PyWebTools Documentation

Release 1.1.2

Mark Hall

January 30, 2017

Contents

The PyWebTools package provides a number of helpers for use with Pyramid, Kajiki, Formencode, and SQLAlchemy packages.

API Documentation:

CHAPTER 1

API Documentation

1.1 pywebtools – Utilities for Kajiki, Pyramid, Formencode, SQLAlchemy

The *pywebtools* module provides helpers and utilities for use with *kajiki*, *pyramid*, *formencode*, and *sqlalchemy*.

1.2 pywebtools.formencode – Utilities for use with Formencode

This module provides a number of `Schema` and `FancyValidator`.

Of particular note is the *CSRFSchema* which can be used as a base-class instead of `Schema` to automatically include CSRF validation.

class pywebtools.formencode.**CSRFSchema**(**args*, ***kw*)

The class:*~pywebtools.formencode.CSRFSchema* is a base formencode.Schema that includes Cross-Site Request Forgery detection.

It should be used as the base class for all specific request schemas.

Messages

badDictType: The input must be dict-like (not a %(type)s: %(value)r)

badType: The input must be a string (not a %(type)s: %(value)r)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field %(name)s was not expected.

singleValueExpected: Please provide only one value

class pywebtools.formencode.**CSRFValidator**(**args*, ***kw*)

Validator that checks a value against the Cross-Site Request Forgery token stored in the user's session.

Messages

badType: The input must be a string (not a %(type)s: %(value)r)

empty: An empty CSRF token was provided. This might indicate a malicious attack.

invalid_csrf_token: The CSRF token is invalid. This might indicate a malicious attack.

missing: No CSRF token was provided. This might indicate a malicious attack.

no_request: No request was specified to validate the CSRFToken

noneType: The input must be a string (not None)

class pywebtools.formencode.**DateValidator**(**args*, ***kw*)

The *DateValidator* provides date validation and conversion from the formats "YYYY-MM-DD" or "DD/MM/YYYY" to a python `datetime.date`.

Messages

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

invalid_format: Please enter a date either as YYYY-MM-DD or DD/MM/YYYY

noneType: The input must be a string (not None)

class pywebtools.formencode.**DictValidator**(**args*, ***kw*)

The *DictValidator* validates that the value is a `dict`.

Messages

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

noneType: The input must be a string (not None)

not_dict: The value is not a dict

class pywebtools.formencode.**DynamicSchema**(*fields=None*, ***kwargs*)

The *DynamicSchema* provides a dynamic `Schema` for which the validation fields are defined from the `list` of (field-name, `FancyValidator`) pairs passed to the constructor.

Messages

badDictType: The input must be dict-like (not a `%(type)s`: `%(value)r`)

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field `%(name)s` was not expected.

singleValueExpected: Please provide only one value

class pywebtools.formencode.**EmailDomainValidator**(**args*, ***kw*)

The *EmailDomainValidator* checks that the given e-mail address is in the list of allowed e-mail address domains.

Requires that the list of allowed domains is available via the `state.email_domains` attribute. If nothing is provided in the `state`, then all e-mail addresses are seen as valid.

Messages

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

noneType: The input must be a string (not None)

wrongdomain: Only e-mail address in the following domains can be used: `%(domains)s`

class `pywebtools.formencode.`**PasswordValidator**(**args*, ***kw*)

The *PasswordValidator* handles the checking of user-provided passwords against the database to allow / dissallow login.

Requires a SQLAlchemy database session to be available via `state.dbsession`.

Messages

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

nologin: No user exists with the given e-mail address or the password does not match

noneType: The input must be a string (not None)

class `pywebtools.formencode.`**State**(***kwargs*)

The *State* provides a blank state object for use with Formencode validation. Any parameters passed to the constructor are automatically set as attributes of the `State`.

class `pywebtools.formencode.`**TimeValidator**(**args*, ***kw*)

The *DateValidator* provides time validation and conversion from the format "HH:MM" to a python `datetime.time`.

Messages

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

invalid_format: Please enter a time as HH:MM

noneType: The input must be a string (not None)

class `pywebtools.formencode.`**UniqueEmailValidator**(**args*, ***kw*)

The *UniqueEmailValidator* checks that the given e-mail address is not already used.

