web2py Documentation

Release 2.14.6-stable

web2py-developers

Nov 01, 2018

Contents

1 admin Module **3**
1.1 Utility functions for the Admin application . . . 3

2 cache Module **7**
2.1 Basic caching classes and methods . . . 7

3 cfs Module **9**
3.1 Functions required to execute app components . . . 9

4 compileapp Module **11**
4.1 Functions required to execute app components . . . 11

5 contenttype Module **15**

6 custom_import Module **17**
6.1 Support for smart import syntax for web2py applications . . . 17

7 dal Module **19**
7.1 Takes care of adapting pyDAL to web2py's needs . . . 19

8 debug Module **21**
8.1 Debugger support classes . . . 21

9 decoder Module **23**

10 fileutils Module **25**
10.1 File operations . . . 25

11 globals Module **27**

12 highlight Module **33**

13 html Module **35**
13.1 Template helpers . . . 35

14 http Module **51**
14.1 HTTP statuses helpers . . . 51

15 languages Module **53**
15.1 Translation system 53

16 main Module **57**
16.1 The gluon wsgi application 57

17 messageboxhandler Module **59**

18 myregex Module **61**
18.1 Useful regexes 61

19 newcron Module **63**

20 portalocker Module **65**

21 recfile Module **67**
21.1 Generates names for cache and session files 67

22 restricted Module **69**
22.1 Restricted environment to execute application's code 69

23 rewrite Module **71**

24 sanitizer Module **75**
24.1 Cross-site scripting (XSS) defense 75

25 scheduler Module **77**
25.1 Background processes made simple 77

26 serializers Module **83**

27 settings Module **85**

28 shell Module **87**
28.1 Web2py environment in the shell 87

29 sql Module **89**
29.1 Just for backward compatibility 89

30 sqlhtml Module **97**

31 storage Module **107**

32 streamer Module **111**
32.1 Facilities to handle file streaming 111

33 template Module **113**
33.1 Templating syntax 113

34 tools Module **117**
34.1 Auth, Mail, PluginManager and various utilities 117

35 utf8 Module **139**
35.1 Utilities and class for UTF8 strings managing 139

36 utils Module **143**
36.1 This file specifically includes utilities for security. 143

37 `validators` Module **145**
37.1 Validators 145

38 `widget` Module **171**
38.1 The widget is called from web2py 171

39 `xmlrpc` Module **173**

40 Indices and tables **175**

Python Module Index **177**

Contents:

CHAPTER 1

admin Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

1.1 Utility functions for the Admin application

gluon.admin.**add_path_first**(*path*)

gluon.admin.**apath**(*path=''*, *r=None*)
Builds a path inside an application folder

Parameters

- **path** (*str*) – path within the application folder
- **r** – the global request object

gluon.admin.**app_cleanup**(*app*, *request*)
Removes session, cache and error files

Parameters

- **app** (*str*) – application name
- **request** – the global request object

Returns True if everything went ok, False otherwise

gluon.admin.**app_compile**(*app*, *request*, *skip_failed_views=False*)
Compiles the application

Parameters

- **app** (*str*) – application name
- **request** – the global request object

Returns None if everything went ok, traceback text if errors are found

gluon.admin.**app_create**(*app*, *request*, *force=False*, *key=None*, *info=False*)
Create a copy of welcome.w2p (scaffolding) app

Parameters

- **app** (*str*) – application name
- **request** – the global request object

gluon.admin.**app_install**(*app*, *fobj*, *request*, *filename*, *overwrite=None*)
Installs an application:

- Identifies file type by filename
- Writes *fobj* contents to the *../deposit/* folder
- Calls *w2p_unpack()* to do the job.

Parameters

- **app** (*str*) – new application name
- **fobj** (*obj*) – file object containing the application to be installed
- **request** – the global request object
- **filename** (*str*) – original filename of the *fobj*, required to determine extension
- **overwrite** (*bool*) – force overwrite of existing application

Returns name of the file where app is temporarily stored or *None* on failure

gluon.admin.**app_pack**(*app*, *request*, *raise_ex=False*, *filenames=None*)
Builds a w2p package for the application

Parameters

- **app** (*str*) – application name
- **request** – the global request object

Returns filename of the w2p file or None on error

gluon.admin.**app_pack_compiled**(*app*, *request*, *raise_ex=False*)
Builds a w2p bytecode-compiled package for the application

Parameters

- **app** (*str*) – application name
- **request** – the global request object

Returns filename of the w2p file or None on error

gluon.admin.**app_uninstall**(*app*, *request*)
Uninstalls the application.

Parameters

- **app** (*str*) – application name
- **request** – the global request object

Returns *True* on success, *False* on failure

gluon.admin.**check_new_version**(*myversion*, *version_url*)
Compares current web2py's version with the latest stable web2py version.

Parameters

- **myversion** – the current version as stored in file *web2py/VERSION*
- **version_URL** – the URL that contains the version of the latest stable release

Returns

state, version

- state : *True* if upgrade available, *False* if current version is up-to-date, -1 on error
- version : the most up-to-version available

Return type tuple

gluon.admin.**create_missing_app_folders**(*request*)

gluon.admin.**create_missing_folders**()

gluon.admin.**plugin_install**(*app*, *fobj*, *request*, *filename*)
Installs a plugin:

- Identifies file type by filename
- Writes *fobj* contents to the *../deposit/* folder
- Calls *w2p_unpack_plugin()* to do the job.

Parameters

- **app** (*str*) – new application name
- **fobj** – file object containing the application to be installed
- **request** – the global request object
- **filename** – original filename of the *fobj*, required to determine extension

Returns name of the file where plugin is temporarily stored or *False* on failure

gluon.admin.**plugin_pack**(*app*, *plugin_name*, *request*)
Builds a w2p package for the plugin

Parameters

- **app** (*str*) – application name
- **plugin_name** (*str*) – the name of the plugin without *plugin_* prefix
- **request** – the current request app

Returns filename of the w2p file or False on error

gluon.admin.**unzip**(*filename*, *dir*, *subfolder=''*)
Unzips filename into dir (.zip only, no .gz etc)

Parameters

- **filename** (*str*) – archive
- **dir** (*str*) – destination
- **subfolder** (*str*) – if != '' unzips only files in subfolder

gluon.admin.**upgrade**(*request*, *url='http://web2py.com'*)

Upgrades web2py (src, osx, win) if a new version is posted. It detects whether src, osx or win is running and downloads the right one

Parameters

- **request** – the current request object (required to determine version and path)
- **url** – the incomplete url where to locate the latest web2py (actual url is url+'/examples/static/web2py_(srclosx|win).zip')

Returns tuple: completed, traceback

- completed: True on success, False on failure (network problem or old version)
- traceback: None on success, raised exception details on failure

CHAPTER 2

cache Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

2.1 Basic caching classes and methods

- Cache - The generic caching object interfacing with the others
- CacheInRam - providing caching in ram
- CacheOnDisk - provides caches on disk

Memcache is also available via a different module (see gluon.contrib.memcache)

When web2py is running on Google App Engine, caching will be provided by the GAE memcache (see gluon.contrib.gae_memcache)

class gluon.cache.**Cache**(*request*)
Bases: object

Sets up generic caching, creating an instance of both CacheInRam and CacheOnDisk. In case of GAE will make use of gluon.contrib.gae_memcache.

- self.ram is an instance of CacheInRam
- self.disk is an instance of CacheOnDisk

action(*time_expire=300*, *cache_model=None*, *prefix=None*, *session=False*, *vars=True*, *lang=True*, *user_agent=False*, *public=True*, *valid_statuses=None*, *quick=None*)
Better fit for caching an action

> **Warning:** Experimental!

Currently only HTTP 1.1 compliant reference : http://code.google.com/p/doctype-mirror/wiki/ArticleHttpCaching

Parameters

- **time_expire** (*int*) – same as @cache
- **cache_model** (*str*) – same as @cache
- **prefix** (*str*) – add a prefix to the calculated key
- **session** (*bool*) – adds response.session_id to the key
- **vars** (*bool*) – adds request.env.query_string
- **lang** (*bool*) – adds T.accepted_language
- **user_agent** (*bool or dict*) – if True, adds is_mobile and is_tablet to the key. Pass a dict to use all the needed values (uses str(.items())) (e.g. user_agent=request.user_agent()). Used only if session is not True
- **public** (*bool*) – if False forces the Cache-Control to be 'private'
- **valid_statuses** – by default only status codes starting with 1,2,3 will be cached. pass an explicit list of statuses on which turn the cache on
- **quick** – Session,Vars,Lang,User-agent,Public: fast overrides with initials, e.g. 'SVLP' or 'VLP', or 'VLP'

autokey = ':%(name)s:%(args)s:%(vars)s'

static with_prefix(*cache_model*, *prefix*)

allow replacing cache.ram with cache.with_prefix(cache.ram,'prefix') it will add prefix to all the cache keys used.

gluon.cache.**lazy_cache**(*key=None*, *time_expire=None*, *cache_model='ram'*)

Can be used to cache any function including ones in modules, as long as the cached function is only called within a web2py request

If a key is not provided, one is generated from the function name *time_expire* defaults to None (no cache expiration)

If cache_model is "ram" then the model is current.cache.ram, etc.

CHAPTER 3

cfs Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

3.1 Functions required to execute app components

Note: FOR INTERNAL USE ONLY

gluon.cfs.**getcfs**(*key*, *filename*, *filter=None*)
Caches the *filtered* file *filename* with *key* until the file is modified.

Parameters

- **key** (*str*) – the cache key
- **filename** – the file to cache
- **filter** – is the function used for filtering. Normally *filename* is a .py file and *filter* is a function that bytecode compiles the file. In this way the bytecode compiled file is cached. (Default = None)

This is used on Google App Engine since pyc files cannot be saved.

CHAPTER 4

compileapp Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

4.1 Functions required to execute app components

Note: FOR INTERNAL USE ONLY

gluon.compileapp.**LOAD**(*c=None*, *f='index'*, *args=None*, *vars=None*, *extension=None*, *target=None*, *ajax=False*, *ajax_trap=False*, *url=None*, *user_signature=False*, *timeout=None*, *times=1*, *content='loading...'*, *post_vars={}*, ***attr*)

LOADs a component into the action's document

Parameters

- **c** (*str*) – controller
- **f** (*str*) – function
- **args** (*tuple or list*) – arguments
- **vars** (*dict*) – vars
- **extension** (*str*) – extension
- **target** (*str*) – id of the target
- **ajax** (*bool*) – True to enable AJAX bahaviour
- **ajax_trap** (*bool*) – True if *ajax* is set to *True*, traps both links and forms "inside" the target
- **url** (*str*) – overrides *c*,'f','args' and *vars*

- **user_signature** (*bool*) – adds hmac signature to all links with a key that is different for every user
- **timeout** (*int*) – in milliseconds, specifies the time to wait before starting the request or the frequency if times is greater than 1 or "infinity"
- **times** (*integer or str*) – how many times the component will be requested "infinity" or "continuous" are accepted to reload indefinitely the component

class gluon.compileapp.**LoadFactory**(*environment*)
Bases: object

Attention: this helper is new and experimental

gluon.compileapp.**build_environment**(*request*, *response*, *session*, *store_current=True*)
Build the environment dictionary into which web2py files are executed.

gluon.compileapp.**compile_application**(*folder*, *skip_failed_views=False*)
Compiles all models, views, controller for the application in *folder*.

gluon.compileapp.**compile_controllers**(*folder*)
Compiles all the controllers in the application specified by *folder*

gluon.compileapp.**compile_models**(*folder*)
Compiles all the models in the application specified by *folder*

gluon.compileapp.**compile_views**(*folder*, *skip_failed_views=False*)
Compiles all the views in the application specified by *folder*

gluon.compileapp.**find_exposed_functions**(*data*)

gluon.compileapp.**local_import_aux**(*name*, *reload_force=False*, *app='welcome'*)
In apps, instead of importing a local module (in applications/app/modules) with:

```
import a.b.c as d
```

you should do:

```
d = local_import('a.b.c')
```

or (to force a reload):

d = local_import('a.b.c', reload=True)

This prevents conflict between applications and un-necessary execs. It can be used to import any module, including regular Python modules.

gluon.compileapp.**model_cmp**(*a*, *b*, *sep='.'*)

gluon.compileapp.**model_cmp_sep**(*a*, *b*, *sep='/'*)

class gluon.compileapp.**mybuiltin**
Bases: object

NOTE could simple use a dict and populate it, NOTE not sure if this changes things though if monkey patching import.....

gluon.compileapp.**re_compile**(*regex*)

gluon.compileapp.**read_pyc**(*filename*)
Read the code inside a bytecode compiled file if the MAGIC number is compatible

Returns a code object

gluon.compileapp.**remove_compiled_application**(*folder*)
Deletes the folder *compiled* containing the compiled application.

gluon.compileapp.**run_controller_in**(*controller*, *function*, *environment*)
Runs the controller.function() (for the app specified by the current folder). It tries pre-compiled controller_function.pyc first before compiling it.

gluon.compileapp.**run_models_in**(*environment*)
Runs all models (in the app specified by the current folder) It tries pre-compiled models first before compiling them.

gluon.compileapp.**run_view_in**(*environment*)
Executes the view for the requested action. The view is the one specified in *response.view* or determined by the url or *view/generic.extension* It tries the pre-compiled views_controller_function.pyc before compiling it.

gluon.compileapp.**save_pyc**(*filename*)
Bytecode compiles the file *filename*

gluon.compileapp.**test**()
Example:

```
>>> import traceback, types
>>> environment={'x':1}
>>> open('a.py', 'w').write('print 1/x')
>>> save_pyc('a.py')
>>> os.unlink('a.py')
>>> if type(read_pyc('a.pyc'))==types.CodeType: print 'code'
code
>>> exec read_pyc('a.pyc') in environment
1
```

CHAPTER 5

contenttype Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

CONTENT_TYPE dictionary created against freedesktop.org's shared mime info database version 1.1.

Deviations from official standards:

- .md: application/x-genesis-rom –> text/x-markdown
- .png: image/x-apple-ios-png –> image/png

Additions:

- .load: text/html
- .json: application/json
- .jsonp: application/jsonp
- .pickle: application/python-pickle
- .w2p': application/w2p

gluon.contenttype.**contenttype**(*filename*, *default='text/plain'*)
Returns the Content-Type string matching extension of the given filename.

CHAPTER 6

custom_import Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

6.1 Support for smart import syntax for web2py applications

exception gluon.custom_import.**CustomImportException**
Bases: exceptions.ImportError

class gluon.custom_import.**TrackImporter**
Bases: object

An importer tracking the date of the module files and reloading them when they are changed.

PACKAGE_PATH_SUFFIX = '/__init__.py'

THREAD_LOCAL = <thread._local object>

gluon.custom_import.**custom_import_install**()

gluon.custom_import.**custom_importer**(*name*, *globals=None*, *locals=None*, *fromlist=None*, *level=-1*)
web2py's custom importer. It behaves like the standard Python importer but it tries to transform import statements as something like "import applications.app_name.modules.x". If the import fails, it falls back on naive_importer

gluon.custom_import.**is_tracking_changes**()

gluon.custom_import.**track_changes**(*track=True*)

CHAPTER 7

`dal` Module

Go to http://pydal.readthedocs.org/en/latest/ for docs on pyDAL

7.1 Takes care of adapting pyDAL to web2py's needs

CHAPTER 8

debug Module

This file is part of the web2py Web Framework
Developed by Massimo Di Pierro <mdipierro@cs.depaul.edu>,
limodou <limodou@gmail.com> and srackham <srackham@gmail.com>.
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

8.1 Debugger support classes

class gluon.debug.**Pipe**(*name*, *mode='r'*, **args*, ***kwargs*)
Bases: Queue.Queue

flush()

read(*count=None*, *timeout=None*)

readline()

write(*data*)

class gluon.debug.**WebDebugger**(*pipe*, *completekey='tab'*, *stdin=None*, *stdout=None*)
Bases: gluon.contrib.qdb.Frontend

Qdb web2py interface

clear_interaction()

do_continue(**args*, ***kwargs*)

do_exec(*statement*)

do_next(**args*, ***kwargs*)

do_quit(**args*, ***kwargs*)

do_return(**args*, ***kwargs*)

do_step(**args*, ***kwargs*)

exception(*title*, *extype*, *exvalue*, *trace*, *request*)
Show a user_exception

interaction(*filename*, *lineno*, *line*, ***context*)

run()
Main method dispatcher (infinite loop)

gluon.debug.**check_interaction**(*fn*)
Decorator to clean and prevent interaction when not available

gluon.debug.**communicate**(*command=None*)
send command to debbuger, wait result

gluon.debug.**set_trace**()
breakpoint shortcut (like pdb)

gluon.debug.**stop_trace**()
stop waiting for the debugger (called atexit)

CHAPTER 9

`decoder` Module

Caller will hand this library a buffer and ask it to either convert it or auto-detect the type.

Based on http://code.activestate.com/recipes/52257/

Licensed under the PSF License

`gluon.decoder.`**`autoDetectXMLEncoding`**(*buffer*)

buffer -> encoding_name The buffer should be at least 4 bytes long. Returns None if encoding cannot be detected. Note that encoding_name might not have an installed decoder (e.g. EBCDIC)

`gluon.decoder.`**`decoder`**(*buffer*)

CHAPTER 10

fileutils Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

10.1 File operations

gluon.fileutils.**parse_version**(*version*)
Attempts to parse SemVer, fallbacks on legacy

gluon.fileutils.**read_file**(*filename*, *mode='r'*)
Returns content from filename, making sure to close the file explicitly on exit.

gluon.fileutils.**write_file**(*filename*, *value*, *mode='w'*)
Writes <value> to filename, making sure to close the file explicitly on exit.

gluon.fileutils.**readlines_file**(*filename*, *mode='r'*)
Applies .split(' ') to the output of *read_file()*

gluon.fileutils.**up**(*path*)

gluon.fileutils.**abspath**(**relpath*, ***base*)
Converts relative path to absolute path based (by default) on applications_parent

gluon.fileutils.**mktree**(*path*)

gluon.fileutils.**listdir**(*path*, *expression='^.+$'*, *drop=True*, *add_dirs=False*, *sort=True*, *maxnum=None*, *exclude_content_from=None*)
Like *os.listdir()* but you can specify a regex pattern to filter files. If *add_dirs* is True, the returned items will have the full path.

gluon.fileutils.**recursive_unlink**(*f*)
Deletes *f*. If it's a folder, also its contents will be deleted

gluon.fileutils.**cleanpath**(*path*)
Turns any expression/path into a valid filename. replaces / with _ and removes special characters.

gluon.fileutils.**tar**(*file*, *dir*, *expression='^.+$'*, *filenames=None*, *exclude_content_from=None*)
Tars dir into file, only tars file that match expression

gluon.fileutils.**untar**(*file*, *dir*)
Untar file into dir

gluon.fileutils.**tar_compiled**(*file*, *dir*, *expression='^.+$'*, *exclude_content_from=None*)
Used to tar a compiled application. The content of models, views, controllers is not stored in the tar file.

gluon.fileutils.**get_session**(*request*, *other_application='admin'*)
Checks that user is authorized to access other_application

gluon.fileutils.**check_credentials**(*request*, *other_application='admin'*, *expiration=3600*, *gae_login=True*)
Checks that user is authorized to access other_application

gluon.fileutils.**w2p_pack**(*filename*, *path*, *compiled=False*, *filenames=None*)
Packs a web2py application.

Parameters

- **filename** (*str*) – path to the resulting archive
- **path** (*str*) – path to the application
- **compiled** (*bool*) – if *True* packs the compiled version
- **filenames** (*list*) – adds filenames to the archive

gluon.fileutils.**w2p_unpack**(*filename*, *path*, *delete_tar=True*)

gluon.fileutils.**w2p_pack_plugin**(*filename*, *path*, *plugin_name*)
Packs the given plugin into a w2p file. Will match files at:

```
<path>/*/plugin_[name].*
<path>/*/plugin_[name]/*
```

gluon.fileutils.**w2p_unpack_plugin**(*filename*, *path*, *delete_tar=True*)

gluon.fileutils.**fix_newlines**(*path*)

gluon.fileutils.**make_fake_file_like_object**()

CHAPTER 11

globals Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

Contains the classes for the global used variables:

- Request
- Response
- Session

class gluon.globals.**Request**(*env*)
Bases: *gluon.storage.Storage*

Defines the request object and the default values of its members

- env: environment variables, by gluon.main.wsgibase()
- cookies
- get_vars
- post_vars
- vars
- folder
- application
- function
- args
- extension
- now: datetime.datetime.now()

- utcnow : datetime.datetime.utcnow()
- is_local
- is_https
- restful()

body

compute_uuid()

get_vars
Lazily parses the query string into get_vars

parse_all_vars()
Merges get_vars and post_vars to vars

parse_get_vars()
Takes the QUERY_STRING and unpacks it to get_vars

parse_post_vars()
Takes the body of the request and unpacks it into post_vars. application/json is also automatically parsed

post_vars
Lazily parse the body into post_vars

requires_https()
If request comes in over HTTP, redirects it to HTTPS and secures the session.

restful()

user_agent()

uuid
Lazily uuid

vars
Lazily parses all get_vars and post_vars to fill vars

class gluon.globals.**Response**
Bases: *gluon.storage.Storage*

Defines the response object and the default values of its members response.write( ) can be used to write in the output html

download(*request*, *db*, *chunk_size=65536*, *attachment=True*, *download_filename=None*)
Example of usage in controller:

```
def download():
    return response.download(request, db)
```

Downloads from http://. . . ./download/filename

include_files(*extensions=None*)
Includes files (usually in the head). Can minify and cache local files By default, caches in ram for 5 minutes. To change, response.cache_includes = (cache_method, time_expire). Example: (cache.disk, 60) # caches to disk for 1 minute.

include_meta()

json(*data*, *default=None*)

render(**a*, ***b*)

stream(*stream*, *chunk_size=65536*, *request=None*, *attachment=False*, *filename=None*)
If in a controller function:

```
return response.stream(file, 100)
```

the file content will be streamed at 100 bytes at the time

Parameters

- **stream** – filename or read()able content
- **chunk_size** (*int*) – Buffer size
- **request** – the request object
- **attachment** (*bool*) – prepares the correct headers to download the file as an attachment. Usually creates a pop-up download window on browsers
- **filename** (*str*) – the name for the attachment

Note: for using the stream name (filename) with attachments the option must be explicitly set as function parameter (will default to the last request argument otherwise)

---

toolbar()

write(*data*, *escape=True*)

xmlrpc(*request*, *methods*)
assuming:

```
def add(a, b):
    return a+b
```

if a controller function "func":

```
return response.xmlrpc(request, [add])
```

the controller will be able to handle xmlrpc requests for the add function. Example:

```
import xmlrpclib
connection = xmlrpclib.ServerProxy(
    'http://hostname/app/contr/func')
print connection.add(3, 4)
```

class gluon.globals.**Session**
Bases: *gluon.storage.Storage*

Defines the session object and the default values of its members (None)

- session_storage_type : 'file', 'db', or 'cookie'
- session_cookie_compression_level :
- session_cookie_expires : cookie expiration
- session_cookie_key : for encrypted sessions in cookies
- session_id : a number or None if no session
- session_id_name :
- session_locked :

- session_masterapp :
- session_new : a new session obj is being created
- session_hash : hash of the pickled loaded session
- session_pickled : picked session

if session in cookie:

- session_data_name : name of the cookie for session data

if session in db:

- session_db_record_id
- session_db_table
- session_db_unique_key

if session in file:

- session_file
- session_filename

clear() → None. Remove all items from D.

clear_session_cookies()

connect(*request=None*, *response=None*, *db=None*, *tablename='web2py_session'*, *masterapp=None*, *migrate=True*, *separate=None*, *check_client=False*, *cookie_key=None*, *cookie_expires=None*, *compression_level=None*)

Used in models, allows to customize Session handling

Parameters

- **request** – the request object
- **response** – the response object
- **db** – to store/retrieve sessions in db (a table is created)
- **tablename** (*str*) – table name
- **masterapp** (*str*) – points to another's app sessions. This enables a "SSO" environment among apps
- **migrate** – passed to the underlying db
- **separate** – with True, creates a folder with the 2 initials of the session id. Can also be a function, e.g.

```
separate=lambda(session_name): session_name[-2:]
```

- **check_client** – if True, sessions can only come from the same ip
- **cookie_key** (*str*) – secret for cookie encryption
- **cookie_expires** – sets the expiration of the cookie
- **compression_level** (*int*) – 0-9, sets zlib compression on the data before the encryption

forget(*response=None*)

is_expired(*seconds=3600*)

is_new()

renew(*clear_session=False*)

save_session_id_cookie()

secure()

CHAPTER 12

`highlight` Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

`gluon.highlight.`**`highlight`**(*code*, *language*, *link='/examples/globals/vars/'*, *counter=1*, *styles=None*, *highlight_line=None*, *context_lines=None*, *attributes=None*)

CHAPTER 13

html Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

13.1 Template helpers

class gluon.html.**A**(**components*, ***attributes*)

Bases: *gluon.html.DIV*

Generates an A() link. A() in web2py is really important and with the included web2py.js allows lots of Ajax interactions in the page

On top of "usual" *_attributes*, it takes

Parameters

- **callback** – an url to call but not redirect to
- **cid** – if you want to load the _href into an element of the page (component) pass its id (without the #) here
- **delete** – element to delete after calling callback
- **target** – same thing as cid
- **confirm** – text to display upon a callback with a delete
- **noconfirm** – don't display alert upon a callback with delete

tag = 'a'

xml()

generates the xml for this component.

gluon.html.**ASSIGNJS**(***kargs*)

Example

ASSIGNJS(var1='1', var2='2') will return the following javascript variables assignations :

var var1 = "1"; var var2 = "2";

Parameters ****kargs** – Any keywords arguments and assigned values.

Returns Javascript vars assignations for the key/value passed.

class gluon.html.**B**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'b'

class gluon.html.**BEAUTIFY**(*component*, ***attributes*)
Bases: *gluon.html.DIV*

Turns any list, dictionary, etc into decent looking html.

Two special attributes are

- sorted: a function that takes the dict and returned sorted keys
- keyfilter: a function that takes a key and returns its representation or None if the key is to be skipped. By default key[:1]=='_' is skipped.

Examples:

```
>>> BEAUTIFY(['a', 'b', {'hello': 'world'}]).xml()
'<div><table><tr><td><div>a</div></td></tr><tr><td><div>b</div></td></tr><tr><td>
↪<div><table><tr><td style="font-weight:bold;vertical-align:top;">hello</td><td␣
↪style="vertical-align:top;">:</td><td><div>world</div></td></tr></table></div></
↪td></tr></table></div>'
```

static no_underscore(*key*)

tag = 'div'

class gluon.html.**BODY**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'body'

class gluon.html.**BR**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'br/'

class gluon.html.**BUTTON**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'button'

class gluon.html.**CENTER**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'center'

class gluon.html.**CAT**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = ''

class gluon.html.**CODE**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

Displays code in HTML with syntax highlighting.

Parameters

- **language** – indicates the language, otherwise PYTHON is assumed
- **link** – can provide a link
- **styles** – for styles

Examples:

{{=CODE("print 'hello world'", language='python', link=None, counter=1, styles={}, highlight_line=None)}}

supported languages are

"python", "html_plain", "c", "cpp", "web2py", "html"

The "html" language interprets {{ and }} tags as "web2py" code, "html_plain" doesn't.

if a link='/examples/global/vars/' is provided web2py keywords are linked to the online docs.

the counter is used for line numbering, counter can be None or a prompt string.

xml()
generates the xml for this component.

class gluon.html.**COL**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'col/'

class gluon.html.**COLGROUP**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'colgroup'

class gluon.html.**DIV**(**components*, ***attributes*)
Bases: gluon.html.XmlComponent

HTML helper, for easy generating and manipulating a DOM structure. Little or no validation is done.

Behaves like a dictionary regarding updating of attributes. Behaves like a list regarding inserting/appending components.

Examples:

```
>>> DIV('hello', 'world', _style='color:red;').xml()
'<div style="color:red;">helloworld</div>'
```

All other HTML helpers are derived from *DIV*.

_something="value" attributes are transparently translated into *something="value"* HTML attributes

append(*value*)
list style appending of components

Examples:

```
>>> a=DIV()
>>> a.append(SPAN('x'))
>>> print a
<div><span>x</span></div>
```

element(**args*, ***kargs*)

Finds the first component that matches the supplied attribute dictionary, or None if nothing could be found

Also the components of the components are searched.

elements(**args*, ***kargs*)

Find all components that match the supplied attribute dictionary, or None if nothing could be found

All components of the components are searched.

Examples:

```
>>> a = DIV(DIV(SPAN('x'),3,DIV(SPAN('y'))))
>>> for c in a.elements('span', first_only=True): c[0]='z'
>>> print a
<div><div><span>z</span>3<div><span>y</span></div></div></div>
>>> for c in a.elements('span'): c[0]='z'
>>> print a
<div><div><span>z</span>3<div><span>z</span></div></div></div>
```

It also supports a syntax compatible with jQuery

Examples:

```
>>> a=TAG('<div><span><a id="1-1" u:v=$>hello</a></span><p class="this is a␣
↪test">world</p></div>')
>>> for e in a.elements('div a#1-1, p.is'): print e.flatten()
hello
world
>>> for e in a.elements('#1-1'): print e.flatten()
hello
>>> a.elements('a[u:v=$]')[0].xml()
'<a id="1-1" u:v="$">hello</a>'
>>> a=FORM( INPUT(_type='text'), SELECT(range(1)), TEXTAREA() )
>>> for c in a.elements('input, select, textarea'): c['_disabled'] = 'disabled
↪'
>>> a.xml()
'<form action="#" enctype="multipart/form-data" method="post"><input disabled=
↪"disabled" type="text" /><select disabled="disabled"><option value="0">0</
↪option></select><textarea cols="40" disabled="disabled" rows="10"></
↪textarea></form>'
```

Elements that are matched can also be replaced or removed by specifying a "replace" argument (note, a list of the original matching elements is still returned as usual).

Examples:

```
>>> a = DIV(DIV(SPAN('x', _class='abc'), DIV(SPAN('y', _class='abc'), SPAN('z
↪', _class='abc'))))
>>> b = a.elements('span.abc', replace=P('x', _class='xyz'))
>>> print a  # We should .xml() here instead of print
<div><div><p class="xyz">x</p><div><p class="xyz">x</p><p class="xyz">x</p></
↪div></div></div>
```

"replace" can be a callable, which will be passed the original element and should return a new element to replace it.

Examples:

```
>>> a = DIV(DIV(SPAN('x', _class='abc'), DIV(SPAN('y', _class='abc'), SPAN('z
↪', _class='abc'))))
>>> b = a.elements('span.abc', replace=lambda el: P(el[0], _class='xyz'))
>>> print a
<div><div><p class="xyz">x</p><div><p class="xyz">y</p><p class="xyz">z</p></
↪div></div></div>
```

If replace=None, matching elements will be removed completely.

Examples:

```
>>> a = DIV(DIV(SPAN('x', _class='abc'), DIV(SPAN('y', _class='abc'), SPAN('z
↪', _class='abc'))))
>>> b = a.elements('span', find='y', replace=None)
>>> print a
<div><div><span class="abc">x</span><div><span class="abc">z</span></div></
↪div></div>
```

If a "find_text" argument is specified, elements will be searched for text components that match find_text, and any matching text components will be replaced (find_text is ignored if "replace" is not also specified). Like the "find" argument, "find_text" can be a string or a compiled regex.

Examples:

```
>>> a = DIV(DIV(SPAN('x', _class='abc'), DIV(SPAN('y', _class='abc'), SPAN('z
↪', _class='abc'))))
>>> b = a.elements(find_text=re.compile('x|y|z'), replace='hello')
>>> print a
<div><div><span class="abc">hello</span><div><span class="abc">hello</span>
↪<span class="abc">hello</span></div></div></div>
```

If other attributes are specified along with find_text, then only components that match the specified attributes will be searched for find_text.

Examples:

```
>>> a = DIV(DIV(SPAN('x', _class='abc'), DIV(SPAN('y', _class='efg'), SPAN('z
↪', _class='abc'))))
>>> b = a.elements('span.efg', find_text=re.compile('x|y|z'), replace='hello')
>>> print a
<div><div><span class="abc">x</span><div><span class="efg">hello</span><span␣
↪class="abc">z</span></div></div></div>
```

flatten(*render=None*)

Returns the text stored by the DIV object rendered by the render function the render function must take text, tagname, and attributes *render=None* is equivalent to *render=lambda text, tag, attr: text*

Examples:

```
>>> markdown = lambda text, tag=None, attributes={}:
↪{None: re.sub('\s+',' ',text),                          'h1':'#'+text+'\n\n
↪',                          'p':text+'\n'}.get(tag,text)
>>> a=TAG('<h1>Header</h1><p>this is a     test</p>')
>>> a.flatten(markdown)
'#Header\n\nthis is a test\n'
```

get(*i*)

insert(*i*, *value*)
List-style inserting of components

Examples:

```
>>> a=DIV()
>>> a.insert(0, SPAN('x'))
>>> print a
<div><span>x</span></div>
```

regex_attr = <_sre.SRE_Pattern object>

regex_class = <_sre.SRE_Pattern object>

regex_id = <_sre.SRE_Pattern object>

regex_tag = <_sre.SRE_Pattern object>

sibling(**args*, ***kargs*)
Finds the first sibling component that match the supplied argument list and attribute dictionary, or None if nothing could be found

siblings(**args*, ***kargs*)
Finds all sibling components that match the supplied argument list and attribute dictionary, or None if nothing could be found

tag = 'div'

update(***kargs*)
dictionary like updating of the tag attributes

xml()
generates the xml for this component.

class gluon.html.**EM**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'em'

class gluon.html.**EMBED**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'embed/'

class gluon.html.**FIELDSET**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'fieldset'

class gluon.html.**FORM**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

Examples:

```
>>> from validators import IS_NOT_EMPTY
>>> form=FORM(INPUT(_name="test", requires=IS_NOT_EMPTY()))
>>> form.xml()
'<form action="#" enctype="multipart/form-data" method="post"><input name="test"␣
↪type="text" /></form>'
```

a FORM is container for INPUT, TEXTAREA, SELECT and other helpers

form has one important method:

```
form.accepts(request.vars, session)
```

if form is accepted (and all validators pass) form.vars contains the accepted vars, otherwise form.errors contains the errors. in case of errors the form is modified to present the errors to the user.

