Trac API Documentation

Release 1.3.3.dev0

The Trac Team

September 14, 2017

Contents

Release 1.3.3.dev0

Date September 14, 2017

This is work in progress. The API is not yet fully covered, but what you'll find here should be accurate, otherwise it's a bug and you're welcome to open a ticket for reporting the problem.

CHAPTER 1

Content

API Reference

trac.about

class trac.about.**AboutModule**
Bases: *trac.core.Component*

"About Trac" page provider, showing version information from third-party packages, as well as configuration information.

trac.admin.api – Trac Administration panels

Primary interface for managing administration panels.

Interfaces

class trac.admin.api.**IAdminPanelProvider**
Bases: *trac.core.Interface*

Extension point interface for adding panels to the web-based administration interface.

See also trac.admin.api.IAdminPanelProvider extension point

get_admin_panels(*req*)
Return a list of available admin panels.

The items returned by this function must be tuples of the form `(category, category_label, page, page_label)`.

render_admin_panel(*req*, *category*, *page*, *path_info*)
Process a request for an admin panel.

This function should return a tuple of the form `(template, data)`, where `template` is the name of the template to use and `data` is the data to use when rendering the template.

Note: When a plugin wants to use a legacy Genshi template instead of a Jinja2 template, it needs to return instead a *triple* of the form `(template, data, None)`, similar to what *IRequestHandler.process_request* does.

---

class trac.admin.api.**IAdminCommandProvider**
Bases: *trac.core.Interface*

Extension point interface for adding commands to the console administration interface `trac-admin`.

See also trac.admin.api.IAdminCommandProvider extension point

get_admin_commands()
Return a list of available admin commands.

The items returned by this function must be tuples of the form `(command, args, help, complete, execute)`, where `command` contains the space-separated command and sub-command names, `args` is a string describing the command arguments and `help` is the help text. The first paragraph of the help text is taken as a short help, shown in the list of commands.

`complete` is called to auto-complete the command arguments, with the current list of arguments as its only argument. It should return a list of relevant values for the last argument in the list.

`execute` is called to execute the command, with the command arguments passed as positional arguments.

Exceptions

class trac.admin.api.**AdminCommandError**(*msg*, *show_usage=False*, *cmd=None*)
Bases: *trac.core.TracError*

Exception raised when an admin command cannot be executed.

Components

class trac.admin.api.**AdminCommandManager**
Bases: *trac.core.Component*

trac-admin command manager.

complete_command(*args*, *cmd_only=False*)
Perform auto-completion on the given arguments.

execute_command(**args*)
Execute a command given by a list of arguments.

get_command_help(*args=[]*)
Return help information for a set of commands.

providers
List of components that implement *IAdminCommandProvider*

Classes

class trac.admin.api.**PathList**
Bases: `list`

A list of paths for command argument auto-completion.

complete(*text*)
Return the items in the list matching text.

class trac.admin.api.**PrefixList**
Bases: list

A list of prefixes for command argument auto-completion.

Helper Functions

trac.admin.api.**get_console_locale**(*env=None*, *lang=None*, *categories=('LANGUAGE', 'LC_ALL', 'LC_MESSAGES', 'LANG')*)
Return negotiated locale for console by locale environments and [trac] default_language.

trac.admin.api.**get_dir_list**(*path*, *dirs_only=False*)
Return a list of paths to filesystem entries in the same directory as the given path.

trac.admin.console

trac.admin.console.**run**(*args=None*)
Main entry point.

trac.admin.web_ui

class trac.admin.web_ui.**AdminModule**
Bases: *trac.core.Component*

Web administration interface provider and panel manager.

panel_providers
List of components that implement *IAdminPanelProvider*

trac.attachment – Attachments for Trac resources

This module contains the *Attachment* model class and the *AttachmentModule* component which manages file attachments for any kind of Trac resources. Currently, the wiki pages, tickets and milestones all support file attachments. You can use the same utility methods from the *AttachmentModule* as they do for easily adding attachments to other kinds of resources.

See also the attach_file_form.html and attachment.html templates which can be used to display the attachments.

Interfaces

class trac.attachment.**IAttachmentChangeListener**
Bases: *trac.core.Interface*

Extension point interface for components that require notification when attachments are created, deleted, renamed or reparented.