Requires a SQLAlchemy database session to be available via `state.dbsession` and the user class to be available via `state.user_class`. If a `state.userid` is provided, then the `user_class` with that `id` can have the same e-mail address.

Messages

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

existing: A user with this e-mail address already exists

noneType: The input must be a string (not None)

1.3 `pywebtools.kajiki` – Utilities for use with Kajiki

This module provides two function definition templates "form.kajiki" and "menu.kajiki".

1.3.1 form.kajiki

Helper functions for creating HTML input elements.

pywebtools.kajiki.**input**(*field_type*, *field_name*, ***kwargs*)

Generates a HTML input field with the given `field_type` and `field_name`. The `kwargs` contains additional parameters:

```
* value: The default value or the value of a checkbox/radio/select input
* values: List of (value, label) pairs for use with a select input
* extra_attrs: Dictionary of additional HTML element attribute key/value pairs
```

Additionally if a `values` dictionary is in the current view, then this will be used to set any input values. This can be used to pre-fill the form with existing data.

Parameters

- **field_type** (`str`) – The HTML field type to generate
- **field_name** (`str`) – The name of the HTML field
- **kwargs** – Additional keyword arguments as described above

pywebtools.kajiki.**field**(*field_type*, *field_name*, *field_label*, ***kwargs*)

Generates a HTML input field with wrapping <label> using the `field_label` value. Sets Foundation / Abide CSS classes if there is an `errors` dictionary in the current view and the `field_name` occurs in that dictionary. Will also output the matching error message.

Parameters

- **field_type** (`str`) – The HTML field type to generate
- **field_name** (`str`) – The name of the HTML field
- **field_name** – The displayed label of the HTML field
- **kwargs** – Additional keyword arguments as described in *input()*

pywebtools.kajiki.**csrf_field**()

Generates a hidden input field with the name "csrf_token" and the value set to the current session's CSRF token. Integrates with the *pywebtools.formencode.CSRFSchema*.

1.3.2 menu.kajiki

Helper function for creating an icon-based menu-bar.

pywebtools.kajiki.**menubar**(*groups*, *class_=None*, *alignment='left'*)

Generates the HTML structure for a Foundation dropdown menu using icons for the top level and text for the drop-down items.

Params groups The list of groups generated by the *MenuBuilder*

Parameters

- **class** (`str`) – An optional CSS class to set on the menu's <ul> element
- **alignment** (`str`) – The alignment of the drop-down items. Defaults to "left", can be set to "right"

1.4 pywebtools.pyramid – Utilities for Pyramid

The *pywebtools.pyramid* module provides helpers and utilities for use with Pyramid. General utilities can be found in *pywebtools.pyramid.util* and a full authentication base-layer in *pywebtools.pyramid.auth*.

1.5 pywebtools.pyramid.auth – Authentication Framework for Pyramid + SQLAlchemy

The *pywebtools.pyramid.auth* module provides an authentication framework for use with Pyramid and SQLAlchemy. It provides a *User* model plus all the views needed to handle registration, login, logout, and password retrieval. The workflow that it implements is registration -> confirmation via e-mail -> password reset -> login -> forgotten password -> password reset.

The framework is configurable via the parameters passed to the *init()* function.

pywebtools.pyramid.auth.**init**(*config*, *renderers=None*, *urls=None*, *redirects=None*, *callbacks=None*)

Initialises the authentication framework.

Routes that can have renderers attached via `renderers`, redirects via ``redirects`, and can have their URLs changed via `urls`:

- user.login - *login()*
- user.logout - *logout()*
- user.register - *register()*
- user.confirm - *confirm()* (no redirection)
- user.forgotten_password - *forgotten_password()*
- user.reset_password - *reset_password()*

If no renderer is provided for a route, then the route will not be registered.

