

# Contents

<b>1</b>	<b>Yaji Manual</b>	<b>1</b>
1.1	License	1
1.2	About	1
1.3	Up-to-date?	1
1.4	Dependencies	1
1.5	A word about the documentation structure	1
1.6	Overview	2
1.6.1	Hello World	2
1.6.2	How applications are shown	2
1.6.3	Views run decoupled from application code	2
1.6.4	Sample applications	3
1.6.5	A typical Yaji application	3
1.6.6	Application starting and stopping	4
1.6.7	Correctly importing it	5
1.7	Basic user interfaces	5
1.7.1	setbodyleft and setbodyright	5
1.7.2	View specifications	5
1.7.2.1	Layouts	6
1.7.2.2	Scroll views	6
1.7.2.3	Sizing	6
1.7.3	Configuring the main view itself	7
1.7.3.1	Header text	7
1.7.3.2	Icons	7

1.7.4	Sidebar and head control . . . . .	7
1.7.5	Finding more information in the API references . . . . .	7
1.7.5.1	Clove . . . . .	7
1.8	Data sources and data bindings . . . . .	8
1.8.1	Server and local data sources . . . . .	8
1.8.2	Data bindings . . . . .	9
1.8.2.1	Direction . . . . .	9
1.8.2.2	Converters . . . . .	9
1.8.3	Data sources beyond bindings . . . . .	9
1.9	Event bindings . . . . .	10
1.9.1	bindevent . . . . .	10
1.9.2	BackendFunction . . . . .	10
1.9.3	BrowserFunction . . . . .	10
1.9.4	Event data . . . . .	10
1.10	Request handlers . . . . .	11
1.10.1	The ‘_do_’ prefix . . . . .	11
1.10.2	URL mapping . . . . .	11
1.10.3	Downstream execution . . . . .	11
1.11	Menus . . . . .	12
1.11.1	Actions . . . . .	12
1.11.2	Submenus . . . . .	12
1.11.3	Separators . . . . .	12
1.12	User feedback . . . . .	12
1.13	Dialogs . . . . .	13
1.14	Internationalization . . . . .	13
1.14.1	Translations . . . . .	13
1.15	Application stopping . . . . .	13
1.16	Forcefully keeping a browser view open . . . . .	14
1.17	Closed notification . . . . .	14
1.18	Shutdown dialog . . . . .	14

1.19 Browser side . . . . .	14
1.19.1 Adding browser side resources . . . . .	14
1.19.2 Custom widgets . . . . .	15
1.19.3 Yaji browser side API . . . . .	15
1.19.3.1 yaji.ready . . . . .	15
1.19.3.2 Ajax . . . . .	15
1.19.3.3 Client events . . . . .	16
1.19.3.4 AppConfig . . . . .	16
1.19.3.5 Curtains . . . . .	17
1.19.4 JSON serialization . . . . .	17
1.20 Middleware . . . . .	17
1.21 Application takeover . . . . .	17
1.22 Storing applications statically . . . . .	18
1.23 Logging . . . . .	18
1.24 Class Hierarchy . . . . .	18
1.25 Class List . . . . .	19
1.26 yaji.app.Application.__ComputeContentMiddleware Class Reference . . . . .	20
1.26.1 Detailed Description . . . . .	21
1.26.2 Member Function Documentation . . . . .	21
1.26.2.1 __callhandler() . . . . .	21
1.26.2.2 request_processor() . . . . .	22
1.27 yaji.request.Request.__InRequest Class Reference . . . . .	22
1.27.1 Detailed Description . . . . .	22
1.28 yaji.app.Application.__PostParamsMiddleware Class Reference . . . . .	22
1.28.1 Detailed Description . . . . .	23
1.28.2 Member Function Documentation . . . . .	23
1.28.2.1 request_processor() . . . . .	23
1.29 yaji.app.Application.__SetStopImplicitlyWhenBrowserClosed Class Reference . . . . .	23
1.29.1 Detailed Description . . . . .	23
1.30 yaji.app.Application.__YajiHTTPRequestHandler Class Reference . . . . .	24

1.30.1 Detailed Description . . . . .	24
1.31 yaji.app.Application.__YajiTCPServer Class Reference . . . . .	24
1.31.1 Detailed Description . . . . .	24
1.32 yaji.gui._AbstractContainer Class Reference . . . . .	24
1.32.1 Detailed Description . . . . .	25
1.32.2 Constructor & Destructor Documentation . . . . .	25
1.32.2.1 __init__() . . . . .	25
1.33 yaji.gui.View._DataBinding Class Reference . . . . .	25
1.33.1 Detailed Description . . . . .	25
1.33.2 Constructor & Destructor Documentation . . . . .	26
1.33.2.1 __init__() . . . . .	26
1.34 yaji.gui.View._EventBinding Class Reference . . . . .	26
1.34.1 Detailed Description . . . . .	26
1.34.2 Constructor & Destructor Documentation . . . . .	26
1.34.2.1 __init__() . . . . .	26
1.35 yaji.guibase.AbstractAction Class Reference . . . . .	27
1.35.1 Detailed Description . . . . .	27
1.35.2 Constructor & Destructor Documentation . . . . .	27
1.35.2.1 __init__() . . . . .	27
1.36 yaji.guibase.AbstractFunction Class Reference . . . . .	28
1.36.1 Detailed Description . . . . .	28
1.37 yaji.guibase.Action Class Reference . . . . .	28
1.37.1 Detailed Description . . . . .	29
1.37.2 Constructor & Destructor Documentation . . . . .	29
1.37.2.1 __init__() . . . . .	29
1.38 yaji.appconfig.AppConfig Class Reference . . . . .	29
1.38.1 Detailed Description . . . . .	30
1.38.2 Member Function Documentation . . . . .	30
1.38.2.1 __markinitialized() . . . . .	30
1.38.2.2 getconfig() . . . . .	30

1.38.2.3	<a href="#">getConfigs()</a>	31
1.38.2.4	<a href="#">setconfig()</a>	31
1.39	<a href="#">yaji.app.Application Class Reference</a>	31
1.39.1	<a href="#">Detailed Description</a>	35
1.39.2	<a href="#">Constructor &amp; Destructor Documentation</a>	35
1.39.2.1	<a href="#">__init__()</a>	35
1.39.3	<a href="#">Member Function Documentation</a>	36
1.39.3.1	<a href="#">__create_datastore_helper()</a>	36
1.39.3.2	<a href="#">__get_content()</a>	36
1.39.3.3	<a href="#">__get_datastore_helper()</a>	37
1.39.3.4	<a href="#">__get_requesthandler_for_url()</a>	37
1.39.3.5	<a href="#">_browser_was_reopened()</a>	38
1.39.3.6	<a href="#">_callhandler()</a>	38
1.39.3.7	<a href="#">_findhandler()</a>	38
1.39.3.8	<a href="#">_get_requestparam_type_converter()</a>	39
1.39.3.9	<a href="#">_get_rootpagecontent()</a>	39
1.39.3.10	<a href="#">_json_make_serializable()</a>	39
1.39.3.11	<a href="#">_middlewares()</a>	39
1.39.3.12	<a href="#">_openbrowser()</a>	40
1.39.3.13	<a href="#">_staticfile_path_to_abspath()</a>	40
1.39.3.14	<a href="#">_tryclosebrowser()</a>	40
1.39.3.15	<a href="#">add_clientscript()</a>	41
1.39.3.16	<a href="#">add_clientstyle()</a>	41
1.39.3.17	<a href="#">add_middleware()</a>	41
1.39.3.18	<a href="#">add_requestparam_type_converter()</a>	42
1.39.3.19	<a href="#">add_staticfile_location()</a>	42
1.39.3.20	<a href="#">add_translations()</a>	42
1.39.3.21	<a href="#">binddata()</a>	43
1.39.3.22	<a href="#">bindevent()</a>	43
1.39.3.23	<a href="#">bindlocal()</a>	43

1.39.3.24 bindserver()	44
1.39.3.25 create_clientlocal_datastore()	44
1.39.3.26 create_datastore()	45
1.39.3.27 create_dialog()	45
1.39.3.28 do_stop_implicitly_when_browser_closed()	45
1.39.3.29 enable_authentication()	46
1.39.3.30 get_clientlocal_datastore()	46
1.39.3.31 get_datastore()	46
1.39.3.32 get_translations()	47
1.39.3.33 get_translations_stringnames()	47
1.39.3.34 icon()	47
1.39.3.35 isstaticapplication()	48
1.39.3.36 mainview_icon()	48
1.39.3.37 onbrowserreopened()	48
1.39.3.38 oninitialize()	48
1.39.3.39 onopenbrowsererror()	48
1.39.3.40 onopenbrowserinformationoutput()	49
1.39.3.41 onprocessrequesterror()	49
1.39.3.42 onunhandledclienterror()	49
1.39.3.43 setactions()	50
1.39.3.44 setbodyleft()	50
1.39.3.45 setbodyright()	50
1.39.3.46 setheadcontrol()	51
1.39.3.47 setsidebar()	51
1.39.3.48 setsplitterposition()	51
1.39.3.49 showonly()	52
1.39.3.50 start()	52
1.39.3.51 stop()	52
1.39.3.52 storeasstaticapplication()	53
1.39.3.53 triggerevent()	54

1.39.3.54	<a href="#">try_addclientresources_from_default_location()</a>	54
1.39.3.55	<a href="#">userfeedback()</a>	54
1.40	<a href="#">yaji.auth.AuthMiddleware Class Reference</a>	55
1.40.1	<a href="#">Detailed Description</a>	55
1.40.2	<a href="#">Constructor &amp; Destructor Documentation</a>	55
1.40.2.1	<a href="#">__init__()</a>	55
1.40.3	<a href="#">Member Function Documentation</a>	56
1.40.3.1	<a href="#">_authenticate_by_password()</a>	56
1.40.3.2	<a href="#">request_processor()</a>	56
1.41	<a href="#">yaji.guibase.BackendFunction Class Reference</a>	56
1.41.1	<a href="#">Detailed Description</a>	57
1.41.2	<a href="#">Constructor &amp; Destructor Documentation</a>	57
1.41.2.1	<a href="#">__init__()</a>	57
1.42	<a href="#">yaji.guibase.BindDirection Class Reference</a>	57
1.42.1	<a href="#">Detailed Description</a>	57
1.42.2	<a href="#">Member Data Documentation</a>	58
1.42.2.1	<a href="#">Bidirectional</a>	58
1.42.2.2	<a href="#">ToDatasource</a>	58
1.42.2.3	<a href="#">ToWidget</a>	58
1.43	<a href="#">yaji.guibase.BrowserFunction Class Reference</a>	58
1.43.1	<a href="#">Detailed Description</a>	59
1.43.2	<a href="#">Constructor &amp; Destructor Documentation</a>	59
1.43.2.1	<a href="#">__init__()</a>	59
1.44	<a href="#">yaji.browser.BrowserHook Class Reference</a>	59
1.44.1	<a href="#">Detailed Description</a>	59
1.44.2	<a href="#">Constructor &amp; Destructor Documentation</a>	60
1.44.2.1	<a href="#">__init__()</a>	60
1.45	<a href="#">yaji.guibase.BrowserSideDatasource Class Reference</a>	60
1.45.1	<a href="#">Detailed Description</a>	60
1.46	<a href="#">yaji.datastore.DataStore Class Reference</a>	60

1.46.1 Detailed Description . . . . .	61
1.46.2 Member Function Documentation . . . . .	61
1.46.2.1 add_onchanged_handler() . . . . .	62
1.46.2.2 appendcolumn() . . . . .	62
1.46.2.3 appendrow() . . . . .	62
1.46.2.4 columncount() . . . . .	63
1.46.2.5 getvalue() . . . . .	63
1.46.2.6 insertcolumn() . . . . .	63
1.46.2.7 insertrow() . . . . .	64
1.46.2.8 removecolumn() . . . . .	64
1.46.2.9 removerow() . . . . .	64
1.46.2.10 rowcount() . . . . .	65
1.47 yaji.gui.Dialog Class Reference . . . . .	65
1.47.1 Detailed Description . . . . .	65
1.47.2 Constructor & Destructor Documentation . . . . .	66
1.47.2.1 __init__() . . . . .	66
1.47.3 Member Function Documentation . . . . .	66
1.47.3.1 add_closed_handler() . . . . .	66
1.47.3.2 dialogid() . . . . .	66
1.48 yaji.dialogcontroller.DialogController Class Reference . . . . .	67
1.48.1 Detailed Description . . . . .	67
1.48.2 Member Function Documentation . . . . .	67
1.48.2.1 close_dialog() . . . . .	67
1.48.2.2 get_dialog_by_id() . . . . .	67
1.48.2.3 show_dialog() . . . . .	68
1.49 yaji.core.Directories Class Reference . . . . .	68
1.49.1 Detailed Description . . . . .	68
1.50 yaji.gui.Grid Class Reference . . . . .	68
1.50.1 Detailed Description . . . . .	69
1.50.2 Constructor & Destructor Documentation . . . . .	69



1.50.2.1	<code>__init__()</code>	69
1.51	<code>yaji.gui.HorizontalStack</code> Class Reference	69
1.51.1	Detailed Description	70
1.51.2	Constructor & Destructor Documentation	70
1.51.2.1	<code>__init__()</code>	70
1.52	<code>yaji.guibase.Icon</code> Class Reference	70
1.52.1	Detailed Description	70
1.52.2	Constructor & Destructor Documentation	70
1.52.2.1	<code>__init__()</code>	70
1.53	<code>yaji.core.Log</code> Class Reference	71
1.53.1	Detailed Description	71
1.54	<code>yaji.core.Middleware</code> Class Reference	71
1.54.1	Detailed Description	72
1.54.2	Member Function Documentation	72
1.54.2.1	<code>request_processor()</code>	72
1.55	<code>yaji.pyhtmlview.PyHtmlView</code> Class Reference	72
1.55.1	Detailed Description	73
1.55.2	Member Function Documentation	73
1.55.2.1	<code>openview()</code>	73
1.56	<code>yaji.request.Request</code> Class Reference	73
1.56.1	Detailed Description	75
1.56.2	Constructor & Destructor Documentation	75
1.56.2.1	<code>__init__()</code>	75
1.56.3	Member Function Documentation	75
1.56.3.1	<code>_in_request()</code>	75
1.56.3.2	<code>_set_lparam()</code>	75
1.56.3.3	<code>_set_runs_downstream()</code>	76
1.56.3.4	<code>get_param()</code>	76
1.56.3.5	<code>get_param_as_list()</code>	76
1.56.3.6	<code>get_response_header()</code>	77

1.56.3.7	<a href="#">params()</a>	77
1.56.3.8	<a href="#">runs_downstream()</a>	77
1.56.3.9	<a href="#">set_response_header()</a>	77
1.56.3.10	<a href="#">skip_processing()</a>	78
1.56.3.11	<a href="#">uieventdata()</a>	78
1.57	<a href="#">yaji.reqhandler.RequestHandler Class Reference</a>	78
1.57.1	<a href="#">Detailed Description</a>	79
1.57.2	<a href="#">Member Function Documentation</a>	79
1.57.2.1	<a href="#">__compile_re_for_urlpattern()</a>	79
1.57.2.2	<a href="#">for_urls()</a>	79
1.57.2.3	<a href="#">run_downstream()</a>	80
1.57.2.4	<a href="#">with_param_type()</a>	80
1.58	<a href="#">yaji.gui.ScrollView Class Reference</a>	80
1.58.1	<a href="#">Detailed Description</a>	81
1.58.2	<a href="#">Constructor &amp; Destructor Documentation</a>	81
1.58.2.1	<a href="#">__init__()</a>	81
1.59	<a href="#">yaji.guibase.Separator Class Reference</a>	81
1.59.1	<a href="#">Detailed Description</a>	81
1.60	<a href="#">yaji.gui.Spacer Class Reference</a>	82
1.60.1	<a href="#">Detailed Description</a>	82
1.61	<a href="#">yaji.guibase.Submenu Class Reference</a>	82
1.61.1	<a href="#">Detailed Description</a>	82
1.61.2	<a href="#">Constructor &amp; Destructor Documentation</a>	82
1.61.2.1	<a href="#">__init__()</a>	82
1.62	<a href="#">yaji.i18n.TrStr Class Reference</a>	83
1.62.1	<a href="#">Detailed Description</a>	83
1.62.2	<a href="#">Constructor &amp; Destructor Documentation</a>	83
1.62.2.1	<a href="#">__init__()</a>	83
1.63	<a href="#">yaji.userfeedback.UserFeedbackController Class Reference</a>	84
1.63.1	<a href="#">Detailed Description</a>	84

1.63.2	Member Function Documentation	84
1.63.2.1	<code>_interpret_response()</code>	84
1.63.2.2	<code>choicedialog()</code>	85
1.63.2.3	<code>filesystemdialog()</code>	85
1.63.2.4	<code>inputdialog()</code>	85
1.63.2.5	<code>make_raw_request()</code>	86
1.63.2.6	<code>messagedialog()</code>	86
1.63.2.7	<code>multilineinputdialog()</code>	87
1.63.2.8	<code>passworddialog()</code>	87
1.64	<code>yaji.datastore.DataStore.ValuePointer</code> Class Reference	87
1.64.1	Detailed Description	87
1.65	<code>yaji.gui.VerticalStack</code> Class Reference	88
1.65.1	Detailed Description	88
1.65.2	Constructor & Destructor Documentation	88
1.65.2.1	<code>__init__()</code>	88
1.66	<code>yaji.gui.View</code> Class Reference	88
1.66.1	Detailed Description	89
1.66.2	Constructor & Destructor Documentation	89
1.66.2.1	<code>__init__()</code>	89
1.67	<code>yaji.guibase.Visibility</code> Class Reference	90
1.67.1	Detailed Description	90
1.67.2	Member Data Documentation	90
1.67.2.1	<code>Invisible</code>	90
1.67.2.2	<code>InvisibleCollapsed</code>	91
1.67.2.3	<code>Visible</code>	91
1.68	<code>yaji.gui.Wrap</code> Class Reference	91
1.68.1	Detailed Description	91
1.68.2	Constructor & Destructor Documentation	91
1.68.2.1	<code>__init__()</code>	91



# Chapter 1

## Yaji Manual

### 1.1 License

Yaji is written by Josef Hahn under the terms of the GPLv3 or higher.

Please read the `LICENSE` file from the package and the [Dependencies](#) section for included third-party stuff.

### 1.2 About

Lorem ipsum dolor sit amet, TODO.

### 1.3 Up-to-date?

Are you currently reading from another source than the homepage? Are you in doubt if that place is up-to-date? If yes, you should visit <https://pseudopolis.eu/wiki/pino/projs/Yaji> and check that. You are currently reading the manual for version 0.1.

### 1.4 Dependencies

There is an external part which is used by Yaji. Many thanks to the project and all participants.

`PyQt5` incl. `WebEngine` *optional*: otherwise you will need a usual web browser

### 1.5 A word about the documentation structure

The following sections in this text (`README.pdf`) are the main part of the Yaji documentation. It is followed by an API reference. There are also some more files, which will be mentioned here later.

## 1.6 Overview

### 1.6.1 Hello World

```
class MyApplication(yaji.Application):  
  
    def __init__(self):  
        super().__init__()  
        self.setbodyleft(yaji.View(label="Hallo Welt."))
```

Yaji is a Python package allowing to implement graphical user interfaces that are accessible by just web browsers. However, it is not a framework for websites. It would lack some features for that on the one hand, while it has some features that would not make sense for that on the other hand. Instead, the entire web server comes up when an end user starts a Yaji-enabled application. That application steers one single user interface, as a usual program with a local user interface would do. The web server is bound to that particular process with all that user interface state, which does not make it a good choice for a website.

Compared to a local user interface, it is a bit more tricky to implement, but it allows to access the user interface easily via the network. It might be not worth taking the additional effort, if this network capability is not a requirement for you.

Writing browser side code is possible, but not needed. Yaji allows implementing rich user interfaces by just Python code. It also allows direct access on browser side for advanced cases, e.g. by additional JavaScript code, but a good practice is to use that in a minimal way (usually not at all).

