(*widget*)

Checks if an object is the parent of widget.

Parameters
widget (Widget) – The widget to check for.

Returns
True, if widget is a child of the object, else False.

isChildOf(*widget*)

Checks if an object is the child of widget.

Parameters
widget (Widget) – The widget to check for.

Returns
True, if object is a child of widget, else False.

hasClass(*className*)

Determine whether the current widget is assigned the given class.

Parameters
className (*str*) – The class name to search for.

addClass(**args*)

Adds a class or a list of classes to the current widget.

If the widget already has the class, it is ignored.

Parameters
args (*list of str | list of list of str*) – A list of class names. This can also be a list.

removeClass(**args*)

Removes a class or a list of classes from the current widget.

Parameters
args (*list of str | list of list of str*) – A list of class names. This can also be a list.

toggleClass(*on*, *off=None*)

Toggles the class `on`.

If the widget contains a class `on`, it is toggled by `off`. `off` can either be a class name that is substituted, or nothing.

Parameters

- **on** (*str*) – Classname to test for. If `on` does not exist, but `off`, `off` is replaced by `on`.
- **off** (*str*) – Classname to replace if `on` existed.

Returns
Returns True, if `on` was switched, else False.

Return type
bool

onBlur(*event*)

onChange(*event*)

onContextMenu(*event*)

onFocus(*event*)

onFocusIn(*event*)

onFocusOut(*event*)

onFormChange(*event*)

onFormInput(*event*)

onInput(*event*)

onInvalid(*event*)

onReset(*event*)

onSelect(*event*)

onSubmit(*event*)

onKeyDown(*event*)

onKeyPress(*event*)

onKeyUp(*event*)

onClick(*event*, *wdg=None*)

onDblClick(*event*)

onDrag(*event*)

onDragEnd(*event*)

onDragEnter(*event*)

onDragLeave(*event*)

onDragOver(*event*)

onDragStart(*event*)

onDrop(*event*)

onMouseDown(*event*)

onMouseMove(*event*)

onMouseOut(*event*)

onMouseOver(*event*)

onMouseUp(*event*)

onMouseWheel(*event*)

onScroll(*event*)

onTouchStart(*event*)

onTouchEnd(*event*)

onTouchMove(*event*)

onTouchCancel(*event*)

focus()

blur()

parent()

children(*n=None*)

Access children of widget.

If n is ommitted, it returns a list of all child-widgets; Else, it returns the N'th child, or None if its out of bounds.

Parameters
n (*int*) – Optional offset of child widget to return.

Returns
Returns all children or only the requested one.

Return type
list | *Widget* | None

sortChildren(*key*, *reversed=False*)

Sorts our direct children. They are rearranged on DOM level.

Key must be a function accepting one widget as parameter and must return the key used to sort these widgets.

fromHTML(*html*, *appendTo=None*, *bindTo=None*, *replace=False*, *vars=None*, ***kwargs*)

Parses html and constructs its elements as part of self.

Parameters

- **html** – HTML code.
- **appendTo** – The entity where the HTML code is constructed below. This defaults to self in usual case.
- **bindTo** – The entity where the named objects are bound to. This defaults to self in usual case.
- **replace** – Clear entire content of appendTo before appending.
- **vars** – Deprecated; Same as kwargs.
- ****kwargs** – Additional variables provided as a dict for {{placeholders}} inside the HTML

Returns

class flare.html5.**_attrLabel**

Bases: object

_getLabel()

_setLabel(*val*)

class flare.html5.**_attrCharset**

Bases: object

_getCharset()

_setCharset(*val*)

class flare.html5.**_attrCite**

Bases: object

_getCite()

_setCite(*val*)

class flare.html5._attrDatetime
Bases: object

_getDatetime()

_setDatetime(*val*)

class flare.html5._attrForm
Bases: object

_getForm()

_setForm(*val*)

class flare.html5._attrAlt
Bases: object

_getAlt()

_setAlt(*val*)

class flare.html5._attrAutofocus
Bases: object

_getAutofocus()

_setAutofocus(*val*)

class flare.html5._attrDisabled
Bases: object

class flare.html5._attrChecked
Bases: object

_getChecked()

_setChecked(*val*)

class flare.html5._attrIndeterminate
Bases: object

_getIndeterminate()

_setIndeterminate(*val*)

class flare.html5._attrName
Bases: object

_getName()

_setName(*val*)

class flare.html5._attrValue
Bases: object

_getValue()

_setValue(*val*)

class flare.html5._attrAutocomplete
Bases: object

_getAutocomplete()

_setAutocomplete(*val*)

class flare.html5._attrRequired
Bases: object

_getRequired()

_setRequired(*val*)

class flare.html5._attrMultiple
Bases: object

_getMultiple()

_setMultiple(*val*)

class flare.html5._attrSize
Bases: object

_getSize()

_setSize(*val*)

class flare.html5._attrFor
Bases: object

_getFor()

_setFor(*val*)

class flare.html5._attrInputs
Bases: *_attrRequired*

_getMaxlength()

_setMaxlength(*val*)

_getPlaceholder()

_setPlaceholder(*val*)

_getReadonly()

_setReadonly(*val*)

class flare.html5._attrFormhead
Bases: object

_getFormaction()

_setFormaction(*val*)

_getFormenctype()

_setFormenctype(*val*)

_getFormmethod()

_setFormmethod(*val*)

_getFormtarget()

_setFormtarget(*val*)

_getFormnovalidate()

_setFormnovalidate(*val*)

class flare.html5._attrHref

Bases: object

_getHref()

Url of a Page.

Parameters
self –

_setHref(*val*)

Url of a Page.

Parameters
val – URL

_getHreflang()

_setHreflang(*val*)

class flare.html5._attrTarget

Bases: object

_getTarget()

_setTarget(*val*)

class flare.html5._attrType

Bases: object

_getType()

_setType(*val*)

class flare.html5._attrMedia

Bases: *_attrType*

_getMedia()

_setMedia(*val*)

class flare.html5._attrDimensions

Bases: object

_getWidth()

_setWidth(*val*)

_getHeight()

_setHeight(*val*)

class flare.html5._attrUsemap

Bases: object

_getUsemap()

_setUsemap(*val*)

class flare.html5._attrMultimedia

Bases: object

_getAutoplay()

_setAutoplay(*val*)

_getPlaysinline()

_setPlaysinline(*val*)

_getControls()

_setControls(*val*)

_getLoop()

_setLoop(*val*)

_getMuted()

_setMuted(*val*)

_getPreload()

_setPreload(*val*)

class flare.html5._attrRel

Bases: object

_getRel()

_setRel(*val*)

class flare.html5._attrSrc

Bases: object

_getSrc()

_setSrc(*val*)

class flare.html5.A(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrHref*, *_attrTarget*, *_attrMedia*, *_attrRel*, *_attrName*

_tagName = 'a'

_getDownload()

The download attribute specifies the path to a download.

Returns

filename

_setDownload(*val*)

The download attribute specifies the path to a download.

Parameters
val – filename

class flare.html5.**Area**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *A*, *_attrAlt*

_tagName = 'area'

_leafTag = True

_getCoords()

_setCoords(*val*)

_getShape()

_setShape(*val*)

class flare.html5.**Audio**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrSrc*, *_attrMultimedia*

_tagName = 'audio'

class flare.html5.**Bdo**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'bdo'

class flare.html5.**Blockquote**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'blockquote'

_getBlockquote()

_setBlockquote(*val*)

class flare.html5.**BodyCls**(**args*, ***kwargs*)

Bases: *Widget*

flare.html5._body

flare.html5.**Body**()

class flare.html5.**Canvas**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDimensions*

_tagName = 'canvas'

class flare.html5.**Command**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrLabel*, *_attrType*, *_attrDisabled*, *_attrChecked*

_tagName = 'command'

_getIcon()

_setIcon(*val*)

_getRadiogroup()

_setRadiogroup(*val*)

class flare.html5._Del(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrCite*, *_attrDatetime*

_tagName = '_del'

class flare.html5.Dialog(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'dialog'

_getOpen()

_setOpen(*val*)

class flare.html5.Abbr(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'abbr'

class flare.html5.Address(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'address'

class flare.html5.Article(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'article'

class flare.html5.Aside(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'aside'

class flare.html5.B(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'b'

class flare.html5.Bdi(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'bdi'

class flare.html5.Br(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'br'

_leafTag = True

class flare.html5.Caption(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'caption'

class flare.html5.**Cite**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'cite'

class flare.html5.**Code**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'code'

class flare.html5.**Datalist**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'datalist'

class flare.html5.**Dfn**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'dfn'

class flare.html5.**Div**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'div'

class flare.html5.**Em**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'em'

class flare.html5.**Embed**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrSrc*, *_attrType*, *_attrDimensions*

_tagName = 'embed'

_leafTag = True

class flare.html5.**Figcaption**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'figcaption'

class flare.html5.**Figure**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'figure'

class flare.html5.**Footer**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'footer'

class flare.html5.**Header**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'header'

class flare.html5.**H1**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'h1'

class flare.html5.H2(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'h2'

class flare.html5.H3(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'h3'

class flare.html5.H4(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'h4'

class flare.html5.H5(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'h5'

class flare.html5.H6(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'h6'

class flare.html5.Hr(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'hr'

_leafTag = True

class flare.html5.I(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'i'

class flare.html5.Kdb(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'kdb'

class flare.html5.Legend(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'legend'

class flare.html5.Mark(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'mark'

class flare.html5.Noscript(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'noscript'

class flare.html5.P(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'p'

class flare.html5.Rq(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'rq'

class flare.html5.Rt(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'rt'

class flare.html5.Ruby(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'ruby'

class flare.html5.S(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 's'

class flare.html5.Samp(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'samp'

class flare.html5.Section(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'section'

class flare.html5.Small(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'small'

class flare.html5.Strong(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'strong'

class flare.html5.Sub(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'sub'

class flare.html5.Summery(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'summery'

class flare.html5.Sup(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'sup'

class flare.html5.**U**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'u'

class flare.html5.**Var**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'var'

class flare.html5.**Wbr**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'wbr'

class flare.html5.**Button**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrType*, *_attrForm*, *_attrAutofocus*, *_attrName*, *_attrValue*, *_attrFormhead*