The following callbacks can be registered via `callbacks`:

- user.created - called from *register()* and *forgotten_password()*
- user.validated - called from *confirm()*
- user.password_reset - called from *forgotten_password()*
- user.password_reset_failed - called from *forgotten_password()*
- user.password_reset_complete - called from *reset_password()*

Parameters

- **config** (`Configurator`) – The Pyramid configuration object to use to set up the configuration
- **renderers** (`dict`) – Renderers to attach to the available views
- **urls** (`dict`) – Alternative URLs to use
- **redirects** (`dict`) – Redirect routes that are used after any request has been completed
- **callbacks** (`dict`) – Callbacks that are called for certain actions on the user

1.6 pywebtools.pyramid.decorators – Decoratorators

The *decorators* module contains function decorators for use with the authentication framework.

pywebtools.pyramid.auth.decorators.**current_user**()
Inserts the currently logged in *User* into the *request* parameter under the attribute current_user. If there is no logged in user, then an anonymous *User* is created.

Used in view functions.

pywebtools.pyramid.auth.decorators.**require_logged_in**()
Checks that the current user is logged in, otherwise redirects to the login page. Requires that the *current_user()* decorator is run first.

pywebtools.pyramid.auth.decorators.**require_permission**(*permission=None*, *class_=None*, *request_key=None*, *action=None*)
Checks whether the current user has the given permission. Supports two modes:

If you provide the permission parameter and it will use *has_permission()* to check whether the current user has the given permission. If not, it raises HTTPUnauthorised.

Alternatively if you provide class_, request_key, and action parameters it will run a SQLAlchemy query for the class_, filtering class_.id == request.matchdict[request_key]. If that returns a result, then it will use the class_'s allow to check whether the current user is allowed to perform the given action. If not it raises HTTPUnauthorised. If no result is returned then it will raise HTTPNotFound.

Parameters

- **permission** (str) – The permission to check the user for
- **class** (class) – The SQLAlchemy ORM class to use for finding the instance that matches the request_key value
- **request_key** (str) – The key to use for getting a unique identifier from the request.matchdict to use in finding an instance of class_
- **action** (str) – The action to check for with the instance of class_

Returns The decorated function's return value

pywebtools.pyramid.auth.decorators.**unauthorised_redirect**(*request*, *redirect_to=None*, *message=None*)
Provides standardised handling of "unauthorised" redirection. Depending on whether the user is currently logged in, it will set the appropriate error message into the session flash and redirect to the appropriate page. If the user is logged in, it will redirect to the root page or to the redirect_to URL if specified. If the user is not logged in, it will always redirect to the login page.

Parameters

- **request** – The pyramid request
- **redirect_to** (*unicode*) – The URL to redirect to, if the user is currently logged in.
- **message** (unicode) – The message to show to the user

1.7 pywebtools.pyramid.auth.models – Authentication SQLAlchemy Models

This module contains all the SQLAlchemy models needed to implement the persistence level of the authentication framework. The main classes of interest are the *User* and *TimeToken*.

class pywebtools.pyramid.auth.models.**Permission**(***kwargs*)

The *Permission* class represents a single permission that can be granted to a *User* or to a *PermissionGroup*.

Instances of *Permission* have the following attributes:

- id – The unique database identifier
- name – The unique name used for permission checking
- permission_groups – List of *PermissionGroup* that contain this *Permission*
- title – The title displayed for this *Permission*
- users – List of *User* that have this *Permission*

class pywebtools.pyramid.auth.models.**PermissionGroup**(***kwargs*)

The *PermissionGroup* groups together one or more *Permission* for easier administration.

Instances of *PermissionGroup* have the following attributes:

- id – The unique database identifier
- permissions – The list of grouped *Permission*
- title – The title displayed for this *PermissionGroup*
- users – List of *User* that have this *PermissionGroup*

class pywebtools.pyramid.auth.models.**TimeToken**(*user_id*, *action*, *timeout*, *data=None*)

The *TimeToken* represents a validation token that has a timeout period.

Instances of the TimeTokne have the following attributes:

- id – The unique database identifier
- user_id – The identifier of the User ` it belongs to
- action – The action the *TimeToken* is associated with
- token – The random token
- timeout – The timeout timestamp until which the *TimeToken* is valid
- data – Any payload data

class pywebtools.pyramid.auth.models.**User**(***kwargs*)

The *User* represents a generic user. Which functionality they can access is determined purely through the individual *User*‘s *Permission*.