See also trac.attachment.IAttachmentChangeListener extension point

attachment_added(*attachment*)
Called when an attachment is added.

attachment_deleted(*attachment*)
Called when an attachment is deleted.

attachment_moved(*attachment*, *old_parent_realm*, *old_parent_id*, *old_filename*)
Called when an attachment is moved.

attachment_reparented(*attachment*, *old_parent_realm*, *old_parent_id*)
Called when an attachment is reparented.

Since 1.3.2 deprecated and will be removed in 1.5.1. Use *attachment_moved* instead.

class trac.attachment.**IAttachmentManipulator**
Bases: *trac.core.Interface*

Extension point interface for components that need to manipulate attachments.

Unlike change listeners, a manipulator can reject changes being committed to the database.

See also trac.attachment.IAttachmentManipulator extension point

prepare_attachment(*req*, *attachment*, *fields*)
Not currently called, but should be provided for future compatibility.

validate_attachment(*req*, *attachment*)
Validate an attachment after upload but before being stored in Trac environment.

Must return a list of `(field, message)` tuples, one for each problem detected. `field` can be any of `description`, `username`, `filename`, `content`, or `None` to indicate an overall problem with the attachment. Therefore, a return value of `[]` means everything is OK.

class trac.attachment.**ILegacyAttachmentPolicyDelegate**
Bases: *trac.core.Interface*

Interface that can be used by plugins to seamlessly participate to the legacy way of checking for attachment permissions.

This should no longer be necessary once it becomes easier to setup fine-grained permissions in the default permission store.

See also trac.attachment.ILegacyAttachmentPolicyDelegate extension point

check_attachment_permission(*action*, *username*, *resource*, *perm*)
Return the usual `True`/`False`/`None` security policy decision appropriate for the requested action on an attachment.

param action one of ATTACHMENT_VIEW, ATTACHMENT_CREATE, ATTACHMENT_DELETE

param username the user string

param resource the *Resource* for the attachment. Note that when ATTACHMENT_CREATE is checked, the resource `.id` will be `None`.

param perm the permission cache for that username and resource

Classes

class trac.attachment.**Attachment**(*env*, *parent_realm_or_attachment_resource*, *parent_id=None*, *filename=None*)
Bases: `object`

Represents an attachment (new or existing).

delete()
Delete the attachment, both the record in the database and the file itself.

classmethod delete_all(*env*, *parent_realm*, *parent_id*)
Delete all attachments of a given resource.

insert(*filename*, *fileobj*, *size*, *t=None*)
Create a new Attachment record and save the file content.

move(*new_realm=None*, *new_id=None*, *new_filename=None*)
Move the attachment, changing one or more of its parent realm, parent id and filename.

The new parent resource must exist.

Since 1.3.2

reparent(*new_realm*, *new_id*)
Change the attachment's `parent_realm` and/or `parent_id`

Since 1.3.2 deprecated and will be removed in 1.5.1. Use the *move* method instead.

classmethod reparent_all(*env*, *parent_realm*, *parent_id*, *new_realm*, *new_id*)
Reparent all attachments of a given resource to another resource.

classmethod select(*env*, *parent_realm*, *parent_id*)
Iterator yielding all *Attachment* instances attached to resource identified by `parent_realm` and `parent_id`.

Returns a tuple containing the `filename`, `description`, `size`, `time` and `author`.

exception trac.attachment.**InvalidAttachment**(*message*, *title=None*, *show_traceback=False*)
Bases: *trac.core.TracError*

Exception raised when attachment validation fails.

Since 1.3.2 deprecated and will be removed in 1.5.1

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

Components

class trac.attachment.**AttachmentModule**
Bases: *trac.core.Component*

attachment_data(*context*)
Return a data dictionary describing the list of viewable attachments in the current context.

change_listeners
List of components that implement *IAttachmentChangeListener*

get_history(*start*, *stop*, *realm*)
Return an iterable of tuples describing changes to attachments on a particular object realm.

The tuples are in the form (change, realm, id, filename, time, description, author). `change` can currently only be `created`.

FIXME: no iterator

get_resource_url(*resource*, *href*, ***kwargs*)
Return an URL to the attachment itself.

A `format` keyword argument equal to `'raw'` will be converted to the raw-attachment prefix.

get_search_results(*req*, *resource_realm*, *terms*)
Return a search result generator suitable for ISearchSource.