### 1.6.2 How applications are shown

By default, a Yaji user interface tries to open a window on your local desktop for your user interface on start. This follows the assumption that local access is the preferred way.

Internally, it uses the WebEngine in PyQt for showing a window that mostly is a tiny browser, just without all the controls around the actual website. If that fails, it tries to open itself in the local browsers.

Furthermore, if it succeeds or not, there is a tiny local web server, which usual web browsers can connect to. Of course, depending on network structure and firewall rules, this can also be a web browser on another machine. It is possible to connect to the user interface with multiple browsers in parallel, even though this is only useful in a few scenarios. Depending on how it is set up by the application around it, it may or may not be possible to disconnect from it completely with all browsers, and to reconnect back much later without loss of state data.

Note: Setting the environment variable `YAJI_CALLCMD` to a custom command line causes a Yaji application to open new browser views by executing that instead of its default routine. This command line should contain `u` for taking the URL to open.

### 1.6.3 Views run decoupled from application code

There is one challenge with this approach by nature that leads to the additional trickyness in comparison to a classic local user interface. In the local case, there is one "UI" thread with a main loop, which all user interface code gets piped through in a mostly first-in-first-out way. This makes it easy to keep things in a consistent state: Whenever any interaction by the user occurs, program code can directly compute a new user interface state, like showing/hiding or enabling/disabling some parts of the user interface depending on some conditions. That typically happens in no time the user could realize. The user is also not able to intervent meanwhile nevertheless, even if computation takes longer, since all further user interactions just take a place in the queue of the main loop until the computation finishes. For example, there is no way to click on a button while a computation is going on which eventually disables it (unless e.g. some multi-threaded application code is explicitly doing it this way).

A Yaji application has to implement such situations differently, because that single main loop does not exist. The application (Python) code and the presentation of the user interface run in a much less encoupled way. The user interface will typically not block while application code is running. Furthermore, application code that modifies the user interface will potentially finish before the view actually reflects that (which is quite understandable because there can be from no to  $n$  views, e.g. by opening it in multiple browser tabs). This makes it harder to keep the user interface in a consistent and 'correct' state (whatever that means for a particular user interface logic). For example, a user could interact in some way that triggers the application code to disable a particular button, but then click on that button before it actually becomes disables. In fact, there are two asynchronous transitions in this case: The user interaction triggering the application code and the application code triggering the button state change. There are some ways to work around that, like curtains, or implementing some pieces of JavaScript code, which will be mentioned in detail later. All of them come with some additional efforts to spend.

### 1.6.4 Sample applications

For many aspects of Yaji there are some small sample applications. The following text will explicitly refer to some of them, but it is a good general advice to study them for better understanding. However, there are some remarks one should keep in mind while reading them:

- They tend to keep things simple and are generally not production ready,
- they typically do completely useless stuff, just for showing a particular feature or aspect of Yaji, and
- they sometimes solve a problem in a 'creative' way, while one would solve it in completely different ways for real code,
- they sometimes play around with different ways of doing the same thing (e.g. writing a data structure in different ways) if that is worth showing.
- However, many of them show only a subset of the available functionality.

The sample applications are located in `./_meta/sampleapps`. Each `.py` file is for one sample application (but `__sampleapp.py` which has a special internal meaning). They are directly executable from command line. Some of them come with additional files, e.g. a browser side JavaScript, which are important parts of the application. For a `foo.py` this would be `__foo.js`. They are automatically loaded by Yaji without an explicit reference to them somewhere in the `.py` code.

### 1.6.5 A typical Yaji application

A typical Yaji application potentially does the same things as local graphical applications do. It implements a user interface based on Yaji typically in the following way (just trying to give an idea about many different things; it is *not* required to understand each point in detail now):

- It implements a subclass of [yaji.app.Application](#). This can either be a real application, or just another abstract enhancement for implementing by deeper subclasses. The following assumes a full, non-abstract implementation.
  - Inside the constructor, it configures the user interface by assembling complete forms from basic parts like buttons, labels and layouts, and assigning that to the main view.
    - \* Those views are *connected* to the application code by data bindings and event bindings. The former keep data on view side in sync with some data on application code side, while the latter trigger some code execution when some event occurs in the view (e.g. the user triggered a button).
    - \* Application code only sets completely assembled views and cannot change only some properties of some subelements or insert/remove/replace subelements of a view; with some important remarks:

- Data bindings are an exception to that rule, since keeping a property of a widget in sync with application data is exactly their primary duty. Furthermore, since `visibility` and `enabled` are also just usual properties of each widget, many kinds of dynamic changes in the user interface structure can be realized by data bindings to them.
- The usual main view is split into a left and a right column, so there are at least two separate views one can set independently from the other. There are also some more situations which involve building a view, like dialogs.
- Setting the main views to new ones at some times during the application execution is possible, so structural changes of the user interface during application execution are possible by completely rebuilding a view each time.
- It provides request handlers for processing all kinds of requests by the view side. For now, those are the handlers used in the event bindings (see above), but they can have other uses. So they react on some kinds of user interactions like triggered buttons or menu items (while *data* bindings are only internally related to that).
  - \* A request handler is a method of your subclass, usually named in a particular way, and annotated in a particular way. It contains the application specific handler code for that event.
- It optionally overrides some methods in order to customize some low-level behavior, e.g. as one piece of the integration into something bigger.
- It optionally provides some browser side resources for advanced cases, e.g. a `.css` style file or a `.js` JavaScript. If there is a file like `__foo.js` for an application `foo.py`, Yaji will automatically load it without any code needed.
  - Browser side code can implement more complex and more tailor-made widget classes that can be used by the application code instead of simpler widgets like buttons and labels. They can react to the user in a more direct way than application code can, so this is needed in some cases.
  - It can do everything user interface related that application code can do, and much more, in a more direct way, meaning: next to the user, but far away from the application code side instead.
  - It has access to data sources, but as mentioned earlier, there is will be a time gap between updates on application code side and browser side.
  - Communication with the application code side is based on Ajax calls and/or higher-level mechanisms based on that.

The following chapters will go more into detail.

## 1.6.6 Application starting and stopping

At some time typically during startup, the application code creates an instance of that subclass and calls `yaji.Application.start()` on it. Eventually, there should be some code somewhere which triggers to stop the application again. A later chapter will go more into details, but the following code is one example of doing both:

```
from yaji import * # do not use * in your code

class MyApplication(Application):
    def __init__(self):
        super().__init__()
        self.setbodyleft(View('Button', label="Stop",
                               OnClicked=self.bindevent(BackendFunction('stop'))))

    @RequestHandler.for_urls(auto=True)
    def _do_stop(self):
        self.stop()

if __name__ == '__main__':
    MyApplication().start()
```



### 1.6.7 Correctly importing it

There is nothing one can particularly do wrong with importing `yaji`. However, there are different ways, of doing it, and mixing them can confuse at first. This documentation does mix it occasionally for different reasons, so you should be aware of it.

The `yaji` package is divided into some submodules, each of them having some direct members, like class or function definitions. The main application base class is `Application` inside the `app` submodule. So, after `import yaji.app` its qualified name is `yaji.app.Application`. The root package itself also includes all members which are typically used by outside code. This is for convenience as it reduces the amount of typing and/or thinking about the correct imports to make, but it can be confusing when seeing `Yaji` class or function names. After just `import yaji` the same class has the qualified name `yaji.Application`. Due to technical reasons it has only this name and not the longer one at all (of course it is possible to make both import statements if that is needed e.g. during some code refactorings).

So, whenever a short name is mentioned somewhere, there is also a longer name (which also includes the submodule name) that is pointing to the same thing. The API reference here lists all module members with their long names, so one way to find out a long name is to search for the very last segment of the name in the API reference.

For keeping lines shorter, most examples in this text assume `from yaji import ...` and would just refer to e.g. `Application`.

## 1.7 Basic user interfaces

### 1.7.1 setbodyleft and setbodyright

Application code sets up the user interface inside the constructor and optionally any time later, by building complex user interface structures from basic ones like buttons or labels, and layout elements, which align widgets (or other layout elements) inside it in a particular way. It applies that by calling `yaji.Application.setbodyleft()` and/or `yaji.Application.setbodyright()`, like the following:

```
def __init__(self):
    ...
    self.setbodyleft(View('Label', label="Foo"))
```

This shows a widget of class `Label` with some properties, precisely a label showing the text "Foo". As the method name suggests, there is a distinction between left and right. This does not mean anything as long as either only `setbodyleft` or `setbodyright` is used. When both are set in parallel, the view splits into a left and a right side. There is a splitter dividing both sides, which the user can move.

### 1.7.2 View specifications

The parameters passed to `yaji.Application.setbodyleft()` and some other functions are instances of `yaji.gui.View`. They completely specify a user interface view with simple elements like buttons, and stack them together in some potentially complex ways with *layouts*. Each subpart within this structure, i.e. each simple elements as well as each layout is a `View`, up to the final complete specification. Each `View` instance has a *class*, which is a string like `'Button'`, `'Label'` or `'HorizontalStack'` and some property assignments. Those assignments can be just any values of types that make sense (e.g. strings for label texts). They can also be data bindings, and for events, they are usually event bindings. The following example shows another view configuration, still without layouts, and kind of nonsensical:

```
View('Button', label=self.binddata('buttonlabeldatasource'), enabled=False,
     OnClicked=self.bindevent(BackendFunction('dosomething')))
```

Data bindings and event bindings are explained in detail later.

There is a shortcut for label, which you could find used in other applications like the samples. Whenever a `label` property is given but no class, `"Label"` class is used. So, the following lines each specify the same thing:

```
View('Label', label="Foo")
View(label="Foo")
```

### 1.7.2.1 Layouts

Building complex views out of simple pieces stacked together uses the same mechanism, with e.g. a `'HorizontalStack'` class widget with one of its properties used for assigning children widgets. However, for builtin layouts, there is a slightly different, shorter notation, since there are special classes available for them. The following example stacks some pieces together:

```
self.setbodyleft(
    VerticalStack(
        View('Label', label="Foo"),
        HorizontalStack(
            View('Label', label="Bar"),
            View('Label', label="Baz")
        )
    )
)
```

Next to [yajui.gui.HorizontalStack](#) and [yajui.gui.VerticalStack](#) there are also some more layout. The [yajui.gui.Grid](#) layout is a quite flexible and powerful one:

```
Grid(
    View('Label', label="Foo", row=0, col=0),
    View('Label', label="Bar", row=1, col=0),
    View('Label', label="Baz", row=1, col=1)
)
```

### 1.7.2.2 Scroll views

Another kind of container, similar to a layout, is [yajui.gui.ScrollView](#). It is used whenever a subelement inside the user interface might be larger than the space available for it. It does so by scroll logic and can be used like this:

```
ScrollView(
    View('SomethingLarge')
)
```

Note: Having more than one widget inside a scroll view requires to place a layout around them before.

There are also some other, more complex containers like tab views or carousels. A later chapter helps to find out details about them. However, there are no shortcuts available for them, so using them requires a complete `View` specification with a class name and some property assignments.

### 1.7.2.3 Sizing

Due to the size the view window has and due to the user interface specification assembled from hierarchies of layouts and actual widgets, sizing for the inner leaf widgets is predetermined to some degree. However, the view window typically does not have exactly the perfect size for presenting the specified user interface. It might be too small in one or both dimensions. That would break the usability of the interface and could be considered a defect. [Scroll views](#) should be used in order to avoid that. Whenever the view is larger than necessary, the additional space is distributed to the widgets. This happens while considering a stretch factor on each widget, which allows to influence how the distribution takes place. Specify the optional `hstretch` and/or `vstretch` properties for that, like this:

```
HorizontalStack(
    View('Label', label="Foo", hstretch=2),
    View('Label', label="Bar", hstretch=5),
    # we can also override the auto sizing (but still stretching)
    View('Label', label="Baz", hstretch=1, width='30pt')
)
```

Since each layout behaves like a widget (in fact is a widget), stretch factors can also be assigned to them. This can be crucial for getting the desired result.

In some pieces of code one might find the usage of those stretch factors by a different name: `horizontalStretchAffinity` and `verticalStretchAffinity` do exactly the same with a more verbose name.

Layouts do not only stretch widgets, but they might also align them on the other axis by giving them the same height or width. The properties `strictHorizontalSizing` and `strictVerticalSizing` can be set to `True` for making a widget ultimately refuse any sizes larger than its native one even for alignment purposes.

### 1.7.3 Configuring the main view itself

After the introduction about how to specify a user interface body, this chapter shows some ways to configure the outer frame of that.

#### 1.7.3.1 Header text

The application view has an optional first and second header text. Both are shown in the header of the main view. Set them like this:

```
class MyApplication(Application):  
    def __init__(self):  
        super().__init__()  
        self.sethead1("My Application")  
        self.sethead2("Foobar Baz")
```

#### 1.7.3.2 Icons

The application icon can be specified like this:

```
class MyApplication(Application):  
    def __init__(self):  
        super().__init__()  
        self.add_staticfile_location("/my/static/files/directory")  
        self.icon = "someimage.png"
```

### 1.7.4 Sidebar and head control

In a similar way as described in [setbodyleft](#) and [setbodyright](#), application code can also specify a sidebar and/or a small widget inside the header. See [yaji.Application.setsidebar\(\)](#) and [yaji.Application.setheadcontrol\(\)](#) for that.

### 1.7.5 Finding more information in the API references

This documentation text introduces each larger block of functionality in some depth, but it might not provide all details and all possibilities for each one; in fact it might not even mention all detail aspects of the common functionality, like listing all available widget classes. The same is also true for the sample applications. This kind of information are provided by the API references instead.

Later in this text, the API reference for the application code side begins, which lists the Python side of the Yaji API.

#### 1.7.5.1 Clove

The foundation and implementation of widgets and user interfaces on the browser side is the *Clove* library. It implements what a widget is, how they play together, and what widget classes exist. There is a dedicated Clove API reference, which lists all that, including descriptions of all those widget classes and what properties they have. It is located in `clove.pdf`.

Note: There is a common naming scheme in all APIs related to Yaji regarding non-public members. Unless there is a better way, they at least have a name starting with "`_`". Some of them are listed in the API references for special purposes, although they are non-public. In most cases it is okay to ignore them.

## 1.8 Data sources and data bindings

Data sources are a high-level mechanism for data exchange (potentially) between application code side and browser side. It is the primary mechanism for showing non-static data in a user interface.

A data source is a special object (i.e. an instance of some particular classes), which keeps any kind of data and which can be used for a data binding somewhere inside a [view specification](#). The data binding will lead to a connection between the data source and some property in that view specification, which keeps them in sync. This can be used in both directions: It updates the bound widget property with new data when the data source content changes, and it updates the data source content when the widget property changes (e.g. when the user typed some text in a field).

The following shows a simple example for a data source usage:

```
class MyApplication(Application):
    def __init__(self):
        super().__init__()
        ds = self.create_datastore()
        self.setbodyleft(View('EditBox', text=self.binddata(ds)))
```

See also [yaji.Application.create\\_datastore\(\)](#) and [yaji.Application.binddata\(\)](#). The variable `ds` is a [yaji.datastore.DataStore](#) object. It can be used to set or get the text content of that `EditBox` in this case.

Note: It is possible to use one data source in more than one data binding.

### 1.8.1 Server and local data sources

There are two different types of data sources. A [yaji.datastore.DataStore](#) as created in the example above comes with an actual data storage on application code side. Data bindings will keep this storage in sync with the user interface state, so application code can read from it and change it. Another type is [yaji.guibase.BrowserSideDatastore](#). Those data sources can be created by [yaji.Application.create\\_clientlocal\\_datastore\(\)](#) instead, and used in the same way as in the example above.

The latter type of data sources (also called a *local data source*, contrary to a *server data source*) are not connected to the application code side. They will not exchange any data between multiple browser windows presenting the same application either. Each view (i.e. each browser window showing the application) has an own, isolated storage for it. Due to that, a single binding like in the example above has limited usage, but having more than one binding for the same local data source might be usable.

Note: Local data sources are a tradeoff. Their obvious disadvantage is the decoupled way in which they keep their data, outside of the range of the application code. Their first duty is to help keeping the user interface in a consistent state. Server data sources need to push their data to the server and to let the server push it to all the other bindings. As mentioned earlier, all data exchange mechanisms between application code side and browser side are inherently indirect and deferred, i.e. a change on one side will eventually be reflected on the other side at a somewhat later time. A local data source reflects the other bindings in a much more direct way, which is very helpful in some situations.

Although application code is not able to make changes to local data sources, it can initialize the data source, like in the following:

```
with self.create_clientlocal_datastore() as ds:
    ds.setvalue("Foo")
self.setbodyleft(View('EditBox', text=self.binddata(ds)))
```

## 1.8.2 Data bindings

The examples above show the usage of `yaji.Application.binddata()` for binding an existing datasource to some widget properties. There is also a slightly different notation, which also can be found in some of the sample applications. It uses `yaji.Application.bindserver()` and `yaji.Application.bindlocal()` instead, typically with a string argument. They do a similar thing, but with a data source name. If there already is a data source with that name, it uses that, otherwise it creates a new one (a server or local one). Note that also `yaji.Application.create_datastore()` and `yaji.Application.create_clientlocal_datastore()` allow to assign a name to the new data source.

There are some configuration switches that control the behavior of a data binding. See the parameters of `yaji.Application.binddata()`.

### 1.8.2.1 Direction

The data direction of a binding specifies if data updates happen either only from the datasource to the widget property, or only the other way around, or in both directions. See the `data_direction` parameter of `yaji.Application.binddata()`, the `bindings_text.py` sample application, and the following example:

```
self.setbodyleft(View('EditBox', text=self.binddata(ds, yaji.BindDirection.ToWidget)))
```

Note: The default is bidirectional flow as this is the only useful thing for typical cases.

### 1.8.2.2 Converters

A data binding can be equipped with a converter that translates between the data representation inside the data-source and some view-specific representation. This is an advanced feature that also involves including browser side code, which is the subject of a much later chapter. See the `convertername` parameter of `yaji.Application.binddata()` and the `bindings_converters.py` sample application.

## 1.8.3 Data sources beyond bindings

There are also other usages of data sources, which do not involve data bindings at all. Most of them use data sources in a much broader manner, storing more complex data structures instead of just single values in them.

A data source allows to have that structures by the following framework:

- A data source *node* can store an arbitrary value, and
  - potentially has a two dimensional grid of child nodes.
- Each data source has a root node.

A data binding only makes use of a small part of that flexibility since it only works with the root cell and no structure of child nodes. But in a broader sense, data sources can also keep data in a list, table or tree structure by that. This is used e.g. in "TreeView" widgets by directly assigning a data source to the `datasource` property (see the `treeview.py` and `tableview.py` sample applications) and for keeping menus (see the `menus.py` sample application).

## 1.9 Event bindings

Data bindings allow to transfer data, including a way to let application code listen for changes e.g. made by the user. But they do not help for some other kinds of user interaction. A button for example, which can be clicked by the user, is not helpful with a data binding. Instead, they provide some events that application code can bind to.

### 1.9.1 bindevent

The available events depend on the widget class, so they can be found in the [Clove API reference](#) for builtin ones. For a button, an event binding could be like this:

```
self.setbodyleft(View('Button', OnClicked=self.bindevent(BackendFunction('something'))))
```

Implementing the handler function involves adding a new method to the [yajl.app.Application](#) implementation and annotating it in a particular way. Later chapters show more details, but for the example above, a handler could be implemented like this:

```
class MyApplication(Application):
    ...

    @RequestHandler.for_urls(auto=True)
    def something(self):
        spectacularly_succeed(fail_on_mondays=True)
```

### 1.9.2 BackendFunction

A [yajl.guibase.BackendFunction](#) as used in the event binding above is a reference to a function/method on the application code side.

Later chapters will also introduce other ways to use them.

They also allow to pass some additional arguments to the function.