REDIRECT_JS = "window.location='%s';return false"

accepts(*request_vars*, *session=None*, *formname='default'*, *keepvalues=False*, *onvalidation=None*, *hideerror=False*, ***kwargs*)
kwargs is not used but allows to specify the same interface for FORM and SQLFORM

add_button(*value*, *url*, *_class=None*)

as_dict(*flat=False*, *sanitize=True*)
EXPERIMENTAL

Sanitize is naive. It should catch any unsafe value for client retrieval.

as_json(*sanitize=True*)

as_xml(*sanitize=True*)

as_yaml(*sanitize=True*)

assert_status(*status*, *request_vars*)

static confirm(*text='OK'*, *buttons=None*, *hidden=None*)

hidden_fields()

process(***kwargs*)
Perform the .validate() method but returns the form

Usage in controllers:

```
# directly on return
def action():
    #some code here
    return dict(form=FORM(...).process(...))
```

You can use it with FORM, SQLFORM or FORM based plugins:

```
# response.flash messages
def action():
    form = SQLFORM(db.table).process(message_onsuccess='Sucess!')
    return dict(form=form)

# callback function
# callback receives True or False as first arg, and a list of args.
def my_callback(status, msg):
    response.flash = "Success! "+msg if status else "Errors occured"

# after argument can be 'flash' to response.flash messages
# or a function name to use as callback or None to do nothing.
def action():
    return dict(form=SQLFORM(db.table).process(onsuccess=my_callback)
```

tag = 'form'

validate(***kwargs*)
This function validates the form, you can use it instead of directly form.accepts.

Usage: In controller:

```
def action():
    form=FORM(INPUT(_name="test", requires=IS_NOT_EMPTY()))
    form.validate() #you can pass some args here - see below
    return dict(form=form)
```

This can receive a bunch of arguments

onsuccess = 'flash' - will show message_onsuccess in response.flash None - will do nothing can be a function (lambda form: pass)

onfailure = 'flash' - will show message_onfailure in response.flash None - will do nothing can be a function (lambda form: pass)

onchange = 'flash' - will show message_onchange in response.flash None - will do nothing can be a function (lambda form: pass)

message_onsuccess message_onfailure message_onchange next = where to redirect in case of success any other kwargs will be passed for form.accepts(...)

xml()
generates the xml for this component.

class gluon.html.**H1**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'h1'

class gluon.html.**H2**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'h2'

class gluon.html.**H3**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'h3'

class gluon.html.**H4**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'h4'

class gluon.html.**H5**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'h5'

class gluon.html.**H6**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'h6'

class gluon.html.**HEAD**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'head'

class gluon.html.**HR**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'hr/'

class gluon.html.**HTML**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

There are four predefined document type definitions. They can be specified in the 'doctype' parameter:

- 'strict' enables strict doctype
- 'transitional' enables transitional doctype (default)
- 'frameset' enables frameset doctype
- 'html5' enables HTML 5 doctype
- any other string will be treated as user's own doctype

'lang' parameter specifies the language of the document. Defaults to 'en'.

See also *DIV*

frameset = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.

html5 = '<!DOCTYPE HTML>\n'

strict = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html

tag = 'html'

transitional = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http:/

xml()
generates the xml for this component.

class gluon.html.**I**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'i'

class gluon.html.**IFRAME**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'iframe'

class gluon.html.**IMG**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'img/'

class gluon.html.**INPUT**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

INPUT Component

Takes two special attributes value= and requires=.

Parameters

- **value** – used to pass the initial value for the input field. value differs from _value because it works for checkboxes, radio, textarea and select/option too. For a checkbox value should be '' or 'on'. For a radio or select/option value should be the _value of the checked/selected item.
- **requires** – should be None, or a validator or a list of validators for the value of the field.

Examples:

```
>>> INPUT(_type='text', _name='name', value='Max').xml()
'<input name="name" type="text" value="Max" />'
```

```
>>> INPUT(_type='checkbox', _name='checkbox', value='on').xml()
'<input checked="checked" name="checkbox" type="checkbox" value="on" />'
```

```
>>> INPUT(_type='radio', _name='radio', _value='yes', value='yes').xml()
'<input checked="checked" name="radio" type="radio" value="yes" />'
```

```
>>> INPUT(_type='radio', _name='radio', _value='no', value='yes').xml()
'<input name="radio" type="radio" value="no" />'
```

tag = 'input/'

xml()
generates the xml for this component.

class gluon.html.**LABEL**(*components, **attributes)
Bases: *gluon.html.DIV*

tag = 'label'

class gluon.html.**LEGEND**(*components, **attributes)
Bases: *gluon.html.DIV*

tag = 'legend'

class gluon.html.**LI**(*components, **attributes)
Bases: *gluon.html.DIV*

tag = 'li'

class gluon.html.**LINK**(*components, **attributes)
Bases: *gluon.html.DIV*

tag = 'link/'

class gluon.html.**OL**(*components, **attributes)
Bases: *gluon.html.UL*

tag = 'ol'

class gluon.html.**UL**(*components, **attributes)
Bases: *gluon.html.DIV*

UL Component.

If subcomponents are not LI-components they will be wrapped in a LI

tag = 'ul'

class gluon.html.**MARKMIN**(*text*, *extra=None*, *allowed=None*, *sep='p'*, *url=None*, *environment=None*, *latex='google'*, *autolinks='default'*, *protolinks='default'*, *class_prefix=''*, *id_prefix='markmin_'*, **kwargs)
Bases: gluon.html.XmlComponent

For documentation: http://web2py.com/examples/static/markmin.html

flatten()

xml()

class gluon.html.**MENU**(*data*, **args)
Bases: *gluon.html.DIV*

Used to build menus

Parameters

- **_class** – defaults to 'web2py-menu web2py-menu-vertical'
- **ul_class** – defaults to 'web2py-menu-vertical'

- **li_class** – defaults to 'web2py-menu-expand'
- **li_first** – defaults to 'web2py-menu-first'
- **li_last** – defaults to 'web2py-menu-last'

Use like:

```
menu = MENU([['name', False, URL(...), [submenu]], ...])
{{=menu}}
```

serialize(*data*, *level=0*)

serialize_mobile(*data*, *select=None*, *prefix=''*)

tag = 'ul'

xml()
generates the xml for this component.

class gluon.html.**META**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'meta/'

class gluon.html.**OBJECT**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'object'

class gluon.html.**OPTION**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'option'

class gluon.html.**P**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

Will replace \n by
 if the *cr2br* attribute is provided.

see also *DIV*

tag = 'p'

xml()
generates the xml for this component.

class gluon.html.**PRE**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'pre'

class gluon.html.**SCRIPT**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'script'

xml()
generates the xml for this component.

class gluon.html.**OPTGROUP**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'optgroup'

class gluon.html.**SELECT**(**components*, ***attributes*)
Bases: *gluon.html.INPUT*

Examples:

```
>>> from validators import IS_IN_SET
>>> SELECT('yes', 'no', _name='selector', value='yes',
...     requires=IS_IN_SET(['yes', 'no'])).xml()
'<select name="selector"><option selected="selected" value="yes">yes</option>
↪<option value="no">no</option></select>'
```

tag = 'select'

class gluon.html.**SPAN**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'span'

class gluon.html.**STRONG**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'strong'

class gluon.html.**STYLE**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'style'

xml()
generates the xml for this component.

class gluon.html.**TABLE**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

TABLE Component.

If subcomponents are not TR/TBODY/THEAD/TFOOT-components they will be wrapped in a TR

tag = 'table'

gluon.html.**TAG**
TAG factory

Examples:

```
>>> print TAG.first(TAG.second('test'), _key = 3)
<first key="3"><second>test</second></first>
```

class gluon.html.**TD**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'td'

class gluon.html.**TEXTAREA**(**components*, ***attributes*)
Bases: *gluon.html.INPUT*

Examples:

```
TEXTAREA(_name='sometext', value='blah ' * 100, requires=IS_NOT_EMPTY())
```

'blah blah blah ...' will be the content of the textarea field.

tag = 'textarea'

class gluon.html.**TH**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'th'

class gluon.html.**THEAD**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'thead'

class gluon.html.**TBODY**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'tbody'

class gluon.html.**TFOOT**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'tfoot'

class gluon.html.**TITLE**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'title'

class gluon.html.**TR**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

TR Component.

If subcomponents are not TD/TH-components they will be wrapped in a TD

tag = 'tr'

class gluon.html.**TT**(**components*, ***attributes*)
Bases: *gluon.html.DIV*

tag = 'tt'

gluon.html.**URL**(*a=None*, *c=None*, *f=None*, *r=None*, *args=None*, *vars=None*, *anchor=''*, *extension=None*, *env=None*, *hmac_key=None*, *hash_vars=True*, *salt=None*, *user_signature=None*, *scheme=None*, *host=None*, *port=None*, *encode_embedded_slash=False*, *url_encode=True*, *language=None*)
generates a url '/a/c/f' corresponding to application a, controller c and function f. If r=request is passed, a, c, f are set, respectively, to r.application, r.controller, r.function.

The more typical usage is:

URL('index')

that generates a url for the index function within the present application and controller.

Parameters

- **a** – application (default to current if r is given)
- **c** – controller (default to current if r is given)
- **f** – function (default to current if r is given)
- **r** – request (optional)
- **args** – any arguments (optional). Additional "path" elements
- **vars** – any variables (optional). Querystring elements
- **anchor** – anchorname, without # (optional)

- **extension** – force an extension
- **hmac_key** – key to use when generating hmac signature (optional)
- **hash_vars** – which of the vars to include in our hmac signature True (default) - hash all vars, False - hash none of the vars, iterable - hash only the included vars ['key1','key2']
- **salt** – salt hashing with this string
- **user_signature** – signs automatically the URL in such way that only the user can access the URL (use with *URL.verify* or *auth.requires_signature()*)
- **scheme** – URI scheme (True, 'http' or 'https', etc); forces absolute URL (optional)
- **host** – string to force absolute URL with host (True means http_host)
- **port** – optional port number (forces absolute URL)
- **encode_embedded_slash** – encode slash characters included in args
- **url_encode** – encode characters included in vars

Raises **SyntaxError** – when no application, controller or function is available or when a CRLF is found in the generated url

Examples:

```
>>> str(URL(a='a', c='c', f='f', args=['x', 'y', 'z'],
...     vars={'p':1, 'q':2}, anchor='1'))
'/a/c/f/x/y/z?p=1&q=2#1'
```

```
>>> str(URL(a='a', c='c', f='f', args=['x', 'y', 'z'],
...     vars={'p':(1,3), 'q':2}, anchor='1'))
'/a/c/f/x/y/z?p=1&p=3&q=2#1'
```

```
>>> str(URL(a='a', c='c', f='f', args=['x', 'y', 'z'],
...     vars={'p':(3,1), 'q':2}, anchor='1'))
'/a/c/f/x/y/z?p=3&p=1&q=2#1'
```

```
>>> str(URL(a='a', c='c', f='f', anchor='1+2'))
'/a/c/f#1%2B2'
```

```
>>> str(URL(a='a', c='c', f='f', args=['x', 'y', 'z'],
...     vars={'p':(1,3), 'q':2}, anchor='1', hmac_key='key'))
'/a/c/f/x/y/z?p=1&p=3&q=2&_signature=a32530f0d0caa80964bb92aad2bedf8a4486a31f#1'
```

```
>>> str(URL(a='a', c='c', f='f', args=['w/x', 'y/z']))
'/a/c/f/w/x/y/z'
```

```
>>> str(URL(a='a', c='c', f='f', args=['w/x', 'y/z'], encode_embedded_slash=True))
'/a/c/f/w%2Fx/y%2Fz'
```

```
>>> str(URL(a='a', c='c', f='f', args=['%(id)d'], url_encode=False))
'/a/c/f/%(id)d'
```

```
>>> str(URL(a='a', c='c', f='f', args=['%(id)d'], url_encode=True))
'/a/c/f/%25%28id%29d'
```

```
>>> str(URL(a='a', c='c', f='f', vars={'id' : '%(id)d' }, url_encode=False))
'/a/c/f?id=%(id)d'
```

```
>>> str(URL(a='a', c='c', f='f', vars={'id' : '%(id)d' }, url_encode=True))
'/a/c/f?id=%25%28id%29d'
```

```
>>> str(URL(a='a', c='c', f='f', anchor='%(id)d', url_encode=False))
'/a/c/f#%(id)d'
```

```
>>> str(URL(a='a', c='c', f='f', anchor='%(id)d', url_encode=True))
'/a/c/f#%25%28id%29d'
```

class gluon.html.**XHTML**(**components*, ***attributes*)

Bases: *gluon.html.DIV*

This is XHTML version of the HTML helper.

There are three predefined document type definitions. They can be specified in the 'doctype' parameter:

- 'strict' enables strict doctype
- 'transitional' enables transitional doctype (default)
- 'frameset' enables frameset doctype
- any other string will be treated as user's own doctype

'lang' parameter specifies the language of the document and the xml document. Defaults to 'en'.

'xmlns' parameter specifies the xml namespace. Defaults to 'http://www.w3.org/1999/xhtml'.

See also *DIV*

frameset = '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.

strict = '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/

tag = 'html'

transitional = '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http:/

xml()

generates the xml for this component.

xmlns = 'http://www.w3.org/1999/xhtml'

class gluon.html.**XML**(*text, sanitize=False, permitted_tags=['a', 'b', 'blockquote', 'br/', 'i', 'li', 'ol', 'ul', 'p', 'cite', 'code', 'pre', 'img/', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'table', 'tr', 'td', 'div', 'strong', 'span'], allowed_attributes={'a': ['href', 'title', 'target'], 'blockquote': ['type'], 'img': ['src', 'alt'], 'td': ['colspan']}*)

Bases: gluon.html.XmlComponent

use it to wrap a string that contains XML/HTML so that it will not be escaped by the template

Examples:

```
>>> XML('<h1>Hello</h1>').xml()
'<h1>Hello</h1>'
```

elements(**args*, ***kargs*)

to be considered experimental since the behavior of this method is questionable another option could be *TAG(self.text).elements(*args, **kwargs)*

flatten(*render=None*)
returns the text stored by the XML object rendered by the *render* function

xml()

gluon.html.**xmlescape**(*data*, *quote=True*)
Returns an escaped string of the provided data

Parameters

- **data** – the data to be escaped
- **quote** – optional (default False)

gluon.html.**embed64**(*filename=None*, *file=None*, *data=None*, *extension='image/gif'*)
helper to encode the provided (binary) data into base64.

Parameters

- **filename** – if provided, opens and reads this file in 'rb' mode
- **file** – if provided, reads this file
- **data** – if provided, uses the provided data

CHAPTER 14

http Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

14.1 HTTP statuses helpers

exception gluon.http.**HTTP**(*status*, *body=''*, *cookies=None*, ***headers*)
Bases: exceptions.Exception

Raises an HTTP response

Parameters

- **status** – usually an integer. If it's a well known status code, the ERROR message will be automatically added. A string can also be passed as *510 Foo Bar* and in that case the status code and the error message will be parsed accordingly
- **body** – what to return as body. If left as is, will return the error code and the status message in the body itself
- **cookies** – pass cookies along (usually not needed)
- **headers** – pass headers as usual dict mapping

cookies2headers(*cookies*)

message
compose a message describing this exception

"status defined_status [web2py_error]"

message elements that are not defined are omitted

to(*responder*, *env=None*)

gluon.http.**redirect**(*location=''*, *how=303*, *client_side=False*, *headers=None*)
Raises a redirect (303)

Parameters

- **location** – the url where to redirect
- **how** – what HTTP status code to use when redirecting
- **client_side** – if set to True, it triggers a reload of the entire page when the fragment has been loaded as a component

CHAPTER 15

languages Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)
Plural subsystem is created by Vladyslav Kozlovskyy (Ukraine) <dbdevelop@gmail.com>

15.1 Translation system

class gluon.languages.**translator**(*langpath*, *http_accept_language*)
Bases: object

This class is instantiated by gluon.compileapp.build_environment as the T object

Example

T.force(None) # turns off translation T.force('fr, it') # forces web2py to translate using fr.py or it.py

T("Hello World") # translates "Hello World" using the selected file

---

Note:

- there is no need to force since, by default, T uses http_accept_language to determine a translation file.
- en and en-en are considered different languages!
- if language xx-yy is not found force() probes other similar languages using such algorithm: *xx-yy.py -> xx.py -> xx-yy*.py -> xx*.py*

M(*message*, *symbols={}*, *language=None*, *lazy=None*, *filter=None*, *ftag=None*, *ns=None*)
Gets cached translated markmin-message with inserted parametes if lazy==True lazyT object is returned

apply_filter(*message*, *symbols={}*, *filter=None*, *ftag=None*)

force(**languages*)
Selects language(s) for translation

if a list of languages is passed as a parameter, the first language from this list that matches the ones from the possible_languages dictionary will be selected

default language will be selected if none of them matches possible_languages.

get_possible_languages()
Gets list of all possible languages for current application

get_possible_languages_info(*lang=None*)
Returns info for selected language or dictionary with all possible languages info from *APP/languages/*.py*
It Returns:

- a tuple containing:

```
langcode, langname, langfile_mtime,
pluraldict_fname, pluraldict_mtime,
prules_langcode, nplurals,
get_plural_id, construct_plural_form

or None
```

- if *lang* is NOT defined a dictionary with all possible languages:

```
{ langcode(from filename):
    ( langcode,         # language code from !langcode!
      langname,
          # language name in national spelling from !langname!
      langfile_mtime,  # m_time of language file
      pluraldict_fname,# name of plural dictionary file or None (when␣
↪default.py is not exist)
      pluraldict_mtime,# m_time of plural dictionary file or 0 if file is␣
↪not exist
      prules_langcode, # code of plural rules language or 'default'
      nplurals,        # nplurals for current language
      get_plural_id,   # get_plural_id() for current language
      construct_plural_form) # construct_plural_form() for current␣
↪language
}
```

Parameters **lang** (*str*) – language

get_t(*message*, *prefix=''*)
Use ## to add a comment into a translation string the comment can be useful do discriminate different possible translations for the same string (for example different locations):

```
T(' hello world ') -> ' hello world '
T(' hello world ## token') -> ' hello world '
T('hello ## world## token') -> 'hello ## world'
```

the ## notation is ignored in multiline strings and strings that start with ##. This is needed to allow markmin syntax to be translated

params_substitution(*message*, *symbols*)
Substitutes parameters from symbols into message using %. also parse *%%{}* placeholders for plural-forms processing.

Returns string with parameters

Note: *symbols* MUST BE OR tuple OR dict of parameters!

plural(*word*, *n*)
Gets plural form of word for number *n* invoked from T()/T.M() in *%%{}* tag

Note: "word" MUST be defined in current language (T.accepted_language)

Parameters

- **word** (*str*) – word in singular
- **n** (*numeric*) – number plural form created for

Returns word in appropriate singular/plural form

Return type word (str)

set_current_languages(**languages*)
Sets current AKA "default" languages Setting one of this languages makes the force() function to turn translation off

translate(*message*, *symbols*)
Gets cached translated message with inserted parameters(symbols)

gluon.languages.**findT**(*path*, *language='en'*)

Note: Must be run by the admin app

gluon.languages.**update_all_languages**(*application_path*)

Note: Must be run by the admin app

CHAPTER 16

main Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

16.1 The gluon wsgi application

gluon.main.**wsgibase**(*environ*, *responder*)

The gluon wsgi application. The first function called when a page is requested (static or dynamic). It can be called by paste.httpserver or by apache mod_wsgi (or any WSGI-compatible server).

- fills request with info
- the environment variables, replacing '.' with '_'
- adds web2py path and version info
- compensates for fcgi missing path_info and query_string
- validates the path in url

The url path must be either:

1. for static pages:

- /<application>/static/<file>

2. for dynamic pages:

- /<application>[/<controller>[/<function>[/<sub>]]][.<extension>]

The naming conventions are:

- application, controller, function and extension may only contain *[a-zA-Z0-9_]*

- file and sub may also contain '-', '=', '.' and '/'

gluon.main.**save_password**(*password*, *port*)
Used by main() to save the password in the parameters_port.py file.

gluon.main.**appfactory**(*wsgiapp=<function wsgibase>*, *logfilename='httpserver.log'*, *profiler_dir=None*, *profilerfilename=None*)
generates a wsgi application that does logging and profiling and calls wsgibase

Parameters

- **wsgiapp** – the base application
- **logfilename** – where to store apache-compatible requests log
- **profiler_dir** – where to store profile files

class gluon.main.**HttpServer**(*ip='127.0.0.1'*, *port=8000*, *password=''*, *pid_filename='httpserver.pid'*, *log_filename='httpserver.log'*, *profiler_dir=None*, *ssl_certificate=None*, *ssl_private_key=None*, *ssl_ca_certificate=None*, *min_threads=None*, *max_threads=None*, *server_name=None*, *request_queue_size=5*, *timeout=10*, *socket_timeout=1*, *shutdown_timeout=None*, *path=None*, *interfaces=None*)
Bases: object

the web2py web server (Rocket)

start()
start the web server

stop(*stoplogging=False*)
stop cron and the web server

CHAPTER 17

messageboxhandler Module

class gluon.messageboxhandler.**MessageBoxHandler**
Bases: logging.Handler

emit(*record*)
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

class gluon.messageboxhandler.**NotifySendHandler**
Bases: logging.Handler

emit(*record*)
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

CHAPTER 18

myregex Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

18.1 Useful regexes

CHAPTER 19

newcron Module

This file is part of the web2py Web Framework
Created by Attila Csipa <web2py@csipa.in.rs>
Modified by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

Cron-style interface

class gluon.newcron.**Token**(*path*)
Bases: object

acquire(*startup=False*)
Returns the time when the lock is acquired or None if cron already running

lock is implemented by writing a pickle (start, stop) in cron.master start is time when cron job starts and stop is time when cron completed stop == 0 if job started but did not yet complete if a cron job started within less than 60 seconds, acquire returns None if a cron job started before 60 seconds and did not stop, a warning is issue "Stale cron.master detected"

release()
Writes into cron.master the time when cron job was completed

gluon.newcron.**absolute_path_link**(*path*)
Returns an absolute path for the destination of a symlink

gluon.newcron.**crondance**(*applications_parent*, *ctype='soft'*, *startup=False*, *apps=None*)

class gluon.newcron.**cronlauncher**(*cmd*, *shell=True*)
Bases: threading.Thread

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class gluon.newcron.**extcron**(*applications_parent*, *apps=None*)
Bases: threading.Thread

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class gluon.newcron.**hardcron**(*applications_parent*)
Bases: threading.Thread

launch()

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

gluon.newcron.**parsecronline**(*line*)

gluon.newcron.**rangetolist**(*s*, *period='min'*)

class gluon.newcron.**softcron**(*applications_parent*)
Bases: threading.Thread

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

gluon.newcron.**stopcron**()
Graceful shutdown of cron

CHAPTER 20

portalocker Module

Cross-platform (posix/nt) API for flock-style file locking.

Synopsis:

```
import portalocker
file = open("somefile", "r+")
portalocker.lock(file, portalocker.LOCK_EX)
file.seek(12)
file.write("foo")
file.close()
```

If you know what you're doing, you may choose to:

```
portalocker.unlock(file)
```

before closing the file, but why?

Methods:

```
lock( file, flags )
unlock( file )
```

Constants:

```
LOCK_EX
LOCK_SH
LOCK_NB
```

I learned the win32 technique for locking files from sample code provided by John Nielsen <nielsenjf@my-deja.com> in the documentation that accompanies the win32 modules.

Author: Jonathan Feinberg <jdf@pobox.com> Version: $Id: portalocker.py,v 1.3 2001/05/29 18:47:55 Administrator Exp $

class gluon.portalocker.**LockedFile**(*filename*, *mode='rb'*)

Bases: object

close()

read(*size=None*)

readline()

readlines()

write(*data*)

gluon.portalocker.**lock**(*file*, *flags*)

gluon.portalocker.**read_locked**(*filename*)

gluon.portalocker.**unlock**(*file*)

gluon.portalocker.**write_locked**(*filename*, *data*)

CHAPTER 21

recfile Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

21.1 Generates names for cache and session files

gluon.recfile.**exists**(*filename*, *path=None*)

gluon.recfile.**generate**(*filename*, *depth=2*, *base=512*)

gluon.recfile.**open**(*filename*, *mode='r'*, *path=None*)

gluon.recfile.**remove**(*filename*, *path=None*)

CHAPTER 22

restricted Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

22.1 Restricted environment to execute application's code

exception gluon.restricted.**RestrictedError**(*layer=''*, *code=''*, *output=''*, *environment=None*)

Bases: exceptions.Exception

Class used to wrap an exception that occurs in the restricted environment below. The traceback is used to log the exception and generate a ticket.

load(*request*, *app*, *ticket_id*)
Loads a logged exception.

log(*request*)
Logs the exception.

gluon.restricted.**restricted**(*code*, *environment=None*, *layer='Unknown'*)
Runs code in environment and returns the output. If an exception occurs in code it raises a RestrictedError containing the traceback. Layer is passed to RestrictedError to identify where the error occurred.

class gluon.restricted.**TicketStorage**(*db=None*, *tablename='web2py_ticket'*)

Bases: *gluon.storage.Storage*

Defines the ticket object and the default values of its members (None)

load(*request*, *app*, *ticket_id*)

store(*request*, *ticket_id*, *ticket_data*)
Stores the ticket. It will figure out if this must be on disk or in db

gluon.restricted.**compile2**(*code*, *layer*)
The +'\n' is necessary else compile fails when code ends in a comment.

CHAPTER 23

rewrite Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

gluon.rewrite parses incoming URLs and formats outgoing URLs for gluon.html.URL.

In addition, it rewrites both incoming and outgoing URLs based on the (optional) user-supplied routes.py, which also allows for rewriting of certain error messages.

routes.py supports two styles of URL rewriting, depending on whether 'routers' is defined. Refer to router.example.py and routes.example.py for additional documentation.

class gluon.rewrite.**MapUrlIn**(*request=None*, *env=None*)
Bases: object

Logic for mapping incoming URLs

arg0
Returns first arg

harg0
Returns first arg with optional hyphen mapping

map_app()
Determines application name

map_controller()
Identifies controller

map_function()
Handles function.extension

map_language()
Handles language (no hyphen mapping)

map_prefix()
Strips path prefix, if present in its entirety

map_root_static()
Handles root-static files (no hyphen mapping)

a root-static file is one whose incoming URL expects it to be at the root, typically robots.txt & favicon.ico

map_static()
Handles static files file_match but no hyphen mapping

pop_arg_if(*dopop*)
Conditionally removes first arg and returns new first arg

sluggify()

update_request()
Updates request from self Builds env.request_uri Makes lower-case versions of http headers in env

validate_args()
Checks args against validation pattern

class gluon.rewrite.**MapUrlOut**(*request*, *env*, *application*, *controller*, *function*, *args*, *other*, *scheme*, *host*, *port*, *language*)
Bases: object

Logic for mapping outgoing URLs

acf()
Converts components to /app/lang/controller/function

build_acf()
Builds a/c/f from components

omit_acf()
Omits what we can of a/c/f

omit_lang()
Omits language if possible

gluon.rewrite.**compile_regex**(*k*, *v*, *env=None*)
Preprocess and compile the regular expressions in routes_app/in/out The resulting regex will match a pattern of the form:

```
[remote address]:[protocol]://[host]:[method] [path]
```

We allow abbreviated regexes on input; here we try to complete them.

gluon.rewrite.**filter_err**(*status*, *application='app'*, *ticket='tkt'*)
doctest/unittest interface to routes_onerror

gluon.rewrite.**filter_url**(*url*, *method='get'*, *remote='0.0.0.0'*, *out=False*, *app=False*, *lang=None*, *domain=(None, None)*, *env=False*, *scheme=None*, *host=None*, *port=None*, *language=None*)
doctest/unittest interface to regex_filter_in() and regex_filter_out()

gluon.rewrite.**fixup_missing_path_info**(*environ*)

gluon.rewrite.**get_effective_router**(*appname*)
Returns a private copy of the effective router for the specified application

gluon.rewrite.**invalid_url**(*routes*)

gluon.rewrite.**load**(*routes='routes.py'*, *app=None*, *data=None*, *rdict=None*)
load: read (if file) and parse routes store results in params (called from main.py at web2py initialization time) If data is present, it's used instead of the routes.py contents. If rdict is present, it must be a dict to be used for routers (unit test)

gluon.rewrite.**load_routers**(*all_apps*)
Load-time post-processing of routers

gluon.rewrite.**log_rewrite**(*string*)
Log rewrite activity under control of routes.py

gluon.rewrite.**map_url_in**(*request*, *env*, *app=False*)
Routes incoming URL

gluon.rewrite.**map_url_out**(*request*, *env*, *application*, *controller*, *function*, *args*, *other*, *scheme*, *host*, *port*, *language=None*)
Supply /a/c/f (or /a/lang/c/f) portion of outgoing url

The basic rule is that we can only make transformations that map_url_in can reverse.

Suppose that the incoming arguments are a,c,f,args,lang and that the router defaults are da, dc, df, dl.

We can perform these transformations trivially if args=[] and lang=None or dl:

```
/da/dc/df => /
/a/dc/df => /a
/a/c/df => /a/c
```

We would also like to be able to strip the default application or application/controller from URLs with function/args present, thus:

```
/da/c/f/args  => /c/f/args
/da/dc/f/args => /f/args
```

We use [applications] and [controllers] and {functions} to suppress ambiguous omissions.

We assume that language names do not collide with a/c/f names.

gluon.rewrite.**regex_filter_in**(*e*)
Regex rewrite incoming URL

gluon.rewrite.**regex_filter_out**(*url*, *e=None*)
Regex rewrite outgoing URL

gluon.rewrite.**regex_select**(*env=None*, *app=None*, *request=None*)
Selects a set of regex rewrite params for the current request

gluon.rewrite.**regex_uri**(*e*, *regexes*, *tag*, *default=None*)
Filters incoming URI against a list of regexes

gluon.rewrite.**regex_url_in**(*request*, *environ*)
Rewrites and parses incoming URL

gluon.rewrite.**sluggify**(*key*)

gluon.rewrite.**try_redirect_on_error**(*http_object*, *request*, *ticket=None*)
Called from main.wsgibase to rewrite the http response

gluon.rewrite.**try_rewrite_on_error**(*http_response*, *request*, *environ*, *ticket=None*)
Called from main.wsgibase to rewrite the http response.

gluon.rewrite.**url_in**(*request*, *environ*)
Parses and rewrites incoming URL

gluon.rewrite.**url_out**(*request*, *environ*, *application*, *controller*, *function*, *args*, *other*, *scheme*, *host*, *port*, *language=None*)

Assembles and rewrites outgoing URL

CHAPTER 24

sanitizer Module

From http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/496942
Submitter: Josh Goldfoot (other recipes)
Last Updated: 2006/08/05
Version: 1.0

24.1 Cross-site scripting (XSS) defense

gluon.sanitizer.**sanitize**(*text, permitted_tags=['a', 'b', 'blockquote', 'br/', 'i', 'li', 'ol', 'ul', 'p', 'cite', 'code', 'pre', 'img/', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'table', 'tbody', 'thead', 'tfoot', 'tr', 'td', 'div', 'strong', 'span'], allowed_attributes={'a': ['href', 'title'], 'blockquote': ['type'], 'img': ['src', 'alt'], 'td': ['colspan']}, escape=True*)

CHAPTER 25

scheduler Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

25.1 Background processes made simple

class gluon.scheduler.**JobGraph**(*db*, *job_name*)
Bases: object

Experimental: dependencies amongs tasks

add_deps(*task_parent*, *task_child*)
Create a dependency between task_parent and task_child.

validate(*job_name=None*)
Validate if all tasks job_name can be completed.

Checks if there are no mutual dependencies among tasks. Commits at the end if successfull, or it rollbacks the entire transaction. Handle with care!

class gluon.scheduler.**MetaScheduler**
Bases: threading.Thread

Base class documenting scheduler's base methods

async(*task*)
Start the background process.

