TiddlyWeb Documentation

Release 1.4

Chris Dent

Oct 09, 2017

Contents

CHAPTER 1

`tiddlyweb` Package

For more complete information also see:

- http://tiddlyweb.com/
- http://docs.tiddlyweb.com/
- http://tiddlyweb.readthedocs.org/

TiddlyWeb is a web service and library for managing and manipulating resources useful in the creation of dynamic wiki-like collections of content and functionality. The model of the data was originally designed for creating custom TiddlyWiki implementations, where the content of the TiddlyWiki can be saved to the server, and shared among multiple users.

TiddlyWeb presents an `HTTP API` for resource management. The API follows, as possible, RESTful principles to keep the API flexible and scalable. The URLs for this interface are kept in a file called `urls.map` found in the tiddlyweb package. `urls.map` dispatches web requests at specific URLs to specific functions in modules in the *`tiddlyweb.web.handler`* package. `urls.map` may be located in another place by changing the `urls_map` key in tiddlywebconfig.py. There are also mechanisms for overriding storage (see *`tiddlyweb.store`*), serialization (see *`tiddlyweb.serializer`*) and authentication (see `tiddlyweb.web.challenger` and *`tiddlyweb.web.extractor`*) systems. There are also `system_plugins` and `twanager_plugins` for further extensibility.

The primary resources presented by the server are Recipes, Bags and Tiddlers. See the *`tiddlyweb.model`* package.

TiddlyWeb includes twanager, a command line tool for doing a variety of TiddlyWeb activities. Run `twanager` without arguments for a list of commands.

See the documentation for other modules and packages within tiddlyweb for additional details.

CHAPTER 2

config Module

The configuration of a particular instance of TiddlyWeb, carried around as a dict in the WSGI `environ` as `tiddlyweb.config`.

If there is a `tiddlywebconfig.py` file in the working directory where `twanager` or the web server is started, its values will override the defaults established in this module.

The server administrator may add additional keys to the config via extensions.

Config Keys

system_plugins A list of Python module names that act as plugins for the running server. At server startup time they are found, compiled, and the function `init(config)` is called on them, where `config` is a reference to the current config. Use this to add functionality to the server that cannot be accomplished from the defaults, such as adding additional web handlers, storage hooks or overriding existing behaviors.

twanager_plugins A list of Python module names that act as plugins for `twanager`, adding command line functionality. As with `system_plugins` `init(config)` is called.

server_store A list containing a module name and a configuration dictionary. The named module is an implementation of *`tiddlyweb.stores.StorageInterface`* (first looked up in the *`tiddlyweb.stores`* package space, then in `sys.path`). The configuration is an arbitrary dictionary of information to be passed to the store (e.g. database `username` and `password`).

server_request_filters A list of WSGI applications which, in order, process the incoming requests made to the server. This can extract, add, or filter information as necessary. The defaults provide query string processing, content negotiation and establish `environ` settings.

server_response_filters A list of WSGI applications which, in order, process the outgoing response from the server. This can transform, log, or handle exceptions as necessary.

server_host The hostname of this server, usually set from whatever starts the server. This is a dictionary with keys: `scheme`, `host`, `port`.

server_prefix A URL path portion which is a prefix to every URL the system uses and produces. Use this to host TiddlyWeb in a subdirectory (e.g. /wiki). Default is ''.

extension_types A dictionary that pairs extension strings used in URLs as human controlled content-negotiation with the MIME types they represent. Add to this if you add to serializers.

serializers Incoming request Accept headers, or extension MIME types paired with a tiddlyweb.serializations.Serializer implementation and an outgoing MIME type for that type of serialization.

extractors An extractor is a credential extractor (see *tiddlyweb.web.extractors.ExtractorInterface*) that looks in an incoming request to attempt to extract information from it that indicates a potential user in the system. This config item is an ordered list of extractors, tried in succession until one returns tiddlyweb.usersign information or there are no more left.

auth_systems A list of challengers available to the system when it needs to ask for a user. (See *tiddlyweb.web.challengers.ChallengerInterface*) If there is more than one challenger the user is presented with a list of those available. If there is only one, the user is automatically directed to just that one. A challenger needs to work with the extractors system so that the challenger provides something for future requests that the extractor can extract.

secret A string used to encrypt the cookie installed by some of the challengers and used by the cookie extractor. **NOTE: EVERY INSTALLATION SHOULD CHANGE THIS IN ITS OWN CONFIGURATION.**

urls_map The file location of the text file that maps URL paths to Python code, doing method dispatch. Usually it is better to use plugins to change the available URLs and handlers.

bag_create_policy A policy statement on who or what kind of user can create new bags on the system through the web API. ANY means any authenticated user can. ADMIN means any user with role ADMIN can. '' means anyone can.

recipe_create_policy A policy statement on who or what kind of user can create new recipes on the system through the web API. See bag_create_policy.

log_file Path and filename of the TiddlyWeb log file.

log_level String of loglevel to log. Pick one of CRITICAL, DEBUG, ERROR, INFO, WARNING.

css_uri A URI of a css file that can be used to style the HTML output of the server. See *tiddlyweb.serializations.html* for the classes and ids used.

wikitext.default_renderer The default module for rendering tiddler.text to HTML when tiddler.type is None.

wikitext.type_render_map A dictionary mapping tiddler.type MIME types to modules with a render() function for turning that type into HTML.

wsgi_server The name of a module that provides a start_server method which starts a server to run this TiddlyWeb instance. Used by the twanager server command only.

special_bag_detectors A list of functions that take an environ and bag name and return a tuple of two functions: the first returns the tiddlers in that bag, the second returns one tiddler from that bag.

collections.use_memory If True *Tiddler Collections* are kept in memory during a single request. Defaults to False to save memory.

CHAPTER 3

`control` Module

control provides routines which integrate the basic `model classes` with the rest of the system. The model classes are intentionally simple. The methods here act as controllers on those classes.

These are primarily related to handling `recipes`.

`tiddlyweb.control.`**`determine_bag_for_tiddler`**(*recipe*, *tiddler*, *environ=None*)

Return the `bag` which this `tiddler` would be in if we were to save it to the named `recipe` rather than to a bag.

This is done reversing the recipe list and seeing if the tiddler passes the constraint of the bag and its associated `filter`. If bag+filter is true, return that bag.

`tiddlyweb.control.`**`determine_bag_from_recipe`**(*recipe*, *tiddler*, *environ=None*)

Given a `recipe` and a `tiddler` determine the `bag` in which this `tiddler` can be found. This is different from `determine_bag_for_tiddler()`. That one finds the bag the tiddler *could* be in. This is the bag the tiddler *is* in.

This is done by reversing the recipe's list, and filtering each bag according to any `filters` present. The resulting tiddlers are checked.

If an `indexer` is configured use the index to determine if a tiddler exists in a bag.

`tiddlyweb.control.`**`filter_tiddlers`**(*tiddlers*, *filters*, *environ=None*)

Return a generator of tiddlers resulting from filtering the provided iterator of tiddlers by the provided `filters`.

If filters is a string, it will be `parsed for filters`.

`tiddlyweb.control.`**`get_tiddlers_from_bag`**(*bag*)

Yield the individual `tiddlers` that are in a `bag`.

The tiddlers return are empty objects that have not been loaded from the `store`.

Rarely used, see `tiddlyweb.store.Store.list_bag_tiddlers()`.

`tiddlyweb.control.`**`get_tiddlers_from_recipe`**(*recipe*, *environ=None*)

Return the list of tiddlers that result from processing the `recipe`.

This list of tiddlers is unique by title with tiddlers later in the recipe taking precedence over those earlier in the recipe.

The tiddlers returned are empty objects (i.e. not loaded from the *`store`*).

`tiddlyweb.control.`**`readable_tiddlers_by_bag`**(*store*, *tiddlers*, *usersign*)

Yield those tiddlers which are readable by the current `usersign`. This means, depending on the `read` constraint on the *`tiddler's bag's policy`*, yield or not.

`tiddlyweb.control.`**`recipe_template`**(*environ*)

Provide a means to specify custom `{{ key }}` values in *`recipes`* which are then replaced with the value specified in `environ['tiddlyweb.recipe_template']`.

This allows recipes to be dynamic in the face of conditions in the current request.

CHAPTER 4

`manage` Module

manage provides the workings for the `twanager` command line tool. `twanager` calls *`handle()`*, making available all commands that have been put into the `COMMANDS` dictionary by the *`make_command()`* decorator. See *`tiddlyweb.commands`* for examples.

Plugins which add commands must be added to the `twanager_plugins` *`config`* setting so they are imported at the proper time.

`tiddlyweb.manage.`**`handle`**`(`*args*`)`
: Dispatch to the proper function for the command given in `args[1]`.

`tiddlyweb.manage.`**`make_command`**`()`
: A decorator that marks the decorated method as a member of the commands dictionary, with associated help.

  The pydoc of the method is used in automatically generated :py:func:usage information.

`tiddlyweb.manage.`**`usage`**`(`*args*`)`
: List this help

CHAPTER 5

serializer Module

Serialize TiddlyWeb entities for the sake of taking input and sending output.