Search results are attachments on resources of the given `resource_realm.realm` whose filename, description or author match the given terms.

get_timeline_events(*req*, *resource_realm*, *start*, *stop*)
Return an event generator suitable for ITimelineEventProvider.

Events are changes to attachments on resources of the given `resource_realm.realm`.

manipulators
List of components that implement *IAttachmentManipulator*

max_size
Maximum allowed file size (in bytes) for attachments.

max_zip_size
Maximum allowed total size (in bytes) for an attachment list to be downloadable as a `zip`. Set this to -1 to disable download as `zip`. (''since 1.0'')

render_unsafe_content
Whether attachments should be rendered in the browser, or only made downloadable.

Pretty much any file may be interpreted as HTML by the browser, which allows a malicious user to attach a file containing cross-site scripting attacks.

For public sites where anonymous users can create attachments it is recommended to leave this option disabled.

viewable_attachments(*context*)
Return the list of viewable attachments in the given context.

Parameters **context** – the *RenderingContext* corresponding to the parent *Resource* for the attachments

class `trac.attachment.`**AttachmentAdmin**
Bases: *trac.core.Component*

trac-admin command provider for attachment administration.

`trac.cache` – Control of cached data coherency

Trac is a server application which may involve multiple concurrent processes. The coherency of the data presented to the clients is ensured by the underlying database and its transaction handling. However, a server process will not systematically retrieve data from the database, as various in-memory caches are used for performance reasons. We could ensure the integrity of those caches in a single process in presence of multiple threads by the appropriate use of locking and by updating the caches as needed, but we also need a mechanism for invalidating the caches in the *other* processes.

The purpose of this module is to provide a `cached` decorator which can annotate a data *retriever* method of a class for turning it into an attribute working like a cache. This means that an access to this attribute will only call the underlying retriever method once on first access, or only once after the cache has been invalidated, even if this invalidation happened in another process.

Public API

trac.cache.**cached**(*fn_or_attr=None*)

Method decorator creating a cached attribute from a data retrieval method.

Accessing the cached attribute gives back the cached value. The data retrieval method is transparently called by the *CacheManager* on first use after the program start or after the cache has been invalidated. Invalidating the cache for this value is done by deleting the attribute.

Note that the cache validity is maintained using the cache table in the database. Cache invalidation is performed within a transaction block, and can be nested within another transaction block.

When the decorator is used in a class for which instances behave as singletons within the scope of a given *Environment* (typically *Component* classes), the key used to identify the attribute in the database is constructed from the names of the containing module, class and retriever method:

```
class WikiSystem(Component):
    @cached
    def pages(self):
        return {name for name, in self.env.db_query(
                    "SELECT DISTINCT name FROM wiki")}
```

Otherwise, when the decorator is used in non-"singleton" objects, a string specifying the name of an attribute containing a string unique to the instance must be passed to the decorator. This value will be appended to the key constructed from module, class and method name:

```
class SomeClass(object):
    def __init__(self, env, name):
        self.env = env
        self.name = name
        self._metadata_id = name

    @cached('_metadata_id')
    def metadata(self):
        ...
```

Note that in this case the key attribute is overwritten with a hash of the key on first access, so it should not be used for any other purpose.

In either case, this decorator requires that the object on which it is used has an env attribute containing the application *Environment*.

Changed in version 1.0: The data retrieval method used to be called with a single argument db containing a reference to a database connection. This is the same connection that can be retrieved via the normal *db_query* or *db_transaction*, so this is no longer needed and is not supported.

Internal API

class trac.cache.**CacheManager**

Bases: *trac.core.Component*

Cache manager.

get(*id*, *retriever*, *instance*)

Get cached or fresh data for the given id.

invalidate(*id*)

Invalidate cached data for the given id.

reset_metadata()
Reset per-request cache metadata.

The following classes are the descriptors created by the *cached* decorator:

class trac.cache.**CachedSingletonProperty**(*retriever*)
Bases: *trac.cache.CachedPropertyBase*

Cached property descriptor for classes behaving as singletons in the scope of one *Environment* instance.