Instances of the *User* have the following attributes:

- id – The unique database identifier
- display_name – The name to display
- email – The e-mail address used for login and communication
- login_limit – Login limitation counter to stop brute-force login attacks
- parts – The UserPartProgress belonging to this User

- `password` – The hashed password
- `permissions` – The *User*'s list of *Permission*.
- `permission_groups` – The *User*'s list of *PermissionGroup*.
- `salt` – The password hash salt

admin_menu(*request*)
Generates the menu bar for the users administration list.

allow(*action*, *user*)
Checks whether the given `user` is allowed to perform the given `action`. Supports the following actions: view, edit, delete.

Parameters

- **action** (*unicode*) – The action to check for
- **user** (*User*) – The user to check

Returns `True` if the `user` may perform the action, `False` otherwise

Return type *bool*

has_option(*key*)
Check if the *User* has the given option.

Parameters **key** (`unicode`) – Option key to check.

Returns Whether the option exists or not

Return type `boolean`

has_permission(*permission*)
Checks whether the user has been granted the given `permission`, either directly or via a *PermissionGroup*.

Parameters **permission** (*unicode*) – The permission to check for

Returns `True` if the user has the permission, `False` otherwise

Return type *bool*

new_password(*password*)
Sets the given `password` as the *User*'s new password. Calls *new_salt()* to generate a new salt for the password.

Parameters **password** (*unicode*) – The new cleartext password

new_salt()
Generates a new `salt`. Will use `os.urandom` if available and the standard pseudo-random if not.

option(*key*)
Get the *User* options with the `key`.

Parameters **key** (`unicode`) – Option key to fetch.

Returns Returns the option value or `None`

password_matches(*password*)
Checks whether the given password matches the hashed, stored password.

Parameters **password** (*unicode*) – The password to check

Returns `True` if the passwords match, `False` otherwise

Return type *bool*

pywebtools.pyramid.auth.models.**groups_permissions = Table('permission_groups_permissions', MetaData(bind**
sqlalchemy.Table to link *PermissionGroup* and *Permission*.

pywebtools.pyramid.auth.models.**init_auth_permissions**(*dbsession*)
Creates the "User Administration" *PermissionGroup* and the four *Permission* "admin.users.view", "admin.users.edit", "admin.users.delete", and "admin.users.permission" needed for the user management views to work.

Parameters **dbsession** (scoped_session()) – The database session to add the new objects to

Returns The group with the new permissions

Return type *PermissionGroup*

pywebtools.pyramid.auth.models.**users_groups = Table('users_permission_groups', MetaData(bind=None), Colun**
sqlalchemy.Table to link *User* and *PermissionGroup*.

pywebtools.pyramid.auth.models.**users_permissions = Table('users_permissions', MetaData(bind=None), Colun**
sqlalchemy.Table to link *User* and *Permission*.

1.8 pywebtools.pyramid.auth.views – Authentication Framework Views

Views that implement the backend for the authentication framework. The current_user() provides a decorator that automatically adds the currently logged in user (or an anonymous not-logged in user) into the current request.

class pywebtools.pyramid.auth.views.**ActionSchema**(**args*, ***kw*)

The ActionSchema handles the validation of user action requests.

Messages

badDictType: The input must be dict-like (not a %(type)s: %(value)r)

badType: The input must be a string (not a %(type)s: %(value)r)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field %(name)s was not expected.

singleValueExpected: Please provide only one value

action = None
The action to apply

confirm = None
Whether the user has confirmed the action

q = None
Optional query parameter for the redirect

start = None
Optional start parameter for the redirect

status = None
Optional status parameter for the redirect

user_id = None
User ids to apply the action to

class pywebtools.pyramid.auth.views.**EditSchema**(**args*, ***kw*)

The class:~*wte.views.user.EditSchema* handles the validation of changes to the `User`.

Messages

badDictType: The input must be dict-like (not a `%(type)s`: `%(value)r`)

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field `%(name)s` was not expected.

singleValueExpected: Please provide only one value

display_name = None
Updated name

email = None
Updated e-mail address

password = None
Updated password

class pywebtools.pyramid.auth.views.**ForgottenPasswordSchema**(**args*, ***kw*)

The *ForgottenPasswordSchema* handles the validation of forgotten password requests.