This module provides the facade for accessing the possibly many modules which act as serializations. It is asked by calling code to provide a serialization for a given MIME type. Plugins may override what MIME types are handled and by what modules. See *tiddlyweb.config* for related configuration settings.

exception tiddlyweb.serializer.**BagFormatError**

Bases: exceptions.Exception

The provided input is insufficient to form a valid Bag.

exception tiddlyweb.serializer.**NoSerializationError**

Bases: exceptions.Exception

There is a NoSerialization of this type for the entity.

exception tiddlyweb.serializer.**RecipeFormatError**

Bases: exceptions.Exception

The provided input is insufficient to form a valid Recipe.

class tiddlyweb.serializer.**Serializer**(*engine*, *environ=None*)

Bases: object

A Serializer is a facade to a Serialization which implements the *tiddlyweb.serializations.SerializationInterface* to turn a TiddlyWeb *entity* into a particular representation or vice versa.

A Serializer can also list collections of entities in a particular representation.

A single Serializer is a reusable tool which can serialize more than one object. You must set serializer.object after initialization and then again for each subsequent object being serialized.

The following example turns the *tiddler* into JSON and vice-versa:

```
tiddler = Tiddler('cow', 'bag')
tiddler.text = 'moo'
serializer = Serializer('json', environ)
serializer.object = tiddler
json_string = serializer.to_string()
```

```
assert '"text": "moo"' in json_string
new_string = json_string.replace('moo', 'meow')
serializer.from_string(new_string)
assert tiddler.text == 'meow'
```

Note that `to_string()` and `from_string()` operate on the Serializer which dispatches to a method in the SerializationInterface implementation based on the class of the object being serialized.

`from_string`(*input_string*)
Turn the provided `input_string` into a TiddlyWeb entity object of the type of `self.object`. That is: populate `self.object` based on `input_string`.

`list_bags`(*bags*)
Provide a (usually unicode) string representation of the provided `bags` in the current serialization.

`list_recipes`(*recipes*)
Provide a (usually unicode) string representation of the provided `recipes` in the current serialization.

`list_tiddlers`(*tiddlers*)
Provide a (usually unicode) string representation of the `tiddlers` in the provided `Tiddlers collection`.

`to_string`()
Provide a (usually unicode) string representation of the `bag`, `recipe` or `tiddler` at `self.object`.

exception `tiddlyweb.serializer.`**`TiddlerFormatError`**
Bases: `exceptions.Exception`

The provided input is insufficient to form a valid Tiddler.

CHAPTER 6

`specialbag` Module

Special bags are a feature implemented in plugins that allow non-standard collections of data to be represented as a *`bag`* of *`tiddlers`*. An example is remotebag.

If `config['special_bag_detectors']` is set, it is a list of functions that take two arguments: a WSGI `environ` and a string and return either:

- two functions
- None

The first function yields tiddlers, like `tiddlyweb.store.list_bag_tiddlers()`. It's arguments are a WSGI *environ* and a string.

The second function returns a single *`tiddler`*. It's arguments are a WSGI `environ` and a tiddler object (with at least `title` and `bag` set).

exception `tiddlyweb.specialbag.`**`SpecialBagError`**

Bases: `exceptions.Exception`

A generic exception to be raised by special bag implementations.

`tiddlyweb.specialbag.`**`get_bag_retriever`**`(`*environ*, *bag*`)`

When loading *`bag`* or *`tiddlers`* within it from the *`store`*, this method is used to inspect `config['special_bag_detectors']` to determine if there is a special handler. If there is, the handler is returned and used for retrieval, otherwise `None` is returned and the store is used as normal.

CHAPTER 7

`store` Module

Store TiddlyWeb entities to a configured persistence layer.

This module provides the facade for accessing one of many possible modules which provide storage for entities. It provides a general interface to get, put, delete or list `entities`.

Each of the single entity methods can be augmented with hooks provided by plugins. This allows actions to be performed based on data in the store being retrieved or updated, such as updating an index.

exception `tiddlyweb.store.`**`NoBagError`**

Bases: `tiddlyweb.store.StoreError`

No `tiddlyweb.model.bag.Bag` was found.

exception `tiddlyweb.store.`**`NoRecipeError`**

Bases: `tiddlyweb.store.StoreError`

No `tiddlyweb.model.recipe.Recipe` was found.

exception `tiddlyweb.store.`**`NoTiddlerError`**

Bases: `tiddlyweb.store.StoreError`

No `tiddlyweb.model.tiddler.Tiddler` was found.

exception `tiddlyweb.store.`**`NoUserError`**

Bases: `tiddlyweb.store.StoreError`

No `tiddlyweb.model.user.User` was found.

class `tiddlyweb.store.`**`Store`**(*engine*, *config=None*, *environ=None*)

Bases: `object`

A Store is a facade to an implementation of `tiddlyweb.stores.StorageInterface` to handle the storage and retrieval of all `entities` in the TiddlyWeb system.

Because of the facade system it is relatively straightforward to create diverse storage systems for all sorts of or multiple media. In addition stores can be layered to provide robust caching and reliability.

The Store distinguishes between single entities and collections. With single entities, an entity is passed to the store and the store is asked to `get()`, `put()` or `delete()` it. When `get()` is used the provided object is updated in place in operation that could be described as population. Dispatch is based on the class of the provided entity.

After any of those operations optional `HOOKS` are called.

With collections there are specific `list` methods:

- *`list_bags()`*
- *`list_recipes()`*
- *`list_bag_tiddlers()`*
- *`list_tiddler_revisions()`*
- *`list_users()`*

Finally a store may optionally provide a *`search()`*. How search works and what it even means is up to the implementation.

`delete`(*thing*)
Delete a thing: recipe, bag, tiddler or user.

`get`(*thing*)
Get a thing: recipe, bag, tiddler or user.

`list_bag_tiddlers`(*bag*)
List all the tiddlers in the bag.

`list_bags`()
List all the available bags in the system.

`list_recipes`()
List all the available recipes in the system.

`list_tiddler_revisions`(*tiddler*)
List the revision ids of the revisions of the indicated tiddler in reverse chronological older (newest first).

`list_users`()
List all the available users in the system.

`put`(*thing*)
Put a thing, recipe, bag, tiddler or user.

`search`(*search_query*)
Search in the store, using a search algorithm specific to the *`tiddlyweb.stores.StorageInterface`* implementation.

exception `tiddlyweb.store.`**`StoreEncodingError`**
Bases: *`tiddlyweb.store.StoreError`*

Something about an entity made it impossible to be encoded to the form required by the store.

exception `tiddlyweb.store.`**`StoreError`**
Bases: `exceptions.IOError`

Base Exception for Store Exceptions.

exception `tiddlyweb.store.`**`StoreLockError`**
Bases: *`tiddlyweb.store.StoreError`*

This process was unable to get a lock on the store.

exception `tiddlyweb.store.`**`StoreMethodNotImplemented`**
Bases: *`tiddlyweb.store.StoreError`*

A `tiddlyweb.stores.StorageInterface` does not implement this method.

`tiddlyweb.store.`**`get_entity`**`(`*entity*, *store*`)`

Load the provided entity from the store if it has not already been loaded. If it can't be found, still return the same entity, just keep it empty.

This works for tiddlers, bags and recipes. Not users!

CHAPTER 8

`util` Module

This module provides a centralized collection of miscellaneous utility functions used throughout TiddlyWeb and plugins.

Web specific utilities are in *`tiddlyweb.web.util`*.

exception `tiddlyweb.util.`**`LockError`**

Bases: `exceptions.IOError`

This process was unable to get a lock.

`tiddlyweb.util.`**`binary_tiddler`**(*tiddler*)

Test if a *`tiddler`* represents binary content (e.g. an image).

Return `True` if this Tiddler has a `type` which suggests the content of the tiddler is non-textual.

`tiddlyweb.util.`**`initialize_logging`**(*config*, *server=False*)

Initialize the system's logging.

If this code is reached from `twanager` when there is no sub_command logging is not started. This avoids spurious `tiddlyweb.log` files popping up all over the place.

`tiddlyweb.util.`**`merge_config`**(*global_config*, *additional_config*, *reconfig=True*)

Update the `global_config` with the additional data provided in the dict `additional_config`. If `reconfig` is `True`, then reread and merge `tiddlywebconfig.py` so its overrides continue to operate.

Note that if the value of a existing key is a dict, then it is updated (merged) with the value from `additional_config`. Otherwise the value is replaced.

Warning: Please ensure (via tests) when using this that it will give the desired results.

`tiddlyweb.util.`**`pseudo_binary`**(*content_type*)

Test if a *`tiddler`* represents textual content that should be treated as a pseudo-binary.

A pseudo binary is defined as textual content for which (this) TiddlyWeb (instance) has no *`serialization`* or is not treated as *`wikitext`*. It is identified by a `MIME` type that looks like `text`, `json`, `xml` or `javascript`.

TiddlyWeb requires that such content be uploaded encoded as `UTF-8`.

`tiddlyweb.util.`**`read_config`**`(`*global_config*`)`
Read in a local configuration override, named `tiddlywebconfig.py`, from the current working directory. If the file exists but can't be imported as valid Python an exception will be thrown, preventing unexpected results.

What is expected in the override file is a dict with the name `config`.

`global_config` is a reference to the currently operational main TiddlyWeb *`config`*. The read configuration data is merged into it.

`tiddlyweb.util.`**`read_utf8_file`**`(`*filename*`)`
Read the `UTF-8` encoded file at `filename` and return unicode.

Allow any exceptions to raise.

`tiddlyweb.util.`**`renderable`**`(`*tiddler*, *environ=None*`)`
Return `True` if the provided *`tiddler's`* `type` is one that can be rendered to HTML by the *`wikitext`* rendering subsystem.