### 1.9.3 BrowserFunction

Similar to [yajl.guibase.BackendFunction](#) there is also [yajl.guibase.BrowserFunction](#). They bind to a function on browser side. This is an advanced feature that typically also involves including browser side code, which is the subject of a much later chapter.

See the `eventbindings.py` sample application for both kinds of function references and ignore the `__do__` prepending the method name for the moment.

### 1.9.4 Event data

Whenever a browser side event occurs, it comes with some event data. Its content depends on the event, and for many events there are no useful event data at all. See [yajl.request.Request.ueventdata](#) and the `eventbindings.py` sample application for (a particularly useless example of) how to get this data.

## 1.10 Request handlers

The [bindevent example above](#) makes use of a [yaji.guibase.BackendFunction](#) and defines a handler method for it, that is annotated to be a *request handler*. Request handlers can be called from browser side, for executing event handlers, but also for other things explained in later chapters.

See also [yaji.reqhandler.RequestHandler](#) for more details.

Note: Request handler calls do not get enqueued in a main loop, but all of them execute in parallel, in separate threads.

### 1.10.1 The ‘\_do\_’ prefix

The easiest way to define a handler method is to just give it the same name as referred to in the [yaji.guibase.BackendFunction](#). There is another naming scheme however, which is recommended to use instead, that means prepending `_do_` to the method name (leaving the name in the event binding untouched). The method name would then be `_do_something` instead.

### 1.10.2 URL mapping

URL mapping is the lookup process that finds the right request handler for the URL of a request from browser side. For the [bindevent example above](#) it would find the `something` (or `_do_something`) method for the request URL `/something`.

Instead of this default naming scheme, it is also possible to associate a request handler with other URL patterns. See the parameters of [yaji.RequestHandler.for\\_urls\(\)](#) for more information. This is optional and mostly for naming aesthetics.

### 1.10.3 Downstream execution

By default, request handlers are executed while leaving the response stream open for the final response. Later chapters show how to work with such responses. For event handlers, however, responses do not have any meaning.

Whenever a request handler potentially runs for a longer time, this can be a problem. There is no precise threshold that defines 'a longer time', as it depends on browsers and network infrastructure, but it is safe to consider anything larger than a few seconds as long. In such cases, request handlers should be annotated as *downstream* ones. They directly return an empty acknowledgment to the browser and execute afterwards. Those handlers are not able to do some things, e.g. to return a response to the browser. The latter is not a restriction for event handlers anyway (in any other case, a custom indirect way to return the response has to be established).

Annotating a request handler by [yaji.RequestHandler.run\\_downstream\(\)](#) makes a request handler a downstream one, like in the following example:

```
@RequestHandler.for_urls(auto=True)
@RequestHandler.run_downstream()
def something(self):
    go_on_summer_vacation()
```

## 1.11 Menus

A Yaji menu is a list of actions and submenus, grouped by separators, very similar to menus in desktop user interfaces. An easy way to provide a menu to a user is to have a main view menu, like in the following example:

```
class MyApplication(Application):
    ...

    def __init__(self):
        super().__init__()
        self.setactions([Action("Do Foo", BackendFunction("foo")),
                          Action("Do Bar", BrowserFunction("bar"))])
```

This example shows a menu with two items, both with a label text and a reference to some function to execute (like for [bindevent](#)). Instead of a list, it is also possible to use a data source, which allows dynamic additions, removals and changes. The `menus.py` sample application shows a larger example.

### 1.11.1 Actions

A menu action is an item in a menu that can be executed. The [menu example above](#) specifies some of them. See [yaji.guibase.Action](#) for more details.

### 1.11.2 Submenus

A submenu is another list of menu items, which the user can pop up on demand. See [yaji.guibase.Submenu](#) for more details.

### 1.11.3 Separators

A separator is a visible highlight like a line for visually grouping items. See [yaji.guibase.Separator](#) for more details.

## 1.12 User feedback

The user feedback subsystem provides simple ways for simple dialog based interactions with the user. Those dialogs can show a message text to the user and have some buttons or ask the user to enter a text, and some other things. See also [yaji.userfeedback.UserFeedbackController](#), [yaji.Application.userfeedback](#), the `eventbindings.py` sample application and the following example:

```
class MyApplication(Application):
    ...

    @RequestHandler.for_urls(auto=True)
    @RequestHandler.run_downstream()
    def _do_something(self):
        self.userfeedback.messageDialog("Hallo Wereld.")
```

Note: User feedback cannot be used in request handlers that are not [downstream](#) ones, because that would inevitably violate the rule of avoiding long execution times with them.



## 1.13 Dialogs

Dialogs are a way of interacting with the user in pseudo popups. Compared to [user feedback](#), dialogs are more powerful and flexible. Dialogs should only be used if the additional flexibility is needed, since it implies more coding.

Showing a dialog needs a [yaji.gui.View](#) specification [as above](#). At first, [yaji.Application.create\\_dialog\(\)](#) creates a new [yaji.gui.Dialog](#) for a view specification, then [yaji.Dialog.show\(\)](#) shows it to the user.

See the `dialog.py` sample application.

## 1.14 Internationalization

There are many aspects of internationalization, many of them handled automatically by the language infrastructures, if program code is straight-forward enough.

### 1.14.1 Translations

One of the other aspects is translating user interface texts, so the user interface can be shown in the native language of the user.

Application code can register text translations during initialization by an internal key and a translation in some languages for each text, like in the following:

```
class MyApplication(Application):  
    def __init__(self):  
        super().__init__()   
        self.add_translations('Welcome', en="Welcome!", it="Benvenuto!", nl="Welkom!")
```

Other application code can refer to those texts like in the following example:

```
self.setbodyleft(View(label=TrStr('Welcome')))
```

If the user language is not available, the English text is shown.

## 1.15 Application stopping

The application shutdown procedure is a bit complicated internally, since there must be a stable and defined interaction between application code side and browser side in different situations. The application code can stop the application, the browser side can trigger it, and the user sometimes can just close all browser windows.

See [yaji.Application.stop\(\)](#) for details.

## 1.16 Forcefully keeping a browser view open

The default behavior of Yaji is to try keeping one view open. When the user closes all browser windows, it will open a new one, so the user is forced to correctly stop the application. The `stop_implicitly_when_browser_closed` parameter of `yaji.Application.__init__()` allows to control this behavior.

It is possible to temporarily override that choice for the execution of a code block with the `yaji.Application.do_stop_implicitly_when_browser_closed()` and `yaji.Application.dont_stop_implicitly_when_browser_closed()` context managers.

## 1.17 Closed notification

If the default behavior of [forcefully keeping a browser view open](#) is enabled, a re-opened view will show a notification. This notification explains why the browser opened again (and allows to close the application in some situations). The `show_browser_closed_notification` parameter of `yaji.Application.__init__()` allows to control this behavior.

## 1.18 Shutdown dialog

In some situations, the application stops without closing all opened views. This usually happens whenever a view is opened in a usual web browser, since application shutdown will not close those browsers or tabs inside them. The default behavior is to show a shutdown dialog in those views, which disables the complete user interface and shows an generic information text. The `skip_shutdown_dialog` parameter of `yaji.Application.__init__()` allows to control this behavior.

## 1.19 Browser side

A Yaji application is allowed to include browser side resources like JavaScripts or style sheets for more flexibility in user interface implementations and for more direct interactions with the user, avoiding expensive data transfer with the application code side.

This leaves the world of Python and the application code context. It is an advanced feature, which should be avoided if a similar implementation without browser side code is possible.

### 1.19.1 Adding browser side resources

The simplest way to include browser side resources is to locate a JavaScript in `__foo.js` and/or a style sheet in `__foo.css` in the same directory as the main application `foo.py`.

More resources can be add with `yaji.Application.add_clientscript()` and `yaji.Application.add_clientstyle()`.

Directories for additional static second-level resources (e.g. images used in style sheets) can be added with `yaji.Application.add_staticfile_location()`.

### 1.19.2 Custom widgets

One good reason for including a JavaScript is to implement a custom widget class, i.e. a class which directly or indirectly extends `clove.Widget`. Details about this are beyond the scope of this text. See the `customwidget↔_simple.py` sample application and the [Clove API reference](#).

### 1.19.3 Yaji browser side API

There is a rich Yaji API on browser side, which provides mechanisms for data exchange with the application code side, user interface features, and more. Some of its basic features are introduced in the next chapters. There is a dedicated API reference for it located in `browserside.pdf`, where you can look up details about the functions mentioned in the following.

#### 1.19.3.1 yaji.ready

An included JavaScript gets executed at an early step of initialization. The infrastructure is not ready for many things at this moment. `yaji.ready` calls a function once the initialization is finished. See the `yaji_ready.py` sample application.

#### 1.19.3.2 Ajax

Ajax requests are a low-level way to communicate with the application code side. They can be done by the XML↔HttpRequest web API or any other ones. There is also `yaji.ajax()`, which can be used for that.

On application code side, [request handlers](#) will answer Ajax requests. Non [downstream](#) request handlers can return an arbitrary serializable data structure that is passed back as response to browser side. Any Ajax API allows to retrieve and process those responses.

##### 1.19.3.2.1 Request parameters

Ajax requests can carry parameters in the usual ways, either encoded in the url or in a POST body, which is usually abstracted by the Ajax API. On application code side, a request handler will receive those parameters by the arguments of the method call. The following example makes an Ajax request in browser code:

```
yaji.ajax({url: 'something', data: {x: 13, y: 37}});
```

The request handler can be defined like in the following example for receiving the parameters:

```
@RequestHandler.for_urls(auto=True)
def _do_something(self, x, y):
    do_something_spectacular_in(x, y)
```

Note: The request handler is required to take exactly the arguments it gets. Violating that is potentially a critical error. Since request handlers are just Python functions, their signatures can specify default values for arguments and variable arguments like `**kwargs` in order to solve this problem.

### 1.19.3.2.2 Request parameter types

By default, request arguments are passed to the request handlers as strings. Type annotations lead to automatic conversions, so the following example is working code:

```
@RequestHandler.for_urls(auto=True)
def _do_something(self, x: int, y: int):
    return 100 * x + y
```

See the `requesthandler.py` sample application. See [yaji.Application.add\\_requestparam\\_type\\_converter\(\)](#) for custom types.

### 1.19.3.2.3 Request objects

Inside a request handler it is possible to access some extended aspects of the request, like the request url, and to participate on building the response in low-level ways, by a request object. See [yaji.request.Request](#), [yaji.Application.current\\_request](#) and the `request.py` sample application.

### 1.19.3.3 Client events

Client events are a low-level mechanism that allows the application code side to broadcast events to the open view(s). They should not to be confused with [event bindings](#), which are a high-level mechanism working in the other direction.

A client event has a name that specifies the kind of event, and some arbitrary event data. Application code can trigger a client event like in the following example:

```
self.triggerevent('somethinghappened', id=1337)
```

See also [yaji.Application.triggerevent\(\)](#).

Browser side code can register a handler during initialization like in the following example:

```
yaji.addEventHandler('somethinghappened', (data) => {
    doSomethingWith(data.id);
});
```

### 1.19.3.4 Appconfig

The appconfig mechanism has some similarities to [data sources](#) but is specialized for global values. It stores some arbitrary global application state data by a key name, backed on application code side, and provides an easy browser side API for consuming them. Application code can set a value like in the following example:

```
self.appconfig.setconfig('headertext', "Schildpad")
```

See also [yaji.appconfig.AppConfig](#) and [yaji.Application.appconfig](#).

Browser side code can consume those values by registering a handler during initialization like in the following example:

```
yaji.appconfig.addHandler('headertext', (value) => {
    setMyFunnyHeader("Een " + value + "je");
});
```

Note: There is no risk in registering an appconfig handler after it was set by application code. When a handler is added, if there already is a value, the handler will be executed once with that value.

### 1.19.3.5 Curtains

Curtains are a way to work around the problems of the [decoupling between application code side and browser side](#). In some defined situation, curtains allow to disable the user interface or to shield it in some other ways, until all transitions between application code side and browser sides are finished. This stops the user from intervening with the user interface while it is not guaranteed to be in a consistent state.

Curtains are implemented on browser side. There is a default `yaji.Curtain` implementation which can also be subclassed for blocking the user interface in different ways (e.g disabling parts of it).

See the `curtain.py` sample application.

### 1.19.4 JSON serialization

JSON serialization is used heavily for data transfer from application code side to browser side (it is also used for some transfers from browser side) and in this direction can also be extended by serialization routines for custom types. Those extensions have an application side code (serialization) part and a browser side (deserialization) code part.

The former is realized by adding a `__to_simple_repr__()` method to the custom type, like in the following example:

```
class MyObject:
    ...
    def __to_simple_repr__(self):
        return 'MyObject', {'foo': self.__foo}
```

This method returns a deserializer name and a dictionary of arbitrary data, which are serializable. The deserializer name can be equal to the class name on application code side, but it actually specifies the class on browser code side that handles the deserialization. Such a class can be like in the following example:

```
class MyObject {
    ...
    static _from_simple_repr_(data) {
        return new MyObject(data.foo, 42);
    }
}
```

## 1.20 Middleware

Middleware are a plugin mechanism for affecting internal request handling in a pluggable and potentially custom way. See [yaji.core.Middleware](#), [yaji.Application.add\\_middleware\(\)](#) and the `middleware.py` sample application.

There are also some methods of [yaji.app.Application](#), often beginning with `on`, that can be overridden for controlling other aspects of low-level behavior.

## 1.21 Application takeover

Application takeover allows multiple Yaji applications to share a browser view by taking it over from a parent application on startup and releasing it when stopping.

This is a feature for exotic scenarios. It can avoid new windows or browser tabs for each new application instance when many applications run in an interleaved way.

See the `takeover.py` sample application.

## 1.22 Storing applications statically

For exotic scenarios, and only if the application implementation is compatible to it, [yaji.Application.storeasstaticapplication\(\)](#) can be used for storing the entire application to a directory of static files. This can again be opened with a usual web browser, but also without any application backend running.

## 1.23 Logging

There is a Python `logging` logger used by Yaji for writing messages for diagnostics. It can also be used by application code. See [yaji.core.Log](#) and `yaji.core.log`.

Note: Setting the environment variable `YAJI_LOGDEBUG` to 1 causes a Yaji application to write verbose debug log messages to the terminal.

## 1.24 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<code>yaji.request.Request.__InRequest</code>	22
<code>yaji.app.Application.__SetStopImplicitlyWhenBrowserClosed</code>	23
<code>yaji.app.Application.__YajiHTTPRequestHandler</code>	24
<code>yaji.app.Application.__YajiTCPServer</code>	24
<code>yaji.gui.View._DataBinding</code>	25
<code>yaji.gui.View._EventBinding</code>	26
<code>yaji.guibase.AbstractAction</code>	27
<code>yaji.guibase.Action</code>	28
<code>yaji.guibase.Separator</code>	81
<code>yaji.guibase.Submenu</code>	82
<code>yaji.guibase.AbstractFunction</code>	28
<code>yaji.guibase.BackendFunction</code>	56
<code>yaji.guibase.BrowserFunction</code>	58
<code>yaji.appconfig.AppConfig</code>	29
<code>yaji.app.Application</code>	31
<code>yaji.guibase.BindDirection</code>	57
<code>yaji.browser.BrowserHook</code>	59
<code>yaji.guibase.BrowserSideDatasource</code>	60
<code>yaji.datastore.DataStore</code>	60
<code>yaji.gui.Dialog</code>	65
<code>yaji.dialogcontroller.DialogController</code>	67
<code>yaji.core.Directories</code>	68
<code>yaji.guibase.Icon</code>	70
<code>yaji.core.Log</code>	71
<code>yaji.core.Middleware</code>	71
<code>yaji.app.Application.__ComputeContentMiddleware</code>	20
<code>yaji.app.Application.__PostParamsMiddleware</code>	22
<code>yaji.auth.AuthMiddleware</code>	55
<code>yaji.pyhtmlview.PyHtmlView</code>	72
<code>yaji.request.Request</code>	73
<code>yaji.reqhandler.RequestHandler</code>	78
<code>yaji.i18n.TrStr</code>	83
<code>yaji.userfeedback.UserFeedbackController</code>	84

yaji.datastore.DataStore.ValuePointer . . . . .	87
yaji.gui.View . . . . .	88
yaji.gui._AbstractContainer . . . . .	24
yaji.gui.Grid . . . . .	68
yaji.gui.HorizontalStack . . . . .	69
yaji.gui.ScrollView . . . . .	80
yaji.gui.VerticalStack . . . . .	88
yaji.gui.Wrap . . . . .	91
yaji.gui.Spacer . . . . .	82
yaji.guibase.Visibility . . . . .	90

## 1.25 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">yaji.app.Application.__ComputeContentMiddleware</a> Middleware which computes the response content, e.g . . . . .	20
<a href="#">yaji.request.Request.__InRequest</a> Context manager for setting the current request . . . . .	22
<a href="#">yaji.app.Application.__PostParamsMiddleware</a> Middleware for parsing http POST parameters . . . . .	22
<a href="#">yaji.app.Application.__SetStopImplicitlyWhenBrowserClosed</a> See <a href="#">do_stop_implicitly_when_browser_closed()</a> and <a href="#">dont_stop_implicitly_when_browser_closed()</a> . . . . .	23
<a href="#">yaji.app.Application.__YajiHTTPRequestHandler</a> The http request handler . . . . .	24
<a href="#">yaji.app.Application.__YajiTCPServer</a> The tcp server . . . . .	24
<a href="#">yaji.gui._AbstractContainer</a> Abstract container view . . . . .	24
<a href="#">yaji.gui.View._DataBinding</a> A data binding . . . . .	25
<a href="#">yaji.gui.View._EventBinding</a> An event binding . . . . .	26
<a href="#">yaji.guibase.AbstractAction</a> Abstract menu action (can be an actual <a href="#">Action</a> , submenus, separators, ...) . . . . .	27
<a href="#">yaji.guibase.AbstractFunction</a> An abstract callable function . . . . .	28
<a href="#">yaji.guibase.Action</a> A menu action . . . . .	28
<a href="#">yaji.appconfig.AppConfig</a> Application configuration storage . . . . .	29
<a href="#">yaji.app.Application</a> Base class for a Yaji web application . . . . .	31
<a href="#">yaji.auth.AuthMiddleware</a> Middleware for authentication . . . . .	55
<a href="#">yaji.guibase.BackendFunction</a> A callable application code side function . . . . .	56
<a href="#">yaji.guibase.BindDirection</a> Data flow directions of data bindings . . . . .	57
<a href="#">yaji.guibase.BrowserFunction</a> A callable browser side function . . . . .	58
<a href="#">yaji.browser.BrowserHook</a> Helper for opening browsers . . . . .	59
<a href="#">yaji.guibase.BrowserSideDatasource</a> A browser side datasource . . . . .	60

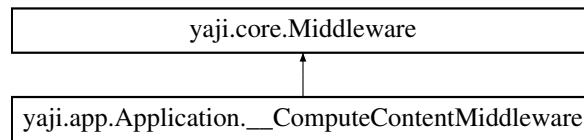
<a href="#">yaji.datastore.DataStore</a>	
A clove-like datastore which then becomes available on browser side as well . . . . .	60
<a href="#">yaji.gui.Dialog</a>	
A dialog . . . . .	65
<a href="#">yaji.dialogcontroller.DialogController</a>	
A controller for dialogs . . . . .	67
<a href="#">yaji.core.Directories</a>	
Some important paths . . . . .	68
<a href="#">yaji.gui.Grid</a>	
A grid view . . . . .	68
<a href="#">yaji.gui.HorizontalStack</a>	
A horizontal stack view . . . . .	69
<a href="#">yaji.guibase.Icon</a>	
Icon . . . . .	70
<a href="#">yaji.core.Log</a>	
Logging . . . . .	71
<a href="#">yaji.core.Middleware</a>	
A middleware controls some internal stuff like request processing in a custom way . . . . .	71
<a href="#">yaji.pyhtmlview.PyHtmlView</a>	
This optional functionality allows to present the user interface of the application in an own bare window instead of the usual browser . . . . .	72
<a href="#">yaji.request.Request</a>	
A request from the client, including ways to give the response . . . . .	73
<a href="#">yaji.reqhandler.RequestHandler</a>	
Decorators for request handler functions, i.e . . . . .	78
<a href="#">yaji.gui.ScrollView</a>	
A scroll view . . . . .	80
<a href="#">yaji.guibase.Separator</a>	
A menu separator . . . . .	81
<a href="#">yaji.gui.Spacer</a>	
A spacer . . . . .	82
<a href="#">yaji.guibase.Submenu</a>	
A submenu of a menu . . . . .	82
<a href="#">yaji.i18n.TrStr</a>	
An i18n-aware string for displaying on browser side . . . . .	83
<a href="#">yaji.userfeedback.UserFeedbackController</a>	
A controller for user feedback operations . . . . .	84
<a href="#">yaji.datastore.DataStore.ValuePointer</a>	
A value pointer is kind of an address or reference to a node (i.e . . . . .	87
<a href="#">yaji.gui.VerticalStack</a>	
A vertical stack view . . . . .	88
<a href="#">yaji.gui.View</a>	
A widget build configuration for user interface definition . . . . .	88
<a href="#">yaji.guibase.Visibility</a>	
The visibility of a widget . . . . .	90
<a href="#">yaji.gui.Wrap</a>	
A wrap view . . . . .	91

## 1.26 yaji.app.Application.\_\_ComputeContentMiddleware Class Reference

Middleware which computes the response content, e.g.