This means there will be no more than one cache to monitor in the database for this kind of cache. Therefore, using only "static" information for the key is enough. For the same reason it is also safe to store the corresponding id as a property of the descriptor instance.

class trac.cache.**CachedProperty**(*retriever*, *key_attr*)
Bases: *trac.cache.CachedPropertyBase*

Cached property descriptor for classes having potentially multiple instances associated to a single *Environment* instance.

As we'll have potentially many different caches to monitor for this kind of cache, the key needs to be augmented by a string unique to each instance of the owner class. As the resulting id will be different for each instance of the owner class, we can't store it as a property of the descriptor class, so we store it back in the attribute used for augmenting the key (`key_attr`).

Both classes inherit from a common base:

class trac.cache.**CachedPropertyBase**(*retriever*)
Bases: property

Base class for cached property descriptors.

Since 1.0.2 inherits from property.

trac.cache.**key_to_id**()

trac.config

class trac.config.**Configuration**(*filename*, *params={}*)
Bases: object

Thin layer over ConfigParser from the Python standard library.

In addition to providing some convenience methods, the class remembers the last modification time of the configuration file, and reparses it when the file has changed.

defaults(*compmgr=None*)
Returns a dictionary of the default configuration values.

If compmgr is specified, return only options declared in components that are enabled in the given ComponentManager.

exists
Return boolean indicating configuration file existence.

Since 1.0.11

get(*section*, *key*, *default=''*)
Return the value of the specified option.

Valid default input is a string. Returns a string.

getbool(*section*, *key*, *default=''*)
Return the specified option as boolean value.

If the value of the option is one of "yes", "true", "enabled", "on", or "1", this method wll return `True`, otherwise `False`.

Valid default input is a string or a bool. Returns a bool.

getfloat(*section*, *key*, *default=''*)
Return the value of the specified option as float.

If the specified option can not be converted to a float, a *ConfigurationError* exception is raised.

Valid default input is a string, float or int. Returns a float.

getint(*section*, *key*, *default=''*)
Return the value of the specified option as integer.

If the specified option can not be converted to an integer, a *ConfigurationError* exception is raised.

Valid default input is a string or an int. Returns an int.

getlist(*section*, *key*, *default=''*, *sep=','*, *keep_empty=False*)
Return a list of values that have been specified as a single comma-separated option.

A different separator can be specified using the `sep` parameter. The `sep` parameter can specify multiple values using a list or a tuple. If the `keep_empty` parameter is set to `True`, empty elements are included in the list.

Valid default input is a string or a list. Returns a string.

getpath(*section*, *key*, *default=''*)
Return a configuration value as an absolute path.

Relative paths are resolved relative to the location of this configuration file.

Valid default input is a string. Returns a normalized path.

has_option(*section*, *option*, *defaults=True*)
Returns True if option exists in section in either the project trac.ini or one of the parents, or is available through the Option registry.

options(*section*, *compmgr=None*)
Return a list of `(name, value)` tuples for every option in the specified section.

This includes options that have default values that haven't been overridden. If `compmgr` is specified, only return default option values for components that are enabled in the given `ComponentManager`.

remove(*section*, *key*)
Remove the specified option.

save()
Write the configuration options to the primary file.

sections(*compmgr=None*, *defaults=True*, *empty=False*)
Return a list of section names.

If `compmgr` is specified, only the section names corresponding to options declared in components that are enabled in the given `ComponentManager` are returned.

Parameters **empty** – If `True`, include sections from the registry that contain no options.

set(*section*, *key*, *value*)
Change a configuration value.

These changes are not persistent unless saved with *save()*.

set_defaults(*compmgr=None*, *component=None*)
Retrieve all default values and store them explicitly in the configuration, so that they can be saved to file.

Values already set in the configuration are not overwritten.

class trac.config.**ConfigSection**(*name*, *doc*, *doc_domain='tracini'*, *doc_args=None*)
Bases: object

Descriptor for configuration sections.

Create the configuration section.

doc
Return localized document of the section

static get_registry(*compmgr=None*)
Return the section registry, as a dict mapping section names to *ConfigSection* objects.

If compmgr is specified, only return sections for components that are enabled in the given ComponentManager.

class trac.config.**Option**(*section*, *name*, *default=None*, *doc=''*, *doc_domain='tracini'*, *doc_args=None*)
Bases: object

Descriptor for configuration options.

Create the configuration option.