Messages

badDictType: The input must be dict-like (not a `%(type)s`: `%(value)r`)

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field `%(name)s` was not expected.

singleValueExpected: Please provide only one value

email = None
E-mail to request a new password or validation token for

return_to = None
URL to redirect to after a successful password request

class pywebtools.pyramid.auth.views.**LoginSchema**(**args*, ***kw*)

The *LoginSchema* handles the validation of a login request.

Messages

badDictType: The input must be dict-like (not a `%(type)s`: `%(value)r`)

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field %(name)s was not expected.

singleValueExpected: Please provide only one value

email = None
E-mail address to log in with

password = None
Password to log in with

return_to = None
URL to redirect to after a successful login (optional)

class pywebtools.pyramid.auth.views.**RegisterSchema**(**args*, ***kw*)

The RegisterSchema handles the validation of registration requests. s

Messages

badDictType: The input must be dict-like (not a %(type)s: %(value)r)

badType: The input must be a string (not a %(type)s: %(value)r)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field %(name)s was not expected.

singleValueExpected: Please provide only one value

email = None
E-mail address to register with

email_confirm = None
Confirmation of the registration e-mail address

name = None
Name of the registering user

return_to = None
URL to redirect to after a successful registration (optional)

class pywebtools.pyramid.auth.views.**ResetPasswordSchema**(**args*, ***kw*)

The ResetPasswordSchema handles the validation of password reset requests.

Messages

badDictType: The input must be dict-like (not a %(type)s: %(value)r)

badType: The input must be a string (not a %(type)s: %(value)r)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field %(name)s was not expected.

singleValueExpected: Please provide only one value

password = None
New password

password_confirm = None
Updated password

pywebtools.pyramid.auth.views.**action**(*request*)
Handles the /users/action URL, applying the given action to the list of selected users. Requires that the current User has the "admin.users.view" Permission.

pywebtools.pyramid.auth.views.**confirm**(*request*)
Handles the "users.confirm" URL, validating that the user with the {token} has access to the e-mail address they provided.

On a successful confirmation, calls the "user.validated" callback with three parameters: the current request object, the new *User*, and the validation *TimeToken*.

If overriding the URL, the URL must only have a {token} parameter.

pywebtools.pyramid.auth.views.**create_user_crumbs**(*request*, *crumbs*)
Creates the base-list of breadcrumbs, depending on the current users authorisation level.

pywebtools.pyramid.auth.views.**delete**(*request*)
Handles the "/users/{uid}/delete" URL, providing the form and backend functionality for deleting a User. Also deletes all the data that is linked to that User.

pywebtools.pyramid.auth.views.**edit**(*request*)
Handles the "/users/{uid}/edit" URL, providing the form and backend functionality to update the user's profile.

pywebtools.pyramid.auth.views.**forgotten_password**(*request*)
Handles the "user.forgotten_password" URL, showing the form where the user can provide their e-mail address.

If the e-mail address provided does not match any known e-mail address, calls the "user.password_reset_failed" callback with the current request. If the e-mail address is known and the *User* status is "unconfirmed", calls the "user.created" callback with the current request, the *User*, and a new *TimeToken*. If the *User* status is "active", calls the "user.password_reset" callback with the current request, the *User*, and a new *TimeToken*.

Uses either the return_to parameter in the request to redirect on success or the "user.forgotten_password" redirection route.

pywebtools.pyramid.auth.views.**login**(*request*)
Handles the "user.login" URL, checking the submitted username and password against the stored *User* and setting the necessary session variables if the login is successful.

Uses either the return_to parameter in the request to redirect on success or the "user.login" redirection route, with parameter replacement "{uid}" will be replaced with the logged in user's identifier.

pywebtools.pyramid.auth.views.**logout**(*request*)
Handles the "user.logout" URL and deletes the current session, thus logging the user out.

Redirects to the "user.logout" redirection route.

pywebtools.pyramid.auth.views.**permissions**(*request*)
Handles the "/users/{uid}/permissions" URL, providing the form and backend functionality for setting the Permission and PermissionGroup that the User belongs to.

pywebtools.pyramid.auth.views.**redirect**(*request*, *redirect_id*, ***kwargs*)
Handles post-action redirects. If the redirection value is a string, uses that as the redirection route name. If it is a dictionary, then the value of the "route" key is used as the route name and the value of the "params" key is passed to the request.route_url function as keyword arguments. Uses *replace_kwargs()* to allow dynamic data.