_tagName = 'button'

class flare.html5.**Fieldset**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrForm*, *_attrName*

_tagName = 'fieldset'

class flare.html5.**Form**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrName*, *_attrTarget*, *_attrAutocomplete*

_tagName = 'form'

_getNovalidate()

_setNovalidate(*val*)

_getAction()

_setAction(*val*)

_getMethod()

_setMethod(*val*)

_getEnctype()

_setEnctype(*val*)

_getAccept_attrCharset()

_setAccept_attrCharset(*val*)

class flare.html5.**Input**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrType*, *_attrForm*, *_attrAlt*, *_attrAutofocus*, *_attrChecked*, *_attrIndeterminate*, *_attrName*, *_attrDimensions*, *_attrValue*, *_attrFormhead*, *_attrAutocomplete*, *_attrInputs*, *_attrMultiple*, *_attrSize*, *_attrSrc*

_tagName = 'input'

_leafTag = True

_getAccept()

_setAccept(*val*)

_getList()

_setList(*val*)

_getMax()

_setMax(*val*)

_getMin()

_setMin(*val*)

_getPattern()

_setPattern(*val*)

_getStep()

_setStep(*val*)

class flare.html5.Label(**args*, *forElem=None*, ***kwargs*)

Bases: *Widget*, *_attrForm*, *_attrFor*

_tagName = 'label'

autoIdCounter = 0

class flare.html5.Optgroup(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrLabel*

_tagName = 'optgroup'

class flare.html5.Option(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrLabel*, *_attrValue*

_tagName = 'option'

_getSelected()

_setSelected(*val*)

class flare.html5.Output(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrForm*, *_attrName*, *_attrFor*

_tagName = 'output'

class flare.html5.Select(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrForm*, *_attrAutofocus*, *_attrName*, *_attrRequired*, *_attrMultiple*, *_attrSize*

_tagName = 'select'

_getSelectedIndex()

_getOptions()

class flare.html5.**Textarea**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrDisabled*, *_attrForm*, *_attrAutofocus*, *_attrName*, *_attrInputs*, *_attrValue*

_tagName = 'textarea'

_getCols()

_setCols(*val*)

_getRows()

_setRows(*val*)

_getWrap()

_setWrap(*val*)

class flare.html5.**HeadCls**(**args*, ***kwargs*)

Bases: *Widget*

flare.html5.**_head**

flare.html5.**Head**()

class flare.html5.**Iframe**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrSrc*, *_attrName*, *_attrDimensions*

_tagName = 'iframe'

_getSandbox()

_setSandbox(*val*)

_getSrcdoc()

_setSrcdoc(*val*)

_getSeamless()

_setSeamless(*val*)

class flare.html5.**Img**(*src=None*, **args*, ***kwargs*)

Bases: *Widget*, *_attrSrc*, *_attrDimensions*, *_attrUsemap*, *_attrAlt*

_tagName = 'img'

_leafTag = True

_getCrossorigin()

_setCrossorigin(*val*)

_getIsmap()

_setIsmap(*val*)

class flare.html5.**Ins**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrCite*, *_attrDatetime*

_tagName = 'ins'

class flare.html5.**Keygen**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Form*, *_attrAutofocus*, *_attrDisabled*

_tagName = 'keygen'

_getChallenge()

_setChallenge(*val*)

_getKeytype()

_setKeytype(*val*)

class flare.html5.**Link**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrHref*, *_attrMedia*, *_attrRel*

_tagName = 'link'

_leafTag = True

_getSizes()

_setSizes(*val*)

class flare.html5.**Ul**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'ul'

class flare.html5.**Ol**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'ol'

class flare.html5.**Li**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'li'

class flare.html5.**Dl**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'dl'

class flare.html5.**Dt**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'dt'

class flare.html5.**Dd**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'dd'

class flare.html5.**Map**(**args*, *forElem=None*, ***kwargs*)

Bases: *Label*, *_attrType*

_tagName = 'map'

class flare.html5.**Menu**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'menu'

class flare.html5.**Meta**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrName*, *_attrCharset*

_tagName = 'meta'

_leafTag = True

_getContent()

_setContent(*val*)

class flare.html5.**Meter**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Form*, *_attrValue*

_tagName = 'meter'

_getHigh()

_setHigh(*val*)

_getLow()

_setLow(*val*)

_getMax()

_setMax(*val*)

_getMin()

_setMin(*val*)

_getOptimum()

_setOptimum(*val*)

class flare.html5.**Nav**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'nav'

class flare.html5.**Object**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Form*, *_attrType*, *_attrName*, *_attrDimensions*, *_attrUsemap*

_tagName = 'object'

class flare.html5.**Param**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrName*, *_attrValue*

_tagName = 'param'

_leafTag = True

class flare.html5.**Progress**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrValue*

_tagName = 'progress'

_getMax()

_setMax(*val*)

class flare.html5.Q(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrCite*

_tagName = 'q'

class flare.html5.Script(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrSrc*, *_attrCharset*

_tagName = 'script'

_getAsync()

_setAsync(*val*)

_getDefer()

_setDefer(*val*)

class flare.html5.Source(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrMedia*, *_attrSrc*

_tagName = 'source'

_leafTag = True

class flare.html5.Span(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'span'

class flare.html5.Details(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'details'

_getOpen()

_setOpen(*val*)

class flare.html5.Summary(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'summary'

class flare.html5.Style(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*, *_attrMedia*

_tagName = 'style'

_getScoped()

_setScoped(*val*)

class flare.html5.**Tr**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'tr'

_getRowspan()

_setRowspan(*span*)

class flare.html5.**Td**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'td'

_getColspan()

_setColspan(*span*)

_getRowspan()

_setRowspan(*span*)

class flare.html5.**Th**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Td*

_tagName = 'th'

class flare.html5.**Thead**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'thead'

class flare.html5.**Tbody**(**args*, *appendTo=None*, *style=None*, ***kwargs*)

Bases: *Widget*

_tagName = 'tbody'

class flare.html5.**ColWrapper**(*parentElem*, **args*, ***kwargs*)

Bases: `object`

__getitem__(*item*)

__setitem__(*key*, *value*)

class flare.html5.**RowWrapper**(*parentElem*, **args*, ***kwargs*)

Bases: `object`

__getitem__(*item*)

class flare.html5.**Table**(**args*, ***kwargs*)

Bases: *Widget*

_tagName = 'table'

prepareRow(*row*)

prepareCol(*row*, *col*)

prepareGrid(*rows*, *cols*)

clear()

_getCell()

getRowCount()

class flare.html5.Time(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrDatetime

_tagName = 'time'

class flare.html5.Track(*args, forElem=None, **kwargs)

Bases: Label, _attrSrc

_tagName = 'track'

_leafTag = True

_getKind()

_setKind(val)

_getSrclang()

_setSrclang(val)

_getDefault()

_setDefault(val)

class flare.html5.Video(*args, appendTo=None, style=None, **kwargs)

Bases: Widget, _attrSrc, _attrDimensions, _attrMultimedia

_tagName = 'video'

_getPoster()

_setPoster(val)

class flare.html5.Template(*args, appendTo=None, style=None, **kwargs)

Bases: Widget

_tagName = 'template'

flare.html5.unescape(val, maxLength=0)

Unquotes several HTML-quoted characters in a string.

Parameters

- **val** (*str*) – The value to be unescaped.
- **maxLength** (*int*) – Cut-off after maxLength characters. A value of 0 means "unlimited". (default)

Returns

The unquoted string.

Return type

str

flare.html5.**doesEventHitWidgetOrParents**(*event*, *widget*)

Test if event 'event' hits widget 'widget' (or *any* of its parents).

flare.html5.**doesEventHitWidgetOrChildren**(*event*, *widget*)

Test if event 'event' hits widget 'widget' (or *any* of its children).

flare.html5.**textToHtml**(*node*, *text*)

Generates html nodes from text by splitting text into content and into line breaks html5.Br.

Parameters

- **node** – The node where the nodes are appended to.
- **text** – The text to be inserted.

flare.html5.**parseInt**(*s*, *ret=0*)

Parses a value as int.

flare.html5.**parseFloat**(*s*, *ret=0.0*)

Parses a value as float.

flare.html5.**getKey**(*event*)

Returns the Key Identifier of the given event.

Available Codes: https://www.w3.org/TR/2006/WD-DOM-Level-3-Events-20060413/keyset.html#KeySet-Set

flare.html5.**isArrowLeft**(*event*)

flare.html5.**isArrowUp**(*event*)

flare.html5.**isArrowRight**(*event*)

flare.html5.**isArrowDown**(*event*)

flare.html5.**isEscape**(*event*)

flare.html5.**isReturn**(*event*)

flare.html5.**isControl**(*event*)

flare.html5.**isShift**(*event*)

flare.html5.**isMeta**(*event*)

flare.html5.**__tags**

flare.html5.**__reVarReplacer**

flare.html5.**registerTag**(*tagName*, *widgetClass*, *override=True*)

flare.html5.**tag**(*arg*)

Decorator to register a sub-class of html5.Widget either under its class-name or an associated tag-name.

Examples

```python # register class Foo as <foo>-Tag @html5.tag class Foo(html5.Div):

    pass

# register class Bar as <baz>-Tag @html5.tag("baz") class Bar(html5.Div):

    pass

```

flare.html5.**_buildTags**(*debug=False*)

Generates a dictionary of all to the html5-library known tags and their associated objects and attributes.

class flare.html5.**HtmlAst**

Bases: list

Abstract syntax tree element used by parseHTML().

flare.html5.**parseHTML**(*html: str*, *debug: bool = False*) → *HtmlAst*

Parses the provided HTML-code according to the tags registered by html5.registerTag() or components that used the html5.tag-decorator.

flare.html5.**fromHTML**(*html: [str,* HtmlAst*]*, *appendTo:* Widget *= None*, *bindTo:* Widget *= None*, *debug: bool = False*, ***kwargs*) → [*Widget*]

Parses the provided HTML code according to the objects defined in the html5-library.

html can also be pre-compiled by *parseHTML()* so that it executes faster.