Inheritance diagram for yaji.app.Application.\_\_ComputeContentMiddleware:





### Static Public Member Functions

- def `request_processor` (index=0)  
*Marks a function as a request processor.*

### Private Member Functions

- def `__callhandler` (self, handler, request)  
*Calls a request handler for a request and returns its result.*

#### 1.26.1 Detailed Description

Middleware which computes the response content, e.g.

by fetching files or executing a request handler.

#### 1.26.2 Member Function Documentation

##### 1.26.2.1 `__callhandler()`

```

def yaji.app.Application.__ComputeContentMiddleware.__callhandler (
    self,
    handler,
    request,
    Tuple,
    Optional,
    Any,
    Optional,
    Exception,
    Optional,
    str ) [private]
  
```

Calls a request handler for a request and returns its result.

##### Parameters

<i>handler</i>	The request handler.
<i>request</i>	The request.

### 1.26.2.2 request\_processor()

```
def yaji.core.Middleware.request_processor (
    index = 0 ) [static], [inherited]
```

Marks a function as a request processor.

It takes a `Request` parameter and reads and/or modifies parts of it.

#### Parameters

<i>index</i>	The execution order index (low value: early execution).
--------------	---

The documentation for this class was generated from the following file:

- yaji/app.py

## 1.27 yaji.request.Request.\_\_InRequest Class Reference

Context manager for setting the current request.

### 1.27.1 Detailed Description

Context manager for setting the current request.

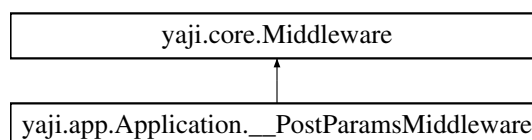
The documentation for this class was generated from the following file:

- yaji/request.py

## 1.28 yaji.app.Application.\_\_PostParamsMiddleware Class Reference

Middleware for parsing http POST parameters.

Inheritance diagram for yaji.app.Application.\_\_PostParamsMiddleware:



## Static Public Member Functions

- def [request\\_processor](#) (index=0)  
*Marks a function as a request processor.*

### 1.28.1 Detailed Description

Middleware for parsing http POST parameters.

### 1.28.2 Member Function Documentation

#### 1.28.2.1 request\_processor()

```
def yaji.core.Middleware.request_processor (
    index = 0 ) [static], [inherited]
```

Marks a function as a request processor.

It takes a `Request` parameter and reads and/or modifies parts of it.

#### Parameters

<i>index</i>	The execution order index (low value: early execution).
--------------	---

The documentation for this class was generated from the following file:

- `yaji/app.py`

## 1.29 yaji.app.Application.\_\_SetStopImplicitlyWhenBrowserClosed Class Reference

See [do\\_stop\\_implicitly\\_when\\_browser\\_closed\(\)](#) and [dont\\_stop\\_implicitly\\_when\\_browser\\_closed\(\)](#).

### 1.29.1 Detailed Description

See [do\\_stop\\_implicitly\\_when\\_browser\\_closed\(\)](#) and [dont\\_stop\\_implicitly\\_when\\_browser\\_closed\(\)](#).

The documentation for this class was generated from the following file:

- `yaji/app.py`

## 1.30 yaji.app.Application.\_\_YajiHTTPRequestHandler Class Reference

The http request handler.

Inherits SimpleHTTPRequestHandler.

### 1.30.1 Detailed Description

The http request handler.

The documentation for this class was generated from the following file:

- yaji/app.py

## 1.31 yaji.app.Application.\_\_YajiTCPServer Class Reference

The tcp server.

Inherits ThreadingTCPServer.

### 1.31.1 Detailed Description

The tcp server.

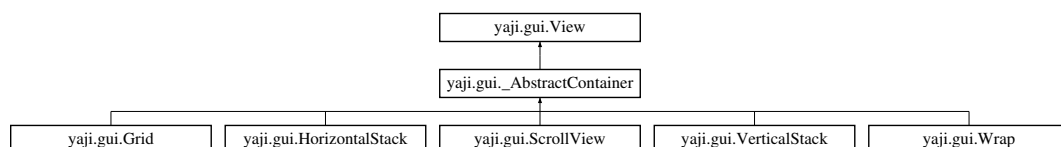
The documentation for this class was generated from the following file:

- yaji/app.py

## 1.32 yaji.gui.\_AbstractContainer Class Reference

Abstract container view.

Inheritance diagram for yaji.gui.\_AbstractContainer:



### Public Member Functions

- def [\\_\\_init\\_\\_](#) (self, clsname, paramname, singlechild, default, children, b)

### 1.32.1 Detailed Description

Abstract container view.

### 1.32.2 Constructor & Destructor Documentation

#### 1.32.2.1 \_\_init\_\_()

```
def yaji.gui._AbstractContainer.__init__ (
    self,
    clsname,
    paramname,
    singlechild,
    default,
    children,
    b )
```

#### Parameters

<i>clsname</i>	The clove classname.
<i>paramname</i>	The clove property name.
<i>singlechild</i>	If this widget is the container for only a single child widget.
<i>default</i>	The default value.
<i>children</i>	Child views.

The documentation for this class was generated from the following file:

- yaji/gui.py

## 1.33 yaji.gui.View.\_DataBinding Class Reference

A data binding.

### Public Member Functions

- def [\\_\\_init\\_\\_](#) (self, datasource, data\_direction, convertername)

#### 1.33.1 Detailed Description

A data binding.

### 1.33.2 Constructor & Destructor Documentation

#### 1.33.2.1 `__init__()`

```
def yaji.gui.View._DataBinding.__init__ (
    self,
    datasource,
    data_direction,
    convertername )
```

##### Parameters

<i>datasource</i>	The internal data source definition.
<i>data_direction</i>	The data direction. See <code>BindDirection</code> .
<i>convertername</i>	Name of the converter class.

The documentation for this class was generated from the following file:

- `yaji/gui.py`

## 1.34 `yaji.gui.View._EventBinding` Class Reference

An event binding.

### Public Member Functions

- `def \_\_init\_\_ (self, function, curtain)`

#### 1.34.1 Detailed Description

An event binding.

### 1.34.2 Constructor & Destructor Documentation

#### 1.34.2.1 `__init__()`

```
def yaji.gui.View._EventBinding.__init__ (
    self,
    function,
    curtain )
```

## Parameters

<i>function</i>	The function to bind to an event.
<i>curtain</i>	The curtain expression.

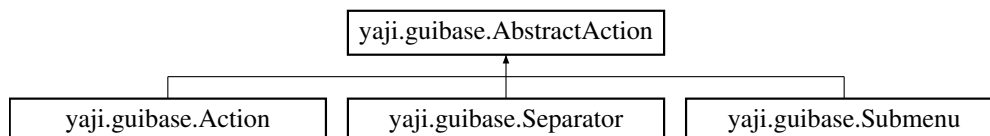
The documentation for this class was generated from the following file:

- yaji/gui.py

## 1.35 yaji.guibase.AbstractAction Class Reference

Abstract menu action (can be an actual [Action](#), submenus, separators, ...).

Inheritance diagram for yaji.guibase.AbstractAction:



### Public Member Functions

- def [\\_\\_init\\_\\_](#) (self, label, icon, checkable, checked, disabled, invisible)

### 1.35.1 Detailed Description

Abstract menu action (can be an actual [Action](#), submenus, separators, ...).

### 1.35.2 Constructor & Destructor Documentation

#### 1.35.2.1 [\\_\\_init\\_\\_\(\)](#)

```

def yaji.guibase.AbstractAction.__init__ (
    self,
    label,
    icon,
    checkable,
    checked,
    disabled,
    invisible )

```

**Parameters**

<i>label</i>	The label of this action.
<i>icon</i>	The action icon.
<i>checkable</i>	If this action is checkable (i.e. has a checkbox in menu).
<i>checked</i>	If this action is checked.
<i>disabled</i>	If this action is disabled (can't be triggered).
<i>invisible</i>	If this action is invisible.

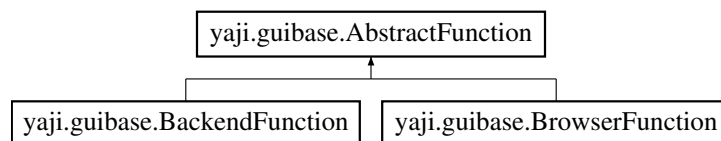
The documentation for this class was generated from the following file:

- `yaji/guibase.py`

## 1.36 `yaji.guibase.AbstractFunction` Class Reference

An abstract callable function.

Inheritance diagram for `yaji.guibase.AbstractFunction`:



### 1.36.1 Detailed Description

An abstract callable function.

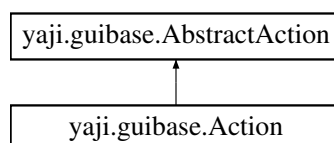
The documentation for this class was generated from the following file:

- `yaji/guibase.py`

## 1.37 `yaji.guibase.Action` Class Reference

A menu action.

Inheritance diagram for `yaji.guibase.Action`:





## Public Member Functions

- `def __init__ (self, label, function, icon=None, checkable=False, checked=False, disabled=False, invisible=False)`

### 1.37.1 Detailed Description

A menu action.

### 1.37.2 Constructor & Destructor Documentation

#### 1.37.2.1 \_\_init\_\_()

```
def yaji.guibase.Action.__init__ (
    self,
    label,
    function,
    icon = None,
    checkable = False,
    checked = False,
    disabled = False,
    invisible = False )
```

#### Parameters

<i>label</i>	The label of this action.
<i>function</i>	The function to call for this action.
<i>icon</i>	The action icon.
<i>checkable</i>	If this action is checkable (i.e. has a checkbox in menu).
<i>checked</i>	If this action is checked.
<i>disabled</i>	If this action is disabled (can't be triggered).
<i>invisible</i>	If this action is invisible.

The documentation for this class was generated from the following file:

- `yaji/guibase.py`

## 1.38 yaji.appconfig.AppConfig Class Reference

Application configuration storage.

## Public Member Functions

- def `getConfigs` (self, `_markinitialized=False`)  
*Returns the complete configuration.*
- def `getConfig` (self, key, `default=None`)  
*Returns a configuration value by key.*
- def `setconfig` (self, key, value)  
*Sets a configuration value by key.*

## Private Member Functions

- def `__markinitialized` (self)  
*Marks the configuration as initialized (in order to handle requests a bit differently).*

### 1.38.1 Detailed Description

Application configuration storage.

This can store arbitrary configuration items, which then become available on browser side as well.

### 1.38.2 Member Function Documentation

#### 1.38.2.1 `__markinitialized()`

```
def yaji.appconfig.AppConfig.__markinitialized (
    self,
    None ) [private]
```

Marks the configuration as initialized (in order to handle requests a bit differently).

#### 1.38.2.2 `getConfig()`

```
def yaji.appconfig.AppConfig.getConfig (
    self,
    key,
    default = None,
    Any )
```

Returns a configuration value by key.

#### Parameters

<i>key</i>	The configuration key.
<i>default</i>	The default value (if no such key).

### 1.38.2.3 getConfigs()

```
def yaji.appconfig.AppConfig.getConfigs (
    self,
    _markinitialized = False,
    Dict,
    str,
    Any )
```

Returns the complete configuration.

#### Parameters

<code>_markinitialized</code>	If to consider the configuration as initialized by this call.
-------------------------------	---

### 1.38.2.4 setconfig()

```
def yaji.appconfig.AppConfig.setconfig (
    self,
    key,
    value,
    None )
```

Sets a configuration value by key.

#### Parameters

<code>key</code>	The configuration key.
<code>value</code>	The new value.

The documentation for this class was generated from the following file:

- `yaji/appconfig.py`

## 1.39 yaji.app.Application Class Reference

Base class for a Yaji web application.

### Classes

- class [\\_\\_ComputeContentMiddleware](#)  
*Middleware which computes the response content, e.g.*

- class `__PostParamsMiddleware`  
*Middleware for parsing http POST parameters.*
- class `__SetStopImplicitlyWhenBrowserClosed`  
*See `do_stop_implicitly_when_browser_closed()` and `dont_stop_implicitly_when_browser_closed()`.*
- class `__YajiHTTPRequestHandler`  
*The http request handler.*
- class `__YajiTCPServer`  
*The tcp server.*

## Public Member Functions

- def `__init__` (self, `parentid`=None, `returntoparent`=True, `show_browser_closed_notification`=False, `stop_implicitly_when_browser_closed`=False, `skip_shutdown_dialog`=False, `browser_hook_heartbeat_threshold`=60 \* 2, `head1`=None, `head2`=None, `icon`=None, `mainview_icon`=None)
- def `head1` (self)  
*The 1st level header text.*
- def `head2` (self)  
*The 2nd level header text.*
- def `bodyleftviewactionlabel` (self)  
*The menu action label for the left main body widget.*
- def `bodyrightviewactionlabel` (self)  
*The menu action label for the right main body widget.*
- def `showonly` (self)  
*Switches to show only the left or right main body widget.*
- def `icon` (self)  
*The application icon.*
- def `mainview_icon` (self)  
*The mainview header icon.*
- def `show_browser_closed_notification` (self)  
*If to show a useful notification (including a way to close the application) after the browser was closed by the user and restarted.*
- def `stop_implicitly_when_browser_closed` (self)  
*If to consider the application as intendedly stopped when the user has closed the browser instead of opening a new one.*
- def `skip_shutdown_dialog` (self)  
*If not to show a ui-blocking 'application stopped' dialog on ui shutdown.*
- def `browser_hook_heartbeat_threshold` (self)  
*The time window within the backend expects a heartbeat from the browser side before it tries to open a new browser window.*
- def `url` (self)  
*The application root url for browser access.*
- def `appconfig` (self)  
*The application AppConfig.*
- def `userfeedback` (self)  
*User feedback handler (for message boxes, ...).*
- def `urlmap` (self)  
*The application's url to request handler map.*
- def `takenover` (self)  
*If the current application is taken over by another one and paused at the moment.*
- def `id` (self)  
*The application id.*

- def [parentid](#) (self)  
*The id of the parent application (if any).*
- def [wasreopened](#) (self)  
*If the browser needed to be opened again.*
- def [returntoparent](#) (self)  
*If to return to the parent application after exit.*
- def [isrunning](#) (self)  
*Checks if the application is currently running (started and no stop triggered yet).*
- def [isstaticapplication](#) (self)  
*Returns if this application is currently running as static one.*
- def [current\\_request](#) (self)  
*Returns the Request associated to the current request (in a request handler).*
- def [create\\_datastore](#) (self, name=None, curtain=None)  
*Creates a new (real serverside) DataStore and returns it.*
- def [get\\_datastore](#) (self, name, create\_if\_not\_exist=True)  
*Returns a (real serverside) DataStore by name.*
- def [create\\_clientlocal\\_datastore](#) (self, name=None)  
*Creates a new browserside data source and returns it.*
- def [get\\_clientlocal\\_datastore](#) (self, name, create\_if\_not\_exist=True)  
*Returns a browserside data source by name.*
- def [start](#) (self)  
*Starts the application.*
- def [stop](#) (self)  
*Stops the application.*
- def [waituntilstopped](#) (self)  
*Blocks execution while the application is running.*
- def [storeasstaticapplication](#) (self, targetdir)  
*Stores the application as static files.*
- def [setbodyleft](#) (self, cfg)  
*Sets the left main body widget.*
- def [setbodyright](#) (self, cfg)  
*Sets the right main body widget.*
- def [setactions](#) (self, actions)  
*Sets the main menu actions.*
- def [setheadcontrol](#) (self, cfg)  
*Sets the head control widget.*
- def [setsidebar](#) (self, cfg)  
*Sets the sidebar widget.*
- def [setsplitterposition](#) (self, v)  
*Sets the main body widgets splitter position.*
- def [create\\_dialog](#) (self, cfg, showcfg)  
*Creates a dialog.*
- def [add\\_clientstyle](#) (self, content=None, frompath=None)  
*Adds additional css application styles.*
- def [add\\_clientscript](#) (self, content=None, frompath=None)  
*Adds application client javascript.*
- def [add\\_middleware](#) (self, middleware)  
*Adds a Middleware.*
- def [add\\_translations](#) (self, stringname, texts)  
*Adds translations in some languages for a string.*
- def [get\\_translations\\_stringnames](#) (self)

- Returns all available stringnames for translations.*
- def `get_translations` (self, stringname, texts)
  - Returns translations in all available languages for a string.*
- def `try_addclientresources_from_default_location` (self, fpath)
  - Tries to add a style file and script file from a file naming convention, so you don't have to call `add_clientstyle/add_clientscript()`.*
- def `add_staticfile_location` (self, path)
  - Adds a local file path as an additional location for reading static files from.*
- def `triggerevent` (self, name, eventdata)
  - Triggers an event on browser side.*
- def `do_stop_implicitly_when_browser_closed` (self, value=True)
  - Use such instances via `with` for temporarily enabling implicit application stop when browser closed.*
- def `dont_stop_implicitly_when_browser_closed` (self)
  - Use such instances via `with` for temporarily disabling implicit application stop when browser closed.*
- def `add_requestparam_type_converter` (self, paramtype, converter)
  - Adds a custom type converter for request params.*
- def `bindserver` (self, datasource, data\_direction=None, convertername=None)
  - Declares binding a property of a `View` to a application code side data source.*
- def `bindlocal` (self, datasource, data\_direction=None, convertername=None)
  - Declares binding a property of a `View` to a local browser side data source.*
- def `binddata` (self, datasource, data\_direction=None, convertername=None)
  - Declares binding a property of a `View` to an application code side or local browser side data source.*
- def `bindevent` (self, function, curtain=None)
  - Declares binding an event property of a `View` to a application code side or local browser side action.*
- def `enable_authentication` (self, authfct, realm=None)
  - Enables authentication by a custom authenticator function.*
- def `oninitialize` (self)
  - Initializes the application.*
- def `onprocessrequesterror` (self, request, error)
  - Reacts on process request errors, e.g.*
- def `onopenbrowsererror` (self)
  - Reacts on errors when opening the browser.*
- def `onunhandledclienterror` (self, error, src, line)
  - Reacts on unhandled client errors.*
- def `onopenbrowserinformationoutput` (self, kind, message)
  - Reacts on information output (e.g.*
- def `onbrowserreopened` (self)
  - Reacts on the fact that the browser was opened again.*

## Private Member Functions

- def `_middlewares` (self)
  - Returns the sorted list of middlewares.*
- def `_openbrowser` (self, url)
  - Opens a browser view for ui rendering.*
- def `_tryclosebrowser` (self, withchecks=True)
  - Tries to close the browser window.*
- def `_browser_was_reopened` (self)
  - Notifies that the browser needed to be reopened.*
- def `_get_requestparam_type_converter` (self, paramtype)

- Returns a type converter for a request param.*
- `def _json_make_serializable (self, o)`  
*Makes some special objects json serializable.*
- `def _staticfile_path_to_abspath (self, relpath)`  
*Searches a file in the pool of static files locations by relative path and returns the absolute path if such a file exists.*
- `def _findhandler (self, request)`  
*Returns the handler function for a request.*
- `def _get_rootpagecontent (self)`  
*Returns the main page content.*
- `def _callhandler (self, handler, request)`  
*Calls a request handler for a request and returns its result, but includes some error handling.*
- `def __get_requesthandler_for_url (self, urlpath, app)`  
*Returns the registered request handler function for a request url path.*
- `def __get_datastore_cell (self, datastorename, id)`  
*Helper for datastore accesses.*
- `def __create_datastore_helper (self, name, createfct)`  
*Creates and adds a new (serverside or browserside) datastore.*
- `def __get_datastore_helper (self, name, create_if_not_exist, creatfct)`  
*Returns a (serverside or browserside) datastore by name if one exists, otherwise see parameters.*

## Static Private Member Functions

- `def __get_content (content, frompath)`  
*Returns a bytes content with an additional ending newline.*

### 1.39.1 Detailed Description

Base class for a Yaji web application.