Parameters

- **request** (Request) – The request to use for redirection
- **redirect_id** (str) – The identifier of the redirect to execute
- **kwargs** – Keyword arguments to use for replacements

pywebtools.pyramid.auth.views.**register**(*request*)
Handles the "user.register" URL, displaying the registration form or if data is POSTed, creating a new user.

On a successful registration, calls the "user.created" callback with three parameters: the current request object, the new *User*, and the validation *TimeToken*.

On a successful registration, redirects to the "user.register" redirection route.

pywebtools.pyramid.auth.views.**replace_kwargs**(*value*, *kwargs*)
Replace any keyword arguments in value with the value specified in kwargs. If the keyword argument does not exist in kwargs, replace with an empty string.

The function can handle both strings and dictionaries. In the case of dictionaries, both the keys and values are replaced.

Params value The value to replace

Parameters **kwargs** (dict) – The replacement values

Returns The value with all keyword arguments replaced

pywebtools.pyramid.auth.views.**reset_password**(*request*)
Handles the "user.forgotten_password" URL, showing the form where the user can provide their e-mail address.

If token is valid, calls the "user.password_reset_complete" callback with the current request and the *User*.

Uses either the return_to parameter in the request to redirect on success or the "user.login" redirection route, with parameter replacement "{uid}" will be replaced with the logged in user's identifier.

If overriding the URL, the URL must only have a {token} parameter.

pywebtools.pyramid.auth.views.**users**(*request*)
Handles the /users URL, displaying all users if the current *User* has the "admin.users.view" *Permission*.

pywebtools.pyramid.auth.views.**view**(*request*)
Handles the "/users/{uid}" URL, showing the user's profile.

1.9 pywebtools.pyramid.decorators – Decoratorators

The *decorators* module contains function decorators for use with views.

pywebtools.pyramid.decorators.**require_method**(*methods*)
Checks that the current request method is in the list of methods that are allowed for the given request.

Parameters **methods** (*list* of *unicode*) – The list of valid request methods

1.10 pywebtools.pyramid.util – Utility Functions for Pyramid

Utility functions for use with the Pyramid framework.

- *request_from_args()* is useful with decorators or any other function where you do not know which argument is the Pyramid Request.
- *get_config_setting()* provides easy access to configuration settings set in the [app:main] section of the INI file.

- `require_method()` is a decorator to enforce HTTP methods.
- *`MenuBuilder`* is a helper class to generate the menu structure used with *`menubar()`*.

class `pywebtools.pyramid.util.`**MenuBuilder**

The *`MenuBuilder`* helps with creating the `list` structure used for creating the icon-menubar used with *`menubar()`*. Call *`group()`* to start a new group of menu items. Call `item()` to add a menu item to the current group. *`generate()`* then generates the final structure for use in the menubar.

generate()

Generate the final menu structure.

Returns The list of menu groups with their menu items

Return_type `list` of menu groups

group(*label*, *icon=None*)

Add a new group to the list of groups in this *`MenuBuilder`*.

Parameters

- **label** (*unicode*) – The menu group's label
- **icon** (*unicode*) – The optional icon for this group. An icon must be provided in order to enable highlighting of menu items

menu(*label*, *href*, *icon=None*, *highlight=False*, *attrs=None*)

Add a new menu item to the current group. Will create a new group with an empty label if *`group()`* has not been called.

Parameters

- **label** (*unicode*) – The menu item's label
- **href** (*unicode*) – The URL that the menu item loads
- **icon** (*unicode*) – The optional icon for this menu item
- **highlight** (*boolean*) – Whether to highlight the menu item by displaying it at the top level
- **attrs** (`dict`) – Additional attributes to set for the menu item link

`pywebtools.pyramid.util.`**confirm_action**(*title*, *message*, *cancel*, *ok*)

Generates a confirmation JSON object for use with the jQuery.postLink() plugin.