Constructs all objects as DOM nodes. The first level is chained into appendTo. If no appendTo is provided, appendTo will be set to html5.Body().

If bindTo is provided, objects are bound to this widget.

```python from vi import html5

div = html5.Div() html5.parse.fromHTML('"

    **<div>Yeah!**
    <a href="hello world" [name]="myLink" class="trullman bernd" disabled> hah ala malla" bababtschga" <img src="/static/images/icon_home.svg" style="background-color: red;"/>st <em>ah</em>ralla <i>malla tralla</i> da </a>lala

    </div>"', div)

div.myLink.appendChild("appended!") ```

## flare.translations

### Submodules

## flare.translations.de

### Module Contents

flare.translations.de.**lngDe**
```

flare.translations.en

Module Contents

flare.translations.en.**lngEn**

Package Contents

flare.translations.**lngDe**

flare.translations.**lngEn**

flare.views

Submodules

flare.views.helpers

Module Contents

Functions

generateView(view, moduleName, actionName[, name, data])	
addView(view[, name])	Add a View and make it available.
updateDefaultView(name)	
removeView(name[, targetView])	
registerViews(root, path)	Add all Views in a folder.
zip_listdir(zip_file, target_dir)	

Attributes

sitepackagespath

flare.views.helpers.**sitepackagespath**

flare.views.helpers.**generateView**(*view:* flare.views.view.View, *moduleName*, *actionName*, *name=None*, *data=()*)

flare.views.helpers.**addView**(*view:* flare.views.view.View, *name=None*)

Add a View and make it available.

flare.views.helpers.**updateDefaultView**(*name*)

flare.views.helpers.**removeView**(*name*, *targetView=None*)

flare.views.helpers.**registerViews**(*root*, *path*)

Add all Views in a folder.

flare.views.helpers.**zip_listdir**(*zip_file*, *target_dir*)

flare.views.view

Module Contents

Classes

View
ViewWidget

class flare.views.view.**View**(*dictOfWidgets=None*, *name=None*)

onActiveViewChanged(*viewName*, **args*, ***kwargs*)

loadView()

class flare.views.view.**ViewWidget**(*view*)

Bases: *flare.html5.Div*

onViewfocusedChanged(*viewname*, **args*, ***kwargs*)

initWidget()

onDetach()

Package Contents

Classes

StateHandler

Attributes

conf

class flare.views.**StateHandler**(*initialize=()*, *widget=None*)

updateState(*key*, *value*)

getState(*key*, *empty=None*)

register(*key*, *widget*)

unregister(*key*, *widget*)

flare.views.**conf**

flare.viur

Subpackages

flare.viur.bones

Expose all bones.

Submodules

flare.viur.bones.base

Collection of Basebone related classes.

Module Contents

Classes

ReadFromClientErrorSeverity	Enum for Errors.
BaseEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
BaseViewWidget	Base class for a bone-compliant view widget implementation using a div.
BaseMultiEditWidgetEntry	Base class for an entry in a MultiBone container.
BaseMultiEditWidget	Class for encapsulating multiple bones inside a container.
BaseMultiViewWidget	
BaseLanguageEditWidget	Class for encapsulating a bone for each language inside a container.
BaseBone	

class flare.viur.bones.base.**ReadFromClientErrorSeverity**

Bases: enum.IntEnum

Enum for Errors.

NotSet = 0

InvalidatesOther = 1

Empty = 2

Invalid = 3

class flare.viur.bones.base.**BaseEditWidget**(*bone*, ***kwargs*)

Bases: flare.ignite.html5.Div

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

class flare.viur.bones.base.**BaseViewWidget**(*bone*, ***kwargs*)

Bases: flare.ignite.html5.Div

Base class for a bone-compliant view widget implementation using a div.

style = ['flr-value']

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

class flare.viur.bones.base.**BaseMultiEditWidgetEntry**(*widget: flare.ignite.html5.Widget*, *errorInformation=None*)

Bases: flare.ignite.html5.Div

Base class for an entry in a MultiBone container.

style = ['flr-bone-widgets-item']

onRemoveBtnClick()

onDragStart(*event*)

onDragOver(*event*)

onDragLeave(*event*)

onDragEnd(*event*)

onDrop(*event*)

class flare.viur.bones.base.**BaseMultiEditWidget**(*bone*, *widgetFactory: callable*, ***kwargs*)

Bases: flare.ignite.html5.Div

Class for encapsulating multiple bones inside a container.

entryFactory

style = ['flr-value-container']

onAddBtnClick()

onRemoveBtnClick()

addEntry(*value=None*)

unserialize(*value*)

serialize()

class flare.viur.bones.base.**BaseMultiViewWidget**(*bone*, *widgetFactory: callable*, ***kwargs*)

Bases: flare.ignite.html5.Ul

unserialize(*value*)

serialize()

class flare.viur.bones.base.**BaseLanguageEditWidget**(*bone*, *widgetFactory: callable*, ***kwargs*)

Bases: flare.ignite.html5.Div

Class for encapsulating a bone for each language inside a container.

onLangBtnClick(*sender*)

unserialize(*value*)

serialize()

class flare.viur.bones.base.**BaseBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: object

editWidgetFactory

viewWidgetFactory

multiEditWidgetFactory

multiViewWidgetFactory

languageEditWidgetFactory

languageViewWidgetFactory

Base "Catch-All" delegate for everything not handled separately.

editWidget(*value=None*, *errorInformation=None*) → flare.ignite.html5.Widget

viewWidget(*value=None*)

labelWidget()

tooltipWidget()

errorWidget()

boneWidget(**args*, ***kwargs*)

flare.viur.bones.boolean

Module Contents

Classes

BooleanEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
BooleanViewWidget	Base class for a bone-compliant view widget implementation using a div.
BooleanBone	

class flare.viur.bones.boolean.**BooleanEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--boolean']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

class flare.viur.bones.boolean.**BooleanViewWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseViewWidget*

Base class for a bone-compliant view widget implementation using a div.

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.boolean.**BooleanBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

viewWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.color

Module Contents

Classes

ColorEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
ColorViewWidget	Base class for a bone-compliant view widget implementation using a div.
ColorBone	

class flare.viur.bones.color.**ColorEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--color']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

onUnsetBtnClick()

serialize()

Serialize the widget value.

class flare.viur.bones.color.**ColorViewWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseViewWidget*

Base class for a bone-compliant view widget implementation using a div.

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.color.**ColorBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

viewWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.date

Module Contents

Classes

DateEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
DateViewWidget	Base class for a bone-compliant view widget implementation using a div.
DateBone	

class flare.viur.bones.date.**DateEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--date']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

class flare.viur.bones.date.**DateViewWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseViewWidget*

Base class for a bone-compliant view widget implementation using a div.

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.date.**DateBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

viewWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.email

Module Contents

Classes

EmailEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
EmailViewWidget	Base class for a bone-compliant view widget implementation using a div.
EmailBone	

class flare.viur.bones.email.**EmailEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

class flare.viur.bones.email.**EmailViewWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseViewWidget*

Base class for a bone-compliant view widget implementation using a div.

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.email.**EmailBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

viewWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.numeric

Module Contents

Classes

NumericEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
NumericViewWidget	Base class for a bone-compliant view widget implementation using a div.
NumericBone	

Functions

_formatCurrencyValue(value, bone)	Internal helper function that formats a numeric value which is a string according to the bone's formatting

flare.viur.bones.numeric.**_formatCurrencyValue**(*value*, *bone*)

Internal helper function that formats a numeric value which is a string according to the bone's formatting

class flare.viur.bones.numeric.**NumericEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--numeric']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

setValue(*value*)

onChange(*event*)

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

class flare.viur.bones.numeric.**NumericViewWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseViewWidget*

Base class for a bone-compliant view widget implementation using a div.

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.numeric.**NumericBone**(**args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

viewWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.password

Module Contents

Classes

PasswordEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
PasswordBone	

class flare.viur.bones.password.**PasswordEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--password', 'flr-value-container', 'input-group']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

serialize()

Serialize the widget value.

class flare.viur.bones.password.**PasswordBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.raw

Module Contents

Classes

RawEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
RawViewWidget	Base class for a bone-compliant view widget implementation using a div.
RawBone	

class flare.viur.bones.raw.**RawEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--raw']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

class flare.viur.bones.raw.**RawViewWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseViewWidget*

Base class for a bone-compliant view widget implementation using a div.

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.raw.**RawBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

viewWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.record

Module Contents

Classes

RecordEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
RecordViewWidget	Base class for a bone-compliant view widget implementation using a div.
RecordBone	

class flare.viur.bones.record.**RecordEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--record']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

class flare.viur.bones.record.**RecordViewWidget**(*bone*, *language=None*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseViewWidget*

Base class for a bone-compliant view widget implementation using a div.

style = ['flr-value', 'flr-value--record']

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.record.**RecordBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

viewWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.relational

Module Contents

Classes

RelationalEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
RelationalViewWidget	
RelationalMultiEditWidget	Class for encapsulating multiple bones inside a container.
RelationalBone	
HierarchyBone	
TreeItemBone	
TreeDirBone	
FileEditDirectWidget	Base class for a bone-compliant edit widget implementation using an input field.
FileViewWidget	
FileMultiEditDirectWidget	Class for encapsulating multiple bones inside a container.
FileDirectBone	
FileEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
FileBone	

Functions

_getDefaultValues(structure)	Gets defaultValues from a structure.

flare.viur.bones.relational.**_getDefaultValues**(*structure*)

Gets defaultValues from a structure.

class flare.viur.bones.relational.**RelationalEditWidget**(*bone*, *language=None*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--relational']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

updateString()

onChange(*event*)

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

onSelectBtnClick()

onDeleteBtnClick()

class flare.viur.bones.relational.**RelationalViewWidget**(*bone*, *language=None*, ***kwargs*)

Bases: *flare.html5.Div*

style = ['flr-value', 'flr-value--relational']

unserialize(*value=None*)

serialize()

class flare.viur.bones.relational.**RelationalMultiEditWidget**(**args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseMultiEditWidget*

Class for encapsulating multiple bones inside a container.

onAddBtnClick()