### 1.39.2 Constructor & Destructor Documentation

#### 1.39.2.1 \_\_init\_\_()

```
def yaji.app.Application.__init__ (
    self,
    parentid = None,
    returntoparent = True,
    show_browser_closed_notification = False,
    stop_implicitly_when_browser_closed = False,
    skip_shutdown_dialog = False,
    browser_hook_heartbeat_threshold = 60*2,
    head1 = None,
    head2 = None,
    icon = None,
    mainview_icon = None )
```

## Parameters

<i>parentid</i>	Optional id of a parent <a href="#">Application</a> , which e.g. can be redirected back to after exit.
<i>returntoparent</i>	If to return back to the <i>parentid</i> application after exit in the user's browser.
<i>show_browser_closed_notification</i>	If to show a useful notification (including a way to close the application) after the browser was closed by the user and restarted.
<i>stop_implicitly_when_browser_closed</i>	If to consider the application as intendedly stopped when the user has closed the browser instead of opening a new one.
<i>skip_shutdown_dialog</i>	If not to show a ui-blocking 'application stopped' dialog on ui shutdown.
<i>browser_hook_heartbeat_threshold</i>	The time window within the backend expects a heartbeat from the browser side before it tries to open a new browser window.
<i>head1</i>	The 1st level header text.
<i>head2</i>	The 2nd level header text.
<i>icon</i>	The window icon.
<i>mainview_icon</i>	The icon of the mainview header.

## 1.39.3 Member Function Documentation

1.39.3.1 `__create_datastore_helper()`

```
def yaji.app.Application.__create_datastore_helper (
    self,
    name,
    createfct,
    Any ) [private]
```

Creates and adds a new (serverside or browserside) datastore.

## Parameters

<i>name</i>	The datastore name (or None).
<i>createfct</i>	Callback function for actually creating a new datastore.

1.39.3.2 `__get_content()`

```
def yaji.app.Application.__get_content (
    content,
    frompath,
    bytes ) [static], [private]
```

Returns a bytes content with an additional ending newline.



## Parameters

<i>content</i>	The content to return (as string or bytes).
<i>frompath</i>	The file path to fetch the content from.

1.39.3.3 `__get_datastore_helper()`

```
def yaji.app.Application.__get_datastore_helper (
    self,
    name,
    create_if_not_exist,
    creatfct,
    Optional,
    Any ) [private]
```

Returns a (serverside or browserside) datastore by name if one exists, otherwise see parameters.

## Parameters

<i>name</i>	The datastore name.
<i>create_if_not_exist</i>	If to create a fresh one (otherwise returns <code>None</code> ) if no one exists with that name.
<i>creatfct</i>	Callback function for actually creating a new datastore.

1.39.3.4 `__get_requesthandler_for_url()`

```
def yaji.app.Application.__get_requesthandler_for_url (
    self,
    urlpath,
    app,
    Tuple,
    Optional,
    Callable,
    Dict,
    str,
    str ) [private]
```

Returns the registered request handler function for a request url path.

## Parameters

<i>urlpath</i>	The path segment of the request url.
----------------	--------------------------------------

#### 1.39.3.5 `_browser_was_reopened()`

```
def yaji.app.Application._browser_was_reopened (
    self,
    None ) [private]
```

Notifies that the browser needed to be reopened.

#### 1.39.3.6 `_callhandler()`

```
def yaji.app.Application._callhandler (
    self,
    handler,
    request,
    Tuple,
    Optional,
    Any,
    Optional,
    Exception,
    Optional,
    str ) [private]
```

Calls a request handler for a request and returns its result, but includes some error handling.

##### Parameters

<i>handler</i>	The request handler.
<i>request</i>	The request.

#### 1.39.3.7 `_findhandler()`

```
def yaji.app.Application._findhandler (
    self,
    request,
    Optional,
    Callable ) [private]
```

Returns the handler function for a request.

##### Parameters

<i>request</i>	The request to serve.
----------------	-----------------------

#### 1.39.3.8 `_get_requestparam_type_converter()`

```
def yaji.app.Application._get_requestparam_type_converter (
    self,
    paramtype ) [private]
```

Returns a type converter for a request param.

##### Parameters

<i>paramtype</i>	The python type of the parameter to convert.
------------------	--

#### 1.39.3.9 `_get_rootpagecontent()`

```
def yaji.app.Application._get_rootpagecontent (
    self,
    bytes ) [private]
```

Returns the main page content.

#### 1.39.3.10 `_json_make_serializable()`

```
def yaji.app.Application._json_make_serializable (
    self,
    o,
    Any ) [private]
```

Makes some special objects json serializable.

##### Parameters

<i>o</i>	The object to serialize.
----------	--------------------------

#### 1.39.3.11 `_middlewares()`

```
def yaji.app.Application._middlewares (
    self,
    List,
    Middleware ) [private]
```

Returns the sorted list of middlewares.

#### 1.39.3.12 `_openbrowser()`

```
def yaji.app.Application._openbrowser (
    self,
    url,
    None ) [private]
```

Opens a browser view for ui rendering.

##### Parameters

<i>url</i>	The url to open.
------------	------------------

#### 1.39.3.13 `_staticfile_path_to_abspath()`

```
def yaji.app.Application._staticfile_path_to_abspath (
    self,
    relpath,
    str ) [private]
```

Searches a file in the pool of static files locations by relative path and returns the absolute path if such a file exists.

See [add\\_staticfile\\_location\(\)](#).

##### Parameters

<i>relpath</i>	The relative static file path.
----------------	--------------------------------

#### 1.39.3.14 `_tryclosebrowser()`

```
def yaji.app.Application._tryclosebrowser (
    self,
    withchecks = True,
    bool ) [private]
```

Tries to close the browser window.

Note: This will not work in all situations, e.g. typically not if the app is opened in the system browser. Your application should handle that situation gracefully (e.g. by the default app shutdown notification).

##### Parameters

<i>withchecks</i>	If set, it makes some checks before, e.g. if implicit stopping is enabled.
-------------------	--

#### 1.39.3.15 add\_clientscript()

```
def yaji.app.Application.add_clientscript (
    self,
    content = None,
    frompath = None,
    None )
```

Adds application client javascript.

Call this from your constructor.

##### Parameters

<i>content</i>	The content to return (as string or bytes).
<i>frompath</i>	The file path to fetch the content from.

#### 1.39.3.16 add\_clientstyle()

```
def yaji.app.Application.add_clientstyle (
    self,
    content = None,
    frompath = None,
    None )
```

Adds additional css application styles.

Call this from your constructor.

##### Parameters

<i>content</i>	The content to return (as string or bytes).
<i>frompath</i>	The file path to fetch the content from.

#### 1.39.3.17 add\_middleware()

```
def yaji.app.Application.add_middleware (
    self,
    middleware,
    None )
```

Adds a Middleware.

##### Parameters

<i>middleware</i>	The middleware to add.
-------------------	------------------------

### 1.39.3.18 add\_requestparam\_type\_converter()

```
def yaji.app.Application.add_requestparam_type_converter (
    self,
    paramtype,
    converter,
    None )
```

Adds a custom type converter for request params.

#### Parameters

<i>paramtype</i>	The python type.
<i>converter</i>	A function which converts a string coming from the web client to a <i>paramtype</i> .

### 1.39.3.19 add\_staticfile\_location()

```
def yaji.app.Application.add_staticfile_location (
    self,
    path,
    None )
```

Adds a local file path as an additional location for reading static files from.

Whenever static files are requested (e.g. by the browser requesting `/static/...`), Yaji tries to find it in one of those locations (in insertion order).

#### Parameters

<i>path</i>	The root path to add as static files source.
-------------	--

### 1.39.3.20 add\_translations()

```
def yaji.app.Application.add_translations (
    self,
    stringname,
    texts,
    None )
```

Adds translations in some languages for a string.

#### Parameters

<i>stringname</i>	The text stringname, as used in <a href="#">i18n.TrStr</a> .
<i>texts</i>	Different translations by two-letter language code.

**1.39.3.21 binddata()**

```
def yaji.app.Application.binddata (
    self,
    datasource,
    data_direction = None,
    convertername = None )
```

Declares binding a property of a `View` to an application code side or local browser side data source.

This is similar to [bindserver\(\)](#) and [bindlocal\(\)](#), but detects automatically if this is a server or local binding. See also [create\\_datastore\(\)](#), [create\\_clientlocal\\_datastore\(\)](#).

**Parameters**

<i>datasource</i>	The data source name or <code>DataStore/BrowserSideDatasource</code> . This data source must already exist.
<i>data_direction</i>	The data flow direction. See <code>BindDirection</code> .
<i>convertername</i>	The converter name.

**1.39.3.22 bindevent()**

```
def yaji.app.Application.bindevent (
    self,
    function,
    curtain = None )
```

Declares binding an event property of a `View` to a application code side or local browser side action.

This triggers the execution of that action whenever that event is triggered.

**Parameters**

<i>function</i>	The function to bind to this event.
<i>curtain</i>	A curtain expression (pointing to a <code>yaji.Curtain</code> in browser side javascript), or <code>True</code> for the default curtain (which might not be a good solution).

**1.39.3.23 bindlocal()**

```
def yaji.app.Application.bindlocal (
    self,
    datasource,
    data_direction = None,
    convertername = None )
```

Declares binding a property of a `View` to a local browser side data source.

Note: You should use [binddata\(\)](#) instead, unless you want to use it with a data source name which might not exist yet.

#### Parameters

<i>datasource</i>	The data source name or <code>BrowserSideDatasource</code> .
<i>data_direction</i>	The data flow direction. See <code>BindDirection</code> .
<i>convertername</i>	The converter name.

#### 1.39.3.24 bindserver()

```
def yaji.app.Application.bindserver (
    self,
    datasource,
    data_direction = None,
    convertername = None )
```

Declares binding a property of a `View` to a application code side data source.

Note: You typically should use [binddata\(\)](#) instead.

#### Parameters

<i>datasource</i>	The data source name or <code>DataStore</code> .
<i>data_direction</i>	The data flow direction. See <code>BindDirection</code> .
<i>convertername</i>	The converter name.

#### 1.39.3.25 create\_clientlocal\_datastore()

```
def yaji.app.Application.create_clientlocal_datastore (
    self,
    name = None,
    BrowserSideDatasource )
```

Creates a new browserside data source and returns it.

See also [get\\_clientlocal\\_datastore\(\)](#), [bindlocal\(\)](#).

#### Parameters

<i>name</i>	The datastore name (e.g. used in <code>yaji.getClientlocalDatastore()</code> on browser side). Default is a random name.
-------------	--



**1.39.3.26 create\_datastore()**

```
def yaji.app.Application.create_datastore (
    self,
    name = None,
    curtain = None,
    DataStore )
```

Creates a new (real serverside) DataStore and returns it.

See also [get\\_datastore\(\)](#), [bindserver\(\)](#).

**Parameters**

<i>name</i>	The datastore name (e.g. used in <a href="#">yaji.getDatastore()</a> on browser side). Default is a random name.
<i>curtain</i>	A curtain expression (pointing to a <a href="#">yaji.Curtain</a> in browser side javascript), or <code>True</code> for the default curtain (which might not be a good solution).

**1.39.3.27 create\_dialog()**

```
def yaji.app.Application.create_dialog (
    self,
    cfg,
    showcfg,
    Dialog )
```

Creates a dialog.

**Parameters**

<i>cfg</i>	The view specification.
<i>showcfg</i>	Additional configuration for dialog presentation.

**1.39.3.28 do\_stop\_implicitly\_when\_browser\_closed()**

```
def yaji.app.Application.do_stop_implicitly_when_browser_closed (
    self,
    value = True,
    ContextManager )
```

Use such instances via `with` for temporarily enabling implicit application stop when browser closed.

**Parameters**

<i>value</i>	If <code>False</code> , does the reverse this, same as <a href="#">dont_stop_implicitly_when_browser_closed()</a> .
--------------	---

### 1.39.3.29 enable\_authentication()

```
def yaji.app.Application.enable_authentication (
    self,
    authfct,
    realm = None,
    None )
```

Enables authentication by a custom authenticator function.

#### Parameters

<i>authfct</i>	The authenticator function: f(username, password) -> True iff authenticated.
<i>realm</i>	The http auth realm name.

### 1.39.3.30 get\_clientlocal\_datastore()

```
def yaji.app.Application.get_clientlocal_datastore (
    self,
    name,
    create_if_not_exist = True,
    Optional,
    BrowserSideDatasource )
```

Returns a browserside data source by name.

If one exists, it returns that, otherwise see parameters. See also [create\\_clientlocal\\_datastore\(\)](#), [bindlocal\(\)](#).

#### Parameters

<i>name</i>	The datastore name.
<i>create_if_not_exist</i>	If to create a new one if no one exists with that name yet (otherwise returns None).

### 1.39.3.31 get\_datastore()

```
def yaji.app.Application.get_datastore (
    self,
    name,
    create_if_not_exist = True,
    Optional,
    DataStore )
```

Returns a (real serverside) DataStore by name.

If one exists, it returns that, otherwise see parameters. See also [create\\_datastore\(\)](#), [bindserver\(\)](#).

## Parameters

<i>name</i>	The datastore name.
<i>create_if_not_exist</i>	If to create a new one if no one exists with that name yet (otherwise returns <code>None</code> ).

1.39.3.32 `get_translations()`

```
def yaji.app.Application.get_translations (
    self,
    stringname,
    texts,
    Dict,
    str,
    str )
```

Returns translations in all available languages for a string.

See also [add\\_translations\(\)](#).

## Parameters

<i>stringname</i>	The text stringname.
-------------------	----------------------

1.39.3.33 `get_translations_stringnames()`

```
def yaji.app.Application.get_translations_stringnames (
    self,
    List,
    str )
```

Returns all available stringnames for translations.

See also [add\\_translations\(\)](#).

1.39.3.34 `icon()`

```
def yaji.app.Application.icon (
    self,
    Optional,
    Icon )
```

The application icon.

You may set it to a path to a png file (relative to `/static` as the web browser sees it). See [add\\_staticfile\\_location\(\)](#).

#### 1.39.3.35 isstaticapplication()

```
def yaji.app.Application.isstaticapplication (
    self,
    bool )
```

Returns if this application is currently running as static one.

See [storeasstaticapplication\(\)](#).

#### 1.39.3.36 mainview\_icon()

```
def yaji.app.Application.mainview_icon (
    self,
    Optional,
    Icon )
```

The mainview header icon.

The application icon is used if no mainview icon is set.

You may set it to a path to a png file (relative to `/static` as the web browser sees it). See [add\\_staticfile\\_location\(\)](#).

#### 1.39.3.37 onbrowserreopened()

```
def yaji.app.Application.onbrowserreopened (
    self,
    None )
```

Reacts on the fact that the browser was opened again.

#### 1.39.3.38 oninitialize()

```
def yaji.app.Application.oninitialize (
    self,
    None )
```

Initializes the application.

This is called during application startup, so later than `__init__` (or never if the app doesn't start). Note: You should just override `__init__` instead if possible!

#### 1.39.3.39 onopenbrowsererror()

```
def yaji.app.Application.onopenbrowsererror (
    self,
    None )
```

Reacts on errors when opening the browser.

#### 1.39.3.40 onopenbrowserinformationoutput()

```
def yaji.app.Application.onopenbrowserinformationoutput (
    self,
    kind,
    message,
    None )
```

Reacts on information output (e.g.

by printing it to stdout) while opening the browser.

##### Parameters

<i>kind</i>	The internal code of the message.
<i>message</i>	The message text.

#### 1.39.3.41 onprocessrequesterror()

```
def yaji.app.Application.onprocessrequesterror (
    self,
    request,
    error,
    None )
```

Reacts on process request errors, e.g.

by logging.

##### Parameters

<i>request</i>	The path part of the request url.
<i>error</i>	The Exception.

#### 1.39.3.42 onunhandledclienterror()

```
def yaji.app.Application.onunhandledclienterror (
    self,
    error,
    src,
    line,
    bool )
```

Reacts on unhandled client errors.

Returning `True` makes the application shut down.

**Parameters**

<i>error</i>	An error description (might be empty).
<i>src</i>	The source file where the error occurred (might be empty).
<i>line</i>	The line in the source file where the error occurred (might be empty).

**1.39.3.43 setactions()**

```
def yaji.app.Application.setactions (
    self,
    actions,
    None )
```

Sets the main menu actions.

**Parameters**

<i>actions</i>	List or datasource of actions.
----------------	--------------------------------

**1.39.3.44 setbodyleft()**

```
def yaji.app.Application.setbodyleft (
    self,
    cfg,
    None )
```

Sets the left main body widget.

**Parameters**

<i>cfg</i>	A view specification.
------------	-----------------------

**1.39.3.45 setbodyright()**

```
def yaji.app.Application.setbodyright (
    self,
    cfg,
    None )
```

Sets the right main body widget.

## Parameters

<i>cfg</i>	A view specification.
------------	-----------------------

## 1.39.3.46 setheadcontrol()

```
def yaji.app.Application.setheadcontrol (
    self,
    cfg,
    None )
```

Sets the head control widget.

## Parameters

<i>cfg</i>	A view specification.
------------	-----------------------

## 1.39.3.47 setsidebar()

```
def yaji.app.Application.setsidebar (
    self,
    cfg,
    None )
```

Sets the sidebar widget.

## Parameters

<i>cfg</i>	A view specification.
------------	-----------------------

## 1.39.3.48 setsplitterposition()

```
def yaji.app.Application.setsplitterposition (
    self,
    v,
    None )
```

Sets the main body widgets splitter position.

## Parameters

<i>v</i>	The splitter position (from left 0 . 0 to right 1 . 0).
----------	---

### 1.39.3.49 showonly()

```
def yaji.app.Application.showonly (
    self,
    Optional,
    int )
```

Switches to show only the left or right main body widget.

Is 0 for left only, 1 for right only, None for both.

### 1.39.3.50 start()

```
def yaji.app.Application.start (
    self,
    None )
```

Starts the application.

Note: If your instance was already started and stopped, you have to create a new instance for another run.

### 1.39.3.51 stop()

```
def yaji.app.Application.stop (
    self,
    None )
```

Stops the application.

- It requests the parent application's return takeover procedure in some situations (see later),
- then invokes stopping the user interface (triggers `_yj_stopui`),
  - which on browser side leads to 'uiShutdown' and back redirection to parent application in some situations
- then shuts down the http server.
  - so the `_yj_stopui` event might never actually arrive, depending on timings (not an actual problem).