Parameters **task** – a *Task* object

Returns

containing:

```
('ok',result,output)
('error',exception,None)
('timeout',None,None)
('terminated',None,None)
```

Return type tuple

die()
Forces termination of the worker process along with any running task

give_up()
Waits for any running task to be executed, then exits the worker process

loop()
Main loop, fetching tasks and starting executor's background processes

pop_task()
Fetches a task ready to be executed

report_task(*task*, *task_report*)
Creates a task report

run()
This is executed by the main thread to send heartbeats

send_heartbeat(*counter*)

sleep()

start_heartbeats()

terminate_process()
Terminates any running tasks (internal use only)

class gluon.scheduler.**Scheduler**(*db*, *tasks=None*, *migrate=True*, *worker_name=None*, *group_names=None*, *heartbeat=3*, *max_empty_runs=0*, *discard_results=False*, *utc_time=False*)

Bases: *gluon.scheduler.MetaScheduler*

Scheduler object

Parameters

- **db** – DAL connection where Scheduler will create its tables
- **tasks** (*dict*) – either a dict containing name–>func or None. If None, functions will be searched in the environment
- **migrate** (*bool*) – turn migration on/off for the Scheduler's tables
- **worker_name** (*str*) – force worker_name to identify each process. Leave it to None to autoassign a name (hostname#pid)
- **group_names** (*list*) – process tasks belonging to this group defaults to ['main'] if nothing gets passed
- **heartbeat** (*int*) – how many seconds the worker sleeps between one execution and the following one. Indirectly sets how many seconds will pass between checks for new tasks
- **max_empty_runs** (*int*) – how many loops are allowed to pass without processing any tasks before exiting the process. 0 to keep always the process alive

- **discard_results** (*bool*) – Scheduler stores executions's details into the scheduler_run table. By default, only if there is a result the details are kept. Turning this to True means discarding results even for tasks that return something
- **utc_time** (*bool*) – do all datetime calculations assuming UTC as the timezone. Remember to pass *start_time* and *stop_time* to tasks accordingly

adj_hibernation()
Used to increase the "sleep" interval for DISABLED workers.

assign_tasks(*db*)
Assign task to workers, that can then pop them from the queue.

Deals with group_name(s) logic, in order to assign linearly tasks to available workers for those groups

being_a_ticker()
Elect a TICKER process that assigns tasks to available workers.

Does its best to elect a worker that is not busy processing other tasks to allow a proper distribution of tasks among all active workers ASAP

define_tables(*db*, *migrate*)
Define Scheduler tables structure.

disable(*group_names=None*, *limit=None*, *worker_name=None*)
Set DISABLED on the workers processing *group_names* tasks.

A DISABLED worker will be kept alive but it won't be able to process any waiting tasks, essentially putting it to sleep. By default, all group_names of Scheduler's instantation are selected

get_workers(*only_ticker=False*)
Returns a dict holding *worker_name : {**columns}* representing all "registered" workers only_ticker returns only the workers running as a TICKER, if there are any

kill(*group_names=None*, *limit=None*, *worker_name=None*)
Sets KILL as worker status. The worker will be killed even if it's processing a task.

loop(*worker_name=None*)
Main loop.

This works basically as a neverending loop that:

- checks if the worker is ready to process tasks (is not DISABLED)
- pops a task from the queue
- if there is a task:
  - spawns the executor background process
  - waits for the process to be finished
  - sleeps *heartbeat* seconds
- if there is not a task:
  - checks for max_empty_runs
  - sleeps *heartbeat* seconds

now()
Shortcut that fetches current time based on UTC preferences.

pop_task(*db*)
Grab a task ready to be executed from the queue.

queue_task(*function*, *pargs=[]*, *pvars={}*, ***kwargs*)
Queue tasks. This takes care of handling the validation of all parameters

Parameters

- **function** – the function (anything callable with a __name__)
- **pargs** – "raw" args to be passed to the function. Automatically jsonified.
- **pvars** – "raw" kwargs to be passed to the function. Automatically jsonified
- **kwargs** – all the parameters available (basically, every *scheduler_task* column). If args and vars are here, they should be jsonified already, and they will override pargs and pvars

Returns a dict just as a normal validate_and_insert(), plus a uuid key holding the uuid of the queued task. If validation is not passed (i.e. some parameters are invalid) both id and uuid will be None, and you'll get an "error" dict holding the errors found.

report_task(*task*, *task_report*)
Take care of storing the result according to preferences.

Deals with logic for repeating tasks.

resume(*group_names=None*, *limit=None*, *worker_name=None*)
Wakes a worker up (it will be able to process queued tasks)

send_heartbeat(*counter*)
Coordination among available workers.

It: - sends the heartbeat - elects a ticker among available workers (the only process that

effectively dispatch tasks to workers)

- deals with worker's statuses
- does "housecleaning" for dead workers
- triggers tasks assignment to workers

set_requirements(*scheduler_task*)
Called to set defaults for lazy_tables connections.

set_worker_status(*group_names=None*, *action='ACTIVE'*, *exclude=None*, *limit=None*, *worker_name=None*)
Internal function to set worker's status.

sleep()
Calculate the number of seconds to sleep.

stop_task(*ref*)
Shortcut for task termination.

If the task is RUNNING it will terminate it, meaning that status will be set as FAILED.

If the task is QUEUED, its stop_time will be set as to "now", the enabled flag will be set to False, and the status to STOPPED

Parameters **ref** – can be

- an integer : lookup will be done by scheduler_task.id
- a string : lookup will be done by scheduler_task.uuid

Returns

- 1 if task was stopped (meaning an update has been done)

- None if task was not found, or if task was not RUNNING or QUEUED

Note: Experimental

task_status(*ref*, *output=False*)
Retrieves task status and optionally the result of the task

Parameters

- **ref** – can be
  - an integer : lookup will be done by scheduler_task.id
  - a string : lookup will be done by scheduler_task.uuid
  - a *Query* : lookup as you wish, e.g.

```
db.scheduler_task.task_name == 'test1'
```

- **output** (*bool*) – if *True*, fetch also the scheduler_run record

Returns a single Row object, for the last queued task. If output == True, returns also the last scheduler_run record. The scheduler_run record is fetched by a left join, so it can have all fields == None

terminate(*group_names=None*, *limit=None*, *worker_name=None*)
Sets TERMINATE as worker status. The worker will wait for any currently running tasks to be executed and then it will exit gracefully

static total_seconds(*td*)
Backport for py2.6.

update_dependencies(*db*, *task_id*)
Unblock execution paths for Jobs.

wrapped_assign_tasks(*db*)
Commodity function to call *assign_tasks* and trap exceptions.

If an exception is raised, assume it happened because of database contention and retries *assign_task* after 0.5 seconds

wrapped_pop_task()
Commodity function to call *pop_task* and trap exceptions.

If an exception is raised, assume it happened because of database contention and retries *pop_task* after 0.5 seconds

wrapped_report_task(*task*, *task_report*)
Commodity function to call *report_task* and trap exceptions.

If an exception is raised, assume it happened because of database contention and retries *pop_task* after 0.5 seconds

class gluon.scheduler.**TYPE**(*myclass=<type 'list'>*, *parse=False*)
Bases: object

Validator that checks whether field is valid json and validates its type. Used for *args* and *vars* of the scheduler_task table

class gluon.scheduler.**Task**(*app*, *function*, *timeout*, *args='[]'*, *vars='{}'*, ***kwargs*)
Bases: object

Defines a "task" object that gets passed from the main thread to the executor's one

class gluon.scheduler.**TaskReport**(*status*, *result=None*, *output=None*, *tb=None*)
Bases: object

Defines a "task report" object that gets passed from the executor's thread to the main one

gluon.scheduler.**executor**(*queue*, *task*, *out*)
The function used to execute tasks in the background process.

gluon.scheduler.**main**()
allows to run worker without python web2py.py .... by simply:

```
python gluon/scheduler.py
```

CHAPTER 26

serializers Module

This file is part of the web2py Web Framework Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu> License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

gluon.serializers.**cast_keys**(*o*, *cast=<type 'str'>*, *encoding='utf-8'*)

Builds a new object with <cast> type keys. Use this function if you are in Python < 2.6.5 This avoids syntax errors when unpacking dictionary arguments.

Parameters

- **o** – is the object input
- **cast** – (defaults to str) is an object type or function which supports conversion such as:

  converted = cast(o)

- **encoding** – (defaults to utf-8) is the encoding for unicode keys. This is not used for custom cast functions

gluon.serializers.**csv**(*value*)

gluon.serializers.**custom_json**(*o*)

gluon.serializers.**ics**(*events*, *title=None*, *link=None*, *timeshift=0*, *calname=True*, ***ignored*)

gluon.serializers.**json**(*value*, *default=<function custom_json>*)

gluon.serializers.**loads_json**(*o*, *unicode_keys=True*, ***kwargs*)

gluon.serializers.**loads_yaml**(*data*)

gluon.serializers.**rss**(*feed*)

gluon.serializers.**safe_encode**(*text*)

gluon.serializers.**xml**(*value*, *encoding='UTF-8'*, *key='document'*, *quote=True*)

gluon.serializers.**xml_rec**(*value*, *key*, *quote=True*)

gluon.serializers.**yaml**(*data*)

CHAPTER 27

`settings` Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

CHAPTER 28

shell Module

This file is part of the web2py Web Framework
Developed by Massimo Di Pierro <mdipierro@cs.depaul.edu>,
limodou <limodou@gmail.com> and srackham <srackham@gmail.com>.
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

28.1 Web2py environment in the shell

gluon.shell.**die**(*msg*)

gluon.shell.**enable_autocomplete_and_history**(*adir*, *env*)

gluon.shell.**env**(*a*, *import_models=False*, *c=None*, *f=None*, *dir=''*, *extra_request={}*)
Returns web2py execution environment for application (a), controller (c), function (f). If import_models is True the exec all application models into the environment.

extra_request allows you to pass along any extra variables to the request object before your models get executed. This was mainly done to support web2py_utils.test_runner, however you can use it with any wrapper scripts that need access to the web2py environment.

gluon.shell.**exec_environment**(*pyfile=''*, *request=None*, *response=None*, *session=None*)
Environment builder and module loader.

Builds a web2py environment and optionally executes a Python file into the environment.

A Storage dictionary containing the resulting environment is returned. The working directory must be web2py root – this is the web2py default.

gluon.shell.**exec_pythonrc**()

gluon.shell.**execute_from_command_line**(*argv=None*)

gluon.shell.**get_usage**()

gluon.shell.**parse_path_info**(*path_info*, *av=False*)

Parses path info formatted like a/c/f where c and f are optional and a leading / is accepted. Return tuple (a, c, f). If invalid path_info a is set to None. If c or f are omitted they are set to None. If av=True, parse args and vars

gluon.shell.**run**(*appname*, *plain=False*, *import_models=False*, *startfile=None*, *bpython=False*, *python_code=False*, *cronjob=False*)

Start interactive shell or run Python script (startfile) in web2py controller environment. appname is formatted like:

- a : web2py application name
- a/c : exec the controller c into the application environment

gluon.shell.**test**(*testpath*, *import_models=True*, *verbose=False*)

Run doctests in web2py environment. testpath is formatted like:

- a: tests all controllers in application a
- a/c: tests controller c in application a
- a/c/f test function f in controller c, application a

Where a, c and f are application, controller and function names respectively. If the testpath is a file name the file is tested. If a controller is specified models are executed by default.

CHAPTER 29

sql Module

This file is part of the web2py Web Framework
Developed by Massimo Di Pierro <mdipierro@cs.depaul.edu>,
limodou <limodou@gmail.com> and srackham <srackham@gmail.com>.
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

29.1 Just for backward compatibility

class gluon.sql.**DAL**(*uri='sqlite://dummy.db'*, *pool_size=0*, *folder=None*, *db_codec='UTF-8'*, *check_reserved=None*, *migrate=True*, *fake_migrate=False*, *migrate_enabled=True*, *fake_migrate_all=False*, *decode_credentials=False*, *driver_args=None*, *adapter_args=None*, *attempts=5*, *auto_import=False*, *bigint_id=False*, *debug=False*, *lazy_tables=False*, *db_uid=None*, *do_connect=True*, *after_connection=None*, *tables=None*, *ignore_field_case=True*, *entity_quoting=False*, *table_hash=None*)

Bases: pydal.helpers.classes.Serializable, pydal.helpers.classes.BasicStorage

An instance of this class represents a database connection

Parameters

- **uri** (*str*) – contains information for connecting to a database. Defaults to *'sqlite://dummy.db'*

  Note: experimental: you can specify a dictionary as uri parameter i.e. with:

  ```
  db = DAL({"uri": "sqlite://storage.sqlite",
            "tables": {...}, ...})
  ```

  for an example of dict input you can check the output of the scaffolding db model with

  db.as_dict()

Note that for compatibility with Python older than version 2.6.5 you should cast your dict input keys to str due to a syntax limitation on kwarg names. for proper DAL dictionary input you can use one of:

```
obj = serializers.cast_keys(dict, [encoding="utf-8"])
#or else (for parsing json input)
obj = serializers.loads_json(data, unicode_keys=False)
```

- **pool_size** – How many open connections to make to the database object.
- **folder** – where .table files will be created. Automatically set within web2py. Use an explicit path when using DAL outside web2py
- **db_codec** – string encoding of the database (default: 'UTF-8')
- **table_hash** – database identifier with .tables. If your connection hash change you can still using old .tables if they have db_hash as prefix
- **check_reserved** – list of adapters to check tablenames and column names against sql/nosql reserved keywords. Defaults to *None*
  - 'common' List of sql keywords that are common to all database types such as "SELECT, INSERT". (recommended)
  - 'all' Checks against all known SQL keywords
  - '<adapternname>" Checks against the specific adapters list of keywords
  - '<adapternname>_nonreserved' Checks against the specific adapters list of nonreserved keywords. (if available)
- **migrate** – sets default migrate behavior for all tables
- **fake_migrate** – sets default fake_migrate behavior for all tables
- **migrate_enabled** – If set to False disables ALL migrations
- **fake_migrate_all** – If set to True fake migrates ALL tables
- **attempts** – Number of times to attempt connecting
- **auto_import** – If set to True, tries import automatically table definitions from the databases folder (works only for simple models)
- **bigint_id** – If set, turn on bigint instead of int for id and reference fields
- **lazy_tables** – delays table definition until table access
- **after_connection** – can a callable that will be executed after the connection

Example

Use as:

```
db = DAL('sqlite://test.db')
```

or:

```
db = DAL(**{"uri": ..., "tables": [...]...}) # experimental

db.define_table('tablename', Field('fieldname1'),
                             Field('fieldname2'))
```

class Field(*fieldname*, *type='string'*, *length=None*, *default=<function <lambda>>*, *required=False*, *requires=<function <lambda>>*, *ondelete='CASCADE'*, *notnull=False*, *unique=False*, *uploadfield=True*, *widget=None*, *label=None*, *comment=None*, *writable=True*, *readable=True*, *update=None*, *authorize=None*, *autodelete=False*, *represent=None*, *uploadfolder=None*, *uploadseparate=False*, *uploadfs=None*, *compute=None*, *custom_store=None*, *custom_retrieve=None*, *custom_retrieve_file_properties=None*, *custom_delete=None*, *filter_in=None*, *filter_out=None*, *custom_qualifier=None*, *map_none=None*, *rname=None*)

Bases: `pydal.objects.Expression`, `pydal.helpers.classes.Serializable`

Lazy
alias of `FieldMethod`

Method
alias of `FieldMethod`

Virtual
alias of `FieldVirtual`

as_dict(*flat=False*, *sanitize=True*)

clone(*point_self_references_to=False*, ***args*)

count(*distinct=None*)

formatter(*value*)

retrieve(*name*, *path=None*, *nameonly=False*)
If *nameonly==True* return (filename, fullfilename) instead of (filename, stream)

retrieve_file_properties(*name*, *path=None*)

set_attributes(**args*, ***attributes*)

sqlsafe

sqlsafe_name

store(*file*, *filename=None*, *path=None*)

validate(*value*)

class Row(**args*, ***kwargs*)

Bases: `pydal.helpers.classes.BasicStorage`

A dictionary that lets you do d['a'] as well as d.a this is only used to store a *Row*

as_dict(*datetime_to_str=False*, *custom_types=None*)

as_json(*mode='object'*, *default=None*, *colnames=None*, *serialize=True*, ***kwargs*)
serializes the row to a JSON object kwargs are passed to .as_dict method only "object" mode supported

serialize = *False* used by Rows.as_json

TODO: return array mode with query column order

mode and colnames are not implemented

as_xml(*row_name='row'*, *colnames=None*, *indent=' '*)

get(*key*, *default=None*)

class Table(*db*, *tablename*, **fields*, ***args*)

Bases: pydal.helpers.classes.Serializable, pydal.helpers.classes.BasicStorage

Represents a database table

Example::

You can create a table as:: db = DAL(...) db.define_table('users', Field('name'))

And then:

```
db.users.insert(name='me') # print db.users._insert(...) to see SQL
db.users.drop()
```

as_dict(*flat=False*, *sanitize=True*)

bulk_insert(*items*)

here items is a list of dictionaries

drop(*mode=''*)

fields

import_from_csv_file(*csvfile*, *id_map=None*, *null='<NULL>'*, *unique='uuid'*, *id_offset=None*, **args*, ***kwargs*)

Import records from csv file. Column headers must have same names as table fields. Field 'id' is ignored. If column names read 'table.file' the 'table.' prefix is ignored.

- 'unique' argument is a field which must be unique (typically a uuid field)
- 'restore' argument is default False; if set True will remove old values in table first.
- 'id_map' if set to None will not map ids

The import will keep the id numbers in the restored table. This assumes that there is an field of type id that is integer and in incrementing order. Will keep the id numbers in restored table.

insert(***fields*)

on(*query*)

sqlsafe

sqlsafe_alias

truncate(*mode=None*)

update(**args*, ***kwargs*)

update_or_insert(*_key=<function <lambda>>*, ***values*)

validate_and_insert(***fields*)

validate_and_update(*_key=<function <lambda>>*, ***fields*)

validate_and_update_or_insert(*_key=<function <lambda>>*, ***fields*)

with_alias(*alias*)

as_dict(*flat=False*, *sanitize=True*)

can_join()

check_reserved_keyword(*name*)

Validates *name* against SQL keywords Uses self.check_reserve which is a list of operators to use.

close()

commit()

define_table(*tablename*, **fields*, ***args*)

static distributed_transaction_begin(**instances*)

static distributed_transaction_commit(**instances*)

executesql(*query*, *placeholders=None*, *as_dict=False*, *fields=None*, *colnames=None*, *as_ordered_dict=False*)

Executes an arbitrary query

Parameters

- **query** (*str*) – the query to submit to the backend
- **placeholders** – is optional and will always be None. If using raw SQL with placeholders, placeholders may be a sequence of values to be substituted in or, (if supported by the DB driver), a dictionary with keys matching named placeholders in your SQL.
- **as_dict** – will always be None when using DAL. If using raw SQL can be set to True and the results cursor returned by the DB driver will be converted to a sequence of dictionaries keyed with the db field names. Results returned with as_dict=True are the same as those returned when applying .to_list() to a DAL query. If "as_ordered_dict"=True the behaviour is the same as when "as_dict"=True with the keys (field names) guaranteed to be in the same order as returned by the select name executed on the database.
- **fields** – list of DAL Fields that match the fields returned from the DB. The Field objects should be part of one or more Table objects defined on the DAL object. The "fields" list can include one or more DAL Table objects in addition to or instead of including Field objects, or it can be just a single table (not in a list). In that case, the Field objects will be extracted from the table(s).

  Note: if either *fields* or *colnames* is provided, the results will be converted to a DAL *Rows* object using the *db._adapter.parse()* method

- **colnames** – list of field names in tablename.fieldname format

Note: It is also possible to specify both "fields" and the associated "colnames". In that case, "fields" can also include DAL Expression objects in addition to Field objects. For Field objects in "fields", the associated "colnames" must still be in tablename.fieldname format. For Expression objects in "fields", the associated "colnames" can be any arbitrary labels.

DAL Table objects referred to by "fields" or "colnames" can be dummy tables and do not have to represent any real tables in the database. Also, note that the "fields" and "colnames" must be in the same order as the fields in the results cursor returned from the DB.

export_to_csv_file(*ofile*, **args*, ***kwargs*)

static get_instances()

Returns a dictionary with uri as key with timings and defined tables:

```
{'sqlite://storage.sqlite': {
    'dbstats': [(select auth_user.email from auth_user, 0.02009)],
    'dbtables': {
        'defined': ['auth_cas', 'auth_event', 'auth_group',
            'auth_membership', 'auth_permission', 'auth_user'],
```

(continues on next page)

(continued from previous page)

```
        'lazy': '[]'
        }
    }
}
```

has_representer(*name*)

import_from_csv_file(*ifile*, *id_map=None*, *null='<NULL>'*, *unique='uuid'*, *map_tablenames=None*, *ignore_missing_tables=False*, **args*, ***kwargs*)

import_table_definitions(*path*, *migrate=False*, *fake_migrate=False*, *tables=None*)

lazy_define_table(*tablename*, **fields*, ***args*)

logger = <logging.Logger object>

parse_as_rest(*patterns*, *args*, *vars*, *queries=None*, *nested_select=True*)

Example

Use as:

```
db.define_table('person',Field('name'),Field('info'))
db.define_table('pet',
    Field('ownedby',db.person),
    Field('name'),Field('info')
)

@request.restful()
def index():
    def GET(*args,**vars):
        patterns = [
            "/friends[person]",
            "/{person.name}/:field",
            "/{person.name}/pets[pet.ownedby]",
            "/{person.name}/pets[pet.ownedby]/{pet.name}",
            "/{person.name}/pets[pet.ownedby]/{pet.name}/:field",
            ("/dogs[pet]", db.pet.info=='dog'),
            ("/dogs[pet]/{pet.name.startswith}", db.pet.info=='dog'),
            ]
        parser = db.parse_as_rest(patterns,args,vars)
        if parser.status == 200:
            return dict(content=parser.response)
        else:
            raise HTTP(parser.status,parser.error)

    def POST(table_name,**vars):
        if table_name == 'person':
            return db.person.validate_and_insert(**vars)
        elif table_name == 'pet':
            return db.pet.validate_and_insert(**vars)
        else:
            raise HTTP(400)
    return locals()
```

represent(*name*, **args*, ***kwargs*)

representers = {'rows_render': <function represent at 0x7fcf10dcdb18>, 'rows_xml': <

rollback()

serializers = {'json': <function custom_json at 0x7fcf11069488>, 'xml': <function xml

static set_folder(*folder*)

smart_query(*fields*, *text*)

tables

uuid()

validators = None

validators_method(*field*)

Field type validation, using web2py's validators mechanism.

makes sure the content of a field is in line with the declared fieldtype

where(*query=None*, *ignore_common_filters=None*)

class gluon.sql.**Field**(*fieldname*, *type='string'*, *length=None*, *default=<function <lambda>>*, *required=False*, *requires=<function <lambda>>*, *ondelete='CASCADE'*, *notnull=False*, *unique=False*, *uploadfield=True*, *widget=None*, *label=None*, *comment=None*, *writable=True*, *readable=True*, *update=None*, *authorize=None*, *autodelete=False*, *represent=None*, *uploadfolder=None*, *uploadseparate=False*, *uploadfs=None*, *compute=None*, *custom_store=None*, *custom_retrieve=None*, *custom_retrieve_file_properties=None*, *custom_delete=None*, *filter_in=None*, *filter_out=None*, *custom_qualifier=None*, *map_none=None*, *rname=None*)

Bases: pydal.objects.Expression, pydal.helpers.classes.Serializable

Lazy

Represents a database field

Example

Usage:

```
a = Field(name, 'string', length=32, default=None, required=False,
    requires=IS_NOT_EMPTY(), ondelete='CASCADE',
    notnull=False, unique=False,
    uploadfield=True, widget=None, label=None, comment=None,
    uploadfield=True, # True means store on disk,
                      # 'a_field_name' means store in this field in db
                      # False means file content will be discarded.
    writable=True, readable=True, update=None, authorize=None,
    autodelete=False, represent=None, uploadfolder=None,
    uploadseparate=False # upload to separate directories by uuid_keys
                         # first 2 character and tablename.fieldname
                         # False - old behavior
                         # True - put uploaded file in
                         #   <uploaddir>/<tablename>.<fieldname>/uuid_key[:2]
                         #        directory)
    uploadfs=None        # a pyfilesystem where to store upload
    )
```

to be used as argument of *DAL.define_table*

alias of FieldMethod

Method
alias of `FieldMethod`

Virtual
alias of `FieldVirtual`

as_dict(*flat=False*, *sanitize=True*)

clone(*point_self_references_to=False*, ***args*)

count(*distinct=None*)

formatter(*value*)

retrieve(*name*, *path=None*, *nameonly=False*)
If *nameonly==True* return (filename, fullfilename) instead of (filename, stream)

retrieve_file_properties(*name*, *path=None*)

set_attributes(**args*, ***attributes*)

sqlsafe

sqlsafe_name

store(*file*, *filename=None*, *path=None*)

validate(*value*)

CHAPTER 30

sqlhtml Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

Holds:

- SQLFORM: provide a form for a table (with/without record)
- SQLTABLE: provides a table for a set of records
- form_factory: provides a SQLFORM for an non-db backed table

class gluon.sqlhtml.**AutocompleteWidget**(*request*, *field*, *id_field=None*, *db=None*, *orderby=None*, *limitby=(0, 10)*, *distinct=False*, *keyword='_autocomplete_%(tablename)s_%(fieldname)s'*, *min_length=2*, *help_fields=None*, *help_string=None*, *at_beginning=True*)

Bases: object

callback()

class gluon.sqlhtml.**BooleanWidget**

Bases: *gluon.sqlhtml.FormWidget*

classmethod widget(*field*, *value*, ***attributes*)

Generates an INPUT checkbox tag.

see also: *FormWidget.widget*

class gluon.sqlhtml.**CacheRepresenter**

Bases: object

class gluon.sqlhtml.**CheckboxesWidget**

Bases: *gluon.sqlhtml.OptionsWidget*

classmethod widget(*field*, *value*, ***attributes*)

Generates a TABLE tag, including INPUT checkboxes (multiple allowed)

see also: *FormWidget.widget*

class gluon.sqlhtml.**DateWidget**
Bases: *gluon.sqlhtml.StringWidget*

class gluon.sqlhtml.**DatetimeWidget**
Bases: *gluon.sqlhtml.StringWidget*

class gluon.sqlhtml.**DecimalWidget**
Bases: *gluon.sqlhtml.StringWidget*

class gluon.sqlhtml.**DoubleWidget**
Bases: *gluon.sqlhtml.StringWidget*

class gluon.sqlhtml.**ExportClass**(*rows*)
Bases: object

content_type = None

export()

file_ext = None

label = None

represented()

class gluon.sqlhtml.**ExporterCSV**(*rows*)
Bases: *gluon.sqlhtml.ExportClass*

content_type = 'text/csv'

export()

file_ext = 'csv'

label = 'CSV'

class gluon.sqlhtml.**ExporterCSV_hidden**(*rows*)
Bases: *gluon.sqlhtml.ExportClass*

content_type = 'text/csv'

export()

file_ext = 'csv'

label = 'CSV'

class gluon.sqlhtml.**ExporterHTML**(*rows*)
Bases: *gluon.sqlhtml.ExportClass*

content_type = 'text/html'

export()

file_ext = 'html'

label = 'HTML'

class gluon.sqlhtml.**ExporterJSON**(*rows*)
Bases: *gluon.sqlhtml.ExportClass*

content_type = 'application/json'

export()

file_ext = 'json'

label = 'JSON'

class gluon.sqlhtml.**ExporterTSV**(*rows*)
Bases: *gluon.sqlhtml.ExportClass*

content_type = 'text/tab-separated-values'

export()

file_ext = 'csv'

label = 'TSV'

class gluon.sqlhtml.**ExporterXML**(*rows*)
Bases: *gluon.sqlhtml.ExportClass*

content_type = 'text/xml'

export()

file_ext = 'xml'

label = 'XML'

class gluon.sqlhtml.**FormWidget**
Bases: object

Helper for SQLFORM to generate form input fields (widget), related to the fieldtype

classmethod widget(*field*, *value*, ***attributes*)
Generates the widget for the field.

When serialized, will provide an INPUT tag:

- id = tablename_fieldname
- class = field.type
- name = fieldname

Parameters

- **field** – the field needing the widget
- **value** – value
- **attributes** – any other attributes to be applied

class gluon.sqlhtml.**IntegerWidget**
Bases: *gluon.sqlhtml.StringWidget*

class gluon.sqlhtml.**JSONWidget**
Bases: *gluon.sqlhtml.FormWidget*

classmethod widget(*field*, *value*, ***attributes*)
Generates a TEXTAREA for JSON notation.

see also: *FormWidget.widget*

class gluon.sqlhtml.**ListWidget**
Bases: *gluon.sqlhtml.StringWidget*

classmethod widget(*field*, *value*, ***attributes*)
Generates an INPUT text tag.

see also: *FormWidget.widget*

class gluon.sqlhtml.**MultipleOptionsWidget**
Bases: *gluon.sqlhtml.OptionsWidget*

classmethod widget(*field*, *value*, *size=5*, ***attributes*)
Generates a SELECT tag, including OPTIONs (multiple options allowed)

see also: *FormWidget.widget*

Parameters **size** – optional param (default=5) to indicate how many rows must be shown

class gluon.sqlhtml.**OptionsWidget**
Bases: *gluon.sqlhtml.FormWidget*

static has_options(*field*)
Checks if the field has selectable options

Parameters **field** – the field needing checking

Returns True if the field has options

classmethod widget(*field*, *value*, ***attributes*)
Generates a SELECT tag, including OPTIONs (only 1 option allowed)

see also: *FormWidget.widget*

class gluon.sqlhtml.**PasswordWidget**
Bases: *gluon.sqlhtml.FormWidget*

classmethod widget(*field*, *value*, ***attributes*)
Generates a INPUT password tag. If a value is present it will be shown as a number of '*', not related to the length of the actual value.

see also: *FormWidget.widget*

class gluon.sqlhtml.**RadioWidget**
Bases: *gluon.sqlhtml.OptionsWidget*

classmethod widget(*field*, *value*, ***attributes*)
Generates a TABLE tag, including INPUT radios (only 1 option allowed)

see also: *FormWidget.widget*

class gluon.sqlhtml.**SQLFORM**(*table, record=None, deletable=False, linkto=None, upload=None, fields=None, labels=None, col3={}, submit_button='Submit', delete_label='Check to delete', showid=True, readonly=False, comments=True, keepopts=[], ignore_rw=False, record_id=None, formstyle=None, buttons=['submit'], separator=None, extra_fields=None, **attributes*)
Bases: *gluon.html.FORM*

SQLFORM is used to map a table (and a current record) into an HTML form.

Given a Table like db.table

Generates an insert form:

```
SQLFORM(db.table)
```

Generates an update form:

```
record=db.table[some_id]
SQLFORM(db.table, record)
```

Generates an update with a delete button:

```
SQLFORM(db.table, record, deletable=True)
```

Parameters

- **table** – *Table* object
- **record** – either an int if the *id* is an int, or the record fetched from the table
- **deletable** – adds the delete checkbox
- **linkto** – the URL of a controller/function to access referencedby records
- **upload** – the URL of a controller/function to download an uploaded file
- **fields** – a list of fields that should be placed in the form, default is all.
- **labels** – a dictionary with labels for each field, keys are the field names.
- **col3** – a dictionary with content for an optional third column (right of each field). keys are field names.
- **submit_button** – text to show in the submit button
- **delete_label** – text to show next to the delete checkbox
- **showid** – shows the id of the record
- **readonly** – doesn't allow for any modification
- **comments** – show comments (stored in *col3* or in Field definition)
- **ignore_rw** – overrides readable/writable attributes
- **record_id** – used to create session key against CSRF
- **formstyle** – what to use to generate the form layout
- **buttons** – override buttons as you please (will be also stored in *form.custom.submit*)
- **separator** – character as separator between labels and inputs

any named optional attribute is passed to the <form> tag for example _class, _id, _style, _action, _method, etc.

AUTOTYPES = {<type 'int'>: ('integer', <gluon.validators.IS_INT_IN_RANGE object at 0x

FIELDKEY_DELETE_RECORD = 'delete_record'

FIELDNAME_REQUEST_DELETE = 'delete_this_record'

ID_LABEL_SUFFIX = '__label'

ID_ROW_SUFFIX = '__row'

accepts(*request_vars*, *session=None*, *formname='%(tablename)s/%(record_id)s'*, *keepvalues=None*, *onvalidation=None*, *dbio=True*, *hideerror=False*, *detect_record_change=False*, ***kwargs*)

Similar to *FORM.accepts* but also does insert, update or delete in DAL. If detect_record_change is *True* than:

- *form.record_changed* = *False* (record is properly validated/submitted)
- *form.record_changed* = *True* (record cannot be submitted because changed)

If detect_record_change == False than:

- *form.record_changed* = *None*

assert_status(*status*, *request_vars*)

static build_query(*fields*, *keywords*)

createform(*xfields*)

static dictform(*dictionary*, ***kwargs*)

static factory(**fields*, ***attributes*)
Generates a SQLFORM for the given fields.