_addEntriesFromSelection(*selector*, *selection*)

class flare.viur.bones.relational.**RelationalBone**(**args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

viewWidgetFactory

multiEditWidgetFactory

selectorAllow = ()

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

class flare.viur.bones.relational.**HierarchyBone**(**args*, ***kwargs*)

Bases: *RelationalBone*

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

class flare.viur.bones.relational.**TreeItemBone**(**args*, ***kwargs*)

Bases: *RelationalBone*

selectorAllow

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

class flare.viur.bones.relational.TreeDirBone(**args*, ***kwargs*)

Bases: *RelationalBone*

selectorAllow

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

class flare.viur.bones.relational.FileEditDirectWidget(*bone*, *language=None*, ***kwargs*)

Bases: *RelationalEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--file']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

onChange(*event*)

startUpload(*file*)

onDragEnter(*event*)

onDragOver(*event*)

onDragLeave(*event*)

onDrop(*event*)

onUploadSuccess(*uploader*, *entry*)

onUploadFailed(*uploader*, *errorCode*)

unserialize(*value=None*)

Unserialize the widget value.

onDeleteBtnClick()

class flare.viur.bones.relational.FileViewWidget(*bone*, *language=None*, ***kwargs*)

Bases: *RelationalViewWidget*

unserialize(*value=None*)

class flare.viur.bones.relational.FileMultiEditDirectWidget(*bone*, *widgetFactory: callable*, ***kwargs*)

Bases: *flare.html5.Div*

Class for encapsulating multiple bones inside a container.

entryFactory

style = ['flr-value-container']

onChange(*event*)

startUpload(*file*)

onDragEnter(*event*)

onDragOver(*event*)

onDragLeave(*event*)

onDrop(*event*)

onUploadSuccess(*uploader*, *entry*)

onUploadFailed(*uploader*, *errorCode*)

addEntry(*value=None*)

unserialize(*value*)

serialize()

class flare.viur.bones.relational.**FileDirectBone**(**args*, ***kwargs*)

Bases: *TreeItemBone*

editWidgetFactory

viewWidgetFactory

multiEditWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

class flare.viur.bones.relational.**FileEditWidget**(*bone*, *language=None*, ***kwargs*)

Bases: *RelationalEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--relational', 'flr-value--file']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.relational.**FileBone**(**args*, ***kwargs*)

Bases: *TreeItemBone*

editWidgetFactory

viewWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.select

Module Contents

Classes

SelectMultipleEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
SelectSingleEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
SelectViewWidget	Base class for a bone-compliant view widget implementation using a div.
SelectMultipleBone	
SelectSingleBone	

class flare.viur.bones.select.**SelectMultipleEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value-container', 'option-group']

entryTemplate

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

class flare.viur.bones.select.**SelectSingleEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

entryTemplate

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

class flare.viur.bones.select.**SelectViewWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseViewWidget*

Base class for a bone-compliant view widget implementation using a div.

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.select.**SelectMultipleBone**(**args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

multiEditWidgetFactory

viewWidgetFactory

Base "Catch-All" delegate for everything not handled separately.

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

class flare.viur.bones.select.**SelectSingleBone**(**args*, ***kwargs*)

Bases: *SelectMultipleBone*

editWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.spatial

Module Contents

Classes

SpatialEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
SpatialBone	

class flare.viur.bones.spatial.**SpatialEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

class flare.viur.bones.spatial.**SpatialBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.string

Module Contents

Classes

StringEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
StringViewWidget	Base class for a bone-compliant view widget implementation using a div.
StringBone	

class flare.viur.bones.string.**StringEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--string']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

onChange(*event*)

onKeyUp(*event*)

renderTimeout()

updateLength()

unserialize(*value=None*)

Unserialize the widget value.

serialize()

Serialize the widget value.

class flare.viur.bones.string.**StringViewWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseViewWidget*

Base class for a bone-compliant view widget implementation using a div.

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.string.**StringBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

viewWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.bones.text

Module Contents

Classes

TextEditWidget	Base class for a bone-compliant edit widget implementation using an input field.
TextViewWidget	Base class for a bone-compliant view widget implementation using a div.
TextBone	

class flare.viur.bones.text.**TextEditWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseEditWidget*

Base class for a bone-compliant edit widget implementation using an input field.

This widget defines the general interface of a bone edit control.

style = ['flr-value', 'flr-value--text']

createWidget()

Function for creating the Widget or multiple Widgets that represent the bone.

updateWidget()

Function for updating the Widget or multiple Widgets that represent the bone.

_setDisabled(*disable*)

class flare.viur.bones.text.**TextViewWidget**(*bone*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseViewWidget*

Base class for a bone-compliant view widget implementation using a div.

unserialize(*value=None*)

Unserialize the widget value.

class flare.viur.bones.text.**TextBone**(*moduleName*, *boneName*, *skelStructure*, *errors=None*, *errorQueue=None*, **args*, ***kwargs*)

Bases: *flare.viur.bones.base.BaseBone*

editWidgetFactory

viewWidgetFactory

static checkFor(*moduleName*, *boneName*, *skelStructure*, **args*, ***kwargs*)

flare.viur.widgets

Submodules

flare.viur.widgets.file

Module Contents

Classes

Search	
FileImagePopup	
FilePreviewImage	
Uploader	Uploads a file to the server while providing visual feedback of the progress.
FileLeafWidget	
FileNodeWidget	
FileWidget	Base Widget that renders a tree.

Functions

getImagePreview(data[, cropped, size])	

flare.viur.widgets.file.**getImagePreview**(*data*, *cropped=False*, *size=150*)

class flare.viur.widgets.file.**Search**(**args*, ***kwargs*)

Bases: flare.ignite.html5.Div

doSearch(**args*, ***kwargs*)

resetSearch()

onKeyDown(*event*)

resetLoadingState()

reevaluate()

focus()

class flare.viur.widgets.file.**FileImagePopup**(*preview*, **args*, ***kwargs*)

Bases: flare.popup.Popup

onClick(*event*)

onDownloadBtnClick(*sender=None*)

class flare.viur.widgets.file.**FilePreviewImage**(*file=None*, *size=150*, **args*, ***kwargs*)

Bases: flare.ignite.html5.Div

setFile(*file*)

download()

onClick(*sender=None*)

class flare.viur.widgets.file.**Uploader**(*file*, *node*, *context=None*, *showResultMessage=True*, *module='file'*, **args*, ***kwargs*)

Bases: flare.ignite.Progress

Uploads a file to the server while providing visual feedback of the progress.

onUploadUrlAvailable(*req*)

Internal callback - the actual upload url (retrieved by calling /file/getUploadURL) is known.

onLoad(**args*, ***kwargs*)

Internal callback - The state of our upload changed.

onUploadAdded(*req*)

onProgress(*event*)

Internal callback - further bytes have been transmitted.

onSuccess(**args*, ***kwargs*)

Internal callback - The upload succeeded.

onFailed(*errorCode*, **args*, ***kwargs*)

replaceWithMessage(*message*, *isSuccess*)

class flare.viur.widgets.file.**FileLeafWidget**(*module*, *data*, *structure*, *widget*, **args*, ***kwargs*)

Bases: *flare.viur.widgets.tree.TreeLeafWidget*

EntryIcon()

Leafs have a different Icon.

setStyle()

Leaf have a different color.

class flare.viur.widgets.file.**FileNodeWidget**(*module*, *data*, *structure*, *widget*, **args*, ***kwargs*)

Bases: *flare.viur.widgets.tree.TreeNodeWidget*

setStyle()

Is used to define the appearance of the element.

class flare.viur.widgets.file.**FileWidget**(*module*, *rootNode=None*, *selectMode=None*, *node=None*, *context=None*, **args*, ***kwargs*)

Bases: *flare.viur.widgets.tree.TreeBrowserWidget*

Base Widget that renders a tree.

leafWidget

nodeWidget

searchWidget()

onStartSearch(*searchStr*, **args*, ***kwargs*)

getChildKey(*widget*)

Derives a string used to sort the entries on each level.

static canHandle(*module*, *moduleInfo*)

flare.viur.widgets.htmleditor

Module Contents

Classes

TextInsertImageAction	Extended version for a button with a text and icon, which binds itself to an event function.
HtmlEditor	

Attributes

summernoteEditor

flare.viur.widgets.htmleditor.**summernoteEditor**

class flare.viur.widgets.htmleditor.**TextInsertImageAction**(*summernote*, *id*, **args*, ***kwargs*)

Bases: *flare.button.Button*

Extended version for a button with a text and icon, which binds itself to an event function.

onClick(*sender=None*)

onSelectionActivated(*selectWdg*, *selection*)

static isSuitableFor(*modul*, *handler*, *actionName*)

resetLoadingState()

class flare.viur.widgets.htmleditor.**HtmlEditor**(**args*, ***kwargs*)

Bases: *flare.html5.Textarea*

initSources = False

_attachSummernote(*retry=0*)

onAttach()

onDetach()

onEditorChange(*e*, **args*, ***kwargs*)

_getValue()

_setValue(*val*)

enable()

Enables an element, in case it is not already enabled.

disable()

Disables an element, in case it is not already disabled.

On disabled elements, events are not triggered anymore.

flare.viur.widgets.list

Module Contents

Classes

ListWidget	Provides the interface to list-applications.
SkellistItem	Extended version for a button with a text and icon, which binds itself to an event function.
ListSelection	

class flare.viur.widgets.list.**ListWidget**(*module*, *filter=None*, *columns=None*, *filterID=None*, *filterDescr=None*, *batchSize=None*, *context=None*, *autoload=True*, **args*, ***kwargs*)

Bases: *flare.html5.Div*

Provides the interface to list-applications.