`tiddlyweb.util.`**`sha`**`(`*data=''*`)`
Create a sha1 digest of the `data`.

`tiddlyweb.util.`**`std_error_message`**`(`*message*`)`
Display `message` on the `stderr` console.

`tiddlyweb.util.`**`superclass_name`**`(`*instance*`)`
Given an instance return the lowerclass name of the penultimate class in the hierarchy (the last is object). This is used to do dynamic method lookups in adaptor classes via serializer.py and store.py while still allowing model entities to be subclassed. Those subclasses must insure that their __mro__ results in Bag, User, Recipe or Tiddler in the penultimate slot.

`tiddlyweb.util.`**`write_lock`**`(`*filename*`)`
Create an advisory lock file based on `filename`.

This is primarily used by the *`text store`*.

`tiddlyweb.util.`**`write_unlock`**`(`*filename*`)`
Unlock the write lock associated with `filename`.

`tiddlyweb.util.`**`write_utf8_file`**`(`*filename*, *content*`)`
Write the unicode string in `content` to a `UTF-8` encoded file named `filename`.

Allow any exceptions to raise.

CHAPTER 9

Subpackages

commands Package

`commands` Package

Command line tools for TiddlyWeb are accessed via the `twanager` script. Each command is named by the first argument passed to the script.

The commands defined in this package are added to a list of available commands using the `twanager` plugin mechanism. That list is extensible via `twanager_plugins` in *`tiddlyweb.config`* and *`tiddlyweb.manage.make_command()`*.

Typical commands do things like starting a server, creating a user and listing existing entities.

`tiddlyweb.commands.`**`init`**`(`*config*`)`
Establish the commands during `twanager` startup.

`interact` Module

This module provides a `twanager` command `interact` which provides a Python shell preloaded with the necessary local variables to interact with the current instance's *`store`* and the entities within. The locals are:

- *`Recipe`*
- *`Bag`*
- *`Tiddler`*
- *`User`*
- *`Policy`*
- *`Serializer`*
- *`control`*
- *`util`*

- *web*
- An `environ` containing `tiddlyweb.config` and *tiddlyweb.store‘* keys and values.
- A `config` containing the current `tiddlyweb.config`.

These are enough to do most operations.

class `tiddlyweb.commands.interact.`**`TabCompleter`**(*namespace=None*)
Bases: `rlcompleter.Completer`

Tab completion for the interactive shell that allows pressing the tab character to indicate an indent.

`complete`(*text*, *state*)
Complete the provided `text`. If there is no text, indent.

class `tiddlyweb.commands.interact.`**`TiddlyWebREPL`**(*locals=None*, *filename='<console>'*)
Bases: `code.InteractiveConsole`

An interactive console for the current TiddlyWeb instance.

This augments it’s super class by adding tab completion and establishing a set of useful local variables.

`tiddlyweb.commands.interact.`**`launch_shell`**(*config*, *store*, *args*)
Establish the basic environment for the shell and then start it.

filters Package

`filters` Package

Overarching handler for TiddlyWeb filters.

Filters provide an extensible syntax for limiting any *`Collection`* by attributes on the entities in the collection. Though primarily for *`Tiddlers`*, *`Bags`* and *`Recipes`* can be filtered as well.

The basic filters provide for selecting and sorting on attributes of the entities and for limiting (the number of) entities. These basic types of filter can be extended with plugins, and the ways attributes are processed can also be extended.

Filters are parsed from a string that is formatted as a `CGI` query string with parameters and arguments. The parameter is a filter type. Each filter is processed in sequence: the first processing all the entities handed to it, the next taking only those that result from the first.

Filters can be extended by adding more parsers to `FILTER_PARSERS`. Parsers for existing filter types may be extended as well (see the documentation for each type).

The call signature for a filter is:

```
filter(entities, indexable=indexable, environ=environ)
```

The attribute and value for which a filter filters is established in the parsing stage and are set as upvalues of the filter closure that gets created.

`indexable` and `environ` are optional parameters that in special cases allow a *`select`* style filter to be optimized with the use of an index. In the current implementation this is only done when:

- the select filter is the first filter in a stack of filters passed to *`recursive_filter()`*
- the entities to be filtered are *`tiddlers`* in the context of a *`bag`* (this helps to constrain the index)

When both of the above are true the system looks for a module named by `tiddlyweb.config['indexer']`, imports it, looks for a function called `indexy_query`, and passes `environ` and information about the bag and the attribute being selected.

What `index_query` does to satisfy the query is up to the module. It should return a list of tiddlers that have been loaded from the `tiddlyweb.store.Store`.

If for some reason `index_query` does not wish to perform the query (e.g. the index cannot satisfy the query) it may raise `FilterIndexRefused` and the normal filtering process will be performed.

Note that testing should be done to determine if using an index is of any benefit. In some stores (for example caching stores) traversing the tiddlers is faster than using an index.

exception `tiddlyweb.filters.`**`FilterError`**

Bases: `exceptions.Exception`

An exception to throw when an attempt is made to filter on an unavailable attribute.

exception `tiddlyweb.filters.`**`FilterIndexRefused`**

Bases: `tiddlyweb.filters.FilterError`

A filter index has refused to satisfy a filter with its index.

`tiddlyweb.filters.`**`parse_for_filters`**(*query_string*, *environ=None*)

Take a string that looks like a `CGI` query string and parse it for filters. Return a tuple of a list of filter functions and a string of whatever was in the query string that did not result in a filter.

`tiddlyweb.filters.`**`recursive_filter`**(*filters*, *entities*, *indexable=False*)

Recursively process the list of `filters` found by `parse_for_filters()` against the given list of `entities`.

Each next filter processes only those entities that were results of the previous filter.

Misnamed, early versions were more truly recursive.

`limit` Module

A `filter` type to limit a group of entities using a syntax similar to SQL Limit:

```
limit=<index>,<count>
limit=<count>
```

`tiddlyweb.filters.limit.`**`limit`**(*entities*, *count=0*, *index=0*)

Make a slice of a list of entities based on a count and index.

`tiddlyweb.filters.limit.`**`limit_parse`**(*count='0'*)

Parse the argument of a `limit` `filter` for a count and index argument, return a function which does the limiting.

Exceptions while parsing are passed up the stack.

`select` Module

A `filter` type for selecting only some entities, usually `tiddlers`, from a collection of entities, usually by an attribute of the tiddlers.

The syntax is:

```
select=attribute:value     # attribute is value
select=attribute:!value    # attribute is not value
select=attribute:>value    # attribute is greater than value
select=attribute:<value    # attribute is less than value
```

ATTRIBUTE_SELECTOR is checked for a function which returns True or False for whether the provided value matches for the entity being tested. The default case is lower case string equality. Other functions may be provided by plugins. Attributes may be virtual, i.e. not real attributes on entity. For example we can check for the presence of a tag in a tiddlers tags attribute with:

```
select=tag:tagvalue
```

An attribute function takes an entity, an attribute name and a value. It may then do anything it wants with it, and must return True or False.

- ! negates a selection, getting all those entities that don't match.
- > gets those entities that sort greater than the value.
- < gets those entities that sort less than the value.

When doing sorting ATTRIBUTE_SORT_KEY is consulted to canonicalize the value. See *tiddlyweb.filters.sort*.

tiddlyweb.filters.select.**bag_in_recipe**(*entity*, *attribute*, *value*)
Return True if the named *bag* is in the *recipe*.

tiddlyweb.filters.select.**default_func**(*entity*, *attribute*, *value*)
Look in the entity for an attribute with the provided value. First real object attributes are checked, then, if available, extended fields. If neither of these are present, return False.

tiddlyweb.filters.select.**field_in_fields**(*entity*, *attribute*, *value*)
Return True if the entity has the named field.

tiddlyweb.filters.select.**select_by_attribute**(*attribute*, *value*, *entities*, *negate=False*, *indexable=None*, *environ=None*)
Select entities where value of attribute matches the provide value.

If negate is True, get those that don't match.

tiddlyweb.filters.select.**select_parse**(*command*)
Parse a select *filter* string into attributes and arguments and return a function (for later use) which will do the selecting.

tiddlyweb.filters.select.**select_relative_attribute**(*attribute*, *value*, *entities*, *greater=False*, *lesser=False*, *environ=None*)
Select entities that sort greater or less than the provided value for the provided attribute.

tiddlyweb.filters.select.**tag_in_tags**(*entity*, *attribute*, *value*)
Return True if the provided entity has a tag of value in its tag list.

tiddlyweb.filters.select.**text_in_text**(*entity*, *attribute*, *value*)
Return True if the provided entity has the string provided in value within its text attribute.

sort Module

A *filter* type to sort a collection of entities by some attribute. The syntax is:

```
sort=attribute    # sort ascending
sort=-attribute   # sort descending
```

Atribute is either a real entity attribute or a key in ATTRIBUTE_SORT_KEY that has as its value a function used to generate a key to pass to the sort. ATTRIBUTE_SORT_KEY can be extended by plugins.

`tiddlyweb.filters.sort.`**`as_int`**`(`*attribute*`)`
Treat attribute as `int` if it looks like one.

`tiddlyweb.filters.sort.`**`date_to_canonical`**`(`*datestring*`)`
Take a (TiddlyWiki-style) string of 14 or less digits and turn it into 14 digits for the sake of comparing entity dates.