A `uiShutdown` disables the user interface by triggering the `OnUiShutdown` event (but only one time; further calls are ignored). `Application` code can register custom handlers to this event, but it does at least this:

- visually disables the user interface and shows a 'shut down' text if `skip_shutdown_dialog==False` (the default)
- if it happened due to the user tried to close a `PyHtmlView` and `show_browser_closed_notification==True` (not the default): makes a '`_yj_application_tryclosebrowser`' request, so `PyHtmlView` support can close the window or parent applications can take control back



The browser side also runs `uiShutdown` e.g. when

- `_yj_pullevent` requests fail (assumption: the application code side has stopped), or
- after a `yaji.stop()` was called (see later).

A `uiShutdown` does not stop stuff like event polling, and will leave `yaji.isrunning==True`! The browser side `yaji.stop()` does the following:

- requests `'_yj_application_stop'`, which calls `Application.stop()` on application code side
  - when backend answered: running `uiShutdown`
  - with disabled user interface during the request if
    - \* `skip_shutdown_dialog==False` (the default) or
    - \* it was explicitly called this way

Browser side calls `yaji.stop()` e.g. in those situations:

- custom application code on browser side triggered it for application exit
- a `'_yj_unhandled_client_error'` occurred and the backend decided for shutdown (which is not the default)
- by `PyHtmlView` support when the user tries to close the window

There is a flag `yaji.isrunning` which is `True` if the backend application is not known to be stopped so far, but might even be `True` after a `uiShutdown`. It will become `False` when

- `_yj_pullevent` requests fail (assumption: the application code side has stopped), or
- the `_yj_stopui` (see above) event was received.

#### 1.39.3.52 storeasstaticapplication()

```
def yaji.app.Application.storeasstaticapplication (
    self,
    targetdir,
    None )
```

Stores the application as static files.

Note: The application has to support that, since there are some pitfalls.

##### Parameters

<code>targetdir</code>	The target directory path.
------------------------	----------------------------

**1.39.3.53 triggerevent()**

```
def yaji.app.Application.triggerevent (
    self,
    name,
    eventdata,
    None )
```

Triggers an event on browser side.

**Parameters**

<i>name</i>	The event name (e.g. used in <code>yaji.addEventHandler()</code> on browser side).
<i>eventdata</i>	Additional event data to pass.

**1.39.3.54 try\_addclientresources\_from\_default\_location()**

```
def yaji.app.Application.try_addclientresources_from_default_location (
    self,
    fpath,
    None )
```

Tries to add a style file and script file from a file naming convention, so you don't have to call `add_clientstyle/add_clientscript()`.

If you pass `"/some/path/foo.py"`, it tries to load `"/some/path/__foo.css"` (and `.js`).

**Parameters**

<i>fpath</i>	Full path of your module file. You could use <code>__file__</code> for that.
--------------	--

**1.39.3.55 userfeedback()**

```
def yaji.app.Application.userfeedback (
    self,
    UserFeedbackController )
```

User feedback handler (for message boxes, ...).

Note: You must not use it in usual request handlers, but only in downstream ones! See [reqhandler.RequestHandler.run\\_downstream\(\)](#).

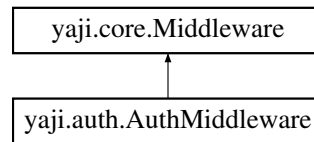
The documentation for this class was generated from the following file:

- `yaji/app.py`

## 1.40 yaji.auth.AuthMiddleware Class Reference

Middleware for authentication.

Inheritance diagram for yaji.auth.AuthMiddleware:



### Public Member Functions

- def `__init__` (self, realm)

### Static Public Member Functions

- def `request_processor` (index=0)  
*Marks a function as a request processor.*

### Private Member Functions

- def `_authenticate_by_password` (self, username, password)  
*This method authenticates a user by a password.*

### 1.40.1 Detailed Description

Middleware for authentication.

### 1.40.2 Constructor & Destructor Documentation

#### 1.40.2.1 `__init__`()

```
def yaji.auth.AuthMiddleware.__init__ (
    self,
    realm )
```

#### Parameters

<code>realm</code>	The http auth realm name.
--------------------	---------------------------

### 1.40.3 Member Function Documentation

#### 1.40.3.1 `_authenticate_by_password()`

```
def yaji.auth.AuthMiddleware._authenticate_by_password (
    self,
    username,
    password,
    bool ) [private]
```

This method authenticates a user by a password.

#### Returns

: True for successful authentication.

#### 1.40.3.2 `request_processor()`

```
def yaji.core.Middleware.request_processor (
    index = 0 ) [static], [inherited]
```

Marks a function as a request processor.

It takes a `Request` parameter and reads and/or modifies parts of it.

#### Parameters

<i>index</i>	The execution order index (low value: early execution).
--------------	---

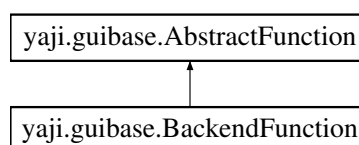
The documentation for this class was generated from the following file:

- `yaji/auth.py`

## 1.41 `yaji.guibase.BackendFunction` Class Reference

A callable application code side function.

Inheritance diagram for `yaji.guibase.BackendFunction`:



## Public Member Functions

- def `__init__` (self, url, funcargs=None)

### 1.41.1 Detailed Description

A callable application code side function.

### 1.41.2 Constructor & Destructor Documentation

#### 1.41.2.1 `__init__`()

```
def yaji.guibase.BackendFunction.__init__ (
    self,
    url,
    funcargs = None )
```

#### Parameters

<i>url</i>	The url to trigger for executing this action.
<i>funcargs</i>	Function arguments.

The documentation for this class was generated from the following file:

- yaji/guibase.py

## 1.42 yaji.guibase.BindDirection Class Reference

Data flow directions of data bindings.

### Static Public Attributes

- string `Bidirectional` = "bidirectional"  
*Bidirectional data transfer.*
- string `ToWidget` = "towidget"  
*Data transfer only from datasource to widget.*
- string `ToDatasource` = "todatasource"  
*Data transfer only from widget to datasource.*

### 1.42.1 Detailed Description

Data flow directions of data bindings.

## 1.42.2 Member Data Documentation

### 1.42.2.1 Bidirectional

```
yaji.guibase.BindDirection.Bidirectional = "bidirectional" [static]
```

Bidirectional data transfer.

### 1.42.2.2 ToDatasource

```
yaji.guibase.BindDirection.ToDatasource = "todatasource" [static]
```

Data transfer only from widget to datasource.

### 1.42.2.3 ToWidget

```
yaji.guibase.BindDirection.ToWidget = "towidget" [static]
```

Data transfer only from datasource to widget.

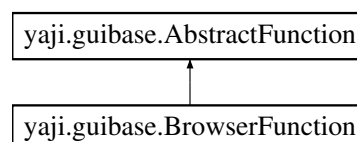
The documentation for this class was generated from the following file:

- `yaji/guibase.py`

## 1.43 yaji.guibase.BrowserFunction Class Reference

A callable browser side function.

Inheritance diagram for `yaji.guibase.BrowserFunction`:



### Public Member Functions

- `def \_\_init\_\_(self, funcname, funcargs=None)`

### 1.43.1 Detailed Description

A callable browser side function.

### 1.43.2 Constructor & Destructor Documentation

#### 1.43.2.1 \_\_init\_\_()

```
def yaji.guibase.BrowserFunction.__init__ (
    self,
    funcname,
    funcargs = None )
```

#### Parameters

<i>funcname</i>	The name of the function to trigger on browser side.
<i>funcargs</i>	Function arguments.

The documentation for this class was generated from the following file:

- yaji/guibase.py

## 1.44 yaji.browser.BrowserHook Class Reference

Helper for opening browsers.

### Public Member Functions

- def [\\_\\_init\\_\\_](#) (self, url, app)
- def [heartbeat](#) (self)  
*Gives a browser heartbeat (so the application code side knows that the browser is still alive).*
- def [start](#) (self)  
*Starts the hook.*
- def [stop](#) (self)  
*Stops the hook.*

### 1.44.1 Detailed Description

Helper for opening browsers.

## 1.44.2 Constructor & Destructor Documentation

### 1.44.2.1 `__init__()`

```
def yaji.browser.BrowserHook.__init__ (
    self,
    url,
    app )
```

#### Parameters

<i>url</i>	The url to open.
------------	------------------

The documentation for this class was generated from the following file:

- `yaji/browser.py`

## 1.45 `yaji.guibase.BrowserSideDatasource` Class Reference

A browser side datasource.

### Public Member Functions

- `def name (self)`  
*The data source name.*

### 1.45.1 Detailed Description

A browser side datasource.

You can only edit it before pushing it to browser side, and only inside a `with` block.

The documentation for this class was generated from the following file:

- `yaji/guibase.py`

## 1.46 `yaji.datastore.DataStore` Class Reference

A clove-like datastore which then becomes available on browser side as well.



## Classes

- class [ValuePointer](#)

*A value pointer is kind of an address or reference to a node (i.e.*

## Public Member Functions

- def [\\_\\_init\\_\\_](#) (self, [name](#), app, [curtain](#)=None, [\\_editor\\_for](#)=None)
- def [add\\_onchanged\\_handler](#) (self, handler)  
*Add a custom handler for changes in this datastore.*
- def [name](#) (self)  
*The datastore name.*
- def [curtain](#) (self)  
*The datastore curtain expression (pointing to a yaji.Curtain in browser side javascript).*
- def [rootnode](#) (self)  
*The root node.*
- def [rowcount](#) (self, ptr=None)  
*Returns the row count of a node (by [ValuePointer](#)).*
- def [columncount](#) (self, ptr=None)  
*Returns the column count of a node (by [ValuePointer](#)).*
- def [appendrow](#) (self, ptr=None, values=None)  
*Appends a row (like inserting to the end).*
- def [appendcolumn](#) (self, ptr=None, values=None)  
*Appends a column (like inserting to the end).*
- def [insertrow](#) (self, i, ptr=None, values=None)  
*Inserts a row.*
- def [insertcolumn](#) (self, i, ptr=None, values=None)  
*Inserts a column.*
- def [removerow](#) (self, i, ptr=None)  
*Removes a row.*
- def [removecolumn](#) (self, i, ptr=None)  
*Removes a column.*
- def [getvalue](#) (self, ptr=None)  
*Returns the value of a node (by [ValuePointer](#)).*

## Static Public Member Functions

- def [getnodebyid](#) (ajaxid)  
*Returns the node for an ajax id.*

### 1.46.1 Detailed Description

A clove-like datastore which then becomes available on browser side as well.

### 1.46.2 Member Function Documentation

#### 1.46.2.1 add\_onchanged\_handler()

```
def yaji.datastore.DataStore.add_onchanged_handler (
    self,
    handler,
    None )
```

Add a custom handler for changes in this datastore.

##### Parameters

<i>handler</i>	A handler function.
----------------	---------------------

#### 1.46.2.2 appendcolumn()

```
def yaji.datastore.DataStore.appendcolumn (
    self,
    ptr = None,
    values = None,
    Optional,
    Node )
```

Appends a column (like inserting to the end).

##### Parameters

<i>ptr</i>	The parent node (by <a href="#">ValuePointer</a> ).
<i>values</i>	An optional list of values to put into the new nodes.

#### 1.46.2.3 appendrow()

```
def yaji.datastore.DataStore.appendrow (
    self,
    ptr = None,
    values = None,
    Optional,
    Node )
```

Appends a row (like inserting to the end).

##### Parameters

<i>ptr</i>	The parent node (by <a href="#">ValuePointer</a> ).
<i>values</i>	An optional list of values to put into the new nodes.

## 1.46.2.4 columncount()

```
def yaji.datastore.DataStore.columncount (
    self,
    ptr = None,
    int )
```

Returns the column count of a node (by [ValuePointer](#)).

## Parameters

<i>ptr</i>	The value pointer of the node to query.
------------	---

## 1.46.2.5 getvalue()

```
def yaji.datastore.DataStore.getvalue (
    self,
    ptr = None,
    Optional,
    Any )
```

Returns the value of a node (by [ValuePointer](#)).

## Parameters

<i>ptr</i>	The value pointer of the node to query.
------------	---

## 1.46.2.6 insertcolumn()

```
def yaji.datastore.DataStore.insertcolumn (
    self,
    i,
    ptr = None,
    values = None,
    Optional,
    Node )
```

Inserts a column.

## Parameters

<i>i</i>	The position to insert the column to.
<i>ptr</i>	The parent node (by <a href="#">ValuePointer</a> ).
<i>values</i>	An optional list of values to put into the new nodes.

#### 1.46.2.7 insertrow()

```
def yaji.datastore.DataStore.insertrow (
    self,
    i,
    ptr = None,
    values = None,
    Optional,
    Node )
```

Inserts a row.

##### Parameters

<i>i</i>	The position to insert the row to.
<i>ptr</i>	The parent node (by <a href="#">ValuePointer</a> ).
<i>values</i>	An optional list of values to put into the new nodes.

#### 1.46.2.8 removecolumn()

```
def yaji.datastore.DataStore.removecolumn (
    self,
    i,
    ptr = None,
    None )
```

Removes a column.

##### Parameters

<i>i</i>	The position to remove the column from.
<i>ptr</i>	The parent node (by <a href="#">ValuePointer</a> ).

#### 1.46.2.9 removerow()

```
def yaji.datastore.DataStore.removerow (
    self,
    i,
    ptr = None,
    None )
```

Removes a row.

##### Parameters

<i>i</i>	The position to remove the row from.
<i>ptr</i>	The parent node (by <a href="#">ValuePointer</a> ).

## 1.46.2.10 rowcount()

```
def yaji.datastore.DataStore.rowcount (
    self,
    ptr = None,
    int )
```

Returns the row count of a node (by [ValuePointer](#)).

## Parameters

<i>ptr</i>	The value pointer of the node to query.
------------	---

The documentation for this class was generated from the following file:

- yaji/datastore.py

## 1.47 yaji.gui.Dialog Class Reference

A dialog.

## Public Member Functions

- def [\\_\\_init\\_\\_](#) (self, ctl, cfg, showcfg)
- def [dialogid](#) (self)  
*The dialog id.*
- def [show](#) (self)  
*Shows the dialog.*
- def [close](#) (self)  
*Closes the dialog.*
- def [add\\_closed\\_handler](#) (self, fct)  
*Adds a closed handler.*

## Private Member Functions

- def [\\_set\\_dialogid](#) (self, dialogid)  
*Sets the dialog id.*
- def [\\_exec\\_closed\\_handlers](#) (self)  
*Executes the closed handlers.*

## 1.47.1 Detailed Description

A dialog.

## 1.47.2 Constructor & Destructor Documentation

### 1.47.2.1 `__init__()`

```
def yaji.gui.Dialog.__init__ (
    self,
    ctl,
    cfg,
    showcfg )
```

#### Parameters

<i>ctl</i>	The dialog controller.
<i>cfg</i>	The dialog view configuration.
<i>showcfg</i>	Additional configuration for dialog presentation.

## 1.47.3 Member Function Documentation

### 1.47.3.1 `add_closed_handler()`

```
def yaji.gui.Dialog.add_closed_handler (
    self,
    fct,
    None )
```

Adds a closed handler.

It gets executed when the dialog closes.

### 1.47.3.2 `dialogid()`

```
def yaji.gui.Dialog.dialogid (
    self,
    int )
```

The dialog id.

The documentation for this class was generated from the following file:

- `yaji/gui.py`

## 1.48 yaji.dialogcontroller.DialogController Class Reference

A controller for dialogs.

### Public Member Functions

- def [show\\_dialog](#) (self, dialog)  
*Shows a dialog.*
- def [close\\_dialog](#) (self, dialog)  
*Closes a dialog.*
- def [get\\_dialogs](#) (self)  
*Returns a list of all open dialogs.*
- def [get\\_dialog\\_by\\_id](#) (self, dialogid)  
*Returns a dialog by id.*

### 1.48.1 Detailed Description

A controller for dialogs.

### 1.48.2 Member Function Documentation

#### 1.48.2.1 close\_dialog()

```
def yaji.dialogcontroller.DialogController.close_dialog (  
    self,  
    dialog )
```

Closes a dialog.

#### Parameters

<i>dialog</i>	The dialog to close.
---------------	----------------------

#### 1.48.2.2 get\_dialog\_by\_id()

```
def yaji.dialogcontroller.DialogController.get_dialog_by_id (  
    self,  
    dialogid,  
    Optional,  
    Dialog )
```

Returns a dialog by id.

**Parameters**

<i>dialogid</i>	The dialog id.
-----------------	----------------

**1.48.2.3 show\_dialog()**

```
def yaji.dialogcontroller.DialogController.show_dialog (
    self,
    dialog )
```

Shows a dialog.

**Parameters**

<i>dialog</i>	The dialog to show.
---------------	---------------------

The documentation for this class was generated from the following file:

- yaji/dialogcontroller.py

**1.49 yaji.core.Directories Class Reference**

Some important paths.

**1.49.1 Detailed Description**

Some important paths.

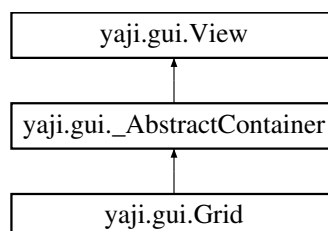
The documentation for this class was generated from the following file:

- yaji/core.py

**1.50 yaji.gui.Grid Class Reference**

A grid view.

Inheritance diagram for yaji.gui.Grid:





## Public Member Functions

- `def \_\_init\_\_ (self, children, b)`

### 1.50.1 Detailed Description

A grid view.

### 1.50.2 Constructor & Destructor Documentation

#### 1.50.2.1 `__init__()`

```
def yaji.gui.Grid.__init__ (
    self,
    children,
    b )
```

#### Parameters

<i>children</i>	The child views.
-----------------	------------------

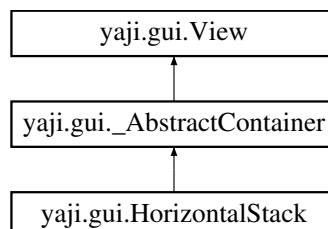
The documentation for this class was generated from the following file:

- `yaji/gui.py`

## 1.51 yaji.gui.HorizontalStack Class Reference

A horizontal stack view.

Inheritance diagram for `yaji.gui.HorizontalStack`:



## Public Member Functions

- `def \_\_init\_\_ (self, children, b)`

### 1.51.1 Detailed Description

A horizontal stack view.

### 1.51.2 Constructor & Destructor Documentation

#### 1.51.2.1 `__init__()`

```
def yaji.gui.HorizontalStack.__init__ (
    self,
    children,
    b )
```

##### Parameters

<code>children</code>	The child views.
-----------------------	------------------

The documentation for this class was generated from the following file:

- `yaji/gui.py`

## 1.52 `yaji.guibase.Icon` Class Reference

`Icon`.

### Public Member Functions

- `def __init__` (self, by\_url=None, by\_symbol=None)
- `def src` (self)  
*The icon source.*
- `def srcfunc` (self)  
*The icon source function.*

### 1.52.1 Detailed Description

`Icon`.

### 1.52.2 Constructor & Destructor Documentation

#### 1.52.2.1 `__init__()`

```
def yaji.guibase.Icon.__init__ (
    self,
    by_url = None,
    by_symbol = None )
```

## Parameters

<i>by_url</i>	A icon by image source url.
<i>by_symbol</i>	A icon by symbol character.

The documentation for this class was generated from the following file:

- yaji/guibase.py

## 1.53 yaji.core.Log Class Reference

Logging.

### Public Member Functions

- def `debug` (self, s, a)  
*Logs a debug message.*
- def `info` (self, s, a)  
*Logs an info message.*
- def `warning` (self, s, a)  
*Logs a warning message.*
- def `error` (self, s, a)  
*Logs an error message.*

### 1.53.1 Detailed Description

Logging.

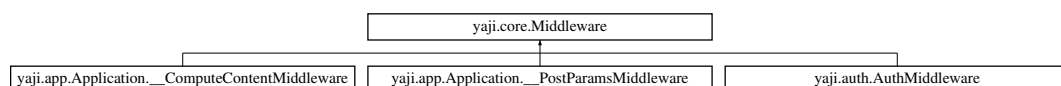
The documentation for this class was generated from the following file:

- yaji/core.py

## 1.54 yaji.core.Middleware Class Reference

A middleware controls some internal stuff like request processing in a custom way.