It acts as a data-provider for a DataTable and binds an action-bar to this table.

setSelector(*callback*, *multi=True*, *allow=None*)

Configures the widget as selector for a relationalBone and shows it.

onAcceptSelectionChanged(*event*, **args*, ***kwargs*)

static canHandle(*moduleName*, *moduleInfo*)

class flare.viur.widgets.list.**SkellistItem**(*skel*)

Bases: *flare.button.Button*

Extended version for a button with a text and icon, which binds itself to an event function.

buildWidget()

onActiveSelectionChanged(*event*, **args*, ***kwargs*)

class flare.viur.widgets.list.**ListSelection**(*modulname*, *filter=None*, *title=None*, *id=None*, *className=None*, *icon=None*, *enableShortcuts=True*, *closeable=True*, *footer=True*, **args*, ***kwargs*)

Bases: *flare.popup.Popup*

requestClients()

onRequestList(*skellist*)

onActiveSelectionChanged(*event*, **args*, ***kwargs*)

activateSelection(*widget*)

reloadList()

buildListSelection()

onApplyfilterChanged(*value*, **args*, ***kwargs*)

onAcceptSelectionChanged(*event*, **args*, ***kwargs*)

onActiveButtonChanged(*event*, **args*, ***kwargs*)

acceptSelection()

setContent(*widget*)

flare.viur.widgets.tree

Module Contents

Classes

TreeItemWidget	
TreeLeafWidget	
TreeNodeWidget	
TreeWidget	Base Widget that renders a tree.
BrowserLeafWidget	
BrowserNodeWidget	
BreadcrumbNodeWidget	
TreeBrowserWidget	Base Widget that renders a tree.

class flare.viur.widgets.tree.**TreeItemWidget**(*module*, *data*, *structure*, *widget*, **args*, ***kwargs*)

Bases: *flare.html5.Li*

setStyle()

Is used to define the appearance of the element.

additionalDropAreas()

Drag and Drop areas.

markDraggedElement()

Mark the current dragged Element.

unmarkDraggedElement()

onDragStart(*event*)

onDragEnd(*event*)

onDragOver(*event*)

Test wherever the current drag would mean.

"make it a child of us", "insert before us" or "insert after us" and apply the correct classes.

onDragLeave(*event*)

Remove all drop indicating classes.

disableDragMarkers()

moveRequest(*params*)

Performs the move operation with the provided params.

onDrop(*event*)

We received a drop.

Test wherever its means "make it a child of us", "insert before us" or "insert after us" and initiate the corresponding NetworkService requests.

EntryIcon()

toggleArrow()

buildDescription()

Creates the visual representation of our entry.

onClick(*event*)

onDblClick(*event*)

toggleExpand()

Toggle a Node and request if needed child elements.

class flare.viur.widgets.tree.**TreeLeafWidget**(*module*, *data*, *structure*, *widget*, **args*, ***kwargs*)

Bases: *TreeItemWidget*

skelType = 'leaf'

setStyle()

Leaf have a different color.

toggleArrow()

Leafes cant be toggled.

EntryIcon()

Leafs have a different Icon.

moveRequest(*params*)

Performs the move operation with the provided params.

onDragOver(*event*)

Test wherever the current drag would mean.

"make it a child of us", "insert before us" or "insert after us" and apply the correct classes.

class flare.viur.widgets.tree.**TreeNodeWidget**(*module*, *data*, *structure*, *widget*, **args*, ***kwargs*)

Bases: *TreeItemWidget*

skelType = 'node'

class flare.viur.widgets.tree.**TreeWidget**(*module*, *rootNode=None*, *node=None*, *context=None*, **args*, ***kwargs*)

Bases: *flare.html5.Div*

Base Widget that renders a tree.

nodeWidget

leafWidget

requestStructure()

receivedStructure(*resp*)

setSelector(*callback*, *multi=True*, *allow=None*)

Configures the widget as selector for a relationalBone and shows it.

selectorReturn()

Returns the current selection to the callback configured with *setSelector*.

onKeyDown(*event*)

onKeyUp(*event*)

getActions()

Returns a list of actions that are being set for the ActionBar. Override this to provide additional actions.

clearSelection()

Empties the current selection.

extendSelection(*element*)

Extends the current selection to element.

This is normally done by clicking or tabbing on an element.

activateSelection(*element*)

Activates the current selection or element.

An activation mostly is an action like selecting or editing an item. This is normally done by double-clicking an element.

requestChildren(*element*)

_showErrorMsg(*req=None*, *code=None*)

Removes all currently visible elements and displayes an error message

onDataChanged(*module*, **args*, ***kwargs*)

onAttach()

onDetach()

itemForKey(*key*, *elem=None*)

Returns the HierarchyWidget displaying the entry with the given key. :param key: The key (id) of the item. :type key: str :returns: HierarchyItem

onSetDefaultRootNode(*req*)

We requested the list of rootNodes for that module and that request just finished. Parse the respone and set our rootNode to the first rootNode received.

setRootNode(*rootNode*, *node=None*)

Set the currently displayed hierarchy to 'rootNode'. :param rootNode: Key of the rootNode which children we shall display :type rootNode: str

reloadData()

Reload the data were displaying.

loadNode(*node*, *skelType=None*, *cursor=None*, *overrideParams=None*)

Fetch the (direct) children of the given node. Once the list is received, append them to their parent node. :param node: Key of the node to fetch :type node: str

_onRequestSucceded(*req*)

The NetworkRequest for a (sub)node finished. Create a new HierarchyItem for each entry received and add them to our view

onDrop(*event*)

We got a drop event. Make that item a direct child of our rootNode

onDragOver(*event*)

Allow dropping children on the rootNode

getChildKey(*widget*)

Order by sortindex

static canHandle(*moduleName*, *moduleInfo*)

class flare.viur.widgets.tree.**BrowserLeafWidget**(*module*, *data*, *structure*, *widget*, **args*, ***kwargs*)

Bases: *TreeLeafWidget*

setStyle()

Leaf have a different color.

class flare.viur.widgets.tree.**BrowserNodeWidget**(*module*, *data*, *structure*, *widget*, **args*, ***kwargs*)

Bases: *TreeNodeWidget*

setStyle()

Is used to define the appearance of the element.

class flare.viur.widgets.tree.**BreadcrumbNodeWidget**(*module*, *data*, *structure*, *widget*, **args*, ***kwargs*)

Bases: *TreeNodeWidget*

setStyle()

Is used to define the appearance of the element.

class flare.viur.widgets.tree.**TreeBrowserWidget**(*module*, *rootNode=None*, *node=None*, *context=None*, **args*, ***kwargs*)

Bases: *TreeWidget*

Base Widget that renders a tree.

leafWidget

nodeWidget

reloadData()

Reload the data were displaying.

rebuildPath()

Rebuild the displayed path-list.

onPathRequestSucceded(*req*)

Rebuild the displayed path-list according to request data

activateSelection(*element*)

Activates the current selection or element.

An activation mostly is an action like selecting or editing an item. This is normally done by double-clicking an element.

static canHandle(*module*, *moduleInfo*)

Submodules

flare.viur.formatString

Module Contents

Functions

formatString(format, data[, structure, language])	Central entryPoint
formatStringHandler(→ str)	
displayStringHandler(→ [flare.html5.Widget])	
evalStringHandler(format, data, structure, language)	

flare.viur.formatString.**formatString**(*format: str*, *data: Dict*, *structure=None*, *language=None*)

Central entryPoint

if string contains $(we use old formatstrings else we use evalStrings (core 3.0 draft)

displayStrings actually only used in relations and records. This handler can be used with display param

flare.viur.formatString.**formatStringHandler**(*format: str*, *value: Dict*, *structure: Dict*, *language: str = 'de'*) → str

flare.viur.formatString.**displayStringHandler**(*display: str*, *value: Dict*, *structure: Dict*, *language: str = 'de'*) → [*flare.html5.Widget*]

flare.viur.formatString.**evalStringHandler**(*format*, *data*, *structure*, *language*)

flare.viur.formconf

Module Contents

flare.viur.formconf.**conf**

A value displayed as “empty value” “emptyValue”: translate(“-“),

Language settings “flare.language.current”: “de”,

Global holder to main admin window “mainWindow”: None,

Modules list “modules”: {“_tasks”: {“handler”: “singleton”, “name”: “Tasks”}},

Language settings “defaultLanguage”: “de”,

Cached selector widgets on relationalBones for re-use “selectors”: {},

flare.viur.formerrors

Module Contents

Functions

collectBoneErrors(errorList, currentKey, boneStructure)	Collect Errors from given errorList.

flare.viur.formerrors.**collectBoneErrors**(*errorList*, *currentKey*, *boneStructure*)

Collect Errors from given errorList.

severity:
NotSet = 0 InvalidatesOther = 1 Empty = 2 Invalid = 3

flare.viur.forms

Module Contents

Classes

ViurForm	Handles an input form for a VIUR skeleton.
ViurFormBone	
ViurFormSubmit	Extended version for a button with a text and icon, which binds itself to an event function.

class flare.viur.forms.**ViurForm**(*formName: str = None*, *moduleName: str = None*, *actionName: str = 'add'*, *skel=None*, *structure=None*, *visible=()*, *ignore=()*, *hide=()*, *errors=None*, *context=None*, **args*, ***kwargs*)

Bases: flare.html5.Form

Handles an input form for a VIUR skeleton.

onChange(*event*)

onBoneChange(*bone*)

_setModulename(*val*)

_setActionname(*val*)

_setFormname(*val*)

buildForm()

Builds a form with save button.

buildInternalForm()

Builds only the form.

registerField(*key*, *widget*)

update()

Updates current form view state regarding conditional input fields.

submitForm()

unserialize(*skel: Dict = None*)

Unserializes a dict of values into this form. :param skel: Either a dict of values to be unserialized into this form, or None for emptying all values.

serialize(*all=False*) → Dict

Serializes all bone's values into a dict to be sent to ViUR or the be evaluated.

actionSuccess(*req*)

handleErrors()

createFormSuccessMessage()

createFormErrorMessage()

actionFailed(*req*, **args*, ***kwargs*)

onFormSuccess(*event*)

onSubmitStatusChanged(*value*, **args*, ***kwargs*)

class flare.viur.forms.**ViurFormBone**(*boneName=None*, *form=None*, *defaultvalue=None*, *hidden=False*, *filter=None*)

Bases: *flare.html5.Div*

onAttach()

onChange(*event*, **args*, ***kwargs*)

unserialize(*data=None*)

serialize()

_setBonename(*val*)

_setLabel(*val*)

_setPlaceholder(*val*)