Internally will build a non-database based data model to hold the fields.

formstyles = {'bootstrap': <function formstyle_bootstrap at 0x7fcf10de99b0>, 'bootstra

static grid(*query*, *fields=None*, *field_id=None*, *left=None*, *headers={}*, *orderby=None*, *groupby=None*, *searchable=True*, *sortable=True*, *paginate=20*, *deletable=True*, *editable=True*, *details=True*, *selectable=None*, *create=True*, *csv=True*, *links=None*, *links_in_grid=True*, *upload='<default>'*, *args=[]*, *user_signature=True*, *maxtextlengths={}*, *maxtextlength=20*, *onvalidation=None*, *onfailure=None*, *oncreate=None*, *onupdate=None*, *ondelete=None*, *sorter_icons=(<gluon.html.XML object>, <gluon.html.XML object>)*, *ui='web2py'*, *showbuttontext=True*, *_class='web2py_grid'*, *formname='web2py_grid'*, *search_widget='default'*, *advanced_search=True*, *ignore_rw=False*, *formstyle=None*, *exportclasses=None*, *formargs={}*, *createargs={}*, *editargs={}*, *viewargs={}*, *selectable_submit_button='Submit'*, *buttons_placement='right'*, *links_placement='right'*, *noconfirm=False*, *cache_count=None*, *client_side_delete=False*, *ignore_common_filters=None*, *auto_pagination=True*, *use_cursor=False*)

static search_menu(*fields*, *search_options=None*, *prefix='w2p'*)

static smartdictform(*session*, *name*, *filename=None*, *query=None*, ***kwargs*)

static smartgrid(*table*, *constraints=None*, *linked_tables=None*, *links=None*, *links_in_grid=True*, *args=None*, *user_signature=True*, *divider='>'*, *breadcrumbs_class=''*, ***kwargs*)
Builds a system of SQLFORM.grid(s) between any referenced tables

Parameters

- **table** – main table
- **constraints** (*dict*) – *{'table':query}* that limits which records can be accessible
- **links** (*dict*) – like *{'tablename':[lambda row: A(....), ...]}* that will add buttons when table tablename is displayed
- **linked_tables** (*list*) – list of tables to be linked

Example

given you defined a model as:

```
db.define_table('person', Field('name'), format='%(name)s')
db.define_table('dog',
    Field('name'), Field('owner', db.person), format='%(name)s')
db.define_table('comment', Field('body'), Field('dog', db.dog))
if db(db.person).isempty():
    from gluon.contrib.populate import populate
    populate(db.person, 300)
    populate(db.dog, 300)
    populate(db.comment, 1000)
```

in a controller, you can do:

```
@auth.requires_login()
def index():
    form=SQLFORM.smartgrid(db[request.args(0) or 'person'])
    return dict(form=form)
```

widgets = {'autocomplete': <class 'gluon.sqlhtml.AutocompleteWidget'>, 'blob': None,

class gluon.sqlhtml.**SQLTABLE**(*sqlrows*, *linkto=None*, *upload=None*, *orderby=None*, *headers={}*, *truncate=16*, *columns=None*, *th_link=''*, *extracolumns=None*, *selectid=None*, *renderstyle=False*, *cid=None*, *colgroup=False*, ***attributes*)

Bases: *gluon.html.TABLE*

Given with a Rows object, as returned by a *db().select()*, generates an html table with the rows.

Parameters

- **sqlrows** – the *Rows* object
- **linkto** – URL (or lambda to generate a URL) to edit individual records
- **upload** – URL to download uploaded files
- **orderby** – Add an orderby link to column headers.
- **headers** – dictionary of headers to headers redefinions headers can also be a string to generate the headers from data for now only headers="fieldname:capitalize", headers="labels" and headers=None are supported
- **truncate** – length at which to truncate text in table cells. Defaults to 16 characters.
- **columns** – a list or dict contaning the names of the columns to be shown Defaults to all
- **th_link** – base link to support orderby headers
- **extracolumns** – a list of dicts
- **selectid** – The id you want to select
- **renderstyle** – Boolean render the style with the table
- **cid** – use this cid for all links
- **colgroup** – #FIXME

Extracolumns example

```
[{'label':A('Extra', _href='#'),
'class': '', #class name of the header
'width':'', #width in pixels or %
'content':lambda row, rc: A('Edit', _href='edit/%s'%row.id),
'selected': False #agregate class selected to this column}]
```

style()

class gluon.sqlhtml.**StringWidget**

Bases: *gluon.sqlhtml.FormWidget*

classmethod widget(*field*, *value*, ***attributes*)

Generates an INPUT text tag.

see also: *FormWidget.widget*

class gluon.sqlhtml.**TextWidget**
Bases: *gluon.sqlhtml.FormWidget*

classmethod widget(*field*, *value*, ***attributes*)
Generates a TEXTAREA tag.

see also: *FormWidget.widget*

class gluon.sqlhtml.**TimeWidget**
Bases: *gluon.sqlhtml.StringWidget*

class gluon.sqlhtml.**UploadWidget**
Bases: *gluon.sqlhtml.FormWidget*

DEFAULT_WIDTH = '150px'

DELETE_FILE = 'delete'

GENERIC_DESCRIPTION = 'file ## download'

ID_DELETE_SUFFIX = '__delete'

static is_image(*value*)
Tries to check if the filename provided references to an image

Checking is based on filename extension. Currently recognized: gif, png, jp(e)g, bmp

Parameters **value** – filename

classmethod represent(*field*, *value*, *download_url=None*)
How to represent the file:

- with download url and if it is an image: <A href=...><IMG ...></A>
- otherwise with download url: <A href=...>file</A>
- otherwise: file

Parameters

- **field** – the field
- **value** – the field value
- **download_url** – url for the file download (default = None)

classmethod widget(*field*, *value*, *download_url=None*, ***attributes*)
generates a INPUT file tag.

Optionally provides an A link to the file, including a checkbox so the file can be deleted.

All is wrapped in a DIV.

see also: *FormWidget.widget*

Parameters

- **field** – the field
- **value** – the field value
- **download_url** – url for the file download (default = None)

gluon.sqlhtml.**add_class**(*a*, *b*)

gluon.sqlhtml.**form_factory**(**fields*, ***attributes*)
Generates a SQLFORM for the given fields.

Internally will build a non-database based data model to hold the fields.

gluon.sqlhtml.**formstyle_bootstrap**(*form*, *fields*)
bootstrap 2.3.x format form layout

gluon.sqlhtml.**formstyle_bootstrap3_inline_factory**(*col_label_size=3*)
bootstrap 3 horizontal form layout

Note: Experimental!

---

gluon.sqlhtml.**formstyle_bootstrap3_stacked**(*form*, *fields*)
bootstrap 3 format form layout

Note: Experimental!

---

gluon.sqlhtml.**formstyle_divs**(*form*, *fields*)
divs only

gluon.sqlhtml.**formstyle_inline**(*form*, *fields*)
divs only, but inline

gluon.sqlhtml.**formstyle_table2cols**(*form*, *fields*)
2 column table

gluon.sqlhtml.**formstyle_table3cols**(*form*, *fields*)
3 column table - default

gluon.sqlhtml.**formstyle_ul**(*form*, *fields*)
unordered list

gluon.sqlhtml.**represent**(*field*, *value*, *record*)

gluon.sqlhtml.**safe_float**(*x*)

gluon.sqlhtml.**safe_int**(*x*)

gluon.sqlhtml.**show_if**(*cond*)

CHAPTER 31

storage Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

Provides:

- List; like list but returns None instead of IndexOutOfBounds
- Storage; like dictionary allowing also for *obj.foo* for *obj['foo']*

class gluon.storage.**List**
Bases: list

Like a regular python list but callable. When a(i) is called if i is out of bounds returns None instead of *IndexError*.

class gluon.storage.**Storage**
Bases: dict

A Storage object is like a dictionary except *obj.foo* can be used in addition to *obj['foo']*, and setting obj.foo = None deletes item foo.

Example:

```
>>> o = Storage(a=1)
>>> print o.a
1

>>> o['a']
1

>>> o.a = 2
>>> print o['a']
2

>>> del o.a
```

(continues on next page)

(continued from previous page)

```
>>> print o.a
None
```

getfirst(*key*, *default=None*)

Returns the first value of a list or the value itself when given a *request.vars* style key.

If the value is a list, its first item will be returned; otherwise, the value will be returned as-is.

Example output for a query string of *?x=abc&y=abc&y=def*:

```
>>> request = Storage()
>>> request.vars = Storage()
>>> request.vars.x = 'abc'
>>> request.vars.y = ['abc', 'def']
>>> request.vars.getfirst('x')
'abc'
>>> request.vars.getfirst('y')
'abc'
>>> request.vars.getfirst('z')
```

getlast(*key*, *default=None*)

Returns the last value of a list or value itself when given a *request.vars* style key.

If the value is a list, the last item will be returned; otherwise, the value will be returned as-is.

Simulated output with a query string of *?x=abc&y=abc&y=def*:

```
>>> request = Storage()
>>> request.vars = Storage()
>>> request.vars.x = 'abc'
>>> request.vars.y = ['abc', 'def']
>>> request.vars.getlast('x')
'abc'
>>> request.vars.getlast('y')
'def'
>>> request.vars.getlast('z')
```

getlist(*key*)

Returns a Storage value as a list.

If the value is a list it will be returned as-is. If object is None, an empty list will be returned. Otherwise, *[value]* will be returned.

Example output for a query string of *?x=abc&y=abc&y=def*:

```
>>> request = Storage()
>>> request.vars = Storage()
>>> request.vars.x = 'abc'
>>> request.vars.y = ['abc', 'def']
>>> request.vars.getlist('x')
['abc']
>>> request.vars.getlist('y')
['abc', 'def']
>>> request.vars.getlist('z')
[]
```

class gluon.storage.**Settings**

Bases: *gluon.storage.Storage*

class gluon.storage.**Messages**(*T*)
Bases: *gluon.storage.Settings*

class gluon.storage.**StorageList**
Bases: *gluon.storage.Storage*

Behaves like Storage but missing elements defaults to [] instead of None

gluon.storage.**load_storage**(*filename*)

gluon.storage.**save_storage**(*storage*, *filename*)

CHAPTER 32

`streamer` Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

32.1 Facilities to handle file streaming

`gluon.streamer.`**`stream_file_or_304_or_206`**(*static_file*, *chunk_size=65536*, *request=None*, *headers={}*, *status=200*, *error_message=None*)

`gluon.streamer.`**`streamer`**(*stream*, *chunk_size=65536*, *bytes=None*)

CHAPTER 33

template Module

This file is part of the web2py Web Framework
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)
Author: Thadeus Burgess
Contributors:
- Massimo Di Pierro for creating the original gluon/template.py
- Jonathan Lundell for extensively testing the regex on Jython.
- Limodou (creater of uliweb) who inspired the block-element support for web2py.

33.1 Templating syntax

class gluon.template.**BlockNode**(*name=''*, *pre_extend=False*, *delimiters=('{{', '}}')*)

Bases: *gluon.template.Node*

Block Container.

This Node can contain other Nodes and will render in a hierarchical order of when nodes were added.

ie:

```
{{ block test }}
    This is default block test
{{ end }}
```

append(*node*)

Adds an element to the nodes.

Parameters **node** – Node object or string to append.

extend(*other*)

Extends the list of nodes with another BlockNode class.

Parameters **other** – BlockNode or Content object to extend from.

output(*blocks*)
Merges all nodes into a single string.

Parameters **blocks** – Dictionary of blocks that are extending from this template.

class gluon.template.**Content**(*name='ContentBlock'*, *pre_extend=False*)
Bases: *gluon.template.BlockNode*

Parent Container – Used as the root level BlockNode.

Contains functions that operate as such.

Parameters **name** – Unique name for this BlockNode

append(*node*)
Adds a node to list. If it is a BlockNode then we assign a block for it.

clear_content()

extend(*other*)
Extends the objects list of nodes with another objects nodes

insert(*other*, *index=0*)
Inserts object at index.

You may pass a list of objects and have them inserted.

class gluon.template.**DummyResponse**

write(*data*, *escape=True*)

class gluon.template.**NOESCAPE**(*text*)
A little helper to avoid escaping.

xml()

class gluon.template.**Node**(*value=None*, *pre_extend=False*)
Bases: object

Basic Container Object

class gluon.template.**SuperNode**(*name=''*, *pre_extend=False*)
Bases: *gluon.template.Node*

class gluon.template.**TemplateParser**(*text*, *name='ParserContainer'*, *context={}*, *path='views/'*, *writer='response.write'*, *lexers={}*, *delimiters=('{{', '}}')*, *_super_nodes=[]*)
Bases: object

Parse all blocks

Parameters

- **text** – text to parse
- **context** – context to parse in
- **path** – folder path to templates
- **writer** – string of writer class to use
- **lexers** – dict of custom lexers to use.
- **delimiters** – for example *('{{','}}')*

- **_super_nodes** – a list of nodes to check for inclusion this should only be set by "self.extend" It contains a list of SuperNodes from a child template that need to be handled.

default_delimiters = ('{{', '}}')

extend(*filename*)
Extends *filename*. Anything not declared in a block defined by the parent will be placed in the parent templates *{{include}}* block.

include(*content*, *filename*)
Includes `filename` here.

parse(*text*)

r_multiline = <_sre.SRE_Pattern object>

r_tag = <_sre.SRE_Pattern object>

re_block = <_sre.SRE_Pattern object>

re_pass = <_sre.SRE_Pattern object>

re_unblock = <_sre.SRE_Pattern object>

reindent(*text*)
Reindents a string of unindented python code.

to_string()
Returns the parsed template with correct indentation.

Used to make it easier to port to python3.

gluon.template.**get_parsed**(*text*)
Returns the indented python code of text. Useful for unit testing.

gluon.template.**output_aux**(*node*, *blocks*)

gluon.template.**parse_template**(*filename*, *path='views/'*, *context={}*, *lexers={}*, *delimiters=('{{', '}}')*)

Parameters

- **filename** – can be a view filename in the views folder or an input stream
- **path** – is the path of a views folder
- **context** – is a dictionary of symbols used to render the template
- **lexers** – dict of custom lexers to use
- **delimiters** – opening and closing tags

gluon.template.**render**(*content='hello world'*, *stream=None*, *filename=None*, *path=None*, *context={}*, *lexers={}*, *delimiters=('{{', '}}')*, *writer='response.write'*)
Generic render function

Parameters

- **content** – default content
- **stream** – file-like obj to read template from
- **filename** – where to find template
- **path** – base path for templates
- **context** – env

- **lexers** – custom lexers to use
- **delimiters** – opening and closing tags
- **writer** – where to inject the resulting stream

Example::

```
>>> render()
'hello world'
>>> render(content='abc')
'abc'
>>> render(content="abc'")
"abc'"
>>> render(content='''a"'bc''')
'a"'bc'
>>> render(content='a\nbc')
'a\nbc'
>>> render(content='a"bcd"e')
'a"bcd"e'
>>> render(content="'''a\nc'''")
"'''a\nc'''"
>>> render(content="'''a\'c'''")
"'''a'c'''"
>>> render(content='{{for i in range(a):}}{{=i}}<br />{{pass}}',␣
↪context=dict(a=5))
'0<br />1<br />2<br />3<br />4<br />'
>>> render(content='{%for i in range(a):%}{%=i%}<br />{%pass%}',␣
↪context=dict(a=5),delimiters=('{%','%}'))
'0<br />1<br />2<br />3<br />4<br />'
>>> render(content="{{='''hello\nworld'''}}")
'hello\nworld'
>>> render(content='{{for i in range(3):\n=i\npass}}')
'012'
```

CHAPTER 34

tools Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

34.1 Auth, Mail, PluginManager and various utilities

class gluon.tools.**Mail**(*server=None*, *sender=None*, *login=None*, *tls=True*)

Bases: object

Class for configuring and sending emails with alternative text / html body, multiple attachments and encryption support

Works with SMTP and Google App Engine.

Parameters

- **server** – SMTP server address in address:port notation
- **sender** – sender email address
- **login** – sender login name and password in login:password notation or None if no authentication is required
- **tls** – enables/disables encryption (True by default)

In Google App Engine use

```
server='gae'
```

For sake of backward compatibility all fields are optional and default to None, however, to be able to send emails at least server and sender must be specified. They are available under following fields:

```
mail.settings.server
mail.settings.sender
mail.settings.login
mail.settings.timeout = 60 # seconds (default)
```

When server is 'logging', email is logged but not sent (debug mode)

Optionally you can use PGP encryption or X509:

```
mail.settings.cipher_type = None
mail.settings.gpg_home = None
mail.settings.sign = True
mail.settings.sign_passphrase = None
mail.settings.encrypt = True
mail.settings.x509_sign_keyfile = None
mail.settings.x509_sign_certfile = None
mail.settings.x509_sign_chainfile = None
mail.settings.x509_nocerts = False
mail.settings.x509_crypt_certfiles = None

cipher_type       : None
                    gpg - need a python-pyme package and gpgme lib
                    x509 - smime
gpg_home          : you can set a GNUPGHOME environment variable
                    to specify home of gnupg
sign              : sign the message (True or False)
sign_passphrase   : passphrase for key signing
encrypt           : encrypt the message (True or False). It defaults
                    to True
                  ... x509 only ...
x509_sign_keyfile : the signers private key filename or
                    string containing the key. (PEM format)
x509_sign_certfile: the signers certificate filename or
                    string containing the cert. (PEM format)
x509_sign_chainfile: sets the optional all-in-one file where you
                     can assemble the certificates of Certification
                     Authorities (CA) which form the certificate
                     chain of email certificate. It can be a
                     string containing the certs to. (PEM format)
x509_nocerts      : if True then no attached certificate in mail
x509_crypt_certfiles: the certificates file or strings to encrypt
                      the messages with can be a file name /
                      string or a list of file names /
                      strings (PEM format)
```

Examples

Create Mail object with authentication data for remote server:

```
mail = Mail('example.com:25', 'me@example.com', 'me:password')
```

Notice for GAE users: attachments have an automatic content_id='attachment-i' where i is progressive number in this way the can be referenced from the HTML as etc.

class Attachment(*payload*, *filename=None*, *content_id=None*, *content_type=None*, *encoding='utf-8'*)

Bases: email.mime.base.MIMEBase

Email attachment

Parameters

- **payload** – path to file or file-like object with read() method
- **filename** – name of the attachment stored in message; if set to None, it will be fetched from payload path; file-like object payload must have explicit filename specified
- **content_id** – id of the attachment; automatically contained within < and >
- **content_type** – content type of the attachment; if set to None, it will be fetched from filename using gluon.contenttype module
- **encoding** – encoding of all strings passed to this function (except attachment body)

Content ID is used to identify attachments within the html body; in example, attached image with content ID 'photo' may be used in html message as a source of img tag **.

Example:: Create attachment from text file:

```
attachment = Mail.Attachment('/path/to/file.txt')

Content-Type: text/plain
MIME-Version: 1.0
Content-Disposition: attachment; filename="file.txt"
Content-Transfer-Encoding: base64

SOMEBASE64CONTENT=
```

Create attachment from image file with custom filename and cid:

```
attachment = Mail.Attachment('/path/to/file.png',
                             filename='photo.png',
                             content_id='photo')

Content-Type: image/png
MIME-Version: 1.0
Content-Disposition: attachment; filename="photo.png"
Content-Id: <photo>
Content-Transfer-Encoding: base64

SOMEOTHERBASE64CONTENT=
```

send(*to*, *subject='[no subject]'*, *message='[no message]'*, *attachments=None*, *cc=None*, *bcc=None*, *reply_to=None*, *sender=None*, *encoding='utf-8'*, *raw=False*, *headers={}*, *from_address=None*, *cipher_type=None*, *sign=None*, *sign_passphrase=None*, *encrypt=None*, *x509_sign_keyfile=None*, *x509_sign_chainfile=None*, *x509_sign_certfile=None*, *x509_crypt_certfiles=None*, *x509_nocerts=None*)

Sends an email using data specified in constructor

Parameters

- **to** – list or tuple of receiver addresses; will also accept single object
- **subject** – subject of the email
- **message** – email body text; depends on type of passed object:

  - if 2-list or 2-tuple is passed: first element will be source of plain text while second of html text;
  - otherwise: object will be the only source of plain text and html source will be set to None

  If text or html source is:

  - None: content part will be ignored,
  - string: content part will be set to it,
  - file-like object: content part will be fetched from it using it's read() method

- **attachments** – list or tuple of Mail.Attachment objects; will also accept single object
- **cc** – list or tuple of carbon copy receiver addresses; will also accept single object
- **bcc** – list or tuple of blind carbon copy receiver addresses; will also accept single object
- **reply_to** – address to which reply should be composed
- **encoding** – encoding of all strings passed to this method (including message bodies)
- **headers** – dictionary of headers to refine the headers just before sending mail, e.g. *{'X-Mailer' : 'web2py mailer'}*
- **from_address** – address to appear in the 'From:' header, this is not the envelope sender. If not specified the sender will be used
- **cipher_type** – gpg - need a python-pyme package and gpgme lib x509 - smime
- **gpg_home** – you can set a GNUPGHOME environment variable to specify home of gnupg
- **sign** – sign the message (True or False)
- **sign_passphrase** – passphrase for key signing
- **encrypt** – encrypt the message (True or False). It defaults to True. ... x509 only ...
- **x509_sign_keyfile** – the signers private key filename or string containing the key. (PEM format)
- **x509_sign_certfile** – the signers certificate filename or string containing the cert. (PEM format)
- **x509_sign_chainfile** – sets the optional all-in-one file where you can assemble the certificates of Certification Authorities (CA) which form the certificate chain of email certificate. It can be a string containing the certs to. (PEM format)
- **x509_nocerts** – if True then no attached certificate in mail
- **x509_crypt_certfiles** – the certificates file or strings to encrypt the messages with can be a file name / string or a list of file names / strings (PEM format)

Examples

Send plain text message to single address:

```
mail.send('you@example.com',
          'Message subject',
          'Plain text body of the message')
```

Send html message to single address:

```
mail.send('you@example.com',
          'Message subject',
          '<html>Plain text body of the message</html>')
```

Send text and html message to three addresses (two in cc):

```
mail.send('you@example.com',
          'Message subject',
          ('Plain text body', '<html>html body</html>'),
          cc=['other1@example.com', 'other2@example.com'])
```

Send html only message with image attachment available from the message by 'photo' content id:

```
mail.send('you@example.com',
          'Message subject',
          (None, '<html><img src="cid:photo" /></html>'),
          Mail.Attachment('/path/to/photo.jpg'
                          content_id='photo'))
```

Send email with two attachments and no body text:

```
mail.send('you@example.com,
          'Message subject',
          None,
          [Mail.Attachment('/path/to/fist.file'),
           Mail.Attachment('/path/to/second.file')])
```

Returns True on success, False on failure.

Before return, method updates two object's fields:

- self.result: return value of smtplib.SMTP.sendmail() or GAE's mail.send_mail() method
- self.error: Exception message or None if above was successful

class gluon.tools.**Auth**(*environment=None*, *db=None*, *mailer=True*, *hmac_key=None*, *controller='default'*, *function='user'*, *cas_provider=None*, *signature=True*, *secure=False*, *csrf_prevention=True*, *propagate_extension=None*, *url_index=None*, *jwt=None*, *host_names=None*)

Bases: object

accessible_query(*name*, *table*, *user_id=None*)

Returns a query with all accessible records for user_id or the current logged in user this method does not work on GAE because uses JOIN and IN

Example

Use as:

```
db(auth.accessible_query('read', db.mytable)).select(db.mytable.ALL)
```

add_group(*role*, *description=''*)

Creates a group associated to a role

add_membership(*group_id=None*, *user_id=None*, *role=None*)

Gives user_id membership of group_id or role if user is None than user_id is that of current logged in user

add_permission(*group_id*, *name='any'*, *table_name=''*, *record_id=0*)
Gives group_id 'name' access to 'table_name' and 'record_id'

allows_jwt(*otherwise=None*)

static archive(*form*, *archive_table=None*, *current_record='current_record'*, *archive_current=False*, *fields=None*)
If you have a table (db.mytable) that needs full revision history you can just do:

```
form = crud.update(db.mytable, myrecord, onaccept=auth.archive)
```

or:

```
form = SQLFORM(db.mytable, myrecord).process(onaccept=auth.archive)
```

crud.archive will define a new table "mytable_archive" and store a copy of the current record (if archive_current=True) or a copy of the previous record (if archive_current=False) in the newly created table including a reference to the current record.

fields allows to specify extra fields that need to be archived.

If you want to access such table you need to define it yourself in a model:

```
db.define_table('mytable_archive',
                Field('current_record', db.mytable),
                db.mytable)
```

Notice such table includes all fields of db.mytable plus one: current_record. crud.archive does not timestamp the stored record unless your original table has a fields like:

```
db.define_table(...,
    Field('saved_on', 'datetime',
          default=request.now, update=request.now, writable=False),
    Field('saved_by', auth.user,
          default=auth.user_id, update=auth.user_id, writable=False),
```

there is nothing special about these fields since they are filled before the record is archived.

If you want to change the archive table name and the name of the reference field you can do, for example:

```
db.define_table('myhistory',
    Field('parent_record', db.mytable), db.mytable)
```

and use it as:

```
form = crud.update(db.mytable, myrecord,
                   onaccept=lambda form:crud.archive(form,
                                                     archive_table=db.
↪myhistory,
                                                     current_record='parent_
↪record'))
```

basic(*basic_auth_realm=False*)
Performs basic login.

Parameters **basic_auth_realm** – optional basic http authentication realm. Can take str or unicode or function or callable or boolean.

reads current.request.env.http_authorization and returns basic_allowed,basic_accepted,user.

if basic_auth_realm is defined is a callable it's return value is used to set the basic authentication realm, if it's a string its content is used instead. Otherwise basic authentication realm is set to the application name. If basic_auth_realm is None or False (the default) the behavior is to skip sending any challenge.

bulk_register(*max_emails=100*)
Creates a form for ther user to send invites to other users to join

cas_login(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*, *version=2*)

cas_validate(*version=2*, *proxy=False*)

change_password(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)
Returns a form that lets the user change password

confirm_registration(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)
Returns a form to confirm user registration

default_messages = {'access_denied': 'Insufficient privileges', 'add_group_log': 'Gr
Class for authentication, authorization, role based access control.

Includes:

- registration and profile
- login and logout
- username and password retrieval
- event logging
- role creation and assignment
- user defined group/role based permission

Parameters

- **environment** – is there for legacy but unused (awful)
- **db** – has to be the database where to create tables for authentication
- **mailer** – *Mail(...)* or None (no mailer) or True (make a mailer)
- **hmac_key** – can be a hmac_key or hmac_key=Auth.get_or_create_key()
- **controller** – (where is the user action?)
- **cas_provider** – (delegate authentication to the URL, CAS2)

Authentication Example:

```
from gluon.contrib.utils import *
mail=Mail()
mail.settings.server='smtp.gmail.com:587'
mail.settings.sender='you@somewhere.com'
mail.settings.login='username:password'
auth=Auth(db)
auth.settings.mailer=mail
# auth.settings....=...
auth.define_tables()
def authentication():
    return dict(form=auth())
```

Exposes:

- *http://...{application}/{controller}/authentication/login*
- *http://...{application}/{controller}/authentication/logout*
- *http://...{application}/{controller}/authentication/register*
- *http://...{application}/{controller}/authentication/verify_email*
- *http://...{application}/{controller}/authentication/retrieve_username*
- *http://...{application}/{controller}/authentication/retrieve_password*
- *http://...{application}/{controller}/authentication/reset_password*
- *http://...{application}/{controller}/authentication/profile*
- *http://...{application}/{controller}/authentication/change_password*

On registration a group with role=new_user.id is created and user is given membership of this group.

You can create a group with:

```
group_id=auth.add_group('Manager', 'can access the manage action')
auth.add_permission(group_id, 'access to manage')
```

Here "access to manage" is just a user defined string. You can give access to a user:

```
auth.add_membership(group_id, user_id)
```

If user id is omitted, the logged in user is assumed

Then you can decorate any action:

```
@auth.requires_permission('access to manage')
def manage():
    return dict()
```

You can restrict a permission to a specific table:

```
auth.add_permission(group_id, 'edit', db.sometable)
@auth.requires_permission('edit', db.sometable)
```

Or to a specific record:

```
auth.add_permission(group_id, 'edit', db.sometable, 45)
@auth.requires_permission('edit', db.sometable, 45)
```

If authorization is not granted calls:

```
auth.settings.on_failed_authorization
```

Other options:

```
auth.settings.mailer=None
auth.settings.expiration=3600 # seconds

...

### these are messages that can be customized
...
```

default_settings = {'allow_basic_login': False, 'allow_basic_login_only': False, 'al

define_signature()

define_tables(*username=None*, *signature=None*, *enable_tokens=False*, *migrate=None*, *fake_migrate=None*)
To be called unless tables are defined manually

Examples

Use as:

```
# defines all needed tables and table files
# 'myprefix_auth_user.table', ...
auth.define_tables(migrate='myprefix_')

# defines all needed tables without migration/table files
auth.define_tables(migrate=False)
```

del_group(*group_id*)
Deletes a group

del_membership(*group_id=None*, *user_id=None*, *role=None*)
Revokes membership from group_id to user_id if user_id is None than user_id is that of current logged in user

del_permission(*group_id*, *name='any'*, *table_name=''*, *record_id=0*)
Revokes group_id 'name' access to 'table_name' and 'record_id'

email_registration(*subject*, *body*, *user*)
Sends and email invitation to a user informing they have been registered with the application

email_reset_password(*user*)

enable_record_versioning(*tables*, *archive_db=None*, *archive_names='%(tablename)s_archive'*, *current_record='current_record'*, *current_record_label=None*)
Used to enable full record versioning (including auth tables):

```
auth = Auth(db)
auth.define_tables(signature=True)
# define our own tables
db.define_table('mything',Field('name'),auth.signature)
auth.enable_record_versioning(tables=db)
```

tables can be the db (all table) or a list of tables. only tables with modified_by and modified_on fiels (as created by auth.signature) will have versioning. Old record versions will be in table 'mything_archive' automatically defined.

when you enable enable_record_versioning, records are never deleted but marked with is_active=False.

enable_record_versioning enables a common_filter for every table that filters out records with is_active = False

---

Note: If you use auth.enable_record_versioning, do not use auth.archive or you will end up with duplicates. auth.archive does explicitly what enable_record_versioning does automatically.

---

static get_or_create_key(*filename=None*, *alg='sha512'*)

get_or_create_user(*keys, update_fields=['email'], login=True, get=True*)
Used for alternate login methods: If the user exists already then password is updated. If the user doesn't yet exist, then they are created.

get_vars_next()

groups()
Displays the groups and their roles for the logged in user

has_membership(*group_id=None*, *user_id=None*, *role=None*)
Checks if user is member of group_id or role

has_permission(*name='any'*, *table_name=''*, *record_id=0*, *user_id=None*, *group_id=None*)
Checks if user_id or current logged in user is member of a group that has 'name' permission on 'table_name' and 'record_id' if group_id is passed, it checks whether the group has the permission

here()

id_group(*role*)
Returns the group_id of the group specified by the role

impersonate(*user_id=<function <lambda>>*)
To use this make a POST to *http://.../impersonate request.post_vars.user_id=<id>*

Set request.post_vars.user_id to 0 to restore original user.

requires impersonator is logged in and:

```
has_permission('impersonate', 'auth_user', user_id)
```

is_impersonating()

is_logged_in()
Checks if the user is logged in and returns True/False. If so user is in auth.user as well as in session.auth.user

jwt()
To use JWT authentication: 1) instantiate auth with:

```
auth = Auth(db, jwt = {'secret_key':'secret'})
```

where 'secret' is your own secret string.

2. Decorate functions that require login but should accept the JWT token credentials:

```
@auth.allows_jwt()
@auth.requires_login()
def myapi(): return 'hello %s' % auth.user.email
```

Notice jwt is allowed but not required. if user is logged in, myapi is accessible.

3. Use it!

Now API users can obtain a token with

http://.../app/default/user/jwt?username=...&password=....

(returns json object with a token attribute) API users can refresh an existing token with

http://.../app/default/user/jwt?token=...

they can authenticate themselves when calling http:/.../myapi by injecting a header

Authorization: Bearer <the jwt token>

Any additional attributes in the jwt argument of Auth() below:

```
auth = Auth(db, jwt = {...})
```

are passed to the constructor of class AuthJWT. Look there for documentation.

log_event(*description*, *vars=None*, *origin='auth'*)

Examples

Use as:

```
auth.log_event(description='this happened', origin='auth')
```

login(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)
Returns a login form

login_bare(*username*, *password*)
Logins user as specified by username (or email) and password

login_user(*user*)
Logins the *user = db.auth_user(id)*

logout(*next=<function <lambda>>*, *onlogout=<function <lambda>>*, *log=<function <lambda>>*)
Logouts and redirects to login

logout_bare()

manage_tokens()

navbar(*prefix='Welcome'*, *action=None*, *separators=' [', ' | ', '] '*), *user_identifier=<function <lambda>>*, *referrer_actions=<function <lambda>>*, *mode='default'*)
Navbar with support for more templates This uses some code from the old navbar.