Parameters

- **section** – the name of the configuration section this option belongs to
- **name** – the name of the option
- **default** – the default value for the option
- **doc** – documentation of the option

doc
Return localized document of the option

dumps(*value*)
Return the value as a string to write to a trac.ini file

static get_registry(*compmgr=None*)
Return the option registry, as a dict mapping (section, key) tuples to *Option* objects.

If compmgr is specified, only return options for components that are enabled in the given ComponentManager.

normalize(*value*)
Normalize the given value to write to a trac.ini file

class trac.config.**BoolOption**(*section*, *name*, *default=None*, *doc=''*, *doc_domain='tracini'*, *doc_args=None*)
Bases: *trac.config.Option*

Descriptor for boolean configuration options.

Create the configuration option.

Parameters

- **section** – the name of the configuration section this option belongs to
- **name** – the name of the option

- **default** – the default value for the option
- **doc** – documentation of the option

class trac.config.**IntOption**(*section*, *name*, *default=None*, *doc=''*, *doc_domain='tracini'*, *doc_args=None*)

Bases: *trac.config.Option*

Descriptor for integer configuration options.

Create the configuration option.

Parameters

- **section** – the name of the configuration section this option belongs to
- **name** – the name of the option
- **default** – the default value for the option
- **doc** – documentation of the option

class trac.config.**FloatOption**(*section*, *name*, *default=None*, *doc=''*, *doc_domain='tracini'*, *doc_args=None*)

Bases: *trac.config.Option*

Descriptor for float configuration options.

Create the configuration option.

Parameters

- **section** – the name of the configuration section this option belongs to
- **name** – the name of the option
- **default** – the default value for the option
- **doc** – documentation of the option

class trac.config.**ListOption**(*section*, *name*, *default=None*, *sep=','*, *keep_empty=False*, *doc=''*, *doc_domain='tracini'*, *doc_args=None*)

Bases: *trac.config.Option*

Descriptor for configuration options that contain multiple values separated by a specific character.

class trac.config.**ChoiceOption**(*section*, *name*, *choices*, *doc=''*, *doc_domain='tracini'*, *doc_args=None*, *case_sensitive=True*)

Bases: *trac.config.Option*

Descriptor for configuration options providing a choice among a list of items.

The default value is the first choice in the list.

class trac.config.**PathOption**(*section*, *name*, *default=None*, *doc=''*, *doc_domain='tracini'*, *doc_args=None*)

Bases: *trac.config.Option*

Descriptor for file system path configuration options.

Relative paths are resolved to absolute paths using the directory containing the configuration file as the reference.

Create the configuration option.

Parameters

- **section** – the name of the configuration section this option belongs to
- **name** – the name of the option

- **default** – the default value for the option
- **doc** – documentation of the option

class trac.config.**ExtensionOption**(*section*, *name*, *interface*, *default=None*, *doc=''*, *doc_domain='tracini'*, *doc_args=None*)
Bases: *trac.config.Option*

Name of a component implementing `interface`. Raises a *ConfigurationError* if the component cannot be found in the list of active components implementing the interface.

class trac.config.**OrderedExtensionsOption**(*section*, *name*, *interface*, *default=None*, *include_missing=True*, *doc=''*, *doc_domain='tracini'*, *doc_args=None*)
Bases: *trac.config.ListOption*

A comma separated, ordered, list of components implementing `interface`. Can be empty.

If `include_missing` is true (the default) all components implementing the interface are returned, with those specified by the option ordered first.

exception trac.config.**ConfigurationError**(*message=None*, *title=None*, *show_traceback=False*)
Bases: *trac.core.TracError*

Exception raised when a value in the configuration file is not valid.

class trac.config.**UnicodeConfigParser**(*ignorecase_option=True*, ***kwargs*)
Bases: ConfigParser.ConfigParser

A Unicode-aware version of ConfigParser. Arguments are encoded to UTF-8 and return values are decoded from UTF-8.

class trac.config.**Section**(*config*, *name*)
Bases: object

Proxy for a specific configuration section.

Objects of this class should not be instantiated directly.

get(*key*, *default=''*)
Return the value of the specified option.

Valid default input is a string. Returns a string.

getbool(*key*, *default=''*)
Return the value of the specified option as boolean.