`tiddlyweb.filters.sort.`**`sort_by_attribute`**`(`*attribute*, *entities*, *reverse=False*, *environ=None*`)`
Sort a group of entities by some attribute. Inspect `ATTRIBUTE_SORT_KEY` to see if there is a special function by which we should generate the value for this attribute.

`tiddlyweb.filters.sort.`**`sort_parse`**`(`*attribute*`)`
Create a function which will sort a collection of entities.

model Package

`model` Package

Models for TiddlyWeb Entities.

Classes representing the important entities in the TiddlyWeb system.

These are intentionally limited, making no effort to handle their own persistence or presentation. That is the job of the *`store`* and *`serializer`*.

`bag` Module

A module containing the *`Bag`* class.

class `tiddlyweb.model.bag.`**`Bag`**`(`*name*, *desc=u''*`)`
Bases: `object`

A Bag is a virtual container for *`tiddlers`*. The bag provides a domain for the tiddlers within identifying those tiddlers uniquely and optionally acting a topical, functional or authorization container for the tiddlers.

A bag can contain many tiddlers but every tiddler must have a different title. The canonical identifier of a tiddler is the combination of the containing bag's name and the tiddler's title.

Containership is achieved via association: There are no methods on a bag to access the contained tiddlers. These are found via *`store.list_bag_tiddlers`*.

A Bag has a *`Policy`* which may be used to control access to both the Bag and the tiddlers within. These controls are optional and are primarily designed for use within the *`web handlers`*.

`collections` Module

Classes representing collections of *`bags`*, *`recipes`* and *`tiddlers`*.

Because the main reason for having a collection is to send it out over the web, the collections keep track of their last-modified time and generate a hash suitable for use as an ETag.

class `tiddlyweb.model.collections.`**`Collection`**`(`*title=''*`)`
Bases: `object`

Base class for all collections.

Can be used directly for general collections if required.

A collection acts as generator, yielding one of its contents when iterated.

add(*thing*)
Add an item to the container, updating the digest and modified information.

hexdigest()
Return the current hex representation of the hash digest of this collection.

class `tiddlyweb.model.collections.`**`Container`**(*title=''*)
Bases: *`tiddlyweb.model.collections.Collection`*

A collection of things which have a `name` attribute.

In TiddlyWeb this is for lists of *`bags`* and *`recipes`*.

class `tiddlyweb.model.collections.`**`Tiddlers`**(*title=''*, *store=None*, *bag=None*, *recipe=None*)
Bases: *`tiddlyweb.model.collections.Collection`*

A Collection specifically for *`tiddlers`*.

This differs from the base class in two ways:

The calculation of the digest is more detailed in order to create stong `ETags` for the collection.

When iterated, if `store` is set on the Collection, then a yielded tiddler will be loaded from the store to fill in all its attributes. When a tiddler is added to the collection, if it is already filled, a non-full copy is made and put into the collection. This is done to save memory and because often the data is not needed.

If `collections.use_memory` is `True` in `config` then the full tiddler is kept in the collection. On servers with adequate memory this can be more efficient.

add(*tiddler*)
Add a reference to the *`tiddler`* to the container, updating the digest and modified information. If the tiddler has recently been deleted, resulting in a *`StoreError`*, don't add it.

`policy` Module

A module containing the *`Policy`* class and associated exceptions.

exception `tiddlyweb.model.policy.`**`ForbiddenError`**
Bases: *`tiddlyweb.model.policy.PermissionsError`*

The provided user cannot do this action.

exception `tiddlyweb.model.policy.`**`PermissionsError`**
Bases: `exceptions.Exception`

Base class for *`Policy`* violations.

class `tiddlyweb.model.policy.`**`Policy`**(*owner=None*, *read=None*, *write=None*, *create=None*, *delete=None*, *manage=None*, *accept=None*)
Bases: `object`

A container for information about the contraints on a *`bag`* or *`recipe`*. Both are containers of *`tiddlers`*. We need to be able to control who can do what to do those tiddlers. We also need to be able to control who can manage those constraints.

The :pu:func:__init__ parameters represent a default policy.

There are six constraints plus one identifying attribute (`owner`). The constraints are listed below with descriptions of what is allowed if the constraint passes.

read View this entity in lists. View the contained entities.

write Edit the contained entities that already exist.

create Create new entities in the container.

delete Remove a contained entity.

manage Change the policy itself.

accept Accept the entity into the container without requiring `validation`.

`allows`(*usersign*, *constraint*)

Is the user encapsulated by the `usersign` dict allowed to perform the action described by `constraint`. If so, return True. If not raise a `UserRequiredError` (if the user is `GUEST`) or `ForbiddenError` exception.

The dict has a `name` key with a string value which is a `username` and a `roles` key with a list of roles as its value. Either may match in the constraint. Usersign is usually populated during the `CredentialsExtractor` phase of a request.

`attributes` = [u'read', u'write', u'create', u'delete', u'manage', u'accept', u'owner']

`user_perms`(*usersign*)

For this policy return a list of constraints for which this usersign passes.

exception `tiddlyweb.model.policy.`**`UserRequiredError`**

Bases: `tiddlyweb.model.policy.PermissionsError`

To do this action a user is required.

`tiddlyweb.model.policy.`**`create_policy_check`**(*environ*, *entity*, *usersign*)

Determine if the user in `usersign` can create `entity` type.

`recipe` Module

The Recipe class.

class `tiddlyweb.model.recipe.`**`Recipe`**(*name*, *desc=u''*)

Bases: `object`

A Recipe is an ordered list that represents a program for creating a collection of `tiddlers`.

Each line in the recipe is the combination of a `bag` name and a `filter` string. This implementation uses list of tuples.

In common usage a recipe contains only strings representing bags and filters, but for the sake of easy testing, the bag argument can be a `Bag` object.

A Recipe has a `Policy` which can be used to control access to the Recipe. These controls are optional and are primarily designed for use within the `web handlers`.

`get_recipe`(*template=None*)

Return the recipe list, as a list of tuple pairs.

`set_recipe`(*recipe_list*)

Set the contents of the recipe list.

`tiddler` Module

A module containing the `Tiddler` class and related functions.

class `tiddlyweb.model.tiddler.`**`Tiddler`**(*title=None*, *bag=None*)

Bases: `object`

The primary content object in the TiddlyWiki and TiddlyWeb universe, representing a distinct piece of content, often vaguely corresponding to a Page in wiki systems. A Tiddler has text and some associated metadata. The text can be anything, often wikitext in some form, or Javascript code. It is possible for a Tiddler to container binary content, such as image data.

A Tiddler is intentionally solely a container of data. That is, it has no methods which change the state of attributes in the Tiddler or manipulate the tiddler. Changing the attributes is done by directly changing the attributes. This is done to make the Tiddler easier to *`store`* and *`serialize`* in many ways.

A Tiddler has several attributes:

title The name of the tiddler. Required.

created A string representing when this tiddler was created.

modified A string representing when this tiddler was last changed. Defaults to now.

modifier A string representing a personage that changed this tiddler in some way. This doesn't necessarily have any assocation with the tiddlyweb.usersign, though it may.

tags A list of strings that describe the tiddler.

fields An arbitrary dictionary of extended (custom) fields on the tiddler.

text The contents of the tiddler. A string.

revision The revision of this tiddler. The type of a revision is unspecified and is *`store`* dependent.

bag The name of the bag in which this tiddler exists, if any.

recipe The name of the recipe in which this tiddler exists, if any.

store A reference to the *`Store`* object which retrieved this tiddler from persistent storage.

`creator`

Get the creator of this tiddler. If it has not been set then use modifier.

Use the creator property instead.

`data_members` **= ['title', 'creator', 'created', 'modifier', 'modified', 'tags', 'fields', 'type', 'text']**

`slots` **= ['title', 'creator', 'created', 'modifier', 'modified', 'tags', 'fields', 'type', 'text', 'revision', 'bag', 'recipe', 'store']**

`tiddlyweb.model.tiddler.`**`current_timestring`**()

Translate the current UTC time into a TiddlyWiki conformat timestring.

`tiddlyweb.model.tiddler.`**`string_to_tags_list`**(*string*)

Given a string representing tags (space-delimited, tags containing spaces are enclosed in in double brackets), parse them into a list of tag strings.

Duplicates are removed.

`tiddlyweb.model.tiddler.`**`tags_list_to_string`**(*tags*)

Given a list of `tags`, turn it into the canonical string representation (space-delimited, enclosing tags containing spaces in double brackets).

`tiddlyweb.model.tiddler.`**`timestring_to_datetime`**(*timestring*)

Turn a TiddlyWiki timestring into a datetime object.

Will raise ValueError if the input is not a 12 or 14 digit timestring.

`user` Module

A class representing a simple user entity.

A User object is not required during TiddlyWeb requests, `credentials extractors` and `policies` may work with arbitrary data for authentication and authorization. However if a locally stored user is required the `User` may be used.

class `tiddlyweb.model.user.`**`User`**(*usersign*, *note=None*)

Bases: `object`

A simple representation of a user. A user is a username, an optional password, an optional list of roles, and an optional note.

`add_role`(*role*)
Add the named `role` (a string) to this user.

`check_password`(*candidate_password*)
Check the password for this user. Return `True` if correct.

`del_role`(*role*)
Remove the named `role` (a string) from this user. If it is not there, do nothing.