_setHide(*val*)

_setValue(*val*)

setInvalid(*errors=None*)

setValid()

class flare.viur.forms.**ViurFormSubmit**(*text=None*, *callback=None*, *className='btn--submit btn--primary'*, *icon=None*, *badge=None*, *form=None*)

Bases: *flare.button.Button*

Extended version for a button with a text and icon, which binds itself to an event function.

onAttach()

sendViurForm(*sender=None*)

onSubmitStatusChanged(*value*, **args*, ***kwargs*)

flare.viur.formtooltip

Module Contents

Classes

ToolTip	Small utility class for providing tooltips.

class flare.viur.formtooltip.**ToolTip**(*shortText=''*, *longText=''*, **args*, ***kwargs*)

Bases: *flare.html5.Div*

Small utility class for providing tooltips.

onClick(*event*)

_setDisabled(*disabled*)

Package Contents

Classes

PriorityQueue	

Functions

formatString(format, data[, structure, language])	Central entryPoint
displayStringHandler(→ [flare.html5.Widget])	

Attributes

conf	# A value displayed as "empty value"
BoneSelector	
ModuleWidgetSelector	
DisplayDelegateSelector	

class flare.viur.**PriorityQueue**

Bases: object

insert(*priority*, *validateFunc*, *generator*)

select(**args*, ***kwargs*)

flare.viur.**conf**

A value displayed as “empty value” “emptyValue”: translate(“-“),

Language settings “flare.language.current”: “de”,

Global holder to main admin window “mainWindow”: None,

Modules list “modules”: {“_tasks”: {“handler”: “singleton”, “name”: “Tasks”}},

Language settings “defaultLanguage”: “de”,

Cached selector widgets on relationalBones for re-use “selectors”: {},

flare.viur.**formatString**(*format: str*, *data: Dict*, *structure=None*, *language=None*)

Central entryPoint

if string contains $(we use old formatstrings else we use evalStrings (core 3.0 draft)

displayStrings actually only used in relations and records. This handler can be used with display param

flare.viur.**displayStringHandler**(*display: str*, *value: Dict*, *structure: Dict*, *language: str = 'de'*) → [*flare.html5.Widget*]

flare.viur.**BoneSelector**

flare.viur.**ModuleWidgetSelector**

flare.viur.**DisplayDelegateSelector**

exception flare.viur.**InvalidBoneValueException**

Bases: ValueError

Inappropriate argument value (of correct type).

flare.widgets

Submodules

flare.widgets.buttonbar

Module Contents

Classes

ButtonBar	
ButtonBarButton	Extended version for a button with a text and icon, which binds itself to an event function.
ButtonBarSearch	

class flare.widgets.buttonbar.**ButtonBar**

Bases: *flare.html5.Div*

onActiveButtonChanged(*event*, **args*, ***kwargs*)

addButton(*name*, *btnStr*)

buttonClicked(*widget*)

class flare.widgets.buttonbar.**ButtonBarButton**

Bases: *flare.button.Button*

Extended version for a button with a text and icon, which binds itself to an event function.

onActiveButtonChanged(*event*, **args*, ***kwargs*)

class flare.widgets.buttonbar.**ButtonBarSearch**

Bases: *flare.html5.Div*

applyFilter(*widget*)

onApplyfilterChanged(*event*, **args*, ***kwargs*)

onActiveButtonChanged(*event*, **args*, ***kwargs*)

Submodules

flare.button

Flare-styled button Widgets.

Module Contents

Classes

Button	Extended version for a button with a text and icon, which binds itself to an event function.

class flare.button.**Button**(*text=None*, *callback=None*, *className=''*, *icon=None*)

Bases: *flare.html5.Button*

Extended version for a button with a text and icon, which binds itself to an event function.

onBind(*widget*, *name*)

Event function that is called on the widget when it is bound to another widget with a name.

This is only done by the HTML parser, a manual binding by the user is not triggered.

onClick(*event*, *widget=None*)

resetIcon()

update()

_setIcon(*icon*)

_getIcon()

_setText(*text*)

_getText()

flare.cache

The cache module is set on top of the network module and caches any entries read.

When the same entry (identified by module and key) is requested, it first is returned from the cache, when already there.

Module Contents

Classes

Cache
Plan

class flare.cache.**Cache**

Bases: object

updateStructure(*module*, *structure*)

update(*module*, *key*, *data*, *structure=None*)

lookup(*module*, *key='current'*)

struct(*module*)

start(*plan*, *finishHandler=None*, *failureHandler=None*)

finish(*plan*)

require(**args*)

invalidate(**args*)

onDataChanged(*module*, *key=None*, ***kwargs*)

request(**args*, *finishHandler=None*, *failureHandler=None*)

class flare.cache.**Plan**(*module*, *action*, *params=None*, *follow=None*, *alias='current'*, *local=True*, *force=False*)

Bases: object

run(*cache*)

finish(*cache*)

_onRequestSuccess(*req*)

_onRequestFailure(*req*, *code*)

flare.config

Flare configuration.

Module Contents

Functions

updateConf(other)	Merges other into conf.

Attributes

conf

flare.config.**updateConf**(*other: Dict*)

Merges other into conf.

flare.config.**conf**

flare.debug

still WIP

Module Contents

Functions

debug([element])	Debug popup
debugElement(element)	recursive debug tree

flare.debug.**debug**(*element=None*)

Debug popup

flare.debug.**debugElement**(*element*)

recursive debug tree

flare.event

Event dispatcher for non-browser Events which occur on Widget state changes.

Module Contents

Classes

EventDispatcher	Base class for event notifier.

class flare.event.**EventDispatcher**(*name*)

Bases: object

Base class for event notifier.

_genTargetFuncName()

Return the name of the function called on the receiving object.

register(*cb*, *reset=False*)

Append "cb" to the list of objects to inform of the given Event.

Does nothing if cb has already subscribed. :param cb: the object to register :type cb: object

unregister(*cb*)

Remove "cb" from the list of objects to inform of the given Event.

Does nothing if cb is not in that list. :param cb: the object to remove :type cb: object

fire(**args*, ***kwargs*)

Fire the event.

Informs all subscribed listeners. All parameters passed to the receiving function.

flare.handler

Flare base handlers for ViUR prototypes.

Module Contents

Classes

requestHandler
ListHandler
SyncHandler

class flare.handler.**requestHandler**(*module*, *action*, *params=()*, *eventName='listUpdated'*, *secure=False*)

requestData(**args*, ***kwargs*)

requestSuccess(*req*)

_requestFailed(*req*, **args*, ***kwargs*)

onListStatusChanged(*event*, **args*, ***kwargs*)

getDescrFromValue(*definition*, *val*)

buildSelectDescr(*skel*, *structure*)

class flare.handler.**ListHandler**(*module*, *action*, *params=()*, *eventName='listUpdated'*, *secure=False*)

Bases: *requestHandler*

reload()

filter(*filterparams*)

getCurrentAmount()

requestNext()

requestSuccess(*req*)

class flare.handler.**SyncHandler**

Bases: object

static request(*url*, *params=None*, *jsonResult=None*)

genReqStr(*params*)

_request(*url*, *params*)

onCompletion(*text*)

onError(*text*, *code*)

flare.i18n

Internationalization tools to easily implement multi-language applications.

Module Contents

Functions

buildTranslations(pathToFolder)	
translate(key[, fallback])	Tries to translate the given string in the currently selected language.
addTranslation(lang, a[, b])	Adds or updates new translations.
setLanguage(lang)	Sets the current language to lang.
getLanguage()	Returns the current language.

Attributes

_currentLanguage
_currentLanguage
_currentLanguage
_currentLanguage
_runtimeTranslations
_lngMap

flare.i18n.**_currentLanguage**

flare.i18n.**_currentLanguage**

flare.i18n.**_currentLanguage = 'en'**

flare.i18n.**_currentLanguage**

flare.i18n.**_runtimeTranslations**

flare.i18n.**_lngMap**

flare.i18n.**buildTranslations**(*pathToFolder*)

flare.i18n.**translate**(*key*, *fallback=None*, ***kwargs*)

Tries to translate the given string in the currently selected language.

Supports replacing markers (using {markerName} syntax).

Parameters

- **key** – The string to translate
- **fallback** – Return string when no translation is found.

Returns

The translated string

flare.i18n.**addTranslation**(*lang*, *a*, *b=None*)

Adds or updates new translations.

flare.i18n.**setLanguage**(*lang*)

Sets the current language to lang.

flare.i18n.**getLanguage**()

Returns the current language.

flare.icons

Components for displaying icons.

Module Contents

Classes

SvgIcon	A raw, embedded SVG icon-component.
Icon	Icon component with first-letter fallback, normally shown as embedded SVG.
BadgeIcon	A badge icon is an icon-component with a little badge, e.g. a number of new messages or items in the cart or so.

class flare.icons.**SvgIcon**(*value=None*, *fallbackIcon=None*, *title=''*)

Bases: *flare.html5.svg.Svg*

A raw, embedded SVG icon-component.

_leafTag = True

_setValue(*value*)

_setTitle(*val*)

Advisory information associated with the element.

Parameters
val – str

getIcon()

replaceSVG(*icondata*)

requestFallBack(*data*, *status*)

class flare.icons.**Icon**(*value=None*, *fallbackIcon=None*, *title=''*, *classes=[]*)

Bases: *flare.html5.I*

Icon component with first-letter fallback, normally shown as embedded SVG.

_leafTag = True

_setValue(*value*)

_setTitle(*val*)

Advisory information associated with the element.

Parameters
val – str

_setFallback(*val*)

onError()

class flare.icons.**BadgeIcon**(*title=''*, *value=None*, *fallbackIcon=None*, *badge=None*)

Bases: *Icon*

A badge icon is an icon-component with a little badge, e.g. a number of new messages or items in the cart or so.

_setBadge(*badge*)

_getBadge()

flare.ignite

Flare-specific form Widgets with specialized classes and behavior.