This method returns `True` if the option value is one of "yes", "true", "enabled", "on", or non-zero numbers, ignoring case. Otherwise `False` is returned.

Valid default input is a string or a bool. Returns a bool.

getfloat(*key*, *default=''*)
Return the value of the specified option as float.

If the specified option can not be converted to a float, a *ConfigurationError* exception is raised.

Valid default input is a string, float or int. Returns a float.

getint(*key*, *default=''*)
Return the value of the specified option as integer.

If the specified option can not be converted to an integer, a *ConfigurationError* exception is raised.

Valid default input is a string or an int. Returns an int.

getlist(*key*, *default=''*, *sep=','*, *keep_empty=True*)
Return a list of values that have been specified as a single comma-separated option.

A different separator can be specified using the `sep` parameter. The `sep` parameter can specify multiple values using a list or a tuple. If the `keep_empty` parameter is set to `True`, empty elements are included in the list.

Valid default input is a string or a list. Returns a list.

getpath(*key*, *default=''*)
Return the value of the specified option as a path, relative to the location of this configuration file.

Valid default input is a string. Returns a normalized path.

iterate(*compmgr=None*, *defaults=True*)
Iterate over the options in this section.

If `compmgr` is specified, only return default option values for components that are enabled in the given `ComponentManager`.

options(*compmgr=None*)
Return `(key, value)` tuples for every option in the section.

This includes options that have default values that haven't been overridden. If `compmgr` is specified, only return default option values for components that are enabled in the given `ComponentManager`.

remove(*key*)
Delete a key from this section.

Like for *`set()`*, the changes won't persist until `save()` gets called.

set(*key*, *value*)
Change a configuration value.

These changes are not persistent unless saved with `save()`.

`trac.config.`**get_configinfo**(*env*)
Returns a list of dictionaries containing the `name` and `options` of each configuration section. The value of `options` is a list of dictionaries containing the `name`, `value` and `modified` state of each configuration option. The `modified` value is True if the value differs from its default.

Since version 1.1.2

Components

class `trac.config.`**ConfigurationAdmin**
Bases: *`trac.core.Component`*

trac-admin command provider for trac.ini administration.

`trac.core` – the Trac "kernel"

Component model

The Trac component model is very simple, it is based on *`Interface`* classes that are used to document a particular set of methods and properties that must be defined by a *`Component`* subclass when it declares it *implements* that interface.

class trac.core.**Interface**
Bases: object

Marker base class for extension point interfaces.

class trac.core.**Component**
Bases: object

Base class for components.

Every component can declare what extension points it provides, as well as what extension points of other components it extends.

static implements(**interfaces*)
Can be used in the class definition of *Component* subclasses to declare the extension points that are extended.

The static method *Component.implements* is never used as such, but rather via the global *implements* function. This globally registers that particular component subclass as an implementation of the listed interfaces.

trac.core.**implements**(**interfaces*)
Can be used in the class definition of *Component* subclasses to declare the extension points that are extended.

For example:

```
class IStuffProvider(Interface):
    """All interfaces start by convention with an "I" and even if it's
    not a convention, in practice most interfaces are "Provider" of
    something ;-)
    """

    def get_stuff(color=None):
        """We usually don't specify "self" here, but try to describe
        as precisely as possible how the method might be called and
        what is the expected return type."""

class ComponentA(Component):

    implements(IStuffProvider)

    # IStuffProvider methods

    def get_stuff(self, color=None):
        if not color or color == 'yellow':
            yield ('duck', "the regular waterproof plastic duck")
```

The benefit of implementing an interface is to possibility to define an *ExtensionPoint* property for an *Interface*, in a *Component* subclass. Such a property provides a convenient way to retrieve *all* registered and enabled component instances for that interface. The enabling of components is the responsibility of the *ComponentManager*, see *is_component_enabled* below.

class trac.core.**ExtensionPoint**(*interface*)
Bases: property

Marker class for extension points in components.

Create the extension point.

Parameters **interface** – the *Interface* subclass that defines the protocol for the extension point

extensions(*component*)
Return a list of components that declare to implement the extension point interface.