`list_roles`()
List (as a list of strings) the roles that this user has.

`set_password`(*password*)
Set the password for this user.

serializations Package

`serializations` Package

Turn entities to and fro various representations.

This is the base class and interface class used to transform strings of various forms to model objects and model objects to strings of various forms.

class `tiddlyweb.serializations.`**`SerializationInterface`**(*environ=None*)

Bases: `object`

A Serialization is a collection of methods that either turn an input string into the object named by the method, or turn the object into a string form. A Serialization is not called directly, instead a `Serializer` facade is used.

The interface is fairly simple: For the core entities that exist in the TiddlyWeb system (`bags`, `recipes` and `tiddlers` there (optionally) exists `<entity>_as` and `as_<entity>` methods in each Serialization.

`*_as` returns a string form of the entity, perhaps as HTML, Text, YAML, Atom, whatever the Serialization does.

`as_*` takes a provided entity and string and updates the skeletal entity to use the information contained in the string (in the Serialization format).

There are also three supporting methods, `list_tiddlers()`, `list_recipes()` and `list_bags()` that provide convenience methods for presenting a collection of entities in the Serialization form. A string is returned.

Strings are usually unicode.

If a method doesn't exist a NoSerializationError is raised and the calling code is expected to do something intelligent when trapping it.

as_bag (*bag*, *input_string*)
Take `input_string` which is a serialized bag and use it to populate the *Bag* in `bag` (if possible).

as_recipe (*recipe*, *input_string*)
Take `input_string` which is a serialized recipe and use it to populate the *Recipe* in `recipe` (if possible).

as_tags (*string*)
Not called directly, but made public for future use. Turn a string into a list of tags.

as_tiddler (*tiddler*, *input_string*)
Take `input_string` which is a serialized tiddler and use it to populate the *Tiddler* in `tiddler` (if possible).

bag_as (*bag*)
Serialize a *Bag* into this serializer's form.

list_bags (*bags*)
Provided a list of *bags*, make a serialized list of those bags (e.g. a a list of HTML links).

list_recipes (*recipes*)
Provided a list of *recipes*, make a serialized list of those recipes (e.g. a a list of HTML links).

list_tiddlers (*bag*)
Provided a *bag*, output the associated *tiddlers*.

recipe_as (*recipe*)
Serialize a :py:*Recipe* into this serializer's form.

tags_as (*tags*)
Not called directly, but made public for future use. Turn a list of tags into a serialized list.

tiddler_as (*tiddler*)
Serialize a *Tiddler* into this serializer's form.

`html` Module

Serialization for HTML.

`HEADER` and `FOOTER` can be overridden to change the basic framing of the system.

class `tiddlyweb.serializations.html.`**Serialization** (*environ=None*)
Bases: *tiddlyweb.serializations.SerializationInterface*

Serialize entities and collections to `HTML` representations. This is primarily used to create browser based presentations. No support is provided for turning `HTML` into entities.

Set `css_uri` in *config* to control CSS.

Set `tiddlyweb.links` in `environ` to a list of `<link>` elements to include those links in the output.

bag_as (*bag*)
Bag as HTML, including a link to the tiddlers within.

list_bags (*bags*)
Yield the provided *bags* as HTML.

list_recipes (*recipes*)
Yield the provided *recipes* as HTML.

list_tiddlers (*tiddlers*)
Yield the provided *tiddlers* as HTML.

This is somewhat more complex than the other list methods as we need to list the tiddler whether it is a revision or not, if it is in a bag or recipe or if it is a search result.

recipe_as(*recipe*)
Recipe as HTML, including a link to the tiddlers within.

tiddler_as(*tiddler*)
Transform the provided *tiddler* into an HTML representation. *Render* the `text` of the tiddler if its `type` is configured.

`json` Module

Serialization for `JSON`.

class `tiddlyweb.serializations.json.`**Serialization**(*environ=None*)
Bases: *tiddlyweb.serializations.SerializationInterface*

Turn entities and collections thereof to and from `JSON`.

as_bag(*bag, input_string*)
Turn a `JSON` dictionary into a *bag* if it is in the proper form. Include the *policy*.

as_recipe(*recipe, input_string*)
Turn a `JSON` dictionary into a *recipe* if it is in the proper form. Include the *policy*.

as_tiddler(*tiddler, input_string*)
Turn a `JSON` dictionary into a *tiddler*. Any keys in the `JSON` which are not recognized will be ignored.

bag_as(*bag*)
A *bag* as a `JSON` dictionary. Includes the bag's *policy*.

list_bags(*bags*)
Create a `JSON` list of *bag* names from the provided `bags`.

list_recipes(*recipes*)
Create a `JSON` list of *recipe* names from the provided `recipes`.

list_tiddlers(*tiddlers*)
List the provided *tiddlers* as `JSON`. The format is a list of dicts in the form described by `_tiddler_dict()`.

If `fat=1` is set in `tiddlyweb.query` include the `text` of each tiddler in the output.

If `render=1` is set in `tiddlyweb.query` include the *rendering* of the `text` of each tiddler in the output, if the tiddler is renderable.

recipe_as(*recipe*)
A *recipe* as a `JSON` dictionary. Includes the recipe's *policy*.

tiddler_as(*tiddler*)
Create a `JSON` dictionary representing a tiddler, as described by `_tiddler_dict()` plus the `text` of the tiddler.

If `fat=0` is set in `tiddlyweb.query` do not include the `text` of the tiddler in the output.

If `render=1` is set in `tiddlyweb.query` include the *rendering* of the `text` of the tiddler in the output, if the tiddler is renderable.

`text` Module

`Serialization` for plain text.

class `tiddlyweb.serializations.text.`**`Serialization`**(*environ=None*)

Bases: `tiddlyweb.serializations.SerializationInterface`

Serialize entities and collections to and from textual representations. This is primarily used by the `text Store`.

`as_recipe`(*recipe*, *input_string*)
Turn a string into a `recipe` if possible.

`as_tiddler`(*tiddler*, *input_string*)
Transform a text representation of a `tiddler` into a tiddler object.

`field` = **'text'**

`fields_as`(*tiddler*)
Turn extended `tiddler` fields into RFC 822-style header strings.

`list_bags`(*bags*)
Return a linefeed separated list of `bag` names in the `bags` list.

`list_recipes`(*recipes*)
Return a linefeed separated list of `recipe` names in the `recipes` list.

`list_tiddlers`(*tiddlers*)
Return a linefeed separated list of `tiddler` titles in the `tiddlers` list.

If the tiddlers are a collection of revisions, include the revision identifier.

`recipe_as`(*recipe*)
Dump a `recipe` as text.

`tiddler_as`(*tiddler*, *omit_empty=False*, *omit_members=None*)
Represent a `tiddler` as a text string: headers, blank line, text.

`omit_*` arguments are non-standard options, usable only when this method is called directly (outside the regular Serializer interface)

If `omit_empty` is True, don't emit empty Tiddler members.

`omit_members` can be used to provide a list of members to not include in the output.

`tiddler_members` = **['creator', 'created', 'modifier', 'modified', 'tags', 'type']**

stores Package

`stores` Package

Storage systems for TiddlyWeb.

The base class and Interface for classes used to get and put data into a storage system.

class `tiddlyweb.stores.`**`StorageInterface`**(*store_config=None*, *environ=None*)

Bases: `object`

An implementation of the StorageInterface is a collection of methods that either store an object or retrieve an object. It is not usually access directly but instead called through a `Store` facade.

The interface is fairly simple: For the data entities that exist in the TiddlyWeb system there (optionally) exists `<entity>_put`, `<entity>_get` and `<entity>_delete` methods.

When `<entity>_get` is used, an empty object is provided. This object is filled by the store method.

There are also five supporting methods, *`list_recipes()`*, *`list_bags()`*, *`list_users()`*, *`list_bag_tiddlers()`*, and *`list_tiddler_revisions()`* that provide methods for getting a collection.

It is useful to understand the classes in the *`tiddlyweb.model`* package when implementing new StorageInterface classes.

If a method is not implemented by the StorageInterface a *`StoreMethodNotImplemented`* exception is raised and the calling code is expected to handle that intelligently.

It is somewhat common to not implement *`list_tiddler_revisions()`*. When this is done it means the instance does not support revisions.

`bag_delete`(*bag*)
: Remove *`bag`* from the store, *including* the *`tiddlers`* contained by the bag.

`bag_get`(*bag*)
: Get the indicated *`bag`* from the store.

`bag_put`(*bag*)
: Put *`bag`* into the store.

`list_bag_tiddlers`(*bag*)
: Retrieve a list of all *`tiddler`* objects in the named *`bag`*.

`list_bags`()
: Retrieve a list of all *`bag`* objects in the system.

`list_recipes`()
: Retrieve a list of all *`recipe`* objects in the system.

`list_tiddler_revisions`(*tiddler*)
: Retrieve a list of all the revision identifiers for the one *`tiddler`*.

`list_users`()
: Retrieve a list of all *`user`* objects in the system.

`recipe_delete`(*recipe*)
: Remove the *`recipe`* from the store, with no impact on the recipe's *`tiddlers`*.

`recipe_get`(*recipe*)
: Get the indicated *`recipe`* from the store.

`recipe_put`(*recipe*)
: Put *`recipe`* into the store.

`search`(*search_query*)
: Search the entire *`tiddler`* store for `search_query`.