Parameters **mode** – see options for list of

not_authorized()
You can change the view for this page to make it look as you like

profile(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)
Returns a form that lets the user change his/her profile

random_password()

register(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)
Returns a registration form

register_bare(***fields*)
Registers a user as specified by username (or email) and a raw password.

request_reset_password(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)
Returns a form to reset the user password

requires(*condition*, *requires_login=True*, *otherwise=None*)
Decorator that prevents access to action if not logged in

requires_login(*otherwise=None*)
Decorator that prevents access to action if not logged in

requires_login_or_token(*otherwise=None*)

requires_membership(*role=None*, *group_id=None*, *otherwise=None*)
Decorator that prevents access to action if not logged in or if user logged in is not a member of group_id. If role is provided instead of group_id then the group_id is calculated.

requires_permission(*name*, *table_name=''*, *record_id=0*, *otherwise=None*)
Decorator that prevents access to action if not logged in or if user logged in is not a member of any group (role) that has 'name' access to 'table_name', 'record_id'.

requires_signature(*otherwise=None*, *hash_vars=True*)
Decorator that prevents access to action if not logged in or if user logged in is not a member of group_id. If role is provided instead of group_id then the group_id is calculated.

reset_password(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)
Returns a form to reset the user password

reset_password_deprecated(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)
Returns a form to reset the user password (deprecated)

retrieve_password(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)

retrieve_username(*next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)
Returns a form to retrieve the user username (only if there is a username field)

run_login_onaccept()

select_host(*host*, *host_names=None*)
checks that host is valid, i.e. in the list of glob host_names if the host is missing, then is it selects the first entry from host_names read more here: https://github.com/web2py/web2py/issues/1196

table_cas()

table_event()

table_group()

table_membership()

table_permission()

table_token()

table_user()

update_groups()

url(*f=None*, *args=None*, *vars=None*, *scheme=False*)

user_group(*user_id=None*)
Returns the group_id of the group uniquely associated to this user i.e. *role=user:[user_id]*

user_group_role(*user_id=None*)

user_id
user.id or None

verify_email(*next=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*)
Action used to verify the registration email

when_is_logged_in_bypass_next_in_url(*next*, *session*)
This function should be use when someone want to avoid asking for user credentials when loaded page contains "user/login?_next=NEXT_COMPONENT" in the URL is refresh but user is already authenticated.

wiki(*slug=None*, *env=None*, *render='markmin'*, *manage_permissions=False*, *force_prefix=''*, *restrict_search=False*, *resolve=True*, *extra=None*, *menu_groups=None*, *templates=None*, *migrate=True*, *controller=None*, *function=None*, *force_render=False*, *groups=None*)

wikimenu()
To be used in menu.py for app wide wiki menus

class gluon.tools.**Recaptcha**(*request=None*, *public_key=''*, *private_key=''*, *use_ssl=False*, *error=None*, *error_message='invalid'*, *label='Verify:'*, *options=''*, *comment=''*, *ajax=False*)

Bases: *gluon.html.DIV*

Examples

Use as:

```
form = FORM(Recaptcha(public_key='...', private_key='...'))
```

or:

```
form = SQLFORM(...)
form.append(Recaptcha(public_key='...', private_key='...'))
```

API_SERVER = 'http://www.google.com/recaptcha/api'

API_SSL_SERVER = 'https://www.google.com/recaptcha/api'

VERIFY_SERVER = 'http://www.google.com/recaptcha/api/verify'

xml()
generates the xml for this component.

class gluon.tools.**Recaptcha2**(*request=None*, *public_key=''*, *private_key=''*, *error_message='invalid'*, *label='Verify:'*, *options=None*, *comment=''*)

Bases: *gluon.html.DIV*

Experimental: Creates a DIV holding the newer Recaptcha from Google (v2)

Parameters

- **request** – the request. If not passed, uses current request
- **public_key** – the public key Google gave you
- **private_key** – the private key Google gave you
- **error_message** – the error message to show if verification fails
- **label** – the label to use
- **options** (*dict*) – takes these parameters
  - hl
  - theme
  - type

  - tabindex
  - callback
  - expired-callback

  see https://developers.google.com/recaptcha/docs/display for docs about those

- **comment** – the comment

Examples

Use as:

```
form = FORM(Recaptcha2(public_key='...', private_key='...'))
```

or:

```
form = SQLFORM(...)
form.append(Recaptcha2(public_key='...', private_key='...'))
```

to protect the login page instead, use:

```
from gluon.tools import Recaptcha2
auth.settings.captcha = Recaptcha2(request, public_key='...', private_key='...')
```

API_URI = 'https://www.google.com/recaptcha/api.js'

VERIFY_SERVER = 'https://www.google.com/recaptcha/api/siteverify'

xml()
generates the xml for this component.

class gluon.tools.**Crud**(*environment*, *db=None*, *controller='default'*)
Bases: object

static archive(*form*, *archive_table=None*, *current_record='current_record'*)

create(*table*, *next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *log=<function <lambda>>*, *message=<function <lambda>>*, *formname=<function <lambda>>*, ***attributes*)

delete(*table*, *record_id*, *next=<function <lambda>>*, *message=<function <lambda>>*)

get_format(*field*)

get_query(*field*, *op*, *value*, *refsearch=False*)

has_permission(*name*, *table*, *record=0*)

log_event(*message*, *vars*)

read(*table*, *record*)

rows(*table*, *query=None*, *fields=None*, *orderby=None*, *limitby=None*)

search(**tables*, ***args*)
Creates a search form and its results for a table .. rubric:: Examples

Use as:

```
form, results = crud.search(db.test,
   queries = ['equals', 'not equal', 'contains'],
   query_labels={'equals':'Equals',
                 'not equal':'Not equal'},
   fields = ['id','children'],
   field_labels = {
       'id':'ID','children':'Children'},
   zero='Please choose',
   query = (db.test.id > 0)&(db.test.id != 3) )
```

select(*table*, *query=None*, *fields=None*, *orderby=None*, *limitby=None*, *headers=None*, ***attr*)

tables()

update(*table*, *record*, *next=<function <lambda>>*, *onvalidation=<function <lambda>>*, *onaccept=<function <lambda>>*, *ondelete=<function <lambda>>*, *log=<function <lambda>>*, *message=<function <lambda>>*, *deletable=<function <lambda>>*, *formname=<function <lambda>>*, ***attributes*)

url(*f=None*, *args=None*, *vars=None*)
This should point to the controller that exposes download and crud

class gluon.tools.**Service**(*environment=None*)
Bases: object

exception JsonRpcException(*code*, *info*)
Bases: exceptions.Exception

amfrpc(*f*)

Example

Use as:

```
service = Service()
@service.amfrpc
def myfunction(a, b):
    return a + b
def call():
    return service()
```

Then call it with:

```
wget http://..../app/default/call/amfrpc/myfunction?a=hello&b=world
```

amfrpc3(*domain='default'*)

Example

Use as:

```
service = Service()
@service.amfrpc3('domain')
def myfunction(a, b):
    return a + b
def call():
    return service()
```

Then call it with:

wget http://. . . ./app/default/call/amfrpc3/myfunction?a=hello&b=world

csv(*f*)

Example

Use as:

```
service = Service()
@service.csv
def myfunction(a, b):
    return a + b
def call():
    return service()
```

Then call it with:

```
wget http://..../app/default/call/csv/myfunction?a=3&b=4
```

error()

json(*f*)

Example

Use as:

```
service = Service()
@service.json
def myfunction(a, b):
    return [{a: b}]
def call():
    return service()
```

Then call it with:;

wget http://. . . ./app/default/call/json/myfunction?a=hello&b=world

jsonrpc(*f*)

Example

Use as:

```
service = Service()
@service.jsonrpc
def myfunction(a, b):
    return a + b
def call():
    return service()
```

Then call it with:

wget http://. . . ./app/default/call/jsonrpc/myfunction?a=hello&b=world

jsonrpc2(*f*)

Example

Use as:

```
service = Service()
@service.jsonrpc2
def myfunction(a, b):
    return a + b
def call():
    return service()
```

Then call it with:

wget –post-data '{"jsonrpc": "2.0", "id": 1, "method": "myfunction", "params": {"a": 1, "b": 2}}' http://..../app/default/call/jsonrpc2

jsonrpc_errors = {-32700: ('Parse error. Invalid JSON was received by the server.', '

rss(*f*)

Example

Use as:

```
service = Service()
@service.rss
def myfunction():
    return dict(title=..., link=..., description=...,
        created_on=..., entries=[dict(title=..., link=...,
            description=..., created_on=...])
def call():
    return service()
```

Then call it with:

wget http://..../app/default/call/rss/myfunction

run(*f*)

Example

Use as:

```
service = Service()
@service.run
def myfunction(a, b):
    return a + b
def call():
    return service()
```

Then call it with:

```
wget http://..../app/default/call/run/myfunction?a=3&b=4
```

serve_amfrpc(*version=0*)

serve_csv(*args=None*)

serve_json(*args=None*)

serve_jsonrpc()

serve_jsonrpc2(*data=None*, *batch_element=False*)

serve_rss(*args=None*)

serve_run(*args=None*)

serve_soap(*version='1.1'*)

serve_xml(*args=None*)

serve_xmlrpc()

soap(*name=None*, *returns=None*, *args=None*, *doc=None*)

Example

Use as:

```
service = Service()
@service.soap('MyFunction',returns={'result':int},args={'a':int,'b':int,})
def myfunction(a, b):
    return a + b
def call():
    return service()
```

Then call it with:

```
from gluon.contrib.pysimplesoap.client import SoapClient
client = SoapClient(wsdl="http://..../app/default/call/soap?WSDL")
response = client.MyFunction(a=1,b=2)
return response['result']
```

It also exposes online generated documentation and xml example messages at *http://.../app/default/call/soap*

xml(*f*)

Example

Use as:

```
service = Service()
@service.xml
def myfunction(a, b):
    return a + b
def call():
    return service()
```

Then call it with:

```
wget http://..../app/default/call/xml/myfunction?a=3&b=4
```

xmlrpc(*f*)

Example

Use as:

```
service = Service()
@service.xmlrpc
def myfunction(a, b):
    return a + b
def call():
    return service()
```

The call it with:

wget http://. . . ./app/default/call/xmlrpc/myfunction?a=hello&b=world

class gluon.tools.**Wiki**(*auth*, *env=None*, *render='markmin'*, *manage_permissions=False*, *force_prefix=''*, *restrict_search=False*, *extra=None*, *menu_groups=None*, *templates=None*, *migrate=True*, *controller=None*, *function=None*, *groups=None*)

Bases: object

automenu()
adds the menu if not present

can_edit(*page=None*)

can_manage()

can_read(*page*)

can_search()

can_see_menu()

cloud()

static component(*text*)
In wiki docs allows *@{component:controller/function/args}* which renders as a *LOAD(. . . , ajax=True)*

create()

edit(*slug*, *from_template=0*)

editmedia(*slug*)

everybody = 'everybody'

first_paragraph(*page*)

fix_hostname(*body*)

get_renderer()

html_render(*page*)

markmin_base(*body*)

markmin_render(*page*)

media(*id*)

menu(*controller='default'*, *function='index'*)

not_authorized(*page=None*)

pages()

preview(*render*)

read(*slug*, *force_render=False*)

render_tags(*tags*)

rows_page = 25

search(*tags=None*, *query=None*, *cloud=True*, *preview=True*, *limitby=(0, 100)*, *orderby=None*)

class gluon.tools.**PluginManager**(*plugin=None*, ***defaults*)

Bases: object

Plugin Manager is similar to a storage object but it is a single level singleton. This means that multiple instances within the same thread share the same attributes. Its constructor is also special. The first argument is the name of the plugin you are defining. The named arguments are parameters needed by the plugin with default values. If the parameters were previous defined, the old values are used.

Example

in some general configuration file:

```
plugins = PluginManager()
plugins.me.param1=3
```

within the plugin model:

```
_ = PluginManager('me',param1=5,param2=6,param3=7)
```

where the plugin is used:

```
>>> print plugins.me.param1
3
>>> print plugins.me.param2
6
>>> plugins.me.param3 = 8
>>> print plugins.me.param3
8
```

Here are some tests:

```
>>> a=PluginManager()
>>> a.x=6
>>> b=PluginManager('check')
>>> print b.x
6
>>> b=PluginManager() # reset settings
>>> print b.x
<Storage {}>
>>> b.x=7
>>> print a.x
7
>>> a.y.z=8
>>> print b.y.z
8
```

(continues on next page)

(continued from previous page)

```
>>> test_thread_separation()
5
>>> plugins=PluginManager('me',db='mydb')
>>> print plugins.me.db
mydb
>>> print 'me' in plugins
True
>>> print plugins.me.installed
True
```

instances = {}

keys()

gluon.tools.**fetch**(*url*, *data=None*, *headers=None*, *cookie={}*, *user_agent='Mozilla/5.0'*)

gluon.tools.**geocode**(*address*)

gluon.tools.**reverse_geocode**(*lat*, *lng*, *lang=None*)

Try to get an approximate address for a given latitude, longitude.

gluon.tools.**prettydate**(*d*, *T=<function <lambda>>*, *utc=False*)

CHAPTER 35

utf8 Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)
Created by Vladyslav Kozlovskyy (Ukraine) <dbdevelop©gmail.com>
for Web2py project

35.1 Utilities and class for UTF8 strings managing

class gluon.utf8.**Utf8**
Bases: str

Class for utf8 string storing and manipulations

The base presupposition of this class usage is: "ALL strings in the application are either of utf-8 or unicode type, even when simple str type is used. UTF-8 is only a "packed" version of unicode, so Utf-8 and unicode strings are interchangeable."

CAUTION! This class is slower than str/unicode! Do NOT use it inside intensive loops. Simply decode string(s) to unicode before loop and encode it back to utf-8 string(s) after intensive calculation.

You can see the benefit of this class in doctests() below

capitalize() → string
Return a copy of the string S with only its first character capitalized.

center(*width*[, *fillchar*]) → string
Return S centered in a string of length width. Padding is done using the specified fill character (default is a space)

count(*sub*[, *start*[, *end*]]) → int
Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

decode(*[encoding[, errors]]*) → object
Decodes S using the codec registered for encoding. encoding defaults to the default encoding. errors may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a UnicodeDecodeError. Other possible values are 'ignore' and 'replace' as well as any other name registered with codecs.register_error that is able to handle UnicodeDecodeErrors.

encode(*[encoding[, errors]]*) → object
Encodes S using the codec registered for encoding. encoding defaults to the default encoding. errors may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that is able to handle UnicodeEncodeErrors.

endswith(*suffix[, start[, end]]*) → bool
Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(*[tabsize]*) → string
Return a copy of S where all tab characters are expanded using spaces. If tabsize is not given, a tab size of 8 characters is assumed.

find(*sub[, start[, end]]*) → int
Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → string
Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

index(*sub[, start[, end]]*) → int
Like S.find() but raise ValueError when the substring is not found.

isalnum() → bool
Return True if all characters in S are alphanumeric and there is at least one character in S, False otherwise.

isalpha() → bool
Return True if all characters in S are alphabetic and there is at least one character in S, False otherwise.

isdigit() → bool
Return True if all characters in S are digits and there is at least one character in S, False otherwise.

islower() → bool
Return True if all cased characters in S are lowercase and there is at least one cased character in S, False otherwise.

isspace() → bool
Return True if all characters in S are whitespace and there is at least one character in S, False otherwise.

istitle() → bool
Return True if S is a titlecased string and there is at least one character in S, i.e. uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

isupper() → bool
Return True if all cased characters in S are uppercase and there is at least one cased character in S, False otherwise.

join(*iterable*) → string
Return a string which is the concatenation of the strings in the iterable. The separator between elements is S.

ljust (*width*[, *fillchar*]) → string
Return S left-justified in a string of length width. Padding is done using the specified fill character (default is a space).

lower () → string
Return a copy of the string S converted to lowercase.

lstrip ([*chars*]) → string or unicode
Return a copy of the string S with leading whitespace removed. If chars is given and not None, remove characters in chars instead. If chars is unicode, S will be converted to unicode before stripping

partition (*sep) -> (head*, *sep*, *tail*)
Search for the separator sep in S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings.

replace (*old*, *new*[, *count*]) → string
Return a copy of string S with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.

rfind (*sub*[, *start*[, *end*]]) → int
Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex (*sub*[, *start*[, *end*]]) → int
Like S.rfind() but raise ValueError when the substring is not found.

rjust (*width*[, *fillchar*]) → string
Return S right-justified in a string of length width. Padding is done using the specified fill character (default is a space)

rpartition (*sep) -> (head*, *sep*, *tail*)
Search for the separator sep in S, starting at the end of S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return two empty strings and S.

rsplit ([*sep*[, *maxsplit*]]) → list of strings
Return a list of the words in the string S, using sep as the delimiter string, starting at the end of the string and working to the front. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator.

rstrip ([*chars*]) → string or unicode
Return a copy of the string S with trailing whitespace removed. If chars is given and not None, remove characters in chars instead. If chars is unicode, S will be converted to unicode before stripping

split ([*sep*[, *maxsplit*]]) → list of strings
Return a list of the words in the string S, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator and empty strings are removed from the result.

splitlines (*keepends=False*) → list of strings
Return a list of the lines in S, breaking at line boundaries. Line breaks are not included in the resulting list unless keepends is given and true.

startswith (*prefix*[, *start*[, *end*]]) → bool
Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip ([*chars*]) → string or unicode
Return a copy of the string S with leading and trailing whitespace removed. If chars is given and not None,

remove characters in chars instead. If chars is unicode, S will be converted to unicode before stripping

swapcase() → string
Return a copy of the string S with uppercase characters converted to lowercase and vice versa.

title() → string
Return a titlecased version of S, i.e. words start with uppercase characters, all remaining cased characters have lowercase.

translate(*table*[, *deletechars*]) → string
Return a copy of the string S, where all characters occurring in the optional argument deletechars are removed, and the remaining characters have been mapped through the given translation table, which must be a string of length 256 or None. If the table argument is None, no translation is applied and the operation simply removes the characters in deletechars.

upper() → string
Return a copy of the string S converted to uppercase.

zfill(*width*) → string
Pad a numeric string S with zeros on the left, to fill a field of the specified width. The string S is never truncated.

CHAPTER 36

utils Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

36.1 This file specifically includes utilities for security.

gluon.utils.**AES_new**(*key*, *IV=None*)
Returns an AES cipher object and random IV if None specified

gluon.utils.**compare**(*a*, *b*)
Compares two strings and not vulnerable to timing attacks

gluon.utils.**fast_urandom16**(*urandom=[]*, *locker=<_RLock owner=None count=0>*)
This is 4x faster than calling os.urandom(16) and prevents the "too many files open" issue with concurrent access to os.urandom()

gluon.utils.**get_callable_argspec**(*fn*)

gluon.utils.**get_digest**(*value*)
Returns a hashlib digest algorithm from a string

gluon.utils.**getipaddrinfo**(*host*)
Filter out non-IP and bad IP addresses from getaddrinfo

gluon.utils.**initialize_urandom**()
This function and the web2py_uuid follow from the following discussion: *http://groups.google.com/group/web2py-developers/browse_thread/thread/7fd5789a7da3f09*

At startup web2py compute a unique ID that identifies the machine by adding uuid.getnode() + int(time.time() * 1e3)

This is a 48-bit number. It converts the number into 16 8-bit tokens. It uses this value to initialize the entropy source ('/dev/urandom') and to seed random.

If os.random() is not supported, it falls back to using random and issues a warning.

gluon.utils.**is_loopback_ip_address**(*ip=None*, *addrinfo=None*)
Determines whether the address appears to be a loopback address. This assumes that the IP is valid.

gluon.utils.**is_valid_ip_address**(*address*)

Examples

Better than a thousand words:

```
>>> is_valid_ip_address('127.0')
False
>>> is_valid_ip_address('127.0.0.1')
True
>>> is_valid_ip_address('2001:660::1')
True
```

gluon.utils.**md5_hash**(*text*)
Generates a md5 hash with the given text

gluon.utils.**pad**(*s*, *n=32*, *padchar=' '*)

gluon.utils.**pbkdf2_hex**(*data*, *salt*, *iterations=1000*, *keylen=24*, *hashfunc=None*)

gluon.utils.**secure_dumps**(*data*, *encryption_key*, *hash_key=None*, *compression_level=None*)

gluon.utils.**secure_loads**(*data*, *encryption_key*, *hash_key=None*, *compression_level=None*)

gluon.utils.**simple_hash**(*text*, *key=''*, *salt=''*, *digest_alg='md5'*)
Generates hash with the given text using the specified digest hashing algorithm

gluon.utils.**web2py_uuid**(*ctokens=(1280781058817960799 8L, 787762416516532974)*)
This function follows from the following discussion: *http://groups.google.com/group/web2py-developers/browse_thread/thread/7fd5789a7da3f09*

It works like uuid.uuid4 except that tries to use os.urandom() if possible and it XORs the output with the tokens uniquely associated with this machine.

CHAPTER 37

validators Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)
Thanks to ga2arch for help with IS_IN_DB and IS_NOT_IN_DB on GAE

37.1 Validators

class gluon.validators.**ANY_OF**(*subs*)
Bases: gluon.validators.Validator

Tests if any of the validators in a list returns successfully:

```
>>> ANY_OF([IS_EMAIL(),IS_ALPHANUMERIC()])('a@b.co')
('a@b.co', None)
>>> ANY_OF([IS_EMAIL(),IS_ALPHANUMERIC()])('abco')
('abco', None)
>>> ANY_OF([IS_EMAIL(),IS_ALPHANUMERIC()])('@ab.co')
('@ab.co', 'enter only letters, numbers, and underscore')
>>> ANY_OF([IS_ALPHANUMERIC(),IS_EMAIL()])('@ab.co')
('@ab.co', 'enter a valid email address')
```

formatter(*value*)
For some validators returns a formatted version (matching the validator) of value. Otherwise just returns the value.

class gluon.validators.**CLEANUP**(*regex=None*)
Bases: gluon.validators.Validator

Examples

Use as:

```
INPUT(_type='text', _name='name', requires=CLEANUP())
```

removes special characters on validation

REGEX_CLEANUP = <_sre.SRE_Pattern object>

class gluon.validators.**CRYPT**(*key=None*, *digest_alg='pbkdf2(1000, 20, sha512)'*, *min_length=0*, *error_message='Too short'*, *salt=True*, *max_length=1024*)

Bases: object

Examples

Use as:

```
INPUT(_type='text', _name='name', requires=CRYPT())
```

encodes the value on validation with a digest.

If no arguments are provided CRYPT uses the MD5 algorithm. If the key argument is provided the HMAC+MD5 algorithm is used. If the digest_alg is specified this is used to replace the MD5 with, for example, SHA512. The digest_alg can be the name of a hashlib algorithm as a string or the algorithm itself.

min_length is the minimal password length (default 4) - IS_STRONG for serious security error_message is the message if password is too short

Notice that an empty password is accepted but invalid. It will not allow login back. Stores junk as hashed password.

Specify an algorithm or by default we will use sha512.

Typical available algorithms: md5, sha1, sha224, sha256, sha384, sha512

If salt, it hashes a password with a salt. If salt is True, this method will automatically generate one. Either case it returns an encrypted password string in the following format:

<algorithm>$<salt>$<hash>

Important: hashed password is returned as a LazyCrypt object and computed only if needed. The LasyCrypt object also knows how to compare itself with an existing salted password

Supports standard algorithms

```
>>> for alg in ('md5','sha1','sha256','sha384','sha512'):
...     print str(CRYPT(digest_alg=alg,salt=True)('test')[0])
md5$...$...
sha1$...$...
sha256$...$...
sha384$...$...
sha512$...$...
```

The syntax is always alg$salt$hash

Supports for pbkdf2

```
>>> alg = 'pbkdf2(1000,20,sha512)'
>>> print str(CRYPT(digest_alg=alg,salt=True)('test')[0])
pbkdf2(1000,20,sha512)$...$...
```

An optional hmac_key can be specified and it is used as salt prefix

```
>>> a = str(CRYPT(digest_alg='md5',key='mykey',salt=True)('test')[0])
>>> print a
md5$...$...
```

Even if the algorithm changes the hash can still be validated

```
>>> CRYPT(digest_alg='sha1',key='mykey',salt=True)('test')[0] == a
True
```

If no salt is specified CRYPT can guess the algorithms from length:

```
>>> a = str(CRYPT(digest_alg='sha1',salt=False)('test')[0])
>>> a
'sha1$$a94a8fe5ccb19ba61c4c0873d391e987982fbbd3'
>>> CRYPT(digest_alg='sha1',salt=False)('test')[0] == a
True
>>> CRYPT(digest_alg='sha1',salt=False)('test')[0] == a[6:]
True
>>> CRYPT(digest_alg='md5',salt=False)('test')[0] == a
True
>>> CRYPT(digest_alg='md5',salt=False)('test')[0] == a[6:]
True
```

class gluon.validators.**IS_ALPHANUMERIC**(*error_message='Enter only letters, numbers, and underscore'*)

Bases: *gluon.validators.IS_MATCH*

Example

Used as:

```
INPUT(_type='text', _name='name', requires=IS_ALPHANUMERIC())

>>> IS_ALPHANUMERIC()('1')
('1', None)
>>> IS_ALPHANUMERIC()('')
('', None)
>>> IS_ALPHANUMERIC()('A_a')
('A_a', None)
>>> IS_ALPHANUMERIC()('!')
('!', 'enter only letters, numbers, and underscore')
```

class gluon.validators.**IS_DATE_IN_RANGE**(*minimum=None, maximum=None, format='%Y-%m-%d', error_message=None*)

Bases: *gluon.validators.IS_DATE*

Examples

Use as:

```
>>> v = IS_DATE_IN_RANGE(minimum=datetime.date(2008,1,1),
↪             maximum=datetime.date(2009,12,31),
↪     format="%m/%d/%Y",error_message="Oops")
```

(continues on next page)

(continued from previous page)

```
>>> v('03/03/2008')
(datetime.date(2008, 3, 3), None)

>>> v('03/03/2010')
('03/03/2010', 'oops')

>>> v(datetime.date(2008,3,3))
(datetime.date(2008, 3, 3), None)

>>> v(datetime.date(2010,3,3))
(datetime.date(2010, 3, 3), 'oops')
```

class gluon.validators.**IS_DATE**(*format='%Y-%m-%d'*, *error_message='Enter date as %(format)s'*)

Bases: gluon.validators.Validator

Examples

Use as:

```
INPUT(_type='text', _name='name', requires=IS_DATE())
```

date has to be in the ISO8960 format YYYY-MM-DD

formatter(*value*)

For some validators returns a formatted version (matching the validator) of value. Otherwise just returns the value.

class gluon.validators.**IS_DATETIME_IN_RANGE**(*minimum=None*, *maximum=None*, *format='%Y-%m-%d %H:%M:%S'*, *error_message=None*, *timezone=None*)

Bases: *gluon.validators.IS_DATETIME*

Examples

Use as::

```
>>> v = IS_DATETIME_IN_RANGE(                    minimum=datetime.
↪datetime(2008,1,1,12,20),                    maximum=datetime.
↪datetime(2009,12,31,12,20),                    format="%m/%d/%Y %H:%M",
↪error_message="Oops")
>>> v('03/03/2008 12:40')
(datetime.datetime(2008, 3, 3, 12, 40), None)
```

```
>>> v('03/03/2010 10:34')
('03/03/2010 10:34', 'oops')
```

```
>>> v(datetime.datetime(2008,3,3,0,0))
(datetime.datetime(2008, 3, 3, 0, 0), None)
```

```
>>> v(datetime.datetime(2010,3,3,0,0))
(datetime.datetime(2010, 3, 3, 0, 0), 'oops')
```

class gluon.validators.**IS_DATETIME**(*format='%Y-%m-%d %H:%M:%S'*, *error_message='Enter date and time as %(format)s'*, *timezone=None*)

Bases: gluon.validators.Validator

Examples

Use as:

```
INPUT(_type='text', _name='name', requires=IS_DATETIME())
```

datetime has to be in the ISO8960 format YYYY-MM-DD hh:mm:ss timezome must be None or a pytz.timezone("America/Chicago") object

formatter(*value*)

For some validators returns a formatted version (matching the validator) of value. Otherwise just returns the value.

isodatetime = '%Y-%m-%d %H:%M:%S'

static nice(*format*)

class gluon.validators.**IS_DECIMAL_IN_RANGE**(*minimum=None*, *maximum=None*, *error_message=None*, *dot='.'*)

Bases: gluon.validators.Validator

Determines that the argument is (or can be represented as) a Python Decimal, and that it falls within the specified inclusive range. The comparison is made with Python Decimal arithmetic.

The minimum and maximum limits can be None, meaning no lower or upper limit, respectively.

Example

Used as:

```
INPUT(_type='text', _name='name', requires=IS_DECIMAL_IN_RANGE(0, 10))

>>> IS_DECIMAL_IN_RANGE(1,5)('4')
(Decimal('4'), None)
>>> IS_DECIMAL_IN_RANGE(1,5)(4)
(Decimal('4'), None)
>>> IS_DECIMAL_IN_RANGE(1,5)(1)
(Decimal('1'), None)
>>> IS_DECIMAL_IN_RANGE(1,5)(5.25)
(5.25, 'enter a number between 1 and 5')
>>> IS_DECIMAL_IN_RANGE(5.25,6)(5.25)
(Decimal('5.25'), None)
>>> IS_DECIMAL_IN_RANGE(5.25,6)('5.25')
(Decimal('5.25'), None)
>>> IS_DECIMAL_IN_RANGE(1,5)(6.0)
(6.0, 'enter a number between 1 and 5')
>>> IS_DECIMAL_IN_RANGE(1,5)(3.5)
(Decimal('3.5'), None)
>>> IS_DECIMAL_IN_RANGE(1.5,5.5)(3.5)
(Decimal('3.5'), None)
>>> IS_DECIMAL_IN_RANGE(1.5,5.5)(6.5)
(6.5, 'enter a number between 1.5 and 5.5')
```

(continues on next page)

(continued from previous page)

```
>>> IS_DECIMAL_IN_RANGE(1.5,None)(6.5)
(Decimal('6.5'), None)
>>> IS_DECIMAL_IN_RANGE(1.5,None)(0.5)
(0.5, 'enter a number greater than or equal to 1.5')
>>> IS_DECIMAL_IN_RANGE(None,5.5)(4.5)
(Decimal('4.5'), None)
>>> IS_DECIMAL_IN_RANGE(None,5.5)(6.5)
(6.5, 'enter a number less than or equal to 5.5')
>>> IS_DECIMAL_IN_RANGE()(6.5)
(Decimal('6.5'), None)
>>> IS_DECIMAL_IN_RANGE(0,99)(123.123)
(123.123, 'enter a number between 0 and 99')
>>> IS_DECIMAL_IN_RANGE(0,99)('123.123')
('123.123', 'enter a number between 0 and 99')
>>> IS_DECIMAL_IN_RANGE(0,99)('12.34')
(Decimal('12.34'), None)
>>> IS_DECIMAL_IN_RANGE()('abc')
('abc', 'enter a number')
```

formatter(*value*)

For some validators returns a formatted version (matching the validator) of value. Otherwise just returns the value.

class gluon.validators.**IS_EMAIL**(*banned=None*, *forced=None*, *error_message='Enter a valid email address'*)

Bases: gluon.validators.Validator

Checks if field's value is a valid email address. Can be set to disallow or force addresses from certain domain(s).

Email regex adapted from http://haacked.com/archive/2007/08/21/i-knew-how-to-validate-an-email-address-until-i.aspx, generally following the RFCs, except that we disallow quoted strings and permit underscores and leading numerics in subdomain labels

Parameters

- **banned** – regex text for disallowed address domains
- **forced** – regex text for required address domains

Both arguments can also be custom objects with a match(value) method.