Parameters

- **title** (`unicode`) – The title of the confirmation dialog box
- **message** (`unicode`) – The main message to show
- **cancel** (`dict` or `unicode`) – The cancel button's settings
- **ok** (`dict` or `unicode`) – The ok button's settings

Returns JSON object

Return type `unicode()`

`pywebtools.pyramid.util.`**confirm_delete**(*obj_type*, *title*, *has_parts=False*)

Generates the confirmation JSON object for use with the jQuery.postLink() plugin.

Parameters

- **obj_type** (`unicode`) – The type of object that is being deleted
- **title** (`unicode`) – The title of the object that is being delete

- **has_parts** (bool) – Whether to add the suffix " and all its parts"

Returns JSON object

Return type unicode()

pywebtools.pyramid.util.**convert_type**(*value*, *target_type*, *default=None*)

Attempts to convert the value to the given target_type. Will return default if the conversion fails.

Supported target_type values are:

•*int* – Convert to an integer value

•*boolean* – Convert to a boolean value (True if the value is the unicode string "true" in any capitalisation

•*list* – Convert to a list, splitting on line-breaks and commas

Parameters

- **value** (*unicode*) – The value to convert
- **target_type** (*unicode*) – The target type to convert to
- **default** – The default value if the conversion fails

Returns The converted value

pywebtools.pyramid.util.**get_config_setting**(*request*, *key*, *target_type=None*, *default=None*)

Gets a configuration setting from the application configuration. Settings are cached for faster access.

Parameters

- **request** (Request) – The request used to access the configuration settings
- **key** (*unicode*) – The configuration key
- **target_type** – If specified, will convert the configuration setting to the given type using *convert_type()*

Returns The configuration setting value or default

pywebtools.pyramid.util.**paginate**(*request*, *route_name*, *query*, *start*, *rows*, *query_params=None*)

Generates the list of pages for a query.

Parameters

- **request** (Request) – The request used to generate URLs
- **route_name** (str()) – The name of the route to use for URLs
- **query** (Query) – The SQLAlchemy query to generate the pagination for
- **start** (int()) – The current starting index
- **rows** (int()) – The number of rows per page
- **query_params** (list() of tuple()) – An optional list of query parameters to include in all URLs that are generated

Returns The list() of pages to use with the "navigation.pagination" helper

Return type list()

pywebtools.pyramid.util.**request_from_args**(**args*)

Returns the Request from the function parameters list args.

Parameters **args** – The parameters passed to a function

Returns The request object

R_type Request

1.11 pywebtools.sqlalchemy – Utilities for use with SQLAlchemy

This module provide the generic DBSession scoped_session for use with Pyramid & SQLAlchemy. It also provides the Base declarative_base() that acts as the base class for any declarative model. If you are starting from a SQLAlchemy + Pyramid starter template in your models module, remove the DBSession and Base and replace them imports from this module. Then do the same for the main function in the main package.

exception pywebtools.sqlalchemy.**DBUpgradeException**(*current*, *required*)
The *DBUpgradeException* is used to indicate that the database requires an upgrade before the Web Teaching Environment system can be used.

class pywebtools.sqlalchemy.**JSONUnicodeText**(**args*, ***kwargs*)
The class:*~pywebtools.sqlalchemy.JSONUnicodeText* is an extension to the UnicodeText column type that does automatic conversion from the JSON string representation stored in the DB to a dict/list representation for use in python.

impl
alias of UnicodeText

process_bind_param(*value*, *dialect*)
Convert the dict/list to JSON for storing.

process_result_value(*value*, *dialect*)
Convert the JSON to dict/list for use.

class pywebtools.sqlalchemy.**MutableDict**(**args*, ***kwargs*)
The *MutableDict* is a dict extension for use with the *JSONUnicodeText* column. It monitors any change to its values and marks the column as dirty, if a change has occurred.

It is smart about its internal structure and will convert any nested dict into *MutableDict* as well, to ensure that all changes are tracked.

classmethod coerce(*key*, *value*)
Automatically coerce any dict to a *MutableDict*. Used by SQLAlchemy.

pywebtools.sqlalchemy.**check_database_version**(*db_version*, *dbsession=None*)
Checks that the current version of the database matches the version specified by db_version. This requires the use of the Alembic database migration library.

Parameters

- **db_version** (str) – The version identifier to check.
- **dbsession** (scoped_session()) – The database session to use for database access. If None will create a new session.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

p

T

U

V