: How search operates is entirely dependent on the StorageInterface implementation. The only requirement is that an iterator of tiddler objects is returned.

`tiddler_delete`(*tiddler*)
: Delete *`tiddler`* (and all its revisions) from the store.

`tiddler_get`(*tiddler*)
: Get a tiddler from the store, returning a populated tiddler object. `tiddler.creator` and `tiddler.created` are based on the modifier and modified of the first revision of a tiddler.

tiddler_put (*tiddler*)
Put `tiddler` into the store.

user_delete (*user*)
Delete `user` from the store.

This will remove the user object but has no impact on other entities which may have been modified by the user.

user_get (*user*)
Get `user` from the store.

user_put (*user*)
Put `user` into the store.

`text` Module

A text-based `StorageInterface` that stores entities in a hierarchy of directories in the filesystem.

class `tiddlyweb.stores.text.`**Store** (*store_config=None*, *environ=None*)
Bases: `tiddlyweb.stores.StorageInterface`

A `StorageInterface` which stores text-based representations in a collection of directories and files.

Some of the entities are serialized to and from text by the `text Serializer`.

bag_delete (*bag*)
Delete `bag` **and** the `tiddlers` within from the system.

bag_get (*bag*)
Fill `bag` with data from the store.

bag_put (*bag*)
Put `bag` into the store.

list_bag_tiddlers (*bag*)
List all the `tiddlers` in the provided `bag`.

list_bags ()
List all the `bags` in the store.

list_recipes ()
List all the `recipes` in the store.

list_tiddler_revisions (*tiddler*)
List all the revisions of one `tiddler`, returning a list of ints.

list_users ()
List all the `users` in the store.

recipe_delete (*recipe*)
Remove a `recipe`, irrevocably, from the system. No impact on `tiddlers`.

recipe_get (*recipe*)
Fill `recipe` with data in the store.

recipe_put (*recipe*)
Put `recipe` into the store.

search (*search_query*)
Search in the store for `tiddlers` that match `search_query`. This is intentionally implemented as a simple and limited grep through files.

tiddler_delete (*tiddler*)
Irrevocably remove `tiddler` from the filesystem.

tiddler_get (*tiddler*)
Fill `tiddler` with data from the store.

tiddler_put (*tiddler*)
Write a `tiddler` into the store. We only write if the tiddler's `bag` already exists. Bag creation is a separate action.

user_delete (*user*)
Delete `user` from the store.

user_get (*user*)
Fill `user` with data from the store.

user_put (*user*)
Put `user` data into the store.

web Package

`web` Package

The routines, modules, etc. that are involved in the presentation and handling of content over HTTP.

These are the parts that makes it TiddlyWeb, not Tiddly something else.

`challenge` Module

WSGI App for running the base challenge system, which lists and links to the available `challengers`. If there is only one challenger, redirect to it.

`tiddlyweb.web.challenge.`**base** (*environ*, *start_response*)
The basic listing page that shows all available `challenger systems`. If there is only one challenger, we redirect to that instead of listing.

`tiddlyweb.web.challenge.`**challenge_get** (*environ*, *start_response*)
Dispatch a `GET` request to the chosen `challenger`.

`tiddlyweb.web.challenge.`**challenge_post** (*environ*, *start_response*)
Dispatch a `POST` request to the chosen `challenger`.

`extractor` Module

Extract of user credentials from incoming web requests. `UserExtract` passes to a stack of extractors. If an `extractor` returns something other than `None`, we have found valid data with which to set `tiddlyweb.usersign`.

class `tiddlyweb.web.extractor.`**UserExtract** (*application*)
Bases: `object`

WSGI Middleware to set the `tiddlyweb.usersign`, if it can be found in the request.

`listentities` Module

Common code used for listing *`bags`* and *`recipes`* in HTTP responses.

`tiddlyweb.web.listentities.`**`list_entities`**(*environ*, *start_response*, *method_name*, *store_list=None*, *serializer_list=None*)

Get an optionally *`filtered`* list of all the *`bags`* or *`recipes`* the current `tiddlyweb.usersign` can read.

`negotiate` Module

WSGI Middleware to do a limited version of content negotiation and put the type in `tiddlyweb.type`. On `GET` and `HEAD` requests the `Accept` header is examined. On `POST` and `PUT`, `Content-Type`. If extensions are provided on a URI used in a `GET` request if the extension matches something in `extension_types` in *`config`*, the type indicated by the extension wins over the Accept header.

class `tiddlyweb.web.negotiate.`**`Negotiate`**(*application*)

Bases: `object`

Perform a form of content negotiation to provide information to the WSGI environment that will later be used to choose serializers.

`tiddlyweb.web.negotiate.`**`figure_type`**(*environ*)

Determine either the `Content-Type` (for `POST` and `PUT`) or `Accept` header (for `GET`) and put that information in `tiddlyweb.type` in the WSGI environment.

`query` Module

WSGI Middleware that extracts `CGI` parameters from the `QUERY_STRING` and puts them in `tiddlyweb.query` in the environ in the same structure that cgi.py uses (dictionary of lists). If the current request is a `POST` of HTML form data, parse that too.

class `tiddlyweb.web.query.`**`Query`**(*application*)

Bases: `object`

Extract `CGI` parameter data from `QUERY_STRING` and POSTed form data.

`extract_query`(*environ*)

Read the `QUERY_STRING` and body (if a POSTed form) to extract query parameters. Put the results in `tiddlyweb.query` in environ. The query names and values are decoded from UTF-8 to unicode.

If there are file uploads in posted form data, the files are not put into `tiddlyweb.query`. Instead the file handles are appended to `tiddlyweb.input_files`.

`sendentity` Module

Send a *`bag`* or *`recipe`* out over HTTP, first *`serializing`* to the correct type.

This consolidates common code for bags and recipes.

`tiddlyweb.web.sendentity.`**`send_entity`**(*environ*, *start_response*, *entity*)

Send a *`bag`* or *`recipe`* out over HTTP, first *`serializing`* to the correct type. If an incoming `Etag` validates, raise a `304` response.

`sendtiddlers` Module

Routines related to sending a list of *tiddlers* out to the web, including optionally *filtering* those tiddlers and validating cache-oriented request headers.

`tiddlyweb.web.sendtiddlers.`**`send_tiddlers`**(*environ*, *start_response*, *tiddlers=None*)
Output the *tiddlers* contained in the provided *Tiddlers collection* in a *Negotiated* representation.

`serve` Module

Functions and Classes for running a TiddlyWeb server, including optionally a built in web server.

class `tiddlyweb.web.serve.`**`Configurator`**(*application*, *config*)
Bases: `object`

WSGI middleware to set `tiddlyweb.config` in `environ` for every request from *config*.

class `tiddlyweb.web.serve.`**`RequestStarter`**(*application*)
Bases: `object`

WSGI middleware that logs basic request information and cleans `PATH_INFO` in the environment.

`PATH_INFO` cleaning is done to ensure that there is a well known encoding of special characters and to support `/` in entity names (see *clean_path_info()*).

`clean_path_info`(*environ*)
Clean `PATH_INFO` in the environment.

This is necessary because WSGI servers tend to decode the URI before putting it in `PATH_INFO`. This means that uri encoded data, such as the `%2F` encoding of `/` will be decoded before we get to route dispatch handling, by which time the `/` is treated as a separator. People say that the right thing to do here is not use `%2F`. This is hogwash. The right thing to do is not decode `PATH_INFO`. In this solution if `REQUEST_URI` is present we use a portion of it to set `PATH_INFO`.

`tiddlyweb.web.serve.`**`load_app`**(*app_prefix=None*, *dirname=None*)
Create our application from a series of layers. The innermost layer is a Selector application based on `urls_map` defined in *config*. This is surrounded by wrappers, which either set something in the environment, modify the request, or transform the response. The wrappers are WSGI middleware defined by `server_request_filters` and `server_response_filters` in *tiddlyweb.config*.

`tiddlyweb.web.serve.`**`start_server`**(*config*)
Start a simple webserver, from `wsgiref`, to run our app.

`util` Module

General utility routines shared by various web related modules.

`tiddlyweb.web.util.`**`bag_etag`**(*environ*, *bag*)
Construct an etag for a *bag*.

`tiddlyweb.web.util.`**`bag_url`**(*environ*, *bag*, *full=True*)
Construct a URL for a *bag*.

`tiddlyweb.web.util.`**`check_bag_constraint`**(*environ*, *bag*, *constraint*)
Check to see if the provided *bag* allows the current `tiddlyweb.usersign` to perform the action described by `constraint`. Lets NoBagError raise if the bag does not exist.

This is a web util because user and store come from the WSGI environ.

`tiddlyweb.web.util.`**`check_incoming_etag`**(*environ*, *etag_string*, *cache_control='no-cache'*, *last_modified=None*, *vary='Accept'*)
Raise 304 if the provided `etag_string` is the same as that found in the `If-None-Match` header of the incoming request.

Return `incoming_etag` to indicate if an etag was there but did not match.

`tiddlyweb.web.util.`**`check_last_modified`**(*environ*, *last_modified_string*, *etag=''*, *cache_control='no-cache'*, *vary='Accept'*)
Raise `304` if an `If-Modified-Since` header matches `last_modified_string`.

`tiddlyweb.web.util.`**`content_length_and_type`**(*environ*)
For `PUT` or `POST` request there must be `Content-Length` and `Content-Type` headers. Raise `400` if not present in the request.