Example

Check for valid email address:

```
INPUT(_type='text', _name='name',
    requires=IS_EMAIL())
```

Check for valid email address that can't be from a .com domain:

```
INPUT(_type='text', _name='name',
    requires=IS_EMAIL(banned='^.*\.com(|\..*)$'))
```

Check for valid email address that must be from a .edu domain:

```
INPUT(_type='text', _name='name',
    requires=IS_EMAIL(forced='^.*\.edu(|\..*)$'))
```

(continues on next page)

(continued from previous page)

```
>>> IS_EMAIL()('a@b.com')
('a@b.com', None)
>>> IS_EMAIL()('abc@def.com')
('abc@def.com', None)
>>> IS_EMAIL()('abc@3def.com')
('abc@3def.com', None)
>>> IS_EMAIL()('abc@def.us')
('abc@def.us', None)
>>> IS_EMAIL()('abc@d_-f.us')
('abc@d_-f.us', None)
>>> IS_EMAIL()('@def.com')           # missing name
('@def.com', 'enter a valid email address')
>>> IS_EMAIL()('"abc@def".com')      # quoted name
('"abc@def".com', 'enter a valid email address')
>>> IS_EMAIL()('abc+def.com')        # no @
('abc+def.com', 'enter a valid email address')
>>> IS_EMAIL()('abc@def.x')          # one-char TLD
('abc@def.x', 'enter a valid email address')
>>> IS_EMAIL()('abc@def.12')         # numeric TLD
('abc@def.12', 'enter a valid email address')
>>> IS_EMAIL()('abc@def..com')       # double-dot in domain
('abc@def..com', 'enter a valid email address')
>>> IS_EMAIL()('abc@.def.com')       # dot starts domain
('abc@.def.com', 'enter a valid email address')
>>> IS_EMAIL()('abc@def.c_m')        # underscore in TLD
('abc@def.c_m', 'enter a valid email address')
>>> IS_EMAIL()('NotAnEmail')         # missing @
('NotAnEmail', 'enter a valid email address')
>>> IS_EMAIL()('abc@NotAnEmail')     # missing TLD
('abc@NotAnEmail', 'enter a valid email address')
>>> IS_EMAIL()('customer/department@example.com')
('customer/department@example.com', None)
>>> IS_EMAIL()('$A12345@example.com')
('$A12345@example.com', None)
>>> IS_EMAIL()('!def!xyz%abc@example.com')
('!def!xyz%abc@example.com', None)
>>> IS_EMAIL()('_Yosemite.Sam@example.com')
('_Yosemite.Sam@example.com', None)
>>> IS_EMAIL()('~@example.com')
('~@example.com', None)
>>> IS_EMAIL()('.wooly@example.com')       # dot starts name
('.wooly@example.com', 'enter a valid email address')
>>> IS_EMAIL()('wo..oly@example.com')      # adjacent dots in name
('wo..oly@example.com', 'enter a valid email address')
>>> IS_EMAIL()('pootietang.@example.com')  # dot ends name
('pootietang.@example.com', 'enter a valid email address')
>>> IS_EMAIL()('.@example.com')            # name is bare dot
('.@example.com', 'enter a valid email address')
>>> IS_EMAIL()('Ima.Fool@example.com')
('Ima.Fool@example.com', None)
>>> IS_EMAIL()('Ima Fool@example.com')     # space in name
('Ima Fool@example.com', 'enter a valid email address')
>>> IS_EMAIL()('localguy@localhost')       # localhost as domain
('localguy@localhost', None)
```

regex = <_sre.SRE_Pattern object at 0x2e0e160>

regex_proposed_but_failed = <_sre.SRE_Pattern object at 0x2e100d0>

class gluon.validators.**IS_LIST_OF_EMAILS**(*error_message='Invalid emails: %s'*)
Bases: object

Example

Used as:

```
Field('emails','list:string',
      widget=SQLFORM.widgets.text.widget,
      requires=IS_LIST_OF_EMAILS(),
      represent=lambda v,r:                          SPAN(*[A(x,_href='mailto:'+x)␣
↪for x in (v or [])])
      )
```

formatter(*value*, *row=None*)

split_emails = <_sre.SRE_Pattern object>

class gluon.validators.**IS_EMPTY_OR**(*other*, *null=None*, *empty_regex=None*)
Bases: gluon.validators.Validator

Dummy class for testing IS_EMPTY_OR:

```
>>> IS_EMPTY_OR(IS_EMAIL())('abc@def.com')
('abc@def.com', None)
>>> IS_EMPTY_OR(IS_EMAIL())('   ')
(None, None)
>>> IS_EMPTY_OR(IS_EMAIL(), null='abc')('   ')
('abc', None)
>>> IS_EMPTY_OR(IS_EMAIL(), null='abc', empty_regex='def')('def')
('abc', None)
>>> IS_EMPTY_OR(IS_EMAIL())('abc')
('abc', 'enter a valid email address')
>>> IS_EMPTY_OR(IS_EMAIL())(' abc ')
('abc', 'enter a valid email address')
```

formatter(*value*)
For some validators returns a formatted version (matching the validator) of value. Otherwise just returns the value.

set_self_id(*id*)

class gluon.validators.**IS_EXPR**(*expression*, *error_message='Invalid expression'*, *environment=None*)
Bases: gluon.validators.Validator

Example

Used as:

```
INPUT(_type='text', _name='name',
    requires=IS_EXPR('5 < int(value) < 10'))
```

The argument of IS_EXPR must be python condition:

```
>>> IS_EXPR('int(value) < 2')('1')
('1', None)

>>> IS_EXPR('int(value) < 2')('2')
('2', 'invalid expression')
```

class gluon.validators.**IS_FLOAT_IN_RANGE**(*minimum=None*, *maximum=None*, *error_message=None*, *dot='.'*)

Bases: gluon.validators.Validator

Determines that the argument is (or can be represented as) a float, and that it falls within the specified inclusive range. The comparison is made with native arithmetic.

The minimum and maximum limits can be None, meaning no lower or upper limit, respectively.

Example

Used as:

```
INPUT(_type='text', _name='name', requires=IS_FLOAT_IN_RANGE(0, 10))

>>> IS_FLOAT_IN_RANGE(1,5)('4')
(4.0, None)
>>> IS_FLOAT_IN_RANGE(1,5)(4)
(4.0, None)
>>> IS_FLOAT_IN_RANGE(1,5)(1)
(1.0, None)
>>> IS_FLOAT_IN_RANGE(1,5)(5.25)
(5.25, 'enter a number between 1 and 5')
>>> IS_FLOAT_IN_RANGE(1,5)(6.0)
(6.0, 'enter a number between 1 and 5')
>>> IS_FLOAT_IN_RANGE(1,5)(3.5)
(3.5, None)
>>> IS_FLOAT_IN_RANGE(1,None)(3.5)
(3.5, None)
>>> IS_FLOAT_IN_RANGE(None,5)(3.5)
(3.5, None)
>>> IS_FLOAT_IN_RANGE(1,None)(0.5)
(0.5, 'enter a number greater than or equal to 1')
>>> IS_FLOAT_IN_RANGE(None,5)(6.5)
(6.5, 'enter a number less than or equal to 5')
>>> IS_FLOAT_IN_RANGE()(6.5)
(6.5, None)
>>> IS_FLOAT_IN_RANGE()('abc')
('abc', 'enter a number')
```

formatter(*value*)

For some validators returns a formatted version (matching the validator) of value. Otherwise just returns the value.

class gluon.validators.**IS_IMAGE**(*extensions=('bmp', 'gif', 'jpeg', 'png')*, *maxsize=(10000, 10000)*, *minsize=(0, 0)*, *error_message='Invalid image'*)

Bases: gluon.validators.Validator

Checks if file uploaded through file input was saved in one of selected image formats and has dimensions (width and height) within given boundaries.

Does *not* check for maximum file size (use IS_LENGTH for that). Returns validation failure if no data was uploaded.

Supported file formats: BMP, GIF, JPEG, PNG.

Code parts taken from http://mail.python.org/pipermail/python-list/2007-June/617126.html

Parameters

- **extensions** – iterable containing allowed *lowercase* image file extensions
- **extension of uploaded file counts as 'jpeg')** (*('jpg')* –
- **maxsize** – iterable containing maximum width and height of the image
- **minsize** – iterable containing minimum width and height of the image

Use (-1, -1) as minsize to pass image size check.

Examples

Check if uploaded file is in any of supported image formats:

INPUT(_type='file', _name='name', requires=IS_IMAGE())

Check if uploaded file is either JPEG or PNG:

INPUT(_type='file', _name='name', requires=IS_IMAGE(extensions=('jpeg', 'png')))

Check if uploaded file is PNG with maximum size of 200x200 pixels:

INPUT(_type='file', _name='name', requires=IS_IMAGE(extensions=('png'), maxsize=(200, 200)))

class gluon.validators.**IS_IN_DB**(*dbset*, *field*, *label=None*, *error_message='Value not in database'*, *orderby=None*, *groupby=None*, *distinct=None*, *cache=None*, *multiple=False*, *zero=''*, *sort=False*, *_and=None*, *left=None*, *delimiter=None*, *auto_add=False*)

Bases: gluon.validators.Validator

Example

Used as:

```
INPUT(_type='text', _name='name',
      requires=IS_IN_DB(db, db.mytable.myfield, zero=''))
```

used for reference fields, rendered as a dropbox

build_set()

maybe_add(*table*, *fieldname*, *value*)

options(*zero=True*)

set_self_id(*id*)

class gluon.validators.**IS_IN_SET**(*theset*, *labels=None*, *error_message='Value not allowed'*, *multiple=False*, *zero=''*, *sort=False*)

Bases: gluon.validators.Validator

Example

Used as:

```
INPUT(_type='text', _name='name',
      requires=IS_IN_SET(['max', 'john'],zero=''))
```

The argument of IS_IN_SET must be a list or set:

```
>>> IS_IN_SET(['max', 'john'])('max')
('max', None)
>>> IS_IN_SET(['max', 'john'])('massimo')
('massimo', 'value not allowed')
>>> IS_IN_SET(['max', 'john'], multiple=True)(('max', 'john'))
(('max', 'john'), None)
>>> IS_IN_SET(['max', 'john'], multiple=True)(('bill', 'john'))
(('bill', 'john'), 'value not allowed')
>>> IS_IN_SET(('id1','id2'), ['first label','second label'])('id1') # Traditional␣
↪way
('id1', None)
>>> IS_IN_SET({'id1':'first label', 'id2':'second label'})('id1')
('id1', None)
>>> import itertools
>>> IS_IN_SET(itertools.chain(['1','3','5'],['2','4','6']))('1')
('1', None)
>>> IS_IN_SET([('id1','first label'), ('id2','second label')])('id1') # Redundant␣
↪way
('id1', None)
```

options(*zero=True*)

class gluon.validators.**IS_INT_IN_RANGE**(*minimum=None*, *maximum=None*, *error_message=None*)

Bases: gluon.validators.Validator

Determines that the argument is (or can be represented as) an int, and that it falls within the specified range. The range is interpreted in the Pythonic way, so the test is: min <= value < max.

The minimum and maximum limits can be None, meaning no lower or upper limit, respectively.

Example

Used as:

```
INPUT(_type='text', _name='name', requires=IS_INT_IN_RANGE(0, 10))

>>> IS_INT_IN_RANGE(1,5)('4')
(4, None)
>>> IS_INT_IN_RANGE(1,5)(4)
(4, None)
>>> IS_INT_IN_RANGE(1,5)(1)
(1, None)
>>> IS_INT_IN_RANGE(1,5)(5)
(5, 'enter an integer between 1 and 4')
>>> IS_INT_IN_RANGE(1,5)(5)
(5, 'enter an integer between 1 and 4')
>>> IS_INT_IN_RANGE(1,5)(3.5)
```

(continues on next page)

(continued from previous page)

```
(3.5, 'enter an integer between 1 and 4')
>>> IS_INT_IN_RANGE(None,5)('4')
(4, None)
>>> IS_INT_IN_RANGE(None,5)('6')
('6', 'enter an integer less than or equal to 4')
>>> IS_INT_IN_RANGE(1,None)('4')
(4, None)
>>> IS_INT_IN_RANGE(1,None)('0')
('0', 'enter an integer greater than or equal to 1')
>>> IS_INT_IN_RANGE()(6)
(6, None)
>>> IS_INT_IN_RANGE()('abc')
('abc', 'enter an integer')
```

class gluon.validators.**IS_IPV4**(*minip='0.0.0.0'*, *maxip='255.255.255.255'*, *invert=False*, *is_localhost=None*, *is_private=None*, *is_automatic=None*, *error_message='Enter valid IPv4 address'*)

Bases: gluon.validators.Validator

Checks if field's value is an IP version 4 address in decimal form. Can be set to force addresses from certain range.

IPv4 regex taken from: http://regexlib.com/REDetails.aspx?regexp_id=1411

Parameters

- **minip** – lowest allowed address; accepts:
  - str, eg. 192.168.0.1
  - list or tuple of octets, eg. [192, 168, 0, 1]
- **maxip** – highest allowed address; same as above
- **invert** – True to allow addresses only from outside of given range; note that range boundaries are not matched this way
- **is_localhost** – localhost address treatment:
  - None (default): indifferent
  - True (enforce): query address must match localhost address (127.0.0.1)
  - False (forbid): query address must not match localhost address
- **is_private** – same as above, except that query address is checked against two address ranges: 172.16.0.0 - 172.31.255.255 and 192.168.0.0 - 192.168.255.255
- **is_automatic** – same as above, except that query address is checked against one address range: 169.254.0.0 - 169.254.255.255

Minip and maxip may also be lists or tuples of addresses in all above forms (str, int, list / tuple), allowing setup of multiple address ranges:

```
minip = (minip1, minip2, ... minipN)
           |       |           |
           |       |           |
maxip = (maxip1, maxip2, ... maxipN)
```

Longer iterable will be truncated to match length of shorter one.

Examples

Check for valid IPv4 address:

INPUT(_type='text', _name='name', requires=IS_IPV4())

Check for valid IPv4 address belonging to specific range:

INPUT(_type='text', _name='name', requires=IS_IPV4(minip='100.200.0.0', maxip='100.200.255.255'))

Check for valid IPv4 address belonging to either 100.110.0.0 - 100.110.255.255 or 200.50.0.0 - 200.50.0.255 address range:

INPUT(_type='text', _name='name',

requires=IS_IPV4(minip=('100.110.0.0', '200.50.0.0'), maxip=('100.110.255.255', '200.50.0.255')))

Check for valid IPv4 address belonging to private address space:

INPUT(_type='text', _name='name', requires=IS_IPV4(is_private=True))

Check for valid IPv4 address that is not a localhost address:

INPUT(_type='text', _name='name', requires=IS_IPV4(is_localhost=False))

```
>>> IS_IPV4()('1.2.3.4')
('1.2.3.4', None)
>>> IS_IPV4()('255.255.255.255')
('255.255.255.255', None)
>>> IS_IPV4()('1.2.3.4 ')
('1.2.3.4 ', 'enter valid IPv4 address')
>>> IS_IPV4()('1.2.3.4.5')
('1.2.3.4.5', 'enter valid IPv4 address')
>>> IS_IPV4()('123.123')
('123.123', 'enter valid IPv4 address')
>>> IS_IPV4()('1111.2.3.4')
('1111.2.3.4', 'enter valid IPv4 address')
>>> IS_IPV4()('0111.2.3.4')
('0111.2.3.4', 'enter valid IPv4 address')
>>> IS_IPV4()('256.2.3.4')
('256.2.3.4', 'enter valid IPv4 address')
>>> IS_IPV4()('300.2.3.4')
('300.2.3.4', 'enter valid IPv4 address')
>>> IS_IPV4(minip='1.2.3.4', maxip='1.2.3.4')('1.2.3.4')
('1.2.3.4', None)
>>> IS_IPV4(minip='1.2.3.5', maxip='1.2.3.9', error_message='Bad ip')('1.
↪2.3.4')
('1.2.3.4', 'bad ip')
>>> IS_IPV4(maxip='1.2.3.4', invert=True)('127.0.0.1')
('127.0.0.1', None)
>>> IS_IPV4(maxip='1.2.3.4', invert=True)('1.2.3.4')
('1.2.3.4', 'enter valid IPv4 address')
>>> IS_IPV4(is_localhost=True)('127.0.0.1')
('127.0.0.1', None)
>>> IS_IPV4(is_localhost=True)('1.2.3.4')
('1.2.3.4', 'enter valid IPv4 address')
>>> IS_IPV4(is_localhost=False)('127.0.0.1')
('127.0.0.1', 'enter valid IPv4 address')
```

(continues on next page)

(continued from previous page)

```
>>> IS_IPV4(maxip='100.0.0.0', is_localhost=True)('127.0.0.1')
('127.0.0.1', 'enter valid IPv4 address')
```

automatic = (2851995648L, 2852061183L)

localhost = 2130706433

numbers = (16777216, 65536, 256, 1)

private = ((2886729728L, 2886795263L), (3232235520L, 3232301055L))

regex = <_sre.SRE_Pattern object at 0x2e26320>

class gluon.validators.**IS_IPV6**(*is_private=None*, *is_link_local=None*, *is_reserved=None*, *is_multicast=None*, *is_routeable=None*, *is_6to4=None*, *is_teredo=None*, *subnets=None*, *error_message='Enter valid IPv6 address'*)

Bases: gluon.validators.Validator

Checks if field's value is an IP version 6 address. First attempts to use the ipaddress library and falls back to contrib/ipaddr.py from Google (https://code.google.com/p/ipaddr-py/)

Parameters

- **is_private** – None (default): indifferent True (enforce): address must be in fc00::/7 range False (forbid): address must NOT be in fc00::/7 range
- **is_link_local** – Same as above but uses fe80::/10 range
- **is_reserved** – Same as above but uses IETF reserved range
- **is_mulicast** – Same as above but uses ff00::/8 range
- **is_routeable** – Similar to above but enforces not private, link_local, reserved or multicast
- **is_6to4** – Same as above but uses 2002::/16 range
- **is_teredo** – Same as above but uses 2001::/32 range
- **subnets** – value must be a member of at least one from list of subnets

Examples

Check for valid IPv6 address:

INPUT(_type='text', _name='name', requires=IS_IPV6())

Check for valid IPv6 address is a link_local address:

INPUT(_type='text', _name='name', requires=IS_IPV6(is_link_local=True))

Check for valid IPv6 address that is Internet routeable:

INPUT(_type='text', _name='name', requires=IS_IPV6(is_routeable=True))

Check for valid IPv6 address in specified subnet:

INPUT(_type='text', _name='name', requires=IS_IPV6(subnets=['2001::/32'])

```
>>> IS_IPV6()('fe80::126c:8ffa:fe22:b3af')
('fe80::126c:8ffa:fe22:b3af', None)
>>> IS_IPV6()('192.168.1.1')
('192.168.1.1', 'enter valid IPv6 address')
>>> IS_IPV6(error_message='Bad ip')('192.168.1.1')
('192.168.1.1', 'bad ip')
>>> IS_IPV6(is_link_local=True)('fe80::126c:8ffa:fe22:b3af')
('fe80::126c:8ffa:fe22:b3af', None)
>>> IS_IPV6(is_link_local=False)('fe80::126c:8ffa:fe22:b3af')
('fe80::126c:8ffa:fe22:b3af', 'enter valid IPv6 address')
>>> IS_IPV6(is_link_local=True)('2001::126c:8ffa:fe22:b3af')
('2001::126c:8ffa:fe22:b3af', 'enter valid IPv6 address')
>>> IS_IPV6(is_multicast=True)('2001::126c:8ffa:fe22:b3af')
('2001::126c:8ffa:fe22:b3af', 'enter valid IPv6 address')
>>> IS_IPV6(is_multicast=True)('ff00::126c:8ffa:fe22:b3af')
('ff00::126c:8ffa:fe22:b3af', None)
>>> IS_IPV6(is_routeable=True)('2001::126c:8ffa:fe22:b3af')
('2001::126c:8ffa:fe22:b3af', None)
>>> IS_IPV6(is_routeable=True)('ff00::126c:8ffa:fe22:b3af')
('ff00::126c:8ffa:fe22:b3af', 'enter valid IPv6 address')
>>> IS_IPV6(subnets='2001::/32')('2001::8ffa:fe22:b3af')
('2001::8ffa:fe22:b3af', None)
>>> IS_IPV6(subnets='fb00::/8')('2001::8ffa:fe22:b3af')
('2001::8ffa:fe22:b3af', 'enter valid IPv6 address')
>>> IS_IPV6(subnets=['fc00::/8','2001::/32'])('2001::8ffa:fe22:b3af')
('2001::8ffa:fe22:b3af', None)
>>> IS_IPV6(subnets='invalidsubnet')('2001::8ffa:fe22:b3af')
('2001::8ffa:fe22:b3af', 'invalid subnet provided')
```

class gluon.validators.**IS_IPADDRESS**(*minip='0.0.0.0'*, *maxip='255.255.255.255'*, *invert=False*, *is_localhost=None*, *is_private=None*, *is_automatic=None*, *is_ipv4=None*, *is_link_local=None*, *is_reserved=None*, *is_multicast=None*, *is_routeable=None*, *is_6to4=None*, *is_teredo=None*, *subnets=None*, *is_ipv6=None*, *error_message='Enter valid IP address'*)

Bases: gluon.validators.Validator

Checks if field's value is an IP Address (v4 or v6). Can be set to force addresses from within a specific range. Checks are done with the correct IS_IPV4 and IS_IPV6 validators.

Uses ipaddress library if found, falls back to PEP-3144 ipaddr.py from Google (in contrib).

Parameters

- **minip** – lowest allowed address; accepts: str, eg. 192.168.0.1 list or tuple of octets, eg. [192, 168, 0, 1]
- **maxip** – highest allowed address; same as above
- **invert** – True to allow addresses only from outside of given range; note that range boundaries are not matched this way

IPv4 specific arguments:

- is_localhost: localhost address treatment:
  - None (default): indifferent
  - True (enforce): query address must match localhost address (127.0.0.1)

  - False (forbid): query address must not match localhost address
- is_private: same as above, except that query address is checked against two address ranges: 172.16.0.0 - 172.31.255.255 and 192.168.0.0 - 192.168.255.255
- is_automatic: same as above, except that query address is checked against one address range: 169.254.0.0 - 169.254.255.255
- is_ipv4: either:
  - None (default): indifferent
  - True (enforce): must be an IPv4 address
  - False (forbid): must NOT be an IPv4 address

IPv6 specific arguments:

- is_link_local: Same as above but uses fe80::/10 range
- is_reserved: Same as above but uses IETF reserved range
- is_mulicast: Same as above but uses ff00::/8 range
- is_routeable: Similar to above but enforces not private, link_local, reserved or multicast
- is_6to4: Same as above but uses 2002::/16 range
- is_teredo: Same as above but uses 2001::/32 range
- subnets: value must be a member of at least one from list of subnets
- is_ipv6: either:
  - None (default): indifferent
  - True (enforce): must be an IPv6 address
  - False (forbid): must NOT be an IPv6 address

Minip and maxip may also be lists or tuples of addresses in all above forms (str, int, list / tuple), allowing setup of multiple address ranges:

```
minip = (minip1, minip2, ... minipN)
           |       |           |
           |       |           |
maxip = (maxip1, maxip2, ... maxipN)
```

Longer iterable will be truncated to match length of shorter one.

```
>>> IS_IPADDRESS()('192.168.1.5')
('192.168.1.5', None)
>>> IS_IPADDRESS(is_ipv6=False)('192.168.1.5')
('192.168.1.5', None)
>>> IS_IPADDRESS()('255.255.255.255')
('255.255.255.255', None)
>>> IS_IPADDRESS()('192.168.1.5 ')
('192.168.1.5 ', 'enter valid IP address')
>>> IS_IPADDRESS()('192.168.1.1.5')
('192.168.1.1.5', 'enter valid IP address')
>>> IS_IPADDRESS()('123.123')
('123.123', 'enter valid IP address')
>>> IS_IPADDRESS()('1111.2.3.4')
('1111.2.3.4', 'enter valid IP address')
>>> IS_IPADDRESS()('0111.2.3.4')
```

(continues on next page)

(continued from previous page)

```
('0111.2.3.4', 'enter valid IP address')
>>> IS_IPADDRESS()('256.2.3.4')
('256.2.3.4', 'enter valid IP address')
>>> IS_IPADDRESS()('300.2.3.4')
('300.2.3.4', 'enter valid IP address')
>>> IS_IPADDRESS(minip='192.168.1.0', maxip='192.168.1.255')('192.168.1.100')
('192.168.1.100', None)
>>> IS_IPADDRESS(minip='1.2.3.5', maxip='1.2.3.9', error_message='Bad ip')('1.2.3.
↪4')
('1.2.3.4', 'bad ip')
>>> IS_IPADDRESS(maxip='1.2.3.4', invert=True)('127.0.0.1')
('127.0.0.1', None)
>>> IS_IPADDRESS(maxip='192.168.1.4', invert=True)('192.168.1.4')
('192.168.1.4', 'enter valid IP address')
>>> IS_IPADDRESS(is_localhost=True)('127.0.0.1')
('127.0.0.1', None)
>>> IS_IPADDRESS(is_localhost=True)('192.168.1.10')
('192.168.1.10', 'enter valid IP address')
>>> IS_IPADDRESS(is_localhost=False)('127.0.0.1')
('127.0.0.1', 'enter valid IP address')
>>> IS_IPADDRESS(maxip='100.0.0.0', is_localhost=True)('127.0.0.1')
('127.0.0.1', 'enter valid IP address')
```

```
>>> IS_IPADDRESS()('fe80::126c:8ffa:fe22:b3af')
('fe80::126c:8ffa:fe22:b3af', None)
>>> IS_IPADDRESS(is_ipv4=False)('fe80::126c:8ffa:fe22:b3af')
('fe80::126c:8ffa:fe22:b3af', None)
>>> IS_IPADDRESS()('fe80::126c:8ffa:fe22:b3af  ')
('fe80::126c:8ffa:fe22:b3af  ', 'enter valid IP address')
>>> IS_IPADDRESS(is_ipv4=True)('fe80::126c:8ffa:fe22:b3af')
('fe80::126c:8ffa:fe22:b3af', 'enter valid IP address')
>>> IS_IPADDRESS(is_ipv6=True)('192.168.1.1')
('192.168.1.1', 'enter valid IP address')
>>> IS_IPADDRESS(is_ipv6=True, error_message='Bad ip')('192.168.1.1')
('192.168.1.1', 'bad ip')
>>> IS_IPADDRESS(is_link_local=True)('fe80::126c:8ffa:fe22:b3af')
('fe80::126c:8ffa:fe22:b3af', None)
>>> IS_IPADDRESS(is_link_local=False)('fe80::126c:8ffa:fe22:b3af')
('fe80::126c:8ffa:fe22:b3af', 'enter valid IP address')
>>> IS_IPADDRESS(is_link_local=True)('2001::126c:8ffa:fe22:b3af')
('2001::126c:8ffa:fe22:b3af', 'enter valid IP address')
>>> IS_IPADDRESS(is_multicast=True)('2001::126c:8ffa:fe22:b3af')
('2001::126c:8ffa:fe22:b3af', 'enter valid IP address')
>>> IS_IPADDRESS(is_multicast=True)('ff00::126c:8ffa:fe22:b3af')
('ff00::126c:8ffa:fe22:b3af', None)
>>> IS_IPADDRESS(is_routeable=True)('2001::126c:8ffa:fe22:b3af')
('2001::126c:8ffa:fe22:b3af', None)
>>> IS_IPADDRESS(is_routeable=True)('ff00::126c:8ffa:fe22:b3af')
('ff00::126c:8ffa:fe22:b3af', 'enter valid IP address')
>>> IS_IPADDRESS(subnets='2001::/32')('2001::8ffa:fe22:b3af')
('2001::8ffa:fe22:b3af', None)
>>> IS_IPADDRESS(subnets='fb00::/8')('2001::8ffa:fe22:b3af')
('2001::8ffa:fe22:b3af', 'enter valid IP address')
>>> IS_IPADDRESS(subnets=['fc00::/8','2001::/32'])('2001::8ffa:fe22:b3af')
('2001::8ffa:fe22:b3af', None)
>>> IS_IPADDRESS(subnets='invalidsubnet')('2001::8ffa:fe22:b3af')
('2001::8ffa:fe22:b3af', 'invalid subnet provided')
```

class gluon.validators.**IS_LENGTH**(*maxsize=255*, *minsize=0*, *error_message='Enter from %(min)g to %(max)g characters'*)

Bases: gluon.validators.Validator

Checks if length of field's value fits between given boundaries. Works for both text and file inputs.

Parameters

- **maxsize** – maximum allowed length / size
- **minsize** – minimum allowed length / size

Examples

Check if text string is shorter than 33 characters:

```
INPUT(_type='text', _name='name', requires=IS_LENGTH(32))
```

Check if password string is longer than 5 characters:

```
INPUT(_type='password', _name='name', requires=IS_LENGTH(minsize=6))
```

Check if uploaded file has size between 1KB and 1MB:

```
INPUT(_type='file', _name='name', requires=IS_LENGTH(1048576, 1024))
```

Other examples:

```
>>> IS_LENGTH()('')
('', None)
>>> IS_LENGTH()('1234567890')
('1234567890', None)
>>> IS_LENGTH(maxsize=5, minsize=0)('1234567890')  # too long
('1234567890', 'enter from 0 to 5 characters')
>>> IS_LENGTH(maxsize=50, minsize=20)('1234567890')  # too short
('1234567890', 'enter from 20 to 50 characters')
```

class gluon.validators.**IS_LIST_OF**(*other=None*, *minimum=0*, *maximum=100*, *error_message=None*)

Bases: gluon.validators.Validator

class gluon.validators.**IS_LOWER**

Bases: gluon.validators.Validator

Converts to lower case:

```
>>> IS_LOWER()('ABC')
('abc', None)
>>> IS_LOWER()('Ñ')
('\xc3\xb1', None)
```

class gluon.validators.**IS_MATCH**(*expression*, *error_message='Invalid expression'*, *strict=False*, *search=False*, *extract=False*, *is_unicode=False*)

Bases: gluon.validators.Validator

Example

Used as:

```
INPUT(_type='text', _name='name', requires=IS_MATCH('.+'))
```

The argument of IS_MATCH is a regular expression:

```
>>> IS_MATCH('.+')('hello')
('hello', None)

>>> IS_MATCH('hell')('hello')
('hello', None)

>>> IS_MATCH('hell.*', strict=False)('hello')
('hello', None)

>>> IS_MATCH('hello')('shello')
('shello', 'invalid expression')

>>> IS_MATCH('hello', search=True)('shello')
('shello', None)

>>> IS_MATCH('hello', search=True, strict=False)('shellox')
('shellox', None)

>>> IS_MATCH('.*hello.*', search=True, strict=False)('shellox')
('shellox', None)

>>> IS_MATCH('.+')('')
('', 'invalid expression')
```

class gluon.validators.**IS_EQUAL_TO**(*expression*, *error_message='No match'*)

Bases: gluon.validators.Validator

Example

Used as:

```
INPUT(_type='text', _name='password')
INPUT(_type='text', _name='password2',
      requires=IS_EQUAL_TO(request.vars.password))
```

The argument of IS_EQUAL_TO is a string:

```
>>> IS_EQUAL_TO('aaa')('aaa')
('aaa', None)

>>> IS_EQUAL_TO('aaa')('aab')
('aab', 'no match')
```

class gluon.validators.**IS_NOT_EMPTY**(*error_message='Enter a value'*, *empty_regex=None*)

Bases: gluon.validators.Validator

Example

Used as:

```
INPUT(_type='text', _name='name', requires=IS_NOT_EMPTY())

>>> IS_NOT_EMPTY()(1)
(1, None)
>>> IS_NOT_EMPTY()(0)
(0, None)
>>> IS_NOT_EMPTY()('x')
('x', None)
>>> IS_NOT_EMPTY()(' x ')
('x', None)
>>> IS_NOT_EMPTY()(None)
(None, 'enter a value')
>>> IS_NOT_EMPTY()('')
('', 'enter a value')
>>> IS_NOT_EMPTY()('  ')
('', 'enter a value')
>>> IS_NOT_EMPTY()(' \n\t')
('', 'enter a value')
>>> IS_NOT_EMPTY()([])
([], 'enter a value')
>>> IS_NOT_EMPTY(empty_regex='def')('def')
('', 'enter a value')
>>> IS_NOT_EMPTY(empty_regex='de[fg]')('deg')
('', 'enter a value')
>>> IS_NOT_EMPTY(empty_regex='def')('abc')
('abc', None)
```

class gluon.validators.**IS_NOT_IN_DB**(*dbset*, *field*, *error_message='Value already in database or empty'*, *allowed_override=[]*, *ignore_common_filters=False*)

Bases: gluon.validators.Validator

Example

Used as:

```
INPUT(_type='text', _name='name', requires=IS_NOT_IN_DB(db, db.table))
```

makes the field unique

set_self_id(*id*)

gluon.validators.**IS_NULL_OR**

alias of *gluon.validators.IS_EMPTY_OR*

class gluon.validators.**IS_SLUG**(*maxlen=80*, *check=False*, *error_message='Must be slug'*, *keep_underscores=False*)

Bases: gluon.validators.Validator

converts arbitrary text string to a slug:

```
>>> IS_SLUG()('abc123')
('abc123', None)
>>> IS_SLUG()('ABC123')
('abc123', None)
>>> IS_SLUG()('abc-123')
('abc-123', None)
```

(continues on next page)

(continued from previous page)

```
>>> IS_SLUG()('abc--123')
('abc-123', None)
>>> IS_SLUG()('abc 123')
('abc-123', None)
>>> IS_SLUG()('abc      _123')
('abc-123', None)
>>> IS_SLUG()('-abc-')
('abc', None)
>>> IS_SLUG()('--a--b--_ -c--')
('a-b-c', None)
>>> IS_SLUG()('abc&amp;123')
('abc123', None)
>>> IS_SLUG()('abc&amp;123&amp;def')
('abc123def', None)
>>> IS_SLUG()('ñ')
('n', None)
>>> IS_SLUG(maxlen=4)('abc123')
('abc1', None)
>>> IS_SLUG()('abc_123')
('abc-123', None)
>>> IS_SLUG(keep_underscores=False)('abc_123')
('abc-123', None)
>>> IS_SLUG(keep_underscores=True)('abc_123')
('abc_123', None)
>>> IS_SLUG(check=False)('abc')
('abc', None)
>>> IS_SLUG(check=True)('abc')
('abc', None)
>>> IS_SLUG(check=False)('a bc')
('a-bc', None)
>>> IS_SLUG(check=True)('a bc')
('a bc', 'must be slug')
```

static urlify(*value*, *maxlen=80*, *keep_underscores=False*)

class gluon.validators.**IS_STRONG**(*min=None*, *max=None*, *upper=None*, *lower=None*, *number=None*, *entropy=None*, *special=None*, *specials='~!@#$%^&*()_+-=?<>,.:;{}[]|'*, *invalid=' "'*, *error_message=None*, *es=False*)

Bases: `object`

Examples

Use as:

```
INPUT(_type='password', _name='passwd',
requires=IS_STRONG(min=10, special=2, upper=2))
```

enforces complexity requirements on a field

```
>>> IS_STRONG(es=True)('Abcd1234')
('Abcd1234',
 'Must include at least 1 of the following: ~!@#$%^&*()_+-=?<>,.:;{}[]|')
>>> IS_STRONG(es=True)('Abcd1234!')
('Abcd1234!', None)
```

(continues on next page)

(continued from previous page)

```
>>> IS_STRONG(es=True, entropy=1)('a')
('a', None)
>>> IS_STRONG(es=True, entropy=1, min=2)('a')
('a', 'Minimum length is 2')
>>> IS_STRONG(es=True, entropy=100)('abc123')
('abc123', 'Entropy (32.35) less than required (100)')
>>> IS_STRONG(es=True, entropy=100)('and')
('and', 'Entropy (14.57) less than required (100)')
>>> IS_STRONG(es=True, entropy=100)('aaa')
('aaa', 'Entropy (14.42) less than required (100)')
>>> IS_STRONG(es=True, entropy=100)('a1d')
('a1d', 'Entropy (15.97) less than required (100)')
>>> IS_STRONG(es=True, entropy=100)('añd')
('a\xc3\xb1d', 'Entropy (18.13) less than required (100)')
```

class gluon.validators.**IS_TIME**(*error_message='Enter time as hh:mm:ss (seconds, am, pm optional)'*)

Bases: gluon.validators.Validator

Example

Use as:

```
INPUT(_type='text', _name='name', requires=IS_TIME())
```

understands the following formats hh:mm:ss [am/pm] hh:mm [am/pm] hh [am/pm]

[am/pm] is optional, ':' can be replaced by any other non-space non-digit:

```
>>> IS_TIME()('21:30')
(datetime.time(21, 30), None)
>>> IS_TIME()('21-30')
(datetime.time(21, 30), None)
>>> IS_TIME()('21.30')
(datetime.time(21, 30), None)
>>> IS_TIME()('21:30:59')
(datetime.time(21, 30, 59), None)
>>> IS_TIME()('5:30')
(datetime.time(5, 30), None)
>>> IS_TIME()('5:30 am')
(datetime.time(5, 30), None)
>>> IS_TIME()('5:30 pm')
(datetime.time(17, 30), None)
>>> IS_TIME()('5:30 whatever')
('5:30 whatever', 'enter time as hh:mm:ss (seconds, am, pm optional)')
>>> IS_TIME()('5:30 20')
('5:30 20', 'enter time as hh:mm:ss (seconds, am, pm optional)')
>>> IS_TIME()('24:30')
('24:30', 'enter time as hh:mm:ss (seconds, am, pm optional)')
>>> IS_TIME()('21:60')
('21:60', 'enter time as hh:mm:ss (seconds, am, pm optional)')
>>> IS_TIME()('21:30::')
('21:30::', 'enter time as hh:mm:ss (seconds, am, pm optional)')
>>> IS_TIME()('')
('', 'enter time as hh:mm:ss (seconds, am, pm optional)')ù
```

class gluon.validators.**IS_UPLOAD_FILENAME**(*filename=None*, *extension=None*, *lastdot=True*, *case=1*, *error_message='Enter valid filename'*)

Bases: gluon.validators.Validator

Checks if name and extension of file uploaded through file input matches given criteria.