Module Contents

Classes

Label
Input
Switch
Check
Radio
Select
Textarea
Progress
Item
Table

class flare.ignite.**Label**(**args*, ***kwargs*)

Bases: *flare.html5.Label*

class flare.ignite.**Input**(**args*, ***kwargs*)

Bases: *flare.html5.Input*

class flare.ignite.**Switch**(**args*, ***kwargs*)

Bases: *flare.html5.Div*

_setChecked(*value*)

_getChecked()

class flare.ignite.**Check**(**args*, ***kwargs*)

Bases: *flare.html5.Input*

class flare.ignite.**Radio**(**args*, ***kwargs*)

Bases: *flare.html5.Div*

class flare.ignite.**Select**(**args*, ***kwargs*)

Bases: *flare.html5.Select*

class flare.ignite.**Textarea**(**args*, ***kwargs*)

Bases: *flare.html5.Textarea*

class flare.ignite.**Progress**(**args*, ***kwargs*)

Bases: *flare.html5.Progress*

class flare.ignite.**Item**(*title=None*, *descr=None*, *className=None*, **args*, ***kwargs*)

Bases: *flare.html5.Div*

class flare.ignite.**Table**(**args*, ***kwargs*)

Bases: *flare.html5.Table*

prepareRow(*row*)

prepareCol(*row*, *col*)

fastGrid(*rows*, *cols*, *createHidden=False*)

flare.input

Input widget with additional event handling.

Module Contents

Classes

Input

class flare.input.**Input**(*type='text'*, *placeholder=None*, *callback=None*, *id=None*, *focusCallback=None*, **args*, ***kwargs*)

Bases: *flare.html5.Input*

onChange(*event*)

onFocus(*event*)

onDetach()

flare.intersectionObserver

Module Contents

Classes

IntersectionObserver	Python wrapper for IntersectionObserver.

class flare.intersectionObserver.**IntersectionObserver**(*callback*, *rootWidget=None*, *rootMargin='0px'*, *threshold=0.2*)

Python wrapper for IntersectionObserver.

Usage: myObserver = IntersectionObserver(myChangeFunction) myObserver.observe(aWidget)

jsObserver

observableWidgets = []

observe(*widget*)

unobserve(*widget*)

flare.log

Generalized Python logging for Pyodide.

Module Contents

Classes

FlareLogRecord	A LogRecord instance represents an event being logged.
JSConsoleHandler	Brings our awesome log messages onto the js console.

Functions

prepareLogger(→ None)	Call this before first usage of logging or getLogger().
getLogger(→ Any)	Creates a child logger of our 'root' logger with a name.

Attributes

loggers

flare.log.**loggers** = []

class flare.log.**FlareLogRecord**(*name*, *level*, *pathname*, *lineno*, *msg*, *args*, *exc_info*, *func=None*, *sinfo=None*, *mergeArgs=False*, ***kwargs*)

Bases: logging.LogRecord

A LogRecord instance represents an event being logged.

LogRecord instances are created every time something is logged. They contain all the information pertinent to the event being logged. The main information passed in is in msg and args, which are combined using str(msg) % args to create the message field of the record. The record also includes information such as when the record was created, the source line where the logging call was made, and any exception information to be logged.

NOTE: This is mostly the same as the original LogRecord. Differences:

- Do not use a single dict as keyword args because pyodites' Proxy objects cannot be used

with isinstance(proxy, collections.abc.Mapping). This will be discussed upstream. * User-supplied arguments to logging messages will not be replaced in message, but will be forwarded to js console via separate arguments.

getMessage() → str

Optionally merge args into message driven by mergeArgs flag in ctor, otherwise this will happen later in js console as objects.

Returns

class flare.log.**JSConsoleHandler**(*stream=None*)

Bases: logging.StreamHandler

Brings our awesome log messages onto the js console.

emit(*record: logging.LogRecord*) → None

Emit a record.

If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using traceback.print_exception and appended to the stream. If the stream has an 'encoding' attribute, it is used to determine how to do the output to the stream.

flare.log.**prepareLogger**(*level: str*, *mergeArgs: bool = False*) → None

Call this before first usage of logging or getLogger().

:param level Log level as str as of all, info, debug, warning, error or critical :param mergeArgs: If True we're merging args into resulting message resulting in possible duplicated output or get the 'raw' message output if False.

flare.log.**getLogger**(*name: str*) → Any

Creates a child logger of our 'root' logger with a name.

Usually it's the __name__ attribute of the module you want to use a logger for.

Parameters

name –

Returns

flare.network

Tools for handling Ajax/fetch-network requests

Module Contents

Classes

DeferredCall	Calls the given function with a fixed delay.
HTTPRequest	Wrapper around XMLHttpRequest.
NetworkService	Generic wrapper around ajax requests.
requestGroup	

Functions

fetch_json(url, callback, **kwargs)	Wrapper that performs a fetch request (with all parameters related to [pyfetch](https://pyodide.org/en/stable/usage/api/python-api.html#pyodide.http.pyfetch).
NiceError(req, code[, params, then])	Displays a descriptive error message using an Alert dialog to the user.
NiceErrorAndThen(function)	Returns a callback which first displays a descriptive error message to the user and then calls another function.
processSkelQueue()	
getUrlHashAsString([urlHash])	
getUrlHashAsObject([urlHash])	
setUrlHash(hash[, param])	

Attributes

skeyRequestQueue

flare.network.**fetch_json**(*url*, *callback*, ***kwargs*)

Wrapper that performs a fetch request (with all parameters related to [pyfetch](https://pyodide.org/en/stable/usage/api/python-api.html#pyodide.http.pyfetch).

Parameters

- **url** – URL to fetch from.

- **then** – Callback for getting the JSON object as parameter, status-code and status-text as parameters. JSON is None in case of an error.
- **catch** – Optional callback for failure, getting the [FetchResponse](https://pyodide.org/en/stable/usage/api/python-api.html#pyodide.http.FetchResponse) as parameter.
- **kwargs** – Any kwargs being passed to pyfetch.

class flare.network.**DeferredCall**(*func*, **args*, ***kwargs*)

Bases: object

Calls the given function with a fixed delay.

This allows assuming that calls to NetworkService are always asynchronous, so its guaranteed that any initialization code can run before the Network-Call yields results.

run()

Internal callback that executes the callback function.

class flare.network.**HTTPRequest**(*method*, *url*, *callbackSuccess=None*, *callbackFailure=None*, *payload=None*, *content_type=None*, *response_type=None*, *asynchronous=True*)

Bases: object

Wrapper around XMLHttpRequest.

onReadyStateChange(**args*, ***kwargs*)

Internal callback.

flare.network.**NiceError**(*req*, *code*, *params=''*, *then=None*)

Displays a descriptive error message using an Alert dialog to the user.

flare.network.**NiceErrorAndThen**(*function*)

Returns a callback which first displays a descriptive error message to the user and then calls another function.

flare.network.**skeyRequestQueue = []**

flare.network.**processSkelQueue**()

class flare.network.**NetworkService**(*module*, *url*, *params*, *successHandler*, *failureHandler*, *finishedHandler*, *modifies*, *secure*, *kickoff*, *group=None*)

Bases: object

Generic wrapper around ajax requests.

Handles caching and multiplexing multiple concurrent requests to the same resource. It also acts as the central proxy to notify currently active widgets of changes made to data on the server.

changeListeners = []

host = ''

prefix = '/json'

defaultFailureHandler

retryCodes

retryMax = 3

retryDelay = 5000

static notifyChange(*module*, ***kwargs*)

Broadcasts a change made to data of module 'module' to all currently registered changeListeners.

Parameters
module (*str*) – Name of the module where the change occured

static registerChangeListener(*listener*)

Registers object 'listener' for change notifications.

'listener' must provide an 'onDataChanged' function accepting one parameter: the name of the module. Does nothing if that object has already registered. :param listener: The object to register :type listener: object

static removeChangeListener(*listener*)

Unregisters the object 'listener' from change notifications.

Parameters
listener (*object*) – The object to unregister. It must be currently registered.

static genReqStr(*params*)

static decode(*req*)

Decodes a response received from the server (ie parsing the json).

Returns
object

static isOkay(*req*)

static urlForArgs(*module*, *path*)

Constructs the final url for that request.

If module is given, it prepends "/prefix" If module is None, path is returned unchanged. :param module: Name of the target module or None :type module: str or None :param path: Path (either relative to 'module' or absolute if 'module' is None :type path: str :returns: str

kickoff()

static request(*module*, *url*, *params=None*, *successHandler=None*, *failureHandler=None*, *finishedHandler=None*, *modifies=False*, *secure=False*, *kickoff=True*, *group=None*)

Performs an AJAX request. Handles caching and security-keys.

Calls made to this function are guaranteed to be async.

Parameters

- **module** (*str or None*) – Target module on the server. Set to None if you want to call anything else
- **url** (*str or None*) – The path (relative to module) or a full url if module is None
- **successHandler** (*callable*) – function beeing called if the request succeeds. Must take one argument (the request).
- **failureHandler** (*callable*) – function beeing called if the request failes. Must take two arguments (the request and an error-code).
- **finishedHandler** (*callable*) – function beeing called if the request finished (regardless wherever it succeeded or not). Must take one argument (the request).
- **modifies** (*bool*) – If set to True, it will automatically broadcast an onDataChanged event for that module.

- **secure** (*bool*) – If true, include a fresh securitykey in this request. Defaults to False.

doFetch(*url*, *params*, *skey*)

Internal function performing the actual AJAX request.

onCompletion(*text*)

Internal hook for the AJAX call.

onError(*text*, *code*)

Internal hook for the AJAX call.

onTimeout(*text*)

Internal hook for the AJAX call.

clear()

onFinished(*success*)

class flare.network.**requestGroup**(*callback=None*)

addRequest(*request*)

call()

onFinished(*success*)

flare.network.**getUrlHashAsString**(*urlHash=None*)

flare.network.**getUrlHashAsObject**(*urlHash=None*)

flare.network.**setUrlHash**(*hash*, *param=None*)

flare.observable

Observed values firing events when changed.

Module Contents

Classes

ObservableValue
StateHandler

class flare.observable.**ObservableValue**(*key*, *value=None*)

Bases: object

value

setValue(*value*)

class flare.observable.**StateHandler**(*initialize=()*, *widget=None*)

updateState(*key*, *value*)

getState(*key*, *empty=None*)

register(*key*, *widget*)

unregister(*key*, *widget*)

flare.popout

Popout menu that is expanded when hovering.