Continuing the example:

```
class StuffModule(Component):

    stuff_providers = ExtensionPoint(IStuffProvider)

    def get_all_stuff(self, color=None):
        stuff = {}
        for provider in self.stuff_provider:
            for name, descr in provider.get_stuff(color) or []:
                stuff[name] = descr
        return stuff
```

Note that besides going through an extension point, *Component* subclass instances can alternatively be retrieved directly by using the instantiation syntax. This is not an usual instantiation though, as this will always return the same instance in the given *ComponentManager* "scope" passed to the constructor:

```
>>> a1 = ComponentA(mgr)
>>> a2 = ComponentA(mgr)
>>> a1 is a2
True
```

The same thing happens when retrieving components via an extension point, the retrieved instances belong to the same "scope" as the instance used to access the extension point:

```
>>> b = StuffModule(mgr)
>>> any(a is a1 for a in b.stuff_providers)
True
```

class trac.core.**ComponentManager**
Bases: object

The component manager keeps a pool of active components.

Initialize the component manager.

component_activated(*component*)
Can be overridden by sub-classes so that special initialization for components can be provided.

disable_component(*component*)
Force a component to be disabled.

Parameters **component** – can be a class or an instance.

enable_component(*component*)
Force a component to be enabled.

Parameters **component** – can be a class or an instance.

Since 1.0.13

is_component_enabled(*cls*)
Can be overridden by sub-classes to veto the activation of a component.

If this method returns False, the component was disabled explicitly. If it returns None, the component was neither enabled nor disabled explicitly. In both cases, the component with the given class will not be available.

is_enabled(*cls*)
Return whether the given component class is enabled.

In practice, there's only one kind of *ComponentManager* in the Trac application itself, the *trac.env.Environment*.

More on components

We have seen above that one way to retrieve a *Component* instance is to call the constructor on a *ComponentManager* instance `mgr`:

```
a1 = ComponentA(mgr)
```

This will eventually trigger the creation of a new `ComponentA` instance if there wasn't already one created for `mgr`*[0]. At this unique occasion, the constructor of the component subclass will be called *without arguments*, so if you define a constructor it must have the following signature:

```
def __init__(self):
    self.all_colors = set()
```

Note that one should try to do as little as possible in a *Component* constructor. The most complex operation could be for example the allocation of a lock to control the concurrent access to some data members and guarantee thread-safe initialization of more costly resources on first use. Never do such costly initializations in the constructor itself.

Exceptions

exception trac.core.**TracBaseError**
Bases: exceptions.Exception

Base class for all exceptions defined in Trac.

exception trac.core.**TracError**(*message*, *title=None*, *show_traceback=False*)
Bases: *trac.core.TracBaseError*

Standard exception for errors in Trac.

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

Miscellaneous

class trac.core.**ComponentMeta**
Bases: type

Meta class for components.

Takes care of component and extension point registration.

Create the component class.

classmethod deregister(*component*)
Remove a component from the registry.

trac.core.**N_**(*string*)
No-op translation marker, inlined here to avoid importing from *trac.util*.

[0] Ok, it *might* happen that more than one component instance get created due to a race condition. This is usually harmless, see #9418.

trac.db.api – Trac DB abstraction layer

Interfaces

class trac.db.api.**IDatabaseConnector**
Bases: *trac.core.Interface*

Extension point interface for components that support the connection to relational databases.

See also trac.db.api.IDatabaseConnector extension point.

backup(*dest*)
Backup the database to a location defined by trac.backup_dir

db_exists(*path*, *log=None*, ***kwargs*)
Return `True` if the database exists.

destroy_db(*path*, *log=None*, ***kwargs*)
Destroy the database.

get_connection(*path*, *log=None*, ***kwargs*)
Create a new connection to the database.

get_exceptions()
Return an object (typically a module) containing all the backend-specific exception types as attributes, named according to the Python Database API (http://www.python.org/dev/peps/pep-0249/).

get_supported_schemes()
Return the connection URL schemes supported by the connector, and their relative priorities as an iterable of `(scheme, priority)` tuples.

If `priority` is a negative number, this is indicative of an error condition with the connector. An error message should be attached to the `error` attribute of the connector.

get_system_info()
Yield a sequence of `(name, version)` tuples describing the name and version information of external packages used by the connector.

init_db(*path*, *schema=None*, *log=None*, ***kwargs*)
Initialize the database.

to_sql(*table*)
Return the DDL statements necessary to create the specified table, including indices.

Classes

The following classes are not meant