Inheritance diagram for yaji.core.Middleware:



## Static Public Member Functions

- def `request_processor` (index=0)  
*Marks a function as a request processor.*

### 1.54.1 Detailed Description

A middleware controls some internal stuff like request processing in a custom way.

See `Application.add_middleware()`.

### 1.54.2 Member Function Documentation

#### 1.54.2.1 `request_processor()`

```
def yaji.core.Middleware.request_processor (
    index = 0 ) [static]
```

Marks a function as a request processor.

It takes a `Request` parameter and reads and/or modifies parts of it.

#### Parameters

<i>index</i>	The execution order index (low value: early execution).
--------------	---

The documentation for this class was generated from the following file:

- `yaji/core.py`

## 1.55 `yaji.pyhtmlview.PyHtmlView` Class Reference

This optional functionality allows to present the user interface of the application in an own bare window instead of the usual browser.

## Static Public Member Functions

- def `openview` (show\_url, icon, wndclassname, scriptedclosing)  
*Opens a new view with some content.*
- def `is_system_compatible` ()  
*Tests if your system is compatible with `PyHtmlView`.*

### 1.55.1 Detailed Description

This optional functionality allows to present the user interface of the application in an own bare window instead of the usual browser.

It is used by default if not configured otherwise and if the system is compatible to it.

It requires PyQt to be available.

### 1.55.2 Member Function Documentation

#### 1.55.2.1 openview()

```
def yaji.pyhtmlview.PyHtmlView.openview (
    show_url,
    icon,
    wndclassname,
    scriptedclosing ) [static]
```

Opens a new view with some content.

#### Parameters

<i>show_url</i>	The url to show content from.
<i>icon</i>	The window icon.
<i>wndclassname</i>	The window class name.
<i>scriptedclosing</i>	A javascript expression to execute for closing.

The documentation for this class was generated from the following file:

- yaji/pyhtmlview.py

## 1.56 yaji.request.Request Class Reference

A request from the client, including ways to give the response.

### Classes

- class [\\_\\_InRequest](#)

*Context manager for setting the current request.*

## Public Member Functions

- `def __init__ (self, urlpath, header, application)`
- `def urlpath (self)`  
*The request url's complete path, like `"/foo/bar?id=13&p=a"`.*
- `def urlbarepath (self)`  
*The request url's path without query string, like `"/foo/bar"`.*
- `def urlpathquery (self)`  
*The request url's querystring, like `"id=13&p=a"`.*
- `def lparams (self)`  
*The request parameters as list (for having no or more than one value by parameter key).*
- `def params (self)`  
*The request parameters in a flat structure (not as list, contrary to `oparams`).*
- `def header (self)`  
*The request headers.*
- `def runs_downstream (self)`  
*If this request is handled downstream.*
- `def uieventdata (self)`  
*The event data dictionary which contains details about an ui event coming from the browser.*
- `def get_param (self, key, defaultval=None)`  
*Returns the value of a request parameter.*
- `def get_param_as_list (self, key)`  
*Returns the value of a request parameter as list (for having no or more than one value by parameter key).*
- `def response_body (self)`  
*The response body.*
- `def response_errortext (self)`  
*The error text (if an error occurred).*
- `def response_httpcode (self)`  
*The http response code.*
- `def set_response_header (self, key, value)`  
*Sets an http response header entry.*
- `def get_response_header (self, key)`  
*Returns the http response header entry as stored before with `set_response_header()`.*
- `def scratchpad (self)`  
*A dictionary for storing some custom things, typically by middleware.*
- `def application (self)`  
*The Application with got this request.*
- `def skip_processing (self)`  
*If to skip further request processing.*

## Static Public Member Functions

- `def current ()`  
*Returns the [Request](#) associated to the current request (in a request handler).*

## Private Member Functions

- `def _set_lparam (self, key, value)`  
*Sets a request parameter value.*
- `def _set_runs_downstream (self)`  
*Sets this request to be handled downstream.*
- `def _in_request (self)`  
*Returns a context manager for setting the current request.*

### 1.56.1 Detailed Description

A request from the client, including ways to give the response.

### 1.56.2 Constructor & Destructor Documentation

#### 1.56.2.1 `__init__()`

```
def yaji.request.Request.__init__ (
    self,
    urlpath,
    header,
    application )
```

##### Parameters

<i>urlpath</i>	The complete request url path, including query string.
<i>header</i>	The http request headers.

### 1.56.3 Member Function Documentation

#### 1.56.3.1 `_in_request()`

```
def yaji.request.Request._in_request (
    self,
    ContextManager ) [private]
```

Returns a context manager for setting the current request.

#### 1.56.3.2 `_set_lparam()`

```
def yaji.request.Request._set_lparam (
    self,
    key,
    value,
    None ) [private]
```

Sets a request parameter value.

Only infrastructure and middleware would use that.

**Parameters**

<i>key</i>	The parameter key.
<i>value</i>	The parameter values (as list).

**1.56.3.3 \_set\_runs\_downstream()**

```
def yaji.request.Request._set_runs_downstream (
    self,
    None ) [private]
```

Sets this request to be handled downstream.

**1.56.3.4 get\_param()**

```
def yaji.request.Request.get_param (
    self,
    key,
    defaultval = None,
    Optional,
    str )
```

Returns the value of a request parameter.

Note: This drops values beyond the first one, so don't use it if you *have* lists.

**Parameters**

<i>key</i>	The parameter key.
<i>defaultval</i>	The default value, if no such parameter exists.

**1.56.3.5 get\_param\_as\_list()**

```
def yaji.request.Request.get_param_as_list (
    self,
    key,
    List,
    str )
```

Returns the value of a request parameter as list (for having no or more than one value by parameter key).

**Parameters**

<i>key</i>	The parameter key.
------------	--------------------



### 1.56.3.6 get\_response\_header()

```
def yaji.request.Request.get_response_header (
    self,
    key,
    str )
```

Returns the http response header entry as stored before with [set\\_response\\_header\(\)](#).

#### Parameters

<i>key</i>	The header key.
------------	-----------------

### 1.56.3.7 params()

```
def yaji.request.Request.params (
    self,
    Dict,
    str,
    str )
```

The request parameters in a flat structure (not as list, contrary to `oparams`).

Note: This drops values beyond the first one, so don't use it if you *have* lists.

### 1.56.3.8 runs\_downstream()

```
def yaji.request.Request.runs_downstream (
    self,
    bool )
```

If this request is handled downstream.

See [reqhandler.RequestHandler.run\\_downstream\(\)](#).

### 1.56.3.9 set\_response\_header()

```
def yaji.request.Request.set_response_header (
    self,
    key,
    value,
    None )
```

Sets an http response header entry.

**Parameters**

<i>key</i>	The header key.
<i>value</i>	The header value.

**1.56.3.10 skip\_processing()**

```
def yaji.request.Request.skip_processing (
    self,
    bool )
```

If to skip further request processing.

Used by middleware.

**1.56.3.11 uieventdata()**

```
def yaji.request.Request.uieventdata (
    self,
    Dict,
    str,
    Optional,
    Any )
```

The event data dictionary which contains details about an ui event coming from the browser.

This only makes sense in a request handler for an event binding (see [Application.bindevent\(\)](#)) and is `None` otherwise.

The documentation for this class was generated from the following file:

- `yaji/request.py`

**1.57 yaji.reqhandler.RequestHandler Class Reference**

Decorators for request handler functions, i.e.

**Public Member Functions**

- `def for_urls (cls, urlpatterns, auto=False)`  
*Makes a method a request handler for a particular url pattern.*
- `def with_param_type (cls, paramname, paramtype)`  
*Defines a parameter of a request handler to be of a particular data type.*
- `def run_downstream (cls)`  
*Defines this request handler to run downstream.*

## Private Member Functions

- `def __compile_re_for_urlpattern (cls, urlpattern)`  
*Returns a compiled `re` regexp for a url pattern string.*

### 1.57.1 Detailed Description

Decorators for request handler functions, i.e.

methods of an Application which handle http requests.

### 1.57.2 Member Function Documentation

#### 1.57.2.1 \_\_compile\_re\_for\_urlpattern()

```
def yaji.reqhandler.RequestHandler.__compile_re_for_urlpattern (
    cls,
    urlpattern,
    Pattern ) [private]
```

Returns a compiled `re` regexp for a url pattern string.

##### Parameters

<i>urlpattern</i>	The url pattern string to convert.
-------------------	------------------------------------

#### 1.57.2.2 for\_urls()

```
def yaji.reqhandler.RequestHandler.for_urls (
    cls,
    urlpatterns,
    auto = False )
```

Makes a method a request handler for a particular url pattern.

A url pattern is a string like "`foos/<fooid>/bars/`", with "`<fooid>`" binding a parameter that you have to add to your handler function signature.

##### Parameters

<i>urlpatterns</i>	The url patterns to bind the request handler to (strings or compiled <code>re</code> regexps).
<i>auto</i>	If to construct the url pattern from the method name.

### 1.57.2.3 run\_downstream()

```
def yaji.reqhandler.RequestHandler.run_downstream (
    cls )
```

Defines this request handler to run downstream.

In this mode, the browser instantly gets an empty response and does not wait longer. This is recommended (and sometimes required; e.g. for user feedback) for operations which can take longer.

Note: This forbids to return any data from the handler, but it does not forbid to modify data sources, to set appconfig, and some other ways to transfer results.

### 1.57.2.4 with\_param\_type()

```
def yaji.reqhandler.RequestHandler.with_param_type (
    cls,
    paramname,
    paramtype )
```

Defines a parameter of a request handler to be of a particular data type.

For custom types, either see `add_do_param_type_converter()` or make your type constructor accept a (str) call. Note: You usually should use type annotations on request handlers instead of this function!

#### Parameters

<i>paramname</i>	The parameter name. This is the name in the <code>...&amp;name=...&amp;...</code> part of the url and of the argument of the request handler.
<i>paramtype</i>	The python type of this parameter. See also <code>Application.add_request_param_type_converter()</code> .

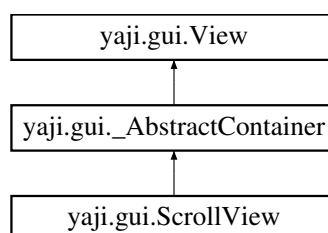
The documentation for this class was generated from the following file:

- `yaji/reqhandler.py`

## 1.58 yaji.gui.ScrollView Class Reference

A scroll view.

Inheritance diagram for `yaji.gui.ScrollView`:



## Public Member Functions

- `def \_\_init\_\_ (self, body, b)`

### 1.58.1 Detailed Description

A scroll view.

### 1.58.2 Constructor & Destructor Documentation

#### 1.58.2.1 `__init__()`

```
def yaji.gui.ScrollView.__init__ (
    self,
    body,
    b )
```

#### Parameters

<i>body</i>	The child view.
-------------	-----------------

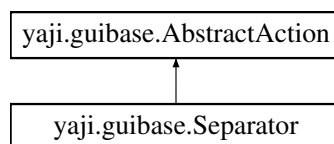
The documentation for this class was generated from the following file:

- `yaji/gui.py`

## 1.59 yaji.guibase.Separator Class Reference

A menu separator.

Inheritance diagram for `yaji.guibase.Separator`:



### 1.59.1 Detailed Description

A menu separator.

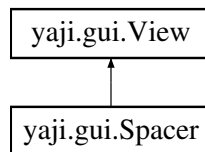
The documentation for this class was generated from the following file:

- `yaji/guibase.py`

## 1.60 yaji.gui.Spacer Class Reference

A spacer.

Inheritance diagram for yaji.gui.Spacer:



### 1.60.1 Detailed Description

A spacer.

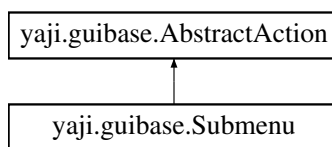
The documentation for this class was generated from the following file:

- yaji/gui.py

## 1.61 yaji.guibase.Submenu Class Reference

A submenu of a menu.

Inheritance diagram for yaji.guibase.Submenu:



### Public Member Functions

- def `__init__` (self, label, icon=None, disabled=False, invisible=False)

### 1.61.1 Detailed Description

A submenu of a menu.

### 1.61.2 Constructor & Destructor Documentation

#### 1.61.2.1 `__init__`()

```
def yaji.guibase.Submenu.__init__ (
    self,
    label,
    icon = None,
    disabled = False,
    invisible = False )
```

## Parameters

<i>label</i>	The label of this action.
<i>icon</i>	The action icon.
<i>disabled</i>	If this action is disabled (can't be triggered).
<i>invisible</i>	If this action is invisible.

The documentation for this class was generated from the following file:

- yaji/guibase.py

## 1.62 yaji.i18n.TrStr Class Reference

An i18n-aware string for displaying on browser side.

### Public Member Functions

- def `__init__` (self, [stringname](#), [args](#))
- def [stringname](#) (self)  
*The string name (as used for referencing the translations for one text).*
- def [args](#) (self)  
*String pieces for replacing "{foo}" patterns in the translation with.*

### 1.62.1 Detailed Description

An i18n-aware string for displaying on browser side.

When returned in a json structure, the browser side transparently gets a translated version.

### 1.62.2 Constructor & Destructor Documentation

#### 1.62.2.1 `__init__()`

```
def yaji.i18n.TrStr.__init__ (
    self,
    stringname,
    args )
```

## Parameters

<i>stringname</i>	The string name (as used in <code>clove.i18n.addString()</code> on browser side).
<i>args</i>	String pieces for replacing "{foo}" patterns in the translation with.

The documentation for this class was generated from the following file:

- `yaji/i18n.py`

## 1.63 `yaji.userfeedback.UserFeedbackController` Class Reference

A controller for user feedback operations.

### Public Member Functions

- `def __init__ (self, app)`
- `def make_raw_request (self, kind, params)`  
*Makes a low-level userfeedback request.*
- `def messagedialog (self, message, buttons=None)`  
*Shows a message dialog to the user and returns the index of the selected button.*
- `def inputdialog (self, question, defaultanswer="")`  
*Shows an input dialog to the user and returns the entered text (or `None` when cancelled).*
- `def choicedialog (self, question, choices)`  
*Shows a choice dialog to the user and returns the selected item (or `None` when cancelled).*
- `def multilineinputdialog (self, question, defaultanswer="")`  
*Like `inputdialog()` but multi-line capable.*
- `def filesystemdialog (self, fstype="file", question="", startpath="/")`  
*Shows a file/directory selection dialog to the user and returns the path to the selected item (or `None`).*
- `def passworddialog (self, question, defaultanswer="")`  
*Shows a password dialog to the user and returns the password (or `None`).*

### Private Member Functions

- `def _interpret_response (self, request)`  
*Processes a userfeedback response from browser side.*

#### 1.63.1 Detailed Description

A controller for user feedback operations.

Use methods inside it for communicating with the user during some operations. Such an object is automatically available during execution, there is no need to create new ones.

#### 1.63.2 Member Function Documentation

##### 1.63.2.1 `_interpret_response()`

```
def yaji.userfeedback.UserFeedbackController._interpret_response (
    self,
    request,
    None ) [private]
```

Processes a userfeedback response from browser side.



## Parameters

<i>request</i>	A client request which contains a userfeedback answer.
----------------	--

## 1.63.2.2 choicedialog()

```
def yaji.userfeedback.UserFeedbackController.choicedialog (
    self,
    question,
    choices,
    Optional,
    int )
```

Shows a choice dialog to the user and returns the selected item (or `None` when cancelled).

## Parameters

<i>question</i>	The question text to show.
<i>choices</i>	The list of choices the user has to select from.

## 1.63.2.3 filesystemdialog()

```
def yaji.userfeedback.UserFeedbackController.filesystemdialog (
    self,
    fstype = "file",
    question = "",
    startpath = "/",
    Optional,
    str )
```

Shows a file/directory selection dialog to the user and returns the path to the selected item (or `None`).

## Parameters

<i>fstype</i>	The type of filesystem items to select.
<i>question</i>	The question text to show.
<i>startpath</i>	The directory path to start in.

## 1.63.2.4 inputdialog()

```
def yaji.userfeedback.UserFeedbackController.inputdialog (
    self,
```

```

    question,
    defaultanswer = "",
    Optional,
    str )

```

Shows an input dialog to the user and returns the entered text (or `None` when cancelled).

#### Parameters

<i>question</i>	The question text to show.
<i>defaultanswer</i>	The default answer text which is written to the text field when showing.

#### 1.63.2.5 make\_raw\_request()

```

def yaji.userfeedback.UserFeedbackController.make_raw_request (
    self,
    kind,
    params,
    Any )

```

Makes a low-level userfeedback request.

#### Parameters

<i>kind</i>	The raw userfeedback kind name as known on browser side. You can register custom kind handlers on browser side with <code>yaji.userfeedback.registerHandler()</code> .
<i>params</i>	Some <code>kind</code> -specific parameters.

#### 1.63.2.6 messagedialog()

```

def yaji.userfeedback.UserFeedbackController.messagedialog (
    self,
    message,
    buttons = None,
    int )

```

Shows a message dialog to the user and returns the index of the selected button.

#### Parameters

<i>message</i>	The message text to show.
<i>buttons</i>	The buttons to offer.

## 1.63.2.7 multilineinputdialog()

```
def yaji.userfeedback.UserFeedbackController.multilineinputdialog (
    self,
    question,
    defaultanswer = "",
    Optional,
    str )
```

Like [inputdialog\(\)](#) but multi-line capable.

## Parameters

<i>question</i>	The question text to show.
<i>defaultanswer</i>	The default answer text which is written to the text field when showing.

## 1.63.2.8 passworddialog()

```
def yaji.userfeedback.UserFeedbackController.passworddialog (
    self,
    question,
    defaultanswer = "",
    Optional,
    str )
```

Shows a password dialog to the user and returns the password (or `None`).

## Parameters

<i>question</i>	The question text to show.
<i>defaultanswer</i>	The default answer text which is written to the password field when showing.

The documentation for this class was generated from the following file:

- `yaji/userfeedback.py`

## 1.64 yaji.datastore.DataStore.ValuePointer Class Reference

A value pointer is kind of an address or reference to a node (i.e.

## 1.64.1 Detailed Description

A value pointer is kind of an address or reference to a node (i.e.

a location) inside a datastore. Note: The root node is usually referenced by `None`.

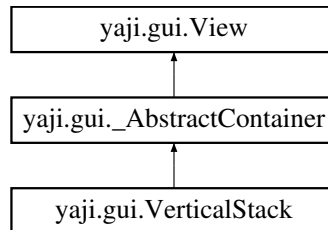
The documentation for this class was generated from the following file:

- `yaji/datastore.py`

## 1.65 yaji.gui.VerticalStack Class Reference

A vertical stack view.

Inheritance diagram for yaji.gui.VerticalStack:



### Public Member Functions

- `def \_\_init\_\_ (self, children, b)`

#### 1.65.1 Detailed Description

A vertical stack view.

#### 1.65.2 Constructor & Destructor Documentation

##### 1.65.2.1 `__init__()`

```
def yaji.gui.VerticalStack.__init__ (
    self,
    children,
    b )
```

##### Parameters

<i>children</i>	The child views.
-----------------	------------------

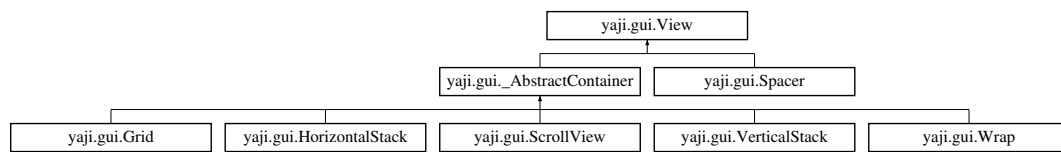
The documentation for this class was generated from the following file:

- `yaji/gui.py`

## 1.66 yaji.gui.View Class Reference

A widget build configuration for user interface definition.