Example:

```html <popout icon="icon-arrowhead-down">

<popout-item @click="onEdit">edit</popout-item> <popout-item @click="onLeave">leave</popout-item> <popout-item @click="onDelete">delete</popout-item>

</popout> ```

#### Module Contents

#### Classes

| | |
|---|---|
| *PopoutItem* | It's an item in a popout menu. |
| *Popout* | Popout menu. |

**class** flare.popout.**PopoutItem**(*\*args*, *appendTo=None*, *style=None*, *\*\*kwargs*)

Bases: *flare.html5.Div*

It's an item in a popout menu.

**style = ['item', 'has-hover']**

**class** flare.popout.**Popout**(*\*args*, *\*\*kwargs*)

Bases: *flare.html5.Div*

Popout menu.

**style = ['popout-opener', 'popout-anchor']**

**_setIcon**(*icon*)

**_getIcon**()

**_setText**(*text*)

**_getText**()
```

flare.popup

Pre-defined dialog widgets for user interaction.

Module Contents

Classes

Popup	
Prompt	
Alert	Just displaying an alerting message box with OK-button.
Confirm	
TextareaDialog	
radioButtonDialog	

class flare.popup.**Popup**(*title=''*, *id=None*, *className=None*, *icon=None*, *enableShortcuts=True*, *closeable=True*, **args*, ***kwargs*)

Bases: *flare.html5.Div*

onAttach()

onDetach()

onDocumentKeyDown(*event*)

close()

onClose()

class flare.popup.**Prompt**(*text*, *value=''*, *successHandler=None*, *abortHandler=None*, *successLbl=None*, *abortLbl=None*, *placeholder=''*, **args*, ***kwargs*)

Bases: *Popup*

onKeyDown(*event*)

onKeyUp(*event*)

onDocumentKeyDown(*event*)

onOkay(**args*, ***kwargs*)

onCancel(**args*, ***kwargs*)

class flare.popup.**Alert**(*msg*, *title=None*, *className=None*, *okCallback=None*, *okLabel=None*, *icon='icon-info'*, *closeable=True*, **args*, ***kwargs*)

Bases: *Popup*

Just displaying an alerting message box with OK-button.

drop()

onOkBtnClick()

onKeyDown(*event*)

class flare.popup.**Confirm**(*question*, *title=None*, *yesCallback=None*, *noCallback=None*, *yesLabel=None*, *noLabel=None*, *icon='icon-question'*, *closeable=True*, **args*, ***kwargs*)

Bases: *Popup*

onKeyDown(*event*)

onDocumentKeyDown(*event*)

drop()

onYesClicked(**args*, ***kwargs*)

onNoClicked(**args*, ***kwargs*)

class flare.popup.**TextareaDialog**(*text*, *value=''*, *successHandler=None*, *abortHandler=None*, *successLbl=None*, *abortLbl=None*, **args*, ***kwargs*)

Bases: *Popup*

onDocumentKeyDown(*event*)

onOkay(**args*, ***kwargs*)

onCancel(**args*, ***kwargs*)

class flare.popup.**radioButtonDialog**(*title*, *radioValues: list*, *radioButtonGroupName='radioButtonGroup'*, *checkedValue=None*, *icon='icon-question'*, *closeable=True*, *successHandler=None*, *abortHandler=None*, *successLbl=None*, *abortLbl=None*, **args*, ***kwargs*)

Bases: *Popup*

onOkay(**args*, ***kwargs*)

onCancel(**args*, ***kwargs*)

flare.priorityqueue

Select object generators by priority.

This is used when implementing pluggable features, which can optionally be registered for specific use-cases.

Module Contents

Classes

PriorityQueue

class flare.priorityqueue.**PriorityQueue**

Bases: object

insert(*priority*, *validateFunc*, *generator*)

select(**args*, ***kwargs*)

flare.safeeval

Here we are trying to provide an secure and safe space for evaluate simple python expressions on some 'data'.

If you only need a oneshot evaluation, you call safeEval and enjoy the result. Otherwise call first compile to get the ast representation and execute that compiled expression multiple times with different data. A plain instance of SafeEval without allowedCallables argument will not accept any method/function like call on execution

Module Contents

Classes

SafeEval	Safely evaluate an expression from an untrusted party.

class flare.safeeval.**SafeEval**(*allowedCallables: None | Dict[str, Any] = None*)

Safely evaluate an expression from an untrusted party.

_BoolOp(*node*, *names*)

Handling ast.BoolOp in a Pythonic style.

callNode(*node: ast.Call*, *names: Dict[str, Any]*) → Any

Evaluates the call if present in allowed callables.

Parameters

- **node** – The call node to evaluate
- **names** – a mapping of local objects which is used as 'locals' namespace

Returns

If allowed to evaluate the node, its result will be returned

compareNode(*node: ast.Compare*, *names: Dict[str, Any]*) → bool

Evaluates an 'if' expression.

These are a bit tricky as they can have more than two operands (eg. "if 1 < 2 < 3")

Parameters

- **node** – The compare node to evaluate
- **names** – a mapping of local objects which is used as 'locals' namespace

listNode(*node*, *names*)

execute(*node: [str, ast.AST]*, *names: Dict[str, Any]*) → Any

Evaluates the current node with optional data.

Parameters

- **node** – The compare node to evaluate
- **names** – a mapping of local objects which is used as 'locals' namespace

Returns
whatever the expression wants to return

compile(*expr: str*) → ast.AST

Compiles a python expression string to an ast.

Afterwards you can use execute to run the compiled ast with optional data. If you only want to run a 'oneshot' expression feel free to use our safeEval method.

Parameters
expr – the expression to compile

Returns
the ready to use ast node

safeEval(*expr: str*, *names: Dict[str, Any]*) → Any

Safely evaluate an expression.

If you want to evaluate the expression multiple times with different variables use compile to generate the AST once and call execute for each set of variables.

Parameters

- **expr** – the string to compile and evaluate
- **names** – a mapping of local objects which is used as 'locals' namespace

Returns
the result of evaluation of the expression with env provided by names

flare.utils

Utility functions.

Module Contents

Functions

unescape(val[, maxLength])	Unquotes several HTML-quoted characters in a string.
doesEventHitWidgetOrParents(event, widget)	Test if event 'event' hits widget 'widget' (or *any* of its parents).
doesEventHitWidgetOrChildren(event, widget)	Test if event 'event' hits widget 'widget' (or *any* of its children).
textToHtml(node, text)	Generates html nodes from text by splitting text into content and into line breaks html5.Br.
parseInt(s[, ret])	Parses a value as int.
parseFloat(s[, ret])	Parses a value as float.
createWorker(pythonCode[, callback, errorCallback, ...])	Generates and starts a Pyodide Webworker.

flare.utils.**unescape**(*val*, *maxLength=0*)

Unquotes several HTML-quoted characters in a string.

Parameters

- **val** (*str*) – The value to be unescaped.
- **maxLength** (*int*) – Cut-off after maxLength characters. A value of 0 means "unlimited". (default)

Returns
The unquoted string.

Return type
str

flare.utils.**doesEventHitWidgetOrParents**(*event*, *widget*)

Test if event 'event' hits widget 'widget' (or *any* of its parents).

flare.utils.**doesEventHitWidgetOrChildren**(*event*, *widget*)

Test if event 'event' hits widget 'widget' (or *any* of its children).

flare.utils.**textToHtml**(*node*, *text*)

Generates html nodes from text by splitting text into content and into line breaks html5.Br.

Parameters

- **node** – The node where the nodes are appended to.
- **text** – The text to be inserted.

flare.utils.**parseInt**(*s*, *ret=0*)

Parses a value as int.

flare.utils.**parseFloat**(*s*, *ret=0.0*)

Parses a value as float.

flare.utils.**createWorker**(*pythonCode*, *callback=None*, *errorCallback=None*, *context={}*)

Generates and starts a Pyodide Webworker.

def calllog(txt=None):
result = txt.data.to_py() if "error" in result:

print(result["error"])

if "msg" in result:
print(result["msg"])

code="'

import statistics,time from js import self as js_self for i in range(0,100):

js_self.postMessage("POST %s"%i)

"'

createWorker(code,calllog,calllog)

context can take variables, these are like global startparameters

Package Contents

Classes

Cache	

Functions

updateConf(other)	Merges other into conf.
_html5WidgetSetHidden(widget, hidden)	
_html5WidgetGetHidden(widget)	
_html5WidgetSetDisabled(widget, disabled)	
loadProjectConf(aconf)	Update the default flare config with project configuration.
bindApp(app, injectdata)	Add the app instance as "app" to global config.

Attributes

conf	

class flare.**Cache**

Bases: object

updateStructure(*module*, *structure*)

update(*module*, *key*, *data*, *structure=None*)

lookup(*module*, *key='current'*)

struct(*module*)

start(*plan*, *finishHandler=None*, *failureHandler=None*)

finish(*plan*)

require(**args*)

invalidate(**args*)

onDataChanged(*module*, *key=None*, ***kwargs*)

request(**args*, *finishHandler=None*, *failureHandler=None*)

flare.**conf**

flare.**updateConf**(*other: Dict*)

Merges other into conf.

flare.**_html5WidgetSetHidden**(*widget*, *hidden*)

flare.**_html5WidgetGetHidden**(*widget*)

flare.**_html5WidgetSetDisabled**(*widget*, *disabled*)

flare.**loadProjectConf**(*aconf*)

Update the default flare config with project configuration.

flare.**bindApp**(*app*, *injectdata*)

Add the app instance as "app" to global config.

1.4.2 webworker_scripts

WARNING! THIS SCRIPTS ARE USED IN A SANDBOX SO ALL DEPENDENCIES SHOULD BE HANDELED HERE!

THIS USES PYODIDE V0.17!

Module Contents

Classes

weblog

Attributes

log

class webworker_scripts.**weblog**

static info(*text*)

static warn(*text*)

static error(*text*)

webworker_scripts.**log**

PYTHON MODULE INDEX

f

w

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z