Does *not* ensure the file type in any way. Returns validation failure if no data was uploaded.

Parameters

- **filename** – filename (before dot) regex
- **extension** – extension (after dot) regex
- **lastdot** – which dot should be used as a filename / extension separator: True means last dot, eg. file.png -> file / png False means first dot, eg. file.tar.gz -> file / tar.gz
- **case** – 0 - keep the case, 1 - transform the string into lowercase (default), 2 - transform the string into uppercase

If there is no dot present, extension checks will be done against empty string and filename checks against whole value.

Examples

Check if file has a pdf extension (case insensitive):

INPUT(_type='file', _name='name', requires=IS_UPLOAD_FILENAME(extension='pdf'))

Check if file has a tar.gz extension and name starting with backup:

INPUT(_type='file', _name='name', requires=IS_UPLOAD_FILENAME(filename='backup.*', extension='tar.gz', lastdot=False))

Check if file has no extension and name matching README (case sensitive):

INPUT(_type='file', _name='name', requires=IS_UPLOAD_FILENAME(filename='^README$', extension='^$', case=0)

class gluon.validators.**IS_UPPER**

Bases: gluon.validators.Validator

Converts to upper case:

```
>>> IS_UPPER()('abc')
('ABC', None)
>>> IS_UPPER()('ñ')
('\xc3\x91', None)
```

class gluon.validators.**IS_URL**(*error_message='Enter a valid URL'*, *mode='http'*, *allowed_schemes=None*, *prepend_scheme='http'*, *allowed_tlds=None*)

Bases: gluon.validators.Validator

Rejects a URL string if any of the following is true:

- The string is empty or None
- The string uses characters that are not allowed in a URL
- The string breaks any of the HTTP syntactic rules
- The URL scheme specified (if one is specified) is not 'http' or 'https'

- The top-level domain (if a host name is specified) does not exist

(These rules are based on RFC 2616: http://www.faqs.org/rfcs/rfc2616.html)

This function only checks the URL's syntax. It does not check that the URL points to a real document, for example, or that it otherwise makes sense semantically. This function does automatically prepend 'http://' in front of a URL in the case of an abbreviated URL (e.g. 'google.ca').

If the parameter mode='generic' is used, then this function's behavior changes. It then rejects a URL string if any of the following is true:

- The string is empty or None
- The string uses characters that are not allowed in a URL
- The URL scheme specified (if one is specified) is not valid

(These rules are based on RFC 2396: http://www.faqs.org/rfcs/rfc2396.html)

The list of allowed schemes is customizable with the allowed_schemes parameter. If you exclude None from the list, then abbreviated URLs (lacking a scheme such as 'http') will be rejected.

The default prepended scheme is customizable with the prepend_scheme parameter. If you set prepend_scheme to None then prepending will be disabled. URLs that require prepending to parse will still be accepted, but the return value will not be modified.

IS_URL is compatible with the Internationalized Domain Name (IDN) standard specified in RFC 3490 (http://tools.ietf.org/html/rfc3490). As a result, URLs can be regular strings or unicode strings. If the URL's domain component (e.g. google.ca) contains non-US-ASCII letters, then the domain will be converted into Punycode (defined in RFC 3492, http://tools.ietf.org/html/rfc3492). IS_URL goes a bit beyond the standards, and allows non-US-ASCII characters to be present in the path and query components of the URL as well. These non-US-ASCII characters will be escaped using the standard '%20' type syntax. e.g. the unicode character with hex code 0x4e86 will become '%4e%86'

Parameters

- **error_message** – a string, the error message to give the end user if the URL does not validate
- **allowed_schemes** – a list containing strings or None. Each element is a scheme the inputed URL is allowed to use
- **prepend_scheme** – a string, this scheme is prepended if it's necessary to make the URL valid

Code Examples:

```
INPUT(_type='text', _name='name', requires=IS_URL())
>>> IS_URL()('abc.com')
('http://abc.com', None)

INPUT(_type='text', _name='name', requires=IS_URL(mode='generic'))
>>> IS_URL(mode='generic')('abc.com')
('abc.com', None)

INPUT(_type='text', _name='name',
    requires=IS_URL(allowed_schemes=['https'], prepend_scheme='https'))
>>> IS_URL(allowed_schemes=['https'], prepend_scheme='https')('https://abc.com')
('https://abc.com', None)

INPUT(_type='text', _name='name',
    requires=IS_URL(prepend_scheme='https'))
```

(continues on next page)

(continued from previous page)

```
>>> IS_URL(prepend_scheme='https')('abc.com')
('https://abc.com', None)

INPUT(_type='text', _name='name',
    requires=IS_URL(mode='generic', allowed_schemes=['ftps', 'https'],
        prepend_scheme='https'))
>>> IS_URL(mode='generic', allowed_schemes=['ftps', 'https'], prepend_scheme=
↪'https')('https://abc.com')
('https://abc.com', None)
>>> IS_URL(mode='generic', allowed_schemes=['ftps', 'https', None], prepend_
↪scheme='https')('abc.com')
('abc.com', None)
```

@author: Jonathan Benn

class gluon.validators.**IS_JSON**(*error_message='Invalid json'*, *native_json=False*)

Bases: gluon.validators.Validator

Example

Used as:

```
INPUT(_type='text', _name='name',
    requires=IS_JSON(error_message="This is not a valid json input")

>>> IS_JSON()('{"a": 100}')
({u'a': 100}, None)

>>> IS_JSON()('spam1234')
('spam1234', 'invalid json')
```

formatter(*value*)

For some validators returns a formatted version (matching the validator) of value. Otherwise just returns the value.

CHAPTER 38

widget Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

38.1 The widget is called from web2py

class gluon.widget.**IO**
Bases: object

write(*data*)

gluon.widget.**check_existent_app**(*options*, *appname*)

gluon.widget.**console**()
Defines the behavior of the console web2py execution

gluon.widget.**get_code_for_scheduler**(*app*, *options*)

gluon.widget.**get_url**(*host*, *path='/'*, *proto='http'*, *port=80*)

gluon.widget.**run_system_tests**(*options*)
Runs unittests for gluon.tests

gluon.widget.**start**(*cron=True*)
Starts server

gluon.widget.**start_browser**(*url*, *startup=False*)

gluon.widget.**start_schedulers**(*options*)

class gluon.widget.**web2pyDialog**(*root*, *options*)
Bases: object

Main window dialog

checkTaskBar()
Checks taskbar status

connect_pages()
Connects pages

error(*message*)
Shows error message

quit(*justHide=False*)
Finishes the program execution

server_ready()

start()
Starts web2py server

start_schedulers(*app*)

stop()
Stops web2py server

try_start_scheduler(*app*)

try_stop_scheduler(*app*)

update(*text*)
Updates app text

update_canvas()
Updates canvas

update_schedulers(*start=False*)

CHAPTER 39

xmlrpc Module

This file is part of the web2py Web Framework
Copyrighted by Massimo Di Pierro <mdipierro@cs.depaul.edu>
License: LGPLv3 (http://www.gnu.org/licenses/lgpl.html)

gluon.xmlrpc.**handler**(*request*, *response*, *methods*)

CHAPTER 40

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

gluon.admin, 3
gluon.cache, 7
gluon.cfs, 9
gluon.compileapp, 11
gluon.contenttype, 15
gluon.custom_import, 17
gluon.dal, 19
gluon.debug, 21
gluon.decoder, 23
gluon.fileutils, 25
gluon.globals, 27
gluon.highlight, 33
gluon.html, 35
gluon.http, 51
gluon.languages, 53
gluon.main, 57
gluon.messageboxhandler, 59
gluon.myregex, 61
gluon.newcron, 63
gluon.portalocker, 65
gluon.recfile, 67
gluon.restricted, 69
gluon.rewrite, 71
gluon.sanitizer, 75
gluon.scheduler, 77
gluon.serializers, 83
gluon.settings, 85
gluon.shell, 87
gluon.sql, 89
gluon.sqlhtml, 97
gluon.storage, 107
gluon.streamer, 111
gluon.template, 113
gluon.tools, 117
gluon.utf8, 139
gluon.utils, 143
gluon.validators, 145
gluon.widget, 171
gluon.xmlrpc, 173

Index

A

A (class in gluon.html), 35
absolute_path_link() (in module gluon.newcron), 63
abspath() (in module gluon.fileutils), 25
accepts() (gluon.html.FORM method), 41
accepts() (gluon.sqlhtml.SQLFORM method), 101
accessible_query() (gluon.tools.Auth method), 121
acf() (gluon.rewrite.MapUrlOut method), 72
acquire() (gluon.newcron.Token method), 63
action() (gluon.cache.Cache method), 7
add_button() (gluon.html.FORM method), 41
add_class() (in module gluon.sqlhtml), 104
add_deps() (gluon.scheduler.JobGraph method), 77
add_group() (gluon.tools.Auth method), 121
add_membership() (gluon.tools.Auth method), 121
add_path_first() (in module gluon.admin), 3
add_permission() (gluon.tools.Auth method), 122
adj_hibernation() (gluon.scheduler.Scheduler method), 79
AES_new() (in module gluon.utils), 143
allows_jwt() (gluon.tools.Auth method), 122
amfrpc() (gluon.tools.Service method), 131
amfrpc3() (gluon.tools.Service method), 131
ANY_OF (class in gluon.validators), 145
apath() (in module gluon.admin), 3
API_SERVER (gluon.tools.Recaptcha attribute), 129
API_SSL_SERVER (gluon.tools.Recaptcha attribute), 129
API_URI (gluon.tools.Recaptcha2 attribute), 130
app_cleanup() (in module gluon.admin), 3
app_compile() (in module gluon.admin), 3
app_create() (in module gluon.admin), 4
app_install() (in module gluon.admin), 4
app_pack() (in module gluon.admin), 4
app_pack_compiled() (in module gluon.admin), 4
app_uninstall() (in module gluon.admin), 4
append() (gluon.html.DIV method), 37
append() (gluon.template.BlockNode method), 113
append() (gluon.template.Content method), 114
appfactory() (in module gluon.main), 58
apply_filter() (gluon.languages.translator method), 53
archive() (gluon.tools.Auth static method), 122
archive() (gluon.tools.Crud static method), 130
arg0 (gluon.rewrite.MapUrlIn attribute), 71
as_dict() (gluon.html.FORM method), 41
as_dict() (gluon.sql.DAL method), 92
as_dict() (gluon.sql.DAL.Field method), 91
as_dict() (gluon.sql.DAL.Row method), 91
as_dict() (gluon.sql.DAL.Table method), 92
as_dict() (gluon.sql.Field method), 96
as_json() (gluon.html.FORM method), 41
as_json() (gluon.sql.DAL.Row method), 91
as_xml() (gluon.html.FORM method), 41
as_xml() (gluon.sql.DAL.Row method), 91
as_yaml() (gluon.html.FORM method), 41
assert_status() (gluon.html.FORM method), 41
assert_status() (gluon.sqlhtml.SQLFORM method), 101
assign_tasks() (gluon.scheduler.Scheduler method), 79
ASSIGNJS() (in module gluon.html), 35
async() (gluon.scheduler.MetaScheduler method), 77
Auth (class in gluon.tools), 121
AutocompleteWidget (class in gluon.sqlhtml), 97
autoDetectXMLEncoding() (in module gluon.decoder), 23
autokey (gluon.cache.Cache attribute), 8
automatic (gluon.validators.IS_IPV4 attribute), 158
automenu() (gluon.tools.Wiki method), 135
AUTOTYPES (gluon.sqlhtml.SQLFORM attribute), 101

B

B (class in gluon.html), 36
basic() (gluon.tools.Auth method), 122
BEAUTIFY (class in gluon.html), 36
being_a_ticker() (gluon.scheduler.Scheduler method), 79
BlockNode (class in gluon.template), 113
BODY (class in gluon.html), 36
body (gluon.globals.Request attribute), 28
BooleanWidget (class in gluon.sqlhtml), 97
BR (class in gluon.html), 36

build_acf() (gluon.rewrite.MapUrlOut method), 72
build_environment() (in module gluon.compileapp), 12
build_query() (gluon.sqlhtml.SQLFORM static method), 101
build_set() (gluon.validators.IS_IN_DB method), 154
bulk_insert() (gluon.sql.DAL.Table method), 92
bulk_register() (gluon.tools.Auth method), 123
BUTTON (class in gluon.html), 36

C

Cache (class in gluon.cache), 7
CacheRepresenter (class in gluon.sqlhtml), 97
callback() (gluon.sqlhtml.AutocompleteWidget method), 97
can_edit() (gluon.tools.Wiki method), 135
can_join() (gluon.sql.DAL method), 92
can_manage() (gluon.tools.Wiki method), 135
can_read() (gluon.tools.Wiki method), 135
can_search() (gluon.tools.Wiki method), 135
can_see_menu() (gluon.tools.Wiki method), 135
capitalize() (gluon.utf8.Utf8 method), 139
cas_login() (gluon.tools.Auth method), 123
cas_validate() (gluon.tools.Auth method), 123
cast_keys() (in module gluon.serializers), 83
CAT (class in gluon.html), 36
CENTER (class in gluon.html), 36
center() (gluon.utf8.Utf8 method), 139
change_password() (gluon.tools.Auth method), 123
check_credentials() (in module gluon.fileutils), 26
check_existent_app() (in module gluon.widget), 171
check_interaction() (in module gluon.debug), 22
check_new_version() (in module gluon.admin), 4
check_reserved_keyword() (gluon.sql.DAL method), 92
CheckboxesWidget (class in gluon.sqlhtml), 97
checkTaskBar() (gluon.widget.web2pyDialog method), 171
cleanpath() (in module gluon.fileutils), 25
CLEANUP (class in gluon.validators), 145
clear() (gluon.globals.Session method), 30
clear_content() (gluon.template.Content method), 114
clear_interaction() (gluon.debug.WebDebugger method), 21
clear_session_cookies() (gluon.globals.Session method), 30
clone() (gluon.sql.DAL.Field method), 91
clone() (gluon.sql.Field method), 96
close() (gluon.portalocker.LockedFile method), 65
close() (gluon.sql.DAL method), 92
cloud() (gluon.tools.Wiki method), 135
CODE (class in gluon.html), 36
COL (class in gluon.html), 37
COLGROUP (class in gluon.html), 37
commit() (gluon.sql.DAL method), 92
communicate() (in module gluon.debug), 22
compare() (in module gluon.utils), 143
compile2() (in module gluon.restricted), 69
compile_application() (in module gluon.compileapp), 12
compile_controllers() (in module gluon.compileapp), 12
compile_models() (in module gluon.compileapp), 12
compile_regex() (in module gluon.rewrite), 72
compile_views() (in module gluon.compileapp), 12
component() (gluon.tools.Wiki static method), 135
compute_uuid() (gluon.globals.Request method), 28
confirm() (gluon.html.FORM static method), 41
confirm_registration() (gluon.tools.Auth method), 123
connect() (gluon.globals.Session method), 30
connect_pages() (gluon.widget.web2pyDialog method), 172
console() (in module gluon.widget), 171
Content (class in gluon.template), 114
content_type (gluon.sqlhtml.ExportClass attribute), 98
content_type (gluon.sqlhtml.ExporterCSV attribute), 98
content_type (gluon.sqlhtml.ExporterCSV_hidden attribute), 98
content_type (gluon.sqlhtml.ExporterHTML attribute), 98
content_type (gluon.sqlhtml.ExporterJSON attribute), 98
content_type (gluon.sqlhtml.ExporterTSV attribute), 99
content_type (gluon.sqlhtml.ExporterXML attribute), 99
contenttype() (in module gluon.contenttype), 15
cookies2headers() (gluon.http.HTTP method), 51
count() (gluon.sql.DAL.Field method), 91
count() (gluon.sql.Field method), 96
count() (gluon.utf8.Utf8 method), 139
create() (gluon.tools.Crud method), 130
create() (gluon.tools.Wiki method), 135
create_missing_app_folders() (in module gluon.admin), 5
create_missing_folders() (in module gluon.admin), 5
createform() (gluon.sqlhtml.SQLFORM method), 102
crondance() (in module gluon.newcron), 63
cronlauncher (class in gluon.newcron), 63
Crud (class in gluon.tools), 130
CRYPT (class in gluon.validators), 146
csv() (gluon.tools.Service method), 132
csv() (in module gluon.serializers), 83
custom_import_install() (in module gluon.custom_import), 17
custom_importer() (in module gluon.custom_import), 17
custom_json() (in module gluon.serializers), 83
CustomImportException, 17

D

DAL (class in gluon.sql), 89
DAL.Field (class in gluon.sql), 91
DAL.Row (class in gluon.sql), 91
DAL.Table (class in gluon.sql), 92
DatetimeWidget (class in gluon.sqlhtml), 98
DateWidget (class in gluon.sqlhtml), 98

DecimalWidget (class in gluon.sqlhtml), 98
decode() (gluon.utf8.Utf8 method), 139
decoder() (in module gluon.decoder), 23
default_delimiters (gluon.template.TemplateParser attribute), 115
default_messages (gluon.tools.Auth attribute), 123
default_settings (gluon.tools.Auth attribute), 124
DEFAULT_WIDTH (gluon.sqlhtml.UploadWidget attribute), 104
define_signature() (gluon.tools.Auth method), 125
define_table() (gluon.sql.DAL method), 93
define_tables() (gluon.scheduler.Scheduler method), 79
define_tables() (gluon.tools.Auth method), 125
del_group() (gluon.tools.Auth method), 125
del_membership() (gluon.tools.Auth method), 125
del_permission() (gluon.tools.Auth method), 125
delete() (gluon.tools.Crud method), 130
DELETE_FILE (gluon.sqlhtml.UploadWidget attribute), 104
dictform() (gluon.sqlhtml.SQLFORM static method), 102
die() (gluon.scheduler.MetaScheduler method), 78
die() (in module gluon.shell), 87
disable() (gluon.scheduler.Scheduler method), 79
distributed_transaction_begin() (gluon.sql.DAL static method), 93
distributed_transaction_commit() (gluon.sql.DAL static method), 93
DIV (class in gluon.html), 37
do_continue() (gluon.debug.WebDebugger method), 21
do_exec() (gluon.debug.WebDebugger method), 21
do_next() (gluon.debug.WebDebugger method), 21
do_quit() (gluon.debug.WebDebugger method), 21
do_return() (gluon.debug.WebDebugger method), 21
do_step() (gluon.debug.WebDebugger method), 21
DoubleWidget (class in gluon.sqlhtml), 98
download() (gluon.globals.Response method), 28
drop() (gluon.sql.DAL.Table method), 92
DummyResponse (class in gluon.template), 114

E

edit() (gluon.tools.Wiki method), 135
editmedia() (gluon.tools.Wiki method), 135
element() (gluon.html.DIV method), 38
elements() (gluon.html.DIV method), 38
elements() (gluon.html.XML method), 49
EM (class in gluon.html), 40
email_registration() (gluon.tools.Auth method), 125
email_reset_password() (gluon.tools.Auth method), 125
EMBED (class in gluon.html), 40
embed64() (in module gluon.html), 50
emit() (gluon.messageboxhandler.MessageBoxHandler method), 59
emit() (gluon.messageboxhandler.NotifySendHandler method), 59
enable_autocomplete_and_history() (in module gluon.shell), 87
enable_record_versioning() (gluon.tools.Auth method), 125
encode() (gluon.utf8.Utf8 method), 140
endswith() (gluon.utf8.Utf8 method), 140
env() (in module gluon.shell), 87
error() (gluon.tools.Service method), 132
error() (gluon.widget.web2pyDialog method), 172
everybody (gluon.tools.Wiki attribute), 135
exception() (gluon.debug.WebDebugger method), 21
exec_environment() (in module gluon.shell), 87
exec_pythonrc() (in module gluon.shell), 87
execute_from_command_line() (in module gluon.shell), 87
executesql() (gluon.sql.DAL method), 93
executor() (in module gluon.scheduler), 82
exists() (in module gluon.recfile), 67
expandtabs() (gluon.utf8.Utf8 method), 140
export() (gluon.sqlhtml.ExportClass method), 98
export() (gluon.sqlhtml.ExporterCSV method), 98
export() (gluon.sqlhtml.ExporterCSV_hidden method), 98
export() (gluon.sqlhtml.ExporterHTML method), 98
export() (gluon.sqlhtml.ExporterJSON method), 98
export() (gluon.sqlhtml.ExporterTSV method), 99
export() (gluon.sqlhtml.ExporterXML method), 99
export_to_csv_file() (gluon.sql.DAL method), 93
ExportClass (class in gluon.sqlhtml), 98
ExporterCSV (class in gluon.sqlhtml), 98
ExporterCSV_hidden (class in gluon.sqlhtml), 98
ExporterHTML (class in gluon.sqlhtml), 98
ExporterJSON (class in gluon.sqlhtml), 98
ExporterTSV (class in gluon.sqlhtml), 99
ExporterXML (class in gluon.sqlhtml), 99
extcron (class in gluon.newcron), 63
extend() (gluon.template.BlockNode method), 113
extend() (gluon.template.Content method), 114
extend() (gluon.template.TemplateParser method), 115

F

factory() (gluon.sqlhtml.SQLFORM static method), 102
fast_urandom16() (in module gluon.utils), 143
fetch() (in module gluon.tools), 137
Field (class in gluon.sql), 95
FIELDKEY_DELETE_RECORD (gluon.sqlhtml.SQLFORM attribute), 101
FIELDNAME_REQUEST_DELETE (gluon.sqlhtml.SQLFORM attribute), 101
fields (gluon.sql.DAL.Table attribute), 92
FIELDSET (class in gluon.html), 40
file_ext (gluon.sqlhtml.ExportClass attribute), 98
file_ext (gluon.sqlhtml.ExporterCSV attribute), 98

file_ext (gluon.sqlhtml.ExporterCSV_hidden attribute), 98
file_ext (gluon.sqlhtml.ExporterHTML attribute), 98
file_ext (gluon.sqlhtml.ExporterJSON attribute), 98
file_ext (gluon.sqlhtml.ExporterTSV attribute), 99
file_ext (gluon.sqlhtml.ExporterXML attribute), 99
filter_err() (in module gluon.rewrite), 72
filter_url() (in module gluon.rewrite), 72
find() (gluon.utf8.Utf8 method), 140
find_exposed_functions() (in module gluon.compileapp), 12
findT() (in module gluon.languages), 55
first_paragraph() (gluon.tools.Wiki method), 135
fix_hostname() (gluon.tools.Wiki method), 135
fix_newlines() (in module gluon.fileutils), 26
fixup_missing_path_info() (in module gluon.rewrite), 72
flatten() (gluon.html.DIV method), 39
flatten() (gluon.html.MARKMIN method), 44
flatten() (gluon.html.XML method), 49
flush() (gluon.debug.Pipe method), 21
force() (gluon.languages.translator method), 53
forget() (gluon.globals.Session method), 30
FORM (class in gluon.html), 40
form_factory() (in module gluon.sqlhtml), 104
format() (gluon.utf8.Utf8 method), 140
formatter() (gluon.sql.DAL.Field method), 91
formatter() (gluon.sql.Field method), 96
formatter() (gluon.validators.ANY_OF method), 145
formatter() (gluon.validators.IS_DATE method), 148
formatter() (gluon.validators.IS_DATETIME method), 149
formatter() (gluon.validators.IS_DECIMAL_IN_RANGE method), 150
formatter() (gluon.validators.IS_EMPTY_OR method), 152
formatter() (gluon.validators.IS_FLOAT_IN_RANGE method), 153
formatter() (gluon.validators.IS_JSON method), 169
formatter() (gluon.validators.IS_LIST_OF_EMAILS method), 152
formstyle_bootstrap() (in module gluon.sqlhtml), 105
formstyle_bootstrap3_inline_factory() (in module gluon.sqlhtml), 105
formstyle_bootstrap3_stacked() (in module gluon.sqlhtml), 105
formstyle_divs() (in module gluon.sqlhtml), 105
formstyle_inline() (in module gluon.sqlhtml), 105
formstyle_table2cols() (in module gluon.sqlhtml), 105
formstyle_table3cols() (in module gluon.sqlhtml), 105
formstyle_ul() (in module gluon.sqlhtml), 105
formstyles (gluon.sqlhtml.SQLFORM attribute), 102
FormWidget (class in gluon.sqlhtml), 99
frameset (gluon.html.HTML attribute), 43
frameset (gluon.html.XHTML attribute), 49

G

generate() (in module gluon.recfile), 67
GENERIC_DESCRIPTION (gluon.sqlhtml.UploadWidget attribute), 104
geocode() (in module gluon.tools), 137
get() (gluon.html.DIV method), 39
get() (gluon.sql.DAL.Row method), 91
get_callable_argspec() (in module gluon.utils), 143
get_code_for_scheduler() (in module gluon.widget), 171
get_digest() (in module gluon.utils), 143
get_effective_router() (in module gluon.rewrite), 72
get_format() (gluon.tools.Crud method), 130
get_instances() (gluon.sql.DAL static method), 93
get_or_create_key() (gluon.tools.Auth static method), 125
get_or_create_user() (gluon.tools.Auth method), 125
get_parsed() (in module gluon.template), 115
get_possible_languages() (gluon.languages.translator method), 54
get_possible_languages_info() (gluon.languages.translator method), 54
get_query() (gluon.tools.Crud method), 130
get_renderer() (gluon.tools.Wiki method), 135
get_session() (in module gluon.fileutils), 26
get_t() (gluon.languages.translator method), 54
get_url() (in module gluon.widget), 171
get_usage() (in module gluon.shell), 87
get_vars (gluon.globals.Request attribute), 28
get_vars_next() (gluon.tools.Auth method), 126
get_workers() (gluon.scheduler.Scheduler method), 79
getcfs() (in module gluon.cfs), 9
getfirst() (gluon.storage.Storage method), 108
getipaddrinfo() (in module gluon.utils), 143
getlast() (gluon.storage.Storage method), 108
getlist() (gluon.storage.Storage method), 108
give_up() (gluon.scheduler.MetaScheduler method), 78
gluon.admin (module), 3
gluon.cache (module), 7
gluon.cfs (module), 9
gluon.compileapp (module), 11
gluon.contenttype (module), 15
gluon.custom_import (module), 17
gluon.dal (module), 19
gluon.debug (module), 21
gluon.decoder (module), 23
gluon.fileutils (module), 25
gluon.globals (module), 27
gluon.highlight (module), 33
gluon.html (module), 35
gluon.http (module), 51
gluon.languages (module), 53
gluon.main (module), 57
gluon.messageboxhandler (module), 59

gluon.myregex (module), 61
gluon.newcron (module), 63
gluon.portalocker (module), 65
gluon.recfile (module), 67
gluon.restricted (module), 69
gluon.rewrite (module), 71
gluon.sanitizer (module), 75
gluon.scheduler (module), 77
gluon.serializers (module), 83
gluon.settings (module), 85
gluon.shell (module), 87
gluon.sql (module), 89
gluon.sqlhtml (module), 97
gluon.storage (module), 107
gluon.streamer (module), 111
gluon.template (module), 113
gluon.tools (module), 117
gluon.utf8 (module), 139
gluon.utils (module), 143
gluon.validators (module), 145
gluon.widget (module), 171
gluon.xmlrpc (module), 173
grid() (gluon.sqlhtml.SQLFORM static method), 102
groups() (gluon.tools.Auth method), 126

H

H1 (class in gluon.html), 42
H2 (class in gluon.html), 42
H3 (class in gluon.html), 42
H4 (class in gluon.html), 42
H5 (class in gluon.html), 42
H6 (class in gluon.html), 42
handler() (in module gluon.xmlrpc), 173
hardcron (class in gluon.newcron), 64
harg0 (gluon.rewrite.MapUrlIn attribute), 71
has_membership() (gluon.tools.Auth method), 126
has_options() (gluon.sqlhtml.OptionsWidget static method), 100
has_permission() (gluon.tools.Auth method), 126
has_permission() (gluon.tools.Crud method), 130
has_representer() (gluon.sql.DAL method), 94
HEAD (class in gluon.html), 42
here() (gluon.tools.Auth method), 126
hidden_fields() (gluon.html.FORM method), 41
highlight() (in module gluon.highlight), 33
HR (class in gluon.html), 42
HTML (class in gluon.html), 42
html5 (gluon.html.HTML attribute), 43
html_render() (gluon.tools.Wiki method), 135
HTTP, 51
HttpServer (class in gluon.main), 58