Inheritance diagram for yaji.gui.View:



## Classes

- class [\\_DataBinding](#)  
*A data binding.*
- class [\\_EventBinding](#)  
*An event binding.*

## Public Member Functions

- def [\\_\\_init\\_\\_](#) (self, clsname=None, enabled=None, visibility=None, hstretch=None, strictHorizontalSizing=None, vstretch=None, strictVerticalSizing=None, styleClass=None, width=None, height=None, b)

### 1.66.1 Detailed Description

A widget build configuration for user interface definition.

Used e.g. as parameters in some `Application` methods. See also the sample applications.

### 1.66.2 Constructor & Destructor Documentation

#### 1.66.2.1 `__init__()`

```

def yaji.gui.View.__init__ (
    self,
    clsname = None,
    enabled = None,
    visibility = None,
    hstretch = None,
    strictHorizontalSizing = None,
    vstretch = None,
    strictVerticalSizing = None,
    styleClass = None,
    width = None,
    height = None,
    b )
  
```

## Parameters

<i>clsname</i>	The clove widget class name. See clove documentation.
<i>enabled</i>	If the widget is enabled (instead of being locked for user interaction).
<i>visibility</i>	The visibility, as one of <a href="#">guibase.Visibility</a> .
<i>hstretch</i>	The horizontal stretch affinity factor.
<i>strictHorizontalSizing</i>	If to not overfulfill horizontal sizing requirements even for alignment purposes.
<i>vstretch</i>	The vertical stretch affinity factor.
<i>strictVerticalSizing</i>	If to not overfulfill vertical sizing requirements even for alignment purposes.
<i>styleClass</i>	Additional CSS classes. See also <a href="#">app.Application.add_clientstyle()</a> .
<i>width</i>	A fixed width (specified as in css).
<i>height</i>	A fixed height (specified as in css).
<i>b</i>	Additional configuration. Depends on the widget class.

The documentation for this class was generated from the following file:

- `yaji/gui.py`

## 1.67 yaji.guibase.Visibility Class Reference

The visibility of a widget.

### Static Public Attributes

- string `Visible` = "clove\_visible"  
*Visible.*
- string `Invisible` = "clove\_invisible"  
*Invisible but taking space.*
- string `InvisibleCollapsed` = "clove\_invisiblecollapsed"  
*Invisible and collapsed.*

### 1.67.1 Detailed Description

The visibility of a widget.

### 1.67.2 Member Data Documentation

#### 1.67.2.1 Invisible

```
yaji.guibase.Visibility.Invisible = "clove_invisible" [static]
```

Invisible but taking space.

### 1.67.2.2 InvisibleCollapsed

```
yaji.guibase.Visibility.InvisibleCollapsed = "clove_invisiblecollapsed" [static]
```

Invisible and collapsed.

### 1.67.2.3 Visible

```
yaji.guibase.Visibility.Visible = "clove_visible" [static]
```

Visible.

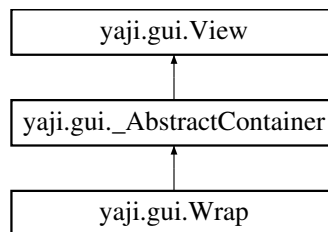
The documentation for this class was generated from the following file:

- yaji/guibase.py

## 1.68 yaji.gui.Wrap Class Reference

A wrap view.

Inheritance diagram for yaji.gui.Wrap:



### Public Member Functions

- def `__init__`(self, children, b)

### 1.68.1 Detailed Description

A wrap view.

### 1.68.2 Constructor & Destructor Documentation

#### 1.68.2.1 `__init__`()

```
def yaji.gui.Wrap.__init__ (
    self,
    children,
    b )
```

**Parameters**

<i>children</i>	The child views.
-----------------	------------------

The documentation for this class was generated from the following file:

- `yaji/gui.py`



# Index

- `__callhandler`
  - `yaji::app::Application::__ComputeContent↔Middleware`, [21](#)
- `__compile_re_for_urlpattern`
  - `yaji::reqhandler::RequestHandler`, [79](#)
- `__create_datastore_helper`
  - `yaji::app::Application`, [36](#)
- `__get_content`
  - `yaji::app::Application`, [36](#)
- `__get_datastore_helper`
  - `yaji::app::Application`, [37](#)
- `__get_requesthandler_for_url`
  - `yaji::app::Application`, [37](#)
- `__init__`
  - `yaji::app::Application`, [35](#)
  - `yaji::auth::AuthMiddleware`, [55](#)
  - `yaji::browser::BrowserHook`, [60](#)
  - `yaji::gui::_AbstractContainer`, [25](#)
  - `yaji::gui::Dialog`, [66](#)
  - `yaji::gui::Grid`, [69](#)
  - `yaji::gui::HorizontalStack`, [70](#)
  - `yaji::gui::ScrollView`, [81](#)
  - `yaji::gui::VerticalStack`, [88](#)
  - `yaji::gui::View`, [89](#)
  - `yaji::gui::View::_DataBinding`, [26](#)
  - `yaji::gui::View::_EventBinding`, [26](#)
  - `yaji::gui::Wrap`, [91](#)
  - `yaji::guibase::AbstractAction`, [27](#)
  - `yaji::guibase::Action`, [29](#)
  - `yaji::guibase::BackendFunction`, [57](#)
  - `yaji::guibase::BrowserFunction`, [59](#)
  - `yaji::guibase::Icon`, [70](#)
  - `yaji::guibase::Submenu`, [82](#)
  - `yaji::i18n::TrStr`, [83](#)
  - `yaji::request::Request`, [75](#)
- `__markinitialized`
  - `yaji::appconfig::AppConfig`, [30](#)
- `_authenticate_by_password`
  - `yaji::auth::AuthMiddleware`, [56](#)
- `_browser_was_reopened`
  - `yaji::app::Application`, [37](#)
- `_callhandler`
  - `yaji::app::Application`, [38](#)
- `_findhandler`
  - `yaji::app::Application`, [38](#)
- `_get_requestparam_type_converter`
  - `yaji::app::Application`, [38](#)
- `_get_rootpagecontent`
  - `yaji::app::Application`, [39](#)
- `_in_request`
  - `yaji::request::Request`, [75](#)
- `_interpret_response`
  - `yaji::userfeedback::UserFeedbackController`, [84](#)
- `_json_make_serializable`
  - `yaji::app::Application`, [39](#)
- `_middlewares`
  - `yaji::app::Application`, [39](#)
- `_openbrowser`
  - `yaji::app::Application`, [39](#)
- `_set_lparam`
  - `yaji::request::Request`, [75](#)
- `_set_runs_downstream`
  - `yaji::request::Request`, [76](#)
- `_staticfile_path_to_abspath`
  - `yaji::app::Application`, [40](#)
- `_tryclosebrowser`
  - `yaji::app::Application`, [40](#)
- `add_clientscript`
  - `yaji::app::Application`, [40](#)
- `add_clientstyle`
  - `yaji::app::Application`, [41](#)
- `add_closed_handler`
  - `yaji::gui::Dialog`, [66](#)
- `add_middleware`
  - `yaji::app::Application`, [41](#)
- `add_onchanged_handler`
  - `yaji::datastore::DataStore`, [61](#)
- `add_requestparam_type_converter`
  - `yaji::app::Application`, [42](#)
- `add_staticfile_location`
  - `yaji::app::Application`, [42](#)
- `add_translations`
  - `yaji::app::Application`, [42](#)
- `appendcolumn`
  - `yaji::datastore::DataStore`, [62](#)
- `appendrow`
  - `yaji::datastore::DataStore`, [62](#)
- `Bidirectional`
  - `yaji::guibase::BindDirection`, [58](#)
- `binddata`
  - `yaji::app::Application`, [43](#)
- `bindevent`
  - `yaji::app::Application`, [43](#)
- `bindlocal`
  - `yaji::app::Application`, [43](#)
- `bindserver`
  - `yaji::app::Application`, [44](#)

- choicedialog
  - yaji::userfeedback::UserFeedbackController, 85
- close\_dialog
  - yaji::dialogcontroller::DialogController, 67
- columncount
  - yaji::datastore::DataStore, 62
- create\_clientlocal\_datastore
  - yaji::app::Application, 44
- create\_datastore
  - yaji::app::Application, 44
- create\_dialog
  - yaji::app::Application, 45
- dialogid
  - yaji::gui::Dialog, 66
- do\_stop\_implicitly\_when\_browser\_closed
  - yaji::app::Application, 45
- enable\_authentication
  - yaji::app::Application, 46
- filesystemdialog
  - yaji::userfeedback::UserFeedbackController, 85
- for\_urls
  - yaji::reqhandler::RequestHandler, 79
- get\_clientlocal\_datastore
  - yaji::app::Application, 46
- get\_datastore
  - yaji::app::Application, 46
- get\_dialog\_by\_id
  - yaji::dialogcontroller::DialogController, 67
- get\_param
  - yaji::request::Request, 76
- get\_param\_as\_list
  - yaji::request::Request, 76
- get\_response\_header
  - yaji::request::Request, 77
- get\_translations
  - yaji::app::Application, 47
- get\_translations\_stringnames
  - yaji::app::Application, 47
- getconfig
  - yaji::appconfig::AppConfig, 30
- getconfigs
  - yaji::appconfig::AppConfig, 31
- getvalue
  - yaji::datastore::DataStore, 63
- icon
  - yaji::app::Application, 47
- inputdialog
  - yaji::userfeedback::UserFeedbackController, 85
- insertcolumn
  - yaji::datastore::DataStore, 63
- insertrow
  - yaji::datastore::DataStore, 63
- Invisible
  - yaji::guibase::Visibility, 90
- InvisibleCollapsed
  - yaji::guibase::Visibility, 90
- isstaticapplication
  - yaji::app::Application, 47
- mainview\_icon
  - yaji::app::Application, 48
- make\_raw\_request
  - yaji::userfeedback::UserFeedbackController, 86
- messagedialog
  - yaji::userfeedback::UserFeedbackController, 86
- multilineinputdialog
  - yaji::userfeedback::UserFeedbackController, 86
- onbrowserreopened
  - yaji::app::Application, 48
- oninitialize
  - yaji::app::Application, 48
- onopenbrowsererror
  - yaji::app::Application, 48
- onopenbrowserinformationoutput
  - yaji::app::Application, 48
- onprocessrequesterror
  - yaji::app::Application, 49
- onunhandledclienterror
  - yaji::app::Application, 49
- openview
  - yaji::pyhtmlview::PyHtmlView, 73
- params
  - yaji::request::Request, 77
- passworddialog
  - yaji::userfeedback::UserFeedbackController, 87
- removecolumn
  - yaji::datastore::DataStore, 64
- removerow
  - yaji::datastore::DataStore, 64
- request\_processor
  - yaji::app::Application::\_\_ComputeContent←  
Middleware, 22
  - yaji::app::Application::\_\_PostParamsMiddleware,  
23
  - yaji::auth::AuthMiddleware, 56
  - yaji::core::Middleware, 72
- rowcount
  - yaji::datastore::DataStore, 65
- run\_downstream
  - yaji::reqhandler::RequestHandler, 80
- runs\_downstream
  - yaji::request::Request, 77
- set\_response\_header
  - yaji::request::Request, 77
- setactions
  - yaji::app::Application, 50
- setbodyleft
  - yaji::app::Application, 50
- setbodyright

- yaji::app::Application, 50
- setconfig
  - yaji::appconfig::AppConfig, 31
- setheadcontrol
  - yaji::app::Application, 51
- setsidebar
  - yaji::app::Application, 51
- setsplitterposition
  - yaji::app::Application, 51
- show\_dialog
  - yaji::dialogcontroller::DialogController, 68
- showonly
  - yaji::app::Application, 52
- skip\_processing
  - yaji::request::Request, 78
- start
  - yaji::app::Application, 52
- stop
  - yaji::app::Application, 52
- storeasstaticapplication
  - yaji::app::Application, 53
- ToDatasource
  - yaji::guibase::BindDirection, 58
- ToWidget
  - yaji::guibase::BindDirection, 58
- triggerevent
  - yaji::app::Application, 53
- try\_addclientresources\_from\_default\_location
  - yaji::app::Application, 54
- uientdata
  - yaji::request::Request, 78
- userfeedback
  - yaji::app::Application, 54
- Visible
  - yaji::guibase::Visibility, 91
- with\_param\_type
  - yaji::reqhandler::RequestHandler, 80
- yaji.app.Application, 31
- yaji.app.Application.\_\_ComputeContentMiddleware, 20
- yaji.app.Application.\_\_PostParamsMiddleware, 22
- yaji.app.Application.\_\_SetStopImplicitlyWhenBrowserClosed, 23
- yaji.app.Application.\_\_YajiHTTPRequestHandler, 24
- yaji.app.Application.\_\_YajiTCPServer, 24
- yaji.appconfig.AppConfig, 29
- yaji.auth.AuthMiddleware, 55
- yaji.browser.BrowserHook, 59
- yaji.core.Directories, 68
- yaji.core.Log, 71
- yaji.core.Middleware, 71
- yaji.datastore.DataStore, 60
- yaji.datastore.DataStore.ValuePointer, 87
- yaji.dialogcontroller.DialogController, 67
- yaji.gui.\_AbstractContainer, 24
- yaji.gui.Dialog, 65
- yaji.gui.Grid, 68
- yaji.gui.HorizontalStack, 69
- yaji.gui.ScrollView, 80
- yaji.gui.Spacer, 82
- yaji.gui.VerticalStack, 88
- yaji.gui.View, 88
- yaji.gui.View.\_DataBinding, 25
- yaji.gui.View.\_EventBinding, 26
- yaji.gui.Wrap, 91
- yaji.guibase.AbstractAction, 27
- yaji.guibase.AbstractFunction, 28
- yaji.guibase.Action, 28
- yaji.guibase.BackendFunction, 56
- yaji.guibase.BindDirection, 57
- yaji.guibase.BrowserFunction, 58
- yaji.guibase.BrowserSideDatasource, 60
- yaji.guibase.Icon, 70
- yaji.guibase.Separator, 81
- yaji.guibase.Submenu, 82
- yaji.guibase.Visibility, 90
- yaji.i18n.TrStr, 83
- yaji.pyhtmlview.PyHtmlView, 72
- yaji.reqhandler.RequestHandler, 78
- yaji.request.Request, 73
- yaji.request.Request.\_\_InRequest, 22
- yaji.userfeedback.UserFeedbackController, 84
- yaji::app::Application
  - \_\_create\_datastore\_helper, 36
  - \_\_get\_content, 36
  - \_\_get\_datastore\_helper, 37
  - \_\_get\_requesthandler\_for\_url, 37
  - \_\_init\_\_, 35
  - \_browser\_was\_reopened, 37
  - \_callhandler, 38
  - \_findhandler, 38
  - \_get\_requestparam\_type\_converter, 38
  - \_get\_rootpagecontent, 39
  - \_json\_make\_serializable, 39
  - \_middlewares, 39
  - \_openbrowser, 39
  - \_staticfile\_path\_to\_abspath, 40
  - \_tryclosebrowser, 40
  - add\_clientscript, 40
  - add\_clientstyle, 41
  - add\_middleware, 41
  - add\_requestparam\_type\_converter, 42
  - add\_staticfile\_location, 42
  - add\_translations, 42
  - binddata, 43
  - bindevent, 43
  - bindlocal, 43
  - bindserver, 44
  - create\_clientlocal\_datastore, 44
  - create\_datastore, 44
  - create\_dialog, 45
  - do\_stop\_implicitly\_when\_browser\_closed, 45
  - enable\_authentication, 46

- get\_clientlocal\_datastore, 46
- get\_datastore, 46
- get\_translations, 47
- get\_translations\_stringnames, 47
- icon, 47
- isstaticapplication, 47
- mainview\_icon, 48
- onbrowserreopened, 48
- oninitialize, 48
- onopenbrowsererror, 48
- onopenbrowserinformationoutput, 48
- onprocessrequesterror, 49
- onunhandledclienterror, 49
- setactions, 50
- setbodyleft, 50
- setbodyright, 50
- setheadcontrol, 51
- setsidebar, 51
- setsplitterposition, 51
- showonly, 52
- start, 52
- stop, 52
- storeasstaticapplication, 53
- triggerevent, 53
- try\_addclientresources\_from\_default\_location, 54
- userfeedback, 54
- yaji::app::Application::\_\_ComputeContentMiddleware
  - \_\_callhandler, 21
  - request\_processor, 22
- yaji::app::Application::\_\_PostParamsMiddleware
  - request\_processor, 23
- yaji::appconfig::AppConfig
  - \_\_markinitialized, 30
  - getconfig, 30
  - getConfigs, 31
  - setconfig, 31
- yaji::auth::AuthMiddleware
  - \_\_init\_\_, 55
  - \_authenticate\_by\_password, 56
  - request\_processor, 56
- yaji::browser::BrowserHook
  - \_\_init\_\_, 60
- yaji::core::Middleware
  - request\_processor, 72
- yaji::datastore::DataStore
  - add\_onchanged\_handler, 61
  - appendcolumn, 62
  - appendrow, 62
  - columncount, 62
  - getvalue, 63
  - insertcolumn, 63
  - insertrow, 63
  - removecolumn, 64
  - removerow, 64
  - rowcount, 65
- yaji::dialogcontroller::DialogController
  - close\_dialog, 67
  - get\_dialog\_by\_id, 67
  - show\_dialog, 68
- yaji::gui::\_AbstractContainer
  - \_\_init\_\_, 25
- yaji::gui::Dialog
  - \_\_init\_\_, 66
  - add\_closed\_handler, 66
  - dialogid, 66
- yaji::gui::Grid
  - \_\_init\_\_, 69
- yaji::gui::HorizontalStack
  - \_\_init\_\_, 70
- yaji::gui::ScrollView
  - \_\_init\_\_, 81
- yaji::gui::VerticalStack
  - \_\_init\_\_, 88
- yaji::gui::View
  - \_\_init\_\_, 89
- yaji::gui::View::\_DataBinding
  - \_\_init\_\_, 26
- yaji::gui::View::\_EventBinding
  - \_\_init\_\_, 26
- yaji::gui::Wrap
  - \_\_init\_\_, 91
- yaji::guibase::AbstractAction
  - \_\_init\_\_, 27
- yaji::guibase::Action
  - \_\_init\_\_, 29
- yaji::guibase::BackendFunction
  - \_\_init\_\_, 57
- yaji::guibase::BindDirection
  - Bidirectional, 58
  - ToDatasource, 58
  - ToWidget, 58
- yaji::guibase::BrowserFunction
  - \_\_init\_\_, 59
- yaji::guibase::Icon
  - \_\_init\_\_, 70
- yaji::guibase::Submenu
  - \_\_init\_\_, 82
- yaji::guibase::Visibility
  - Invisible, 90
  - InvisibleCollapsed, 90
  - Visible, 91
- yaji::i18n::TrStr
  - \_\_init\_\_, 83
- yaji::pyhtmlview::PyHtmlView
  - openview, 73
- yaji::reqhandler::RequestHandler
  - \_\_compile\_re\_for\_urlpattern, 79
  - for\_urls, 79
  - run\_downstream, 80
  - with\_param\_type, 80
- yaji::request::Request
  - \_\_init\_\_, 75
  - \_in\_request, 75
  - \_set\_lparam, 75
  - \_set\_runs\_downstream, 76
  - get\_param, 76

- get\_param\_as\_list, [76](#)
- get\_response\_header, [77](#)
- params, [77](#)
- runs\_downstream, [77](#)
- set\_response\_header, [77](#)
- skip\_processing, [78](#)
- uieventdata, [78](#)
- yaji::userfeedback::UserFeedbackController
  - \_interpret\_response, [84](#)
  - choicedialog, [85](#)
  - filesystemdialog, [85](#)
  - inputdialog, [85](#)
  - make\_raw\_request, [86](#)
  - messagedialog, [86](#)
  - multilineinputdialog, [86](#)
  - passworddialog, [87](#)