`tiddlyweb.web.util.`**`datetime_from_http_date`**(*http_datestring*)
Turn an HTTP formatted date into a datetime object. Return `None` if the date string is invalid.

`tiddlyweb.web.util.`**`encode_name`**(*name*)
Encode a unicode value as utf-8 and then URL encode that string. Use for entity titles in URLs.

`tiddlyweb.web.util.`**`entity_etag`**(*environ*, *entity*)
Construct an Etag from the digest of the *`JSON`* reprepresentation of an entity.

The JSON representation provides a reasonably repeatable and unique string of data.

`tiddlyweb.web.util.`**`escape_attribute_value`**(*text*)
Escape common HTML character entities, including double quotes in attribute values

This assumes values are enclosed in double quotes (key="value").

`tiddlyweb.web.util.`**`get_route_value`**(*environ*, *name*)
Retrieve and decode `name` from data provided in WSGI route.

If `name` is not present in the route, allow KeyError to raise.

`tiddlyweb.web.util.`**`get_serialize_type`**(*environ*, *collection=False*, *accept_type=False*)
Look in the `environ` to determine which *`serializer`* should be used for this request.

If `collection` is `True`, then the presence of an extension on the URI which does not match any serializer should lead to a `415`.

`tiddlyweb.web.util.`**`handle_extension`**(*environ*, *resource_name*)
Look for an extension (as defined in *`config`*) on the provided `resource_name` and trim it off to give the "real" resource name.

`tiddlyweb.web.util.`**`html_encode`**(*text*)
Encode `&`, < and > entities in `text` that will be used in or as HTML.

`tiddlyweb.web.util.`**`html_frame`**(*environ*, *title=''*)
Return the header and footer from the current HTML *`serialization`*.

`tiddlyweb.web.util.`**`http_date_from_timestamp`**(*timestamp*)
Turn a modifier or created tiddler `timestamp` into a properly formatted HTTP date. If the timestamp is invalid use the current time as the timestamp.

`tiddlyweb.web.util.`**`make_cookie`**(*name*, *value*, *mac_key=None*, *path=None*, *expires=None*, *httponly=True*, *domain=None*)
Create a cookie string, optionally with a MAC, path and expires value. If `expires` is provided, its value should be in seconds.

`tiddlyweb.web.util.`**`read_request_body`**(*environ*, *length*)
Read the `wsgi.input` handle to get the request body.

Length is a required parameter because it is tested for existence earlier in the process.

`tiddlyweb.web.util.`**`recipe_etag`**(*environ*, *recipe*)
Construct an etag for a *`recipe`*.

`tiddlyweb.web.util.`**`recipe_url`**(*environ*, *recipe*, *full=True*)
Construct a URL for a *`recipe`*.

`tiddlyweb.web.util.`**`server_base_url`**(*environ*)
Using information in *`tiddlyweb.config`*, construct the base URL of the server, without the trailing `/`.

`tiddlyweb.web.util.`**`server_host_url`**(*environ*)
Generate the scheme and host portion of our server url.

`tiddlyweb.web.util.`**`tiddler_etag`**(*environ*, *tiddler*)
Construct an etag for a *`tiddler`* from the tiddler's attributes, but not its text.

`tiddlyweb.web.util.`**`tiddler_url`**(*environ*, *tiddler*, *container='bags'*, *full=True*)
Construct a URL for a *`tiddler`*.

`validator` Module

A collection of routines for validating, santizing and otherwise messing with content coming in from the web to be `tiddlers`, *`bags`* or *`recipes`*.

The validators can be extended by adding functions to the `BAG_VALIDATORS`, `RECIPE_VALIDATORS` and `TIDDLER_VALIDATORS`. The functions take an entity object, and an optional WSGI `environ` dict.

exception `tiddlyweb.web.validator.`**`InvalidBagError`**
Bases: `exceptions.Exception`

The provided *`bag`* has not passed a validation routine and has been rejected. The caller should stop processing and return an error to calling code or user-agent.

exception `tiddlyweb.web.validator.`**`InvalidRecipeError`**
Bases: `exceptions.Exception`

The provided *`recipe`* has not passed a validation routine and has been rejected. The caller should stop processing and return an error to calling code or user-agent.

exception `tiddlyweb.web.validator.`**`InvalidTiddlerError`**
Bases: `exceptions.Exception`

The provided *`tiddler`* has not passed a validation routine and has been rejected. The caller should stop processing and return an error to calling code or user-agent.

`tiddlyweb.web.validator.`**`sanitize_desc`**(*entity*, *environ*)
Strip any dangerous HTML which may be present in a *`bag`* or *`recipe`* description.

`tiddlyweb.web.validator.`**`sanitize_html_fragment`**(*fragment*)
Santize an HTML `fragment`, returning a copy of the fragment that has been cleaned up.

`tiddlyweb.web.validator.`**`validate_bag`**(*bag*, *environ=None*)
Pass the *`bag`* to each of the functions in `BAG_VALIDATORS`, in order, either changing the content of the bags's attributes, or if some aspect of the bag can not be accepted raising *`InvalidBagError`*.

`BAG_VALIDATORS` may be extended by plugins.

`validate_bag` is called whenever a bag is `PUT` via HTTP.

`tiddlyweb.web.validator.`**`validate_recipe`**(*recipe*, *environ=None*)
Pass the *`recipe`* to each of the functions in `RECIPE_VALIDATORS`, in order, either changing the content of the recipes's attributes, or if some aspect of the recipe can not be accepted raising *`InvalidRecipeError`*.

RECIPE_VALIDATORS may be extended by plugins.

validate_recipe is called whenever a recipe is PUT via HTTP.

tiddlyweb.web.validator.**validate_tiddler**(*tiddler*, *environ=None*)

Pass the *tiddler* to each of the functions in TIDDLER_VALIDATORS, in order, either changing the content of the tiddler's attributes, or if some aspect of the tiddler can not be accepted raising *InvalidTiddlerError*.

TIDDLER_VALIDATORS is an empty list which may be extended by plugins.

validate_tiddler is called from *web handlers*, when the accept constraint on the *policy* of the *bag* containing the tiddler does not pass.

wsgi Module

WSGI Middleware apps that haven't gotten around to being extracted to their own modules.

class tiddlyweb.web.wsgi.**EncodeUTF8**(*application*)

Bases: object

WSGI Middleware to ensure that the unicode content sent out the pipe is encoded to UTF-8. Within the application string-based content is *unicode* (i.e. not encoded).

class tiddlyweb.web.wsgi.**Header**(*application*)

Bases: object

If REQUEST_METHOD is HEAD, change it internally to GET and consume the generated output so the response has no body.

class tiddlyweb.web.wsgi.**PermissionsExceptor**(*application*)

Bases: object

Trap *permissions exceptions* and turn them into HTTP exceptions so the errors are propagated to clients.

class tiddlyweb.web.wsgi.**SimpleLog**(*application*)

Bases: object

WSGI Middleware to write a very simple log to stdout.

Borrowed from Paste Translogger

format = '%(REMOTE_ADDR)s - %(REMOTE_USER)s [%(time)s] "%(REQUEST_METHOD)s %(REQUEST_URI)

write_log(*environ*, *req_uri*, *status*, *size*)

Write the log info out in a formatted form to logging.info.

This is rather more complex than desirable because there is a mix of str and unicode in the gathered data and it needs to be made acceptable for output.

class tiddlyweb.web.wsgi.**StoreSet**(*application*)

Bases: object

WSGI Middleware that sets our choice of *Store* in the environ. That is, initialize the store for each request.

class tiddlyweb.web.wsgi.**TransformProtect**(*application*)

Bases: object

WSGI Middleware to add a Cache-Control: no-transform` header so that mobile companies that transcode content over their 3G (etc) networks don't, as it will break various JavaScript things, including TiddlyWiki.

Subpackages

challengers Package

`challengers` Package

The ChallengerInterface class.

class `tiddlyweb.web.challengers.`**`ChallengerInterface`**

Bases: `object`

An interface for challenging users for authentication purposes. The chalenger basically does whatever is required and *may* result in doing something to a response that causes the user agent's next request to pass an *`extractor`*.

Though there is no requirement for there to be a one to one correspondence between a Challenger and an *`Extractor`*, it will often be the case that a Challenger will need a particular Extractor in order to be effective.

A Challenger is a WSGI application.

`challenge_get`(*environ*, *start_response*)

Respond to a `GET` request.

`challenge_post`(*environ*, *start_response*)

Respond to a `POST` request.

`cookie_form` Module

A *`challenger`* that presents or validates a form for getting a username and password.

class `tiddlyweb.web.challengers.cookie_form.`**`Challenger`**

Bases: *`tiddlyweb.web.challengers.ChallengerInterface`*

A simple login challenger that asks the user agent, via an HTML form, for a username and password and vaidates it against a *`User entity`* in the *`store`*.

If valid, a cookie is set in the response. This is used in subsequent requests by the *`simple_cookie credentials extractor`*.

`challenge_get`(*environ*, *start_response*)

Respond to a `GET` request by sending a form.

`challenge_post`(*environ*, *start_response*)

Respond to a `POST` by processing data sent from a form. The form should include a username and password. If it does not, send the form aagain. If it does, validate the data.

`desc` = 'TiddlyWeb username and password'

extractors Package

`extractors` Package

The ExtractorInterface class, used to extract and validate information in web requests that may identify a user. Often, but not always, that information was originally created by a *`challenger`*.

class `tiddlyweb.web.extractors.`**`ExtractorInterface`**

Bases: `object`

An interface for user extraction.