I

I (class in gluon.html), 43
ics() (in module gluon.serializers), 83
ID_DELETE_SUFFIX (gluon.sqlhtml.UploadWidget attribute), 104
id_group() (gluon.tools.Auth method), 126
ID_LABEL_SUFFIX (gluon.sqlhtml.SQLFORM attribute), 101
ID_ROW_SUFFIX (gluon.sqlhtml.SQLFORM attribute), 101
IFRAME (class in gluon.html), 43
IMG (class in gluon.html), 43
impersonate() (gluon.tools.Auth method), 126
import_from_csv_file() (gluon.sql.DAL method), 94
import_from_csv_file() (gluon.sql.DAL.Table method), 92
import_table_definitions() (gluon.sql.DAL method), 94
include() (gluon.template.TemplateParser method), 115
include_files() (gluon.globals.Response method), 28
include_meta() (gluon.globals.Response method), 28
index() (gluon.utf8.Utf8 method), 140
initialize_urandom() (in module gluon.utils), 143
INPUT (class in gluon.html), 43
insert() (gluon.html.DIV method), 40
insert() (gluon.sql.DAL.Table method), 92
insert() (gluon.template.Content method), 114
instances (gluon.tools.PluginManager attribute), 137
IntegerWidget (class in gluon.sqlhtml), 99
interaction() (gluon.debug.WebDebugger method), 22
invalid_url() (in module gluon.rewrite), 72
IO (class in gluon.widget), 171
IS_ALPHANUMERIC (class in gluon.validators), 147
IS_DATE (class in gluon.validators), 148
IS_DATE_IN_RANGE (class in gluon.validators), 147
IS_DATETIME (class in gluon.validators), 148
IS_DATETIME_IN_RANGE (class in gluon.validators), 148
IS_DECIMAL_IN_RANGE (class in gluon.validators), 149
IS_EMAIL (class in gluon.validators), 150
IS_EMPTY_OR (class in gluon.validators), 152
IS_EQUAL_TO (class in gluon.validators), 163
is_expired() (gluon.globals.Session method), 30
IS_EXPR (class in gluon.validators), 152
IS_FLOAT_IN_RANGE (class in gluon.validators), 153
IS_IMAGE (class in gluon.validators), 153
is_image() (gluon.sqlhtml.UploadWidget static method), 104
is_impersonating() (gluon.tools.Auth method), 126
IS_IN_DB (class in gluon.validators), 154
IS_IN_SET (class in gluon.validators), 154
IS_INT_IN_RANGE (class in gluon.validators), 155
IS_IPADDRESS (class in gluon.validators), 159
IS_IPV4 (class in gluon.validators), 156
IS_IPV6 (class in gluon.validators), 158
IS_JSON (class in gluon.validators), 169

IS_LENGTH (class in gluon.validators), 161
IS_LIST_OF (class in gluon.validators), 162
IS_LIST_OF_EMAILS (class in gluon.validators), 152
is_logged_in() (gluon.tools.Auth method), 126
is_loopback_ip_address() (in module gluon.utils), 144
IS_LOWER (class in gluon.validators), 162
IS_MATCH (class in gluon.validators), 162
is_new() (gluon.globals.Session method), 30
IS_NOT_EMPTY (class in gluon.validators), 163
IS_NOT_IN_DB (class in gluon.validators), 164
IS_NULL_OR (in module gluon.validators), 164
IS_SLUG (class in gluon.validators), 164
IS_STRONG (class in gluon.validators), 165
IS_TIME (class in gluon.validators), 166
is_tracking_changes() (in module gluon.custom_import), 17
IS_UPLOAD_FILENAME (class in gluon.validators), 166
IS_UPPER (class in gluon.validators), 167
IS_URL (class in gluon.validators), 167
is_valid_ip_address() (in module gluon.utils), 144
isalnum() (gluon.utf8.Utf8 method), 140
isalpha() (gluon.utf8.Utf8 method), 140
isdigit() (gluon.utf8.Utf8 method), 140
islower() (gluon.utf8.Utf8 method), 140
isodatetime (gluon.validators.IS_DATETIME attribute), 149
isspace() (gluon.utf8.Utf8 method), 140
istitle() (gluon.utf8.Utf8 method), 140
isupper() (gluon.utf8.Utf8 method), 140

J

JobGraph (class in gluon.scheduler), 77
join() (gluon.utf8.Utf8 method), 140
json() (gluon.globals.Response method), 28
json() (gluon.tools.Service method), 132
json() (in module gluon.serializers), 83
jsonrpc() (gluon.tools.Service method), 132
jsonrpc2() (gluon.tools.Service method), 132
jsonrpc_errors (gluon.tools.Service attribute), 133
JSONWidget (class in gluon.sqlhtml), 99
jwt() (gluon.tools.Auth method), 126

K

keys() (gluon.tools.PluginManager method), 137
kill() (gluon.scheduler.Scheduler method), 79

L

LABEL (class in gluon.html), 44
label (gluon.sqlhtml.ExportClass attribute), 98
label (gluon.sqlhtml.ExporterCSV attribute), 98
label (gluon.sqlhtml.ExporterCSV_hidden attribute), 98
label (gluon.sqlhtml.ExporterHTML attribute), 98
label (gluon.sqlhtml.ExporterJSON attribute), 98
label (gluon.sqlhtml.ExporterTSV attribute), 99
label (gluon.sqlhtml.ExporterXML attribute), 99
launch() (gluon.newcron.hardcron method), 64
Lazy (gluon.sql.DAL.Field attribute), 91
Lazy (gluon.sql.Field attribute), 95
lazy_cache() (in module gluon.cache), 8
lazy_define_table() (gluon.sql.DAL method), 94
LEGEND (class in gluon.html), 44
LI (class in gluon.html), 44
LINK (class in gluon.html), 44
List (class in gluon.storage), 107
listdir() (in module gluon.fileutils), 25
ListWidget (class in gluon.sqlhtml), 99
ljust() (gluon.utf8.Utf8 method), 140
load() (gluon.restricted.RestrictedError method), 69
load() (gluon.restricted.TicketStorage method), 69
LOAD() (in module gluon.compileapp), 11
load() (in module gluon.rewrite), 72
load_routers() (in module gluon.rewrite), 73
load_storage() (in module gluon.storage), 109
LoadFactory (class in gluon.compileapp), 12
loads_json() (in module gluon.serializers), 83
loads_yaml() (in module gluon.serializers), 83
local_import_aux() (in module gluon.compileapp), 12
localhost (gluon.validators.IS_IPV4 attribute), 158
lock() (in module gluon.portalocker), 66
LockedFile (class in gluon.portalocker), 65
log() (gluon.restricted.RestrictedError method), 69
log_event() (gluon.tools.Auth method), 127
log_event() (gluon.tools.Crud method), 130
log_rewrite() (in module gluon.rewrite), 73
logger (gluon.sql.DAL attribute), 94
login() (gluon.tools.Auth method), 127
login_bare() (gluon.tools.Auth method), 127
login_user() (gluon.tools.Auth method), 127
logout() (gluon.tools.Auth method), 127
logout_bare() (gluon.tools.Auth method), 127
loop() (gluon.scheduler.MetaScheduler method), 78
loop() (gluon.scheduler.Scheduler method), 79
lower() (gluon.utf8.Utf8 method), 141
lstrip() (gluon.utf8.Utf8 method), 141

M

M() (gluon.languages.translator method), 53
Mail (class in gluon.tools), 117
Mail.Attachment (class in gluon.tools), 118
main() (in module gluon.scheduler), 82
make_fake_file_like_object() (in module gluon.fileutils), 26
manage_tokens() (gluon.tools.Auth method), 127
map_app() (gluon.rewrite.MapUrlIn method), 71
map_controller() (gluon.rewrite.MapUrlIn method), 71
map_function() (gluon.rewrite.MapUrlIn method), 71
map_language() (gluon.rewrite.MapUrlIn method), 71

map_prefix() (gluon.rewrite.MapUrlIn method), 71
map_root_static() (gluon.rewrite.MapUrlIn method), 72
map_static() (gluon.rewrite.MapUrlIn method), 72
map_url_in() (in module gluon.rewrite), 73
map_url_out() (in module gluon.rewrite), 73
MapUrlIn (class in gluon.rewrite), 71
MapUrlOut (class in gluon.rewrite), 72
MARKMIN (class in gluon.html), 44
markmin_base() (gluon.tools.Wiki method), 135
markmin_render() (gluon.tools.Wiki method), 135
maybe_add() (gluon.validators.IS_IN_DB method), 154
md5_hash() (in module gluon.utils), 144
media() (gluon.tools.Wiki method), 135
MENU (class in gluon.html), 44
menu() (gluon.tools.Wiki method), 135
message (gluon.http.HTTP attribute), 51
MessageBoxHandler (class in gluon.messageboxhandler), 59
Messages (class in gluon.storage), 108
META (class in gluon.html), 45
MetaScheduler (class in gluon.scheduler), 77
Method (gluon.sql.DAL.Field attribute), 91
Method (gluon.sql.Field attribute), 95
mktree() (in module gluon.fileutils), 25
model_cmp() (in module gluon.compileapp), 12
model_cmp_sep() (in module gluon.compileapp), 12
MultipleOptionsWidget (class in gluon.sqlhtml), 99
mybuiltin (class in gluon.compileapp), 12

N

navbar() (gluon.tools.Auth method), 127
nice() (gluon.validators.IS_DATETIME static method), 149
no_underscore() (gluon.html.BEAUTIFY static method), 36
Node (class in gluon.template), 114
NOESCAPE (class in gluon.template), 114
not_authorized() (gluon.tools.Auth method), 127
not_authorized() (gluon.tools.Wiki method), 135
NotifySendHandler (class in gluon.messageboxhandler), 59
now() (gluon.scheduler.Scheduler method), 79
numbers (gluon.validators.IS_IPV4 attribute), 158

O

OBJECT (class in gluon.html), 45
OL (class in gluon.html), 44
omit_acf() (gluon.rewrite.MapUrlOut method), 72
omit_lang() (gluon.rewrite.MapUrlOut method), 72
on() (gluon.sql.DAL.Table method), 92
open() (in module gluon.recfile), 67
OPTGROUP (class in gluon.html), 45
OPTION (class in gluon.html), 45
options() (gluon.validators.IS_IN_DB method), 154
options() (gluon.validators.IS_IN_SET method), 155
OptionsWidget (class in gluon.sqlhtml), 100
output() (gluon.template.BlockNode method), 113
output_aux() (in module gluon.template), 115

P

P (class in gluon.html), 45
PACKAGE_PATH_SUFFIX (gluon.custom_import.TrackImporter attribute), 17
pad() (in module gluon.utils), 144
pages() (gluon.tools.Wiki method), 136
params_substitution() (gluon.languages.translator method), 54
parse() (gluon.template.TemplateParser method), 115
parse_all_vars() (gluon.globals.Request method), 28
parse_as_rest() (gluon.sql.DAL method), 94
parse_get_vars() (gluon.globals.Request method), 28
parse_path_info() (in module gluon.shell), 87
parse_post_vars() (gluon.globals.Request method), 28
parse_template() (in module gluon.template), 115
parse_version() (in module gluon.fileutils), 25
parsecronline() (in module gluon.newcron), 64
partition() (gluon.utf8.Utf8 method), 141
PasswordWidget (class in gluon.sqlhtml), 100
pbkdf2_hex() (in module gluon.utils), 144
Pipe (class in gluon.debug), 21
plugin_install() (in module gluon.admin), 5
plugin_pack() (in module gluon.admin), 5
PluginManager (class in gluon.tools), 136
plural() (gluon.languages.translator method), 55
pop_arg_if() (gluon.rewrite.MapUrlIn method), 72
pop_task() (gluon.scheduler.MetaScheduler method), 78
pop_task() (gluon.scheduler.Scheduler method), 79
post_vars (gluon.globals.Request attribute), 28
PRE (class in gluon.html), 45
prettydate() (in module gluon.tools), 137
preview() (gluon.tools.Wiki method), 136
private (gluon.validators.IS_IPV4 attribute), 158
process() (gluon.html.FORM method), 41
profile() (gluon.tools.Auth method), 127

Q

queue_task() (gluon.scheduler.Scheduler method), 79
quit() (gluon.widget.web2pyDialog method), 172

R

r_multiline (gluon.template.TemplateParser attribute), 115
r_tag (gluon.template.TemplateParser attribute), 115
RadioWidget (class in gluon.sqlhtml), 100
random_password() (gluon.tools.Auth method), 127
rangetolist() (in module gluon.newcron), 64
re_block (gluon.template.TemplateParser attribute), 115

re_compile() (in module gluon.compileapp), 12
re_pass (gluon.template.TemplateParser attribute), 115
re_unblock (gluon.template.TemplateParser attribute), 115
read() (gluon.debug.Pipe method), 21
read() (gluon.portalocker.LockedFile method), 66
read() (gluon.tools.Crud method), 130
read() (gluon.tools.Wiki method), 136
read_file() (in module gluon.fileutils), 25
read_locked() (in module gluon.portalocker), 66
read_pyc() (in module gluon.compileapp), 12
readline() (gluon.debug.Pipe method), 21
readline() (gluon.portalocker.LockedFile method), 66
readlines() (gluon.portalocker.LockedFile method), 66
readlines_file() (in module gluon.fileutils), 25
Recaptcha (class in gluon.tools), 129
Recaptcha2 (class in gluon.tools), 129
recursive_unlink() (in module gluon.fileutils), 25
redirect() (in module gluon.http), 51
REDIRECT_JS (gluon.html.FORM attribute), 41
regex (gluon.validators.IS_EMAIL attribute), 151
regex (gluon.validators.IS_IPV4 attribute), 158
regex_attr (gluon.html.DIV attribute), 40
regex_class (gluon.html.DIV attribute), 40
REGEX_CLEANUP (gluon.validators.CLEANUP attribute), 146
regex_filter_in() (in module gluon.rewrite), 73
regex_filter_out() (in module gluon.rewrite), 73
regex_id (gluon.html.DIV attribute), 40
regex_proposed_but_failed (gluon.validators.IS_EMAIL attribute), 151
regex_select() (in module gluon.rewrite), 73
regex_tag (gluon.html.DIV attribute), 40
regex_uri() (in module gluon.rewrite), 73
regex_url_in() (in module gluon.rewrite), 73
register() (gluon.tools.Auth method), 127
register_bare() (gluon.tools.Auth method), 127
reindent() (gluon.template.TemplateParser method), 115
release() (gluon.newcron.Token method), 63
remove() (in module gluon.recfile), 67
remove_compiled_application() (in module gluon.compileapp), 12
render() (gluon.globals.Response method), 28
render() (in module gluon.template), 115
render_tags() (gluon.tools.Wiki method), 136
renew() (gluon.globals.Session method), 30
replace() (gluon.utf8.Utf8 method), 141
report_task() (gluon.scheduler.MetaScheduler method), 78
report_task() (gluon.scheduler.Scheduler method), 80
represent() (gluon.sql.DAL method), 94
represent() (gluon.sqlhtml.UploadWidget class method), 104
represent() (in module gluon.sqlhtml), 105
represented() (gluon.sqlhtml.ExportClass method), 98
representers (gluon.sql.DAL attribute), 94
Request (class in gluon.globals), 27
request_reset_password() (gluon.tools.Auth method), 127
requires() (gluon.tools.Auth method), 127
requires_https() (gluon.globals.Request method), 28
requires_login() (gluon.tools.Auth method), 127
requires_login_or_token() (gluon.tools.Auth method), 127
requires_membership() (gluon.tools.Auth method), 128
requires_permission() (gluon.tools.Auth method), 128
requires_signature() (gluon.tools.Auth method), 128
reset_password() (gluon.tools.Auth method), 128
reset_password_deprecated() (gluon.tools.Auth method), 128
Response (class in gluon.globals), 28
restful() (gluon.globals.Request method), 28
restricted() (in module gluon.restricted), 69
RestrictedError, 69
resume() (gluon.scheduler.Scheduler method), 80
retrieve() (gluon.sql.DAL.Field method), 91
retrieve() (gluon.sql.Field method), 96
retrieve_file_properties() (gluon.sql.DAL.Field method), 91
retrieve_file_properties() (gluon.sql.Field method), 96
retrieve_password() (gluon.tools.Auth method), 128
retrieve_username() (gluon.tools.Auth method), 128
reverse_geocode() (in module gluon.tools), 137
rfind() (gluon.utf8.Utf8 method), 141
rindex() (gluon.utf8.Utf8 method), 141
rjust() (gluon.utf8.Utf8 method), 141
rollback() (gluon.sql.DAL method), 94
rows() (gluon.tools.Crud method), 130
rows_page (gluon.tools.Wiki attribute), 136
rpartition() (gluon.utf8.Utf8 method), 141
rsplit() (gluon.utf8.Utf8 method), 141
rss() (gluon.tools.Service method), 133
rss() (in module gluon.serializers), 83
rstrip() (gluon.utf8.Utf8 method), 141
run() (gluon.debug.WebDebugger method), 22
run() (gluon.newcron.cronlauncher method), 63
run() (gluon.newcron.extcron method), 64
run() (gluon.newcron.hardcron method), 64
run() (gluon.newcron.softcron method), 64
run() (gluon.scheduler.MetaScheduler method), 78
run() (gluon.tools.Service method), 133
run() (in module gluon.shell), 88
run_controller_in() (in module gluon.compileapp), 13
run_login_onaccept() (gluon.tools.Auth method), 128
run_models_in() (in module gluon.compileapp), 13
run_system_tests() (in module gluon.widget), 171
run_view_in() (in module gluon.compileapp), 13

S

safe_encode() (in module gluon.serializers), 83
safe_float() (in module gluon.sqlhtml), 105
safe_int() (in module gluon.sqlhtml), 105
sanitize() (in module gluon.sanitizer), 75
save_password() (in module gluon.main), 58
save_pyc() (in module gluon.compileapp), 13
save_session_id_cookie() (gluon.globals.Session method), 31
save_storage() (in module gluon.storage), 109
Scheduler (class in gluon.scheduler), 78
SCRIPT (class in gluon.html), 45
search() (gluon.tools.Crud method), 130
search() (gluon.tools.Wiki method), 136
search_menu() (gluon.sqlhtml.SQLFORM static method), 102
secure() (gluon.globals.Session method), 31
secure_dumps() (in module gluon.utils), 144
secure_loads() (in module gluon.utils), 144
SELECT (class in gluon.html), 45
select() (gluon.tools.Crud method), 131
select_host() (gluon.tools.Auth method), 128
send() (gluon.tools.Mail method), 119
send_heartbeat() (gluon.scheduler.MetaScheduler method), 78
send_heartbeat() (gluon.scheduler.Scheduler method), 80
serialize() (gluon.html.MENU method), 45
serialize_mobile() (gluon.html.MENU method), 45
serializers (gluon.sql.DAL attribute), 95
serve_amfrpc() (gluon.tools.Service method), 133
serve_csv() (gluon.tools.Service method), 134
serve_json() (gluon.tools.Service method), 134
serve_jsonrpc() (gluon.tools.Service method), 134
serve_jsonrpc2() (gluon.tools.Service method), 134
serve_rss() (gluon.tools.Service method), 134
serve_run() (gluon.tools.Service method), 134
serve_soap() (gluon.tools.Service method), 134
serve_xml() (gluon.tools.Service method), 134
serve_xmlrpc() (gluon.tools.Service method), 134
server_ready() (gluon.widget.web2pyDialog method), 172
Service (class in gluon.tools), 131
Service.JsonRpcException, 131
Session (class in gluon.globals), 29
set_attributes() (gluon.sql.DAL.Field method), 91
set_attributes() (gluon.sql.Field method), 96
set_current_languages() (gluon.languages.translator method), 55
set_folder() (gluon.sql.DAL static method), 95
set_requirements() (gluon.scheduler.Scheduler method), 80
set_self_id() (gluon.validators.IS_EMPTY_OR method), 152
set_self_id() (gluon.validators.IS_IN_DB method), 154
set_self_id() (gluon.validators.IS_NOT_IN_DB method), 164
set_trace() (in module gluon.debug), 22
set_worker_status() (gluon.scheduler.Scheduler method), 80
Settings (class in gluon.storage), 108
show_if() (in module gluon.sqlhtml), 105
sibling() (gluon.html.DIV method), 40
siblings() (gluon.html.DIV method), 40
simple_hash() (in module gluon.utils), 144
sleep() (gluon.scheduler.MetaScheduler method), 78
sleep() (gluon.scheduler.Scheduler method), 80
sluggify() (gluon.rewrite.MapUrlIn method), 72
sluggify() (in module gluon.rewrite), 73
smart_query() (gluon.sql.DAL method), 95
smartdictform() (gluon.sqlhtml.SQLFORM static method), 102
smartgrid() (gluon.sqlhtml.SQLFORM static method), 102
soap() (gluon.tools.Service method), 134
softcron (class in gluon.newcron), 64
SPAN (class in gluon.html), 46
split() (gluon.utf8.Utf8 method), 141
split_emails (gluon.validators.IS_LIST_OF_EMAILS attribute), 152
splitlines() (gluon.utf8.Utf8 method), 141
SQLFORM (class in gluon.sqlhtml), 100
sqlsafe (gluon.sql.DAL.Field attribute), 91
sqlsafe (gluon.sql.DAL.Table attribute), 92
sqlsafe (gluon.sql.Field attribute), 96
sqlsafe_alias (gluon.sql.DAL.Table attribute), 92
sqlsafe_name (gluon.sql.DAL.Field attribute), 91
sqlsafe_name (gluon.sql.Field attribute), 96
SQLTABLE (class in gluon.sqlhtml), 103
start() (gluon.main.HttpServer method), 58
start() (gluon.widget.web2pyDialog method), 172
start() (in module gluon.widget), 171
start_browser() (in module gluon.widget), 171
start_heartbeats() (gluon.scheduler.MetaScheduler method), 78
start_schedulers() (gluon.widget.web2pyDialog method), 172
start_schedulers() (in module gluon.widget), 171
startswith() (gluon.utf8.Utf8 method), 141
stop() (gluon.main.HttpServer method), 58
stop() (gluon.widget.web2pyDialog method), 172
stop_task() (gluon.scheduler.Scheduler method), 80
stop_trace() (in module gluon.debug), 22
stopcron() (in module gluon.newcron), 64
Storage (class in gluon.storage), 107
StorageList (class in gluon.storage), 109
store() (gluon.restricted.TicketStorage method), 69
store() (gluon.sql.DAL.Field method), 91
store() (gluon.sql.Field method), 96

stream() (gluon.globals.Response method), 28
stream_file_or_304_or_206() (in module gluon.streamer), 111
streamer() (in module gluon.streamer), 111
strict (gluon.html.HTML attribute), 43
strict (gluon.html.XHTML attribute), 49
StringWidget (class in gluon.sqlhtml), 103
strip() (gluon.utf8.Utf8 method), 141
STRONG (class in gluon.html), 46
STYLE (class in gluon.html), 46
style() (gluon.sqlhtml.SQLTABLE method), 103
SuperNode (class in gluon.template), 114
swapcase() (gluon.utf8.Utf8 method), 142

T

TABLE (class in gluon.html), 46
table_cas() (gluon.tools.Auth method), 128
table_event() (gluon.tools.Auth method), 128
table_group() (gluon.tools.Auth method), 128
table_membership() (gluon.tools.Auth method), 128
table_permission() (gluon.tools.Auth method), 128
table_token() (gluon.tools.Auth method), 128
table_user() (gluon.tools.Auth method), 128
tables (gluon.sql.DAL attribute), 95
tables() (gluon.tools.Crud method), 131
tag (gluon.html.A attribute), 35
tag (gluon.html.B attribute), 36
tag (gluon.html.BEAUTIFY attribute), 36
tag (gluon.html.BODY attribute), 36
tag (gluon.html.BR attribute), 36
tag (gluon.html.BUTTON attribute), 36
tag (gluon.html.CAT attribute), 36
tag (gluon.html.CENTER attribute), 36
tag (gluon.html.COL attribute), 37
tag (gluon.html.COLGROUP attribute), 37
tag (gluon.html.DIV attribute), 40
tag (gluon.html.EM attribute), 40
tag (gluon.html.EMBED attribute), 40
tag (gluon.html.FIELDSET attribute), 40
tag (gluon.html.FORM attribute), 41
tag (gluon.html.H1 attribute), 42
tag (gluon.html.H2 attribute), 42
tag (gluon.html.H3 attribute), 42
tag (gluon.html.H4 attribute), 42
tag (gluon.html.H5 attribute), 42
tag (gluon.html.H6 attribute), 42
tag (gluon.html.HEAD attribute), 42
tag (gluon.html.HR attribute), 42
tag (gluon.html.HTML attribute), 43
tag (gluon.html.I attribute), 43
tag (gluon.html.IFRAME attribute), 43
tag (gluon.html.IMG attribute), 43
tag (gluon.html.INPUT attribute), 44
tag (gluon.html.LABEL attribute), 44
tag (gluon.html.LEGEND attribute), 44
tag (gluon.html.LI attribute), 44
tag (gluon.html.LINK attribute), 44
tag (gluon.html.MENU attribute), 45
tag (gluon.html.META attribute), 45
tag (gluon.html.OBJECT attribute), 45
tag (gluon.html.OL attribute), 44
tag (gluon.html.OPTGROUP attribute), 45
tag (gluon.html.OPTION attribute), 45
tag (gluon.html.P attribute), 45
tag (gluon.html.PRE attribute), 45
tag (gluon.html.SCRIPT attribute), 45
tag (gluon.html.SELECT attribute), 46
tag (gluon.html.SPAN attribute), 46
tag (gluon.html.STRONG attribute), 46
tag (gluon.html.STYLE attribute), 46
tag (gluon.html.TABLE attribute), 46
tag (gluon.html.TBODY attribute), 47
tag (gluon.html.TD attribute), 46
tag (gluon.html.TEXTAREA attribute), 46
tag (gluon.html.TFOOT attribute), 47
tag (gluon.html.TH attribute), 47
tag (gluon.html.THEAD attribute), 47
tag (gluon.html.TITLE attribute), 47
tag (gluon.html.TR attribute), 47
tag (gluon.html.TT attribute), 47
tag (gluon.html.UL attribute), 44
tag (gluon.html.XHTML attribute), 49
TAG (in module gluon.html), 46
tar() (in module gluon.fileutils), 26
tar_compiled() (in module gluon.fileutils), 26
Task (class in gluon.scheduler), 81
task_status() (gluon.scheduler.Scheduler method), 81
TaskReport (class in gluon.scheduler), 82
TBODY (class in gluon.html), 47
TD (class in gluon.html), 46
TemplateParser (class in gluon.template), 114
terminate() (gluon.scheduler.Scheduler method), 81
terminate_process() (gluon.scheduler.MetaScheduler method), 78
test() (in module gluon.compileapp), 13
test() (in module gluon.shell), 88
TEXTAREA (class in gluon.html), 46
TextWidget (class in gluon.sqlhtml), 103
TFOOT (class in gluon.html), 47
TH (class in gluon.html), 46
THEAD (class in gluon.html), 47
THREAD_LOCAL (gluon.custom_import.TrackImporter attribute), 17
TicketStorage (class in gluon.restricted), 69
TimeWidget (class in gluon.sqlhtml), 104
TITLE (class in gluon.html), 47
title() (gluon.utf8.Utf8 method), 142
to() (gluon.http.HTTP method), 51

to_string() (gluon.template.TemplateParser method), 115
Token (class in gluon.newcron), 63
toolbar() (gluon.globals.Response method), 29
total_seconds() (gluon.scheduler.Scheduler static method), 81
TR (class in gluon.html), 47
track_changes() (in module gluon.custom_import), 17
TrackImporter (class in gluon.custom_import), 17
transitional (gluon.html.HTML attribute), 43
transitional (gluon.html.XHTML attribute), 49
translate() (gluon.languages.translator method), 55
translate() (gluon.utf8.Utf8 method), 142
translator (class in gluon.languages), 53
truncate() (gluon.sql.DAL.Table method), 92
try_redirect_on_error() (in module gluon.rewrite), 73
try_rewrite_on_error() (in module gluon.rewrite), 73
try_start_scheduler() (gluon.widget.web2pyDialog method), 172
try_stop_scheduler() (gluon.widget.web2pyDialog method), 172
TT (class in gluon.html), 47
TYPE (class in gluon.scheduler), 81

U

UL (class in gluon.html), 44
unlock() (in module gluon.portalocker), 66
untar() (in module gluon.fileutils), 26
unzip() (in module gluon.admin), 5
up() (in module gluon.fileutils), 25
update() (gluon.html.DIV method), 40
update() (gluon.sql.DAL.Table method), 92
update() (gluon.tools.Crud method), 131
update() (gluon.widget.web2pyDialog method), 172
update_all_languages() (in module gluon.languages), 55
update_canvas() (gluon.widget.web2pyDialog method), 172
update_dependencies() (gluon.scheduler.Scheduler method), 81
update_groups() (gluon.tools.Auth method), 128
update_or_insert() (gluon.sql.DAL.Table method), 92
update_request() (gluon.rewrite.MapUrlIn method), 72
update_schedulers() (gluon.widget.web2pyDialog method), 172
upgrade() (in module gluon.admin), 5
UploadWidget (class in gluon.sqlhtml), 104
upper() (gluon.utf8.Utf8 method), 142
url() (gluon.tools.Auth method), 128
url() (gluon.tools.Crud method), 131
URL() (in module gluon.html), 47
url_in() (in module gluon.rewrite), 73
url_out() (in module gluon.rewrite), 73
urlify() (gluon.validators.IS_SLUG static method), 165
user_agent() (gluon.globals.Request method), 28
user_group() (gluon.tools.Auth method), 128
user_group_role() (gluon.tools.Auth method), 128
user_id (gluon.tools.Auth attribute), 128
Utf8 (class in gluon.utf8), 139
uuid (gluon.globals.Request attribute), 28
uuid() (gluon.sql.DAL method), 95

V

validate() (gluon.html.FORM method), 41
validate() (gluon.scheduler.JobGraph method), 77
validate() (gluon.sql.DAL.Field method), 91
validate() (gluon.sql.Field method), 96
validate_and_insert() (gluon.sql.DAL.Table method), 92
validate_and_update() (gluon.sql.DAL.Table method), 92
validate_and_update_or_insert() (gluon.sql.DAL.Table method), 92
validate_args() (gluon.rewrite.MapUrlIn method), 72
validators (gluon.sql.DAL attribute), 95
validators_method() (gluon.sql.DAL method), 95
vars (gluon.globals.Request attribute), 28
verify_email() (gluon.tools.Auth method), 128
VERIFY_SERVER (gluon.tools.Recaptcha attribute), 129
VERIFY_SERVER (gluon.tools.Recaptcha2 attribute), 130
Virtual (gluon.sql.DAL.Field attribute), 91
Virtual (gluon.sql.Field attribute), 96

W

w2p_pack() (in module gluon.fileutils), 26
w2p_pack_plugin() (in module gluon.fileutils), 26
w2p_unpack() (in module gluon.fileutils), 26
w2p_unpack_plugin() (in module gluon.fileutils), 26
web2py_uuid() (in module gluon.utils), 144
web2pyDialog (class in gluon.widget), 171
WebDebugger (class in gluon.debug), 21
when_is_logged_in_bypass_next_in_url() (gluon.tools.Auth method), 128
where() (gluon.sql.DAL method), 95
widget() (gluon.sqlhtml.BooleanWidget class method), 97
widget() (gluon.sqlhtml.CheckboxesWidget class method), 97
widget() (gluon.sqlhtml.FormWidget class method), 99
widget() (gluon.sqlhtml.JSONWidget class method), 99
widget() (gluon.sqlhtml.ListWidget class method), 99
widget() (gluon.sqlhtml.MultipleOptionsWidget class method), 100
widget() (gluon.sqlhtml.OptionsWidget class method), 100
widget() (gluon.sqlhtml.PasswordWidget class method), 100
widget() (gluon.sqlhtml.RadioWidget class method), 100
widget() (gluon.sqlhtml.StringWidget class method), 103
widget() (gluon.sqlhtml.TextWidget class method), 104

widget() (gluon.sqlhtml.UploadWidget class method), 104
widgets (gluon.sqlhtml.SQLFORM attribute), 103
Wiki (class in gluon.tools), 135
wiki() (gluon.tools.Auth method), 129
wikimenu() (gluon.tools.Auth method), 129
with_alias() (gluon.sql.DAL.Table method), 92
with_prefix() (gluon.cache.Cache static method), 8
wrapped_assign_tasks() (gluon.scheduler.Scheduler method), 81
wrapped_pop_task() (gluon.scheduler.Scheduler method), 81
wrapped_report_task() (gluon.scheduler.Scheduler method), 81
write() (gluon.debug.Pipe method), 21
write() (gluon.globals.Response method), 29
write() (gluon.portalocker.LockedFile method), 66
write() (gluon.template.DummyResponse method), 114
write() (gluon.widget.IO method), 171
write_file() (in module gluon.fileutils), 25
write_locked() (in module gluon.portalocker), 66
wsgibase() (in module gluon.main), 57

X

XHTML (class in gluon.html), 49
XML (class in gluon.html), 49
xml() (gluon.html.A method), 35
xml() (gluon.html.CODE method), 37
xml() (gluon.html.DIV method), 40
xml() (gluon.html.FORM method), 42
xml() (gluon.html.HTML method), 43
xml() (gluon.html.INPUT method), 44
xml() (gluon.html.MARKMIN method), 44
xml() (gluon.html.MENU method), 45
xml() (gluon.html.P method), 45
xml() (gluon.html.SCRIPT method), 45
xml() (gluon.html.STYLE method), 46
xml() (gluon.html.XHTML method), 49
xml() (gluon.html.XML method), 50
xml() (gluon.template.NOESCAPE method), 114
xml() (gluon.tools.Recaptcha method), 129
xml() (gluon.tools.Recaptcha2 method), 130
xml() (gluon.tools.Service method), 134
xml() (in module gluon.serializers), 83
xml_rec() (in module gluon.serializers), 83
xmlescape() (in module gluon.html), 50
xmlns (gluon.html.XHTML attribute), 49
xmlrpc() (gluon.globals.Response method), 29
xmlrpc() (gluon.tools.Service method), 134

Y

yaml() (in module gluon.serializers), 83

Z

zfill() (gluon.utf8.Utf8 method), 142