Given a WSGI environ, figure out if the request contains information which can be used to identify a valid user. If it does, return a dict including information about that user.

If it doesn't return *False*.

`extract`(*environ*, *start_response*)
Look at the incoming request and try to extract a user.

`load_user`(*environ*, *usersign*)
Check the *`User`* database in the *`store`* for a user matching this usersign. The user is not required to exist, but if it does it can be used to get additional information about the user, such as roles.

`http_basic` Module

A very simple *`extractor`* that looks at the HTTP `Authorization` header and looks for Basic auth information therein.

class `tiddlyweb.web.extractors.http_basic.`**`Extractor`**
Bases: *`tiddlyweb.web.extractors.ExtractorInterface`*

An *`extractor`* for HTTP Basic Authentication. If there is an *Authorization* header attempt to get a username and password out of it and compare with *`User`* information in the *`Store`*. If the password is valid, return the user information. Otherwise return `False`.

`extract`(*environ*, *start_response*)
Look in the request for an `Authorization` header.

`simple_cookie` Module

An *`extractor`* that looks at a cookie named `tiddlyweb_user`.

class `tiddlyweb.web.extractors.simple_cookie.`**`Extractor`**
Bases: *`tiddlyweb.web.extractors.ExtractorInterface`*

Look in the headers for a cookie named `tiddlyweb_user`.

If it is there and the associated hashed value validates against a server side secret, return the indicated user.

`extract`(*environ*, *start_response*)
Extract the cookie, if there, from the headers and attempt to validate its contents.

handler Package

`handler` Package

Convenience routines for presenting the root of the web server.

Here because nowhere else seems right.

`tiddlyweb.web.handler.`**`root`**(*environ*, *start_response*)
Convenience application to provide an entry point at root.

`bag` Module

Methods for accessing *`Bag`* entities.

`tiddlyweb.web.handler.bag.`**`delete`**`(`*environ*, *start_response*`)`

Handle `DELETE` on a single bag URI.

Remove the *`bag`* and the *`tiddlers`* within from the *`store`*.

How the store chooses to handle remove and what it means is up to the store.

`tiddlyweb.web.handler.bag.`**`get`**`(`*environ*, *start_response*`)`

Handle `GET` on a single bag URI.

Get a representation in some serialization determined by *`tiddlyweb.web.negotiate`* of a *`bag`* (the bag itself, not the tiddlers within).

`tiddlyweb.web.handler.bag.`**`get_tiddlers`**`(`*environ*, *start_response*`)`

Handle `GET` on a tiddlers-within-a-bag URI.

Get a list representation of the *`tiddlers`* in a *`bag`*.

The information sent is dependent on the serialization chosen via *`tiddlyweb.web.negotiate`*.

`tiddlyweb.web.handler.bag.`**`list_bags`**`(`*environ*, *start_response*`)`

Handle `GET` on the bags URI.

List all the *`bags`* that are readable by the current usersign.

The information sent is dependent on the serialization chosen via *`tiddlyweb.web.negotiate`*.

`tiddlyweb.web.handler.bag.`**`put`**`(`*environ*, *start_response*`)`

Handle `PUT` on a single bag URI.

Put a *`bag`* to the server, meaning the description and policy of the bag, if *`policy`* allows.

`chronicle` Module

A chronicle is a stack of *`tiddlers`*, usually revisions of one tiddler. By POSTing a chronicle of tiddlers originally named A to tiddler B, it is possible to rename a tiddler while preserving revision history.

`tiddlyweb.web.handler.chronicle.`**`post_revisions`**`(`*environ*, *start_response*`)`

Handle a `POST` of a chronicle of *`tiddlers`* at a tiddler revisions URI.

Take a collection of `JSON` tiddlers, each with a text key and value, and process them into the store.

`recipe` Module

Methods for accessing *`Recipe`* entities.

`tiddlyweb.web.handler.recipe.`**`delete`**`(`*environ*, *start_response*`)`

Handle `DELETE` on a single recipe URI.

Delete a *`recipe`*. This just removes the recipe, not any associated *`bags`* or *`tiddlers`*.

`tiddlyweb.web.handler.recipe.`**`get`**`(`*environ*, *start_response*`)`

Handle `GET` on a single recipe URI.

Get a representation in some serialization determined by *`tiddlyweb.web.negotiate`* of a *`recipe`* (just the recipe itself, not the tiddlers it can produce).

`tiddlyweb.web.handler.recipe.`**`get_tiddlers`**`(`*environ*, *start_response*`)`

Handle `GET` on a tiddlers-within-a-recipe URI.

Get a list representation of the *`tiddlers`* generated from a *`recipe`*.

The information sent is dependent on the serialization chosen via `tiddlyweb.web.negotiate`.

`tiddlyweb.web.handler.recipe.`**`list_recipes`**`(`*environ*, *start_response*`)`
Handle `GET` on the recipes URI.

List all the `recipes` that are readable by the current usersign.

The information sent is dependent on the serialization chosen via `tiddlyweb.web.negotiate`.

`tiddlyweb.web.handler.recipe.`**`put`**`(`*environ*, *start_response*`)`
Handle `PUT` on a single recipe URI.

Put a `recipe` to the server, meaning the description, policy and recipe list of the recipe, if `policy` allows.

`search` Module

Handle searches for `tiddlers` if the configured `store` supports search.

`tiddlyweb.web.handler.search.`**`get`**`(`*environ*, *start_response*`)`
Handle `GET` on the search URI.

Perform a search against the `store`.

What search means and what results are returned is dependent on the search implementation (if any) in the `chosen store`.

`tiddlyweb.web.handler.search.`**`get_search_query`**`(`*environ*`)`
Inspect `tiddlyweb.query` in the environment to find the search query in a parameter named `q`.

`tiddlyweb.web.handler.search.`**`get_tiddlers`**`(`*environ*`)`
Call search in the `store` to get the generator of `tiddlers` matching the query found by `get_search_query()`.

`tiddler` Module

Methods for accessing `Tiddler` entities.

`tiddlyweb.web.handler.tiddler.`**`delete`**`(`*environ*, *start_response*`)`
Handle `DELETE` on a single tiddler URI.

Delete a `tiddler` from the `store`.

What delete means is up to the store.

`tiddlyweb.web.handler.tiddler.`**`get`**`(`*environ*, *start_response*`)`
Handle `GET` on a single tiddler or tiddler revision URI.

Get a representation in some serialization determined by `tiddlyweb.web.negotiate` of a `tiddler`.

`tiddlyweb.web.handler.tiddler.`**`get_revisions`**`(`*environ*, *start_response*`)`
Handle `GET` on the collection of revisions of single tiddler URI.

Get a list representation in some serialization determined by `tiddlyweb.web.negotiate` of the revisions of a `tiddler`.

`tiddlyweb.web.handler.tiddler.`**`put`**`(`*environ*, *start_response*`)`
Handle `PUT` on a single tiddler URI.

Put a `tiddler` to the server.

`tiddlyweb.web.handler.tiddler.`**`validate_tiddler_headers`**(*environ*, *tiddler*)

Check ETag and last modified header information to see if a) on `GET` the user agent can use its cached tiddler b) on `PUT` we have edit contention.

wikitext Package

`wikitext` Package

Functions for rendering any *`tiddler`* that has been identified as wikitext into the rendered form (usually HTML) of that wikitext.

Wikitext rendering is engaged when a tiddler is requested via a `GET`, when the negotiated media-type of the request is html, and when `tiddler.type` is either `None` or in the keys of the dictionary associated with the *`tiddlyweb.config['wikitext.type_render_map']`*.

When `tiddler.type` is `None`, the renderer named in `tiddlyweb.config['wiktext.default_renderer']` is used. This is either a module in the *`tiddlyweb.wikitext`* package, or a module on `sys.path`.

When `tiddler.type` is something other than `None`, the renderer is determined by looking up the type in `tiddlyweb.config['wikitext.type_render_map']`. The found value is a module of the same type described above.

The renderer module has a function `render`.

`tiddlyweb.wikitext.`**`render_wikitext`**(*tiddler=None*, *environ=None*)

Take a *`tiddler`* and render wikitext in `tiddler.text` to some kind of HTML format.

`raw` Module

A default simple wikitext renderer which does not render the wikitext but instead wraps it in `pre` tags.

`tiddlyweb.wikitext.raw.`**`render`**(*tiddler*, *environ*)

Wrap HTML encoded wikitext with `pre` tags.

CHAPTER 10

TiddlyWeb

TiddlyWeb is an open source HTTP API for storing and accessing flexible and composable microcontent. It is also a toolkit for tiddlers on the web and a robust server side for TiddlyWiki.

TiddlyWeb by itself provides the base HTTP API, storage engine and default serializations. A large variety of plugins provide additional functionality.

Quick Start

The quickest way to get going with an operational installation of TiddlyWeb is to install tiddlywebwiki.

See the quick start documentation for that.

Additional Documentation

Besides package documentation starting at *tiddlyweb Package*, additional documentation can be found at http://tiddlyweb.com and http://docs.tiddlyweb.com/.

Note that all of this documentation is in a constant state of flux, as it should be. If you find an error please help to fix it.

Source

The TiddlyWeb source is kept at GitHub.

- genindex
- modindex
- search

Python Module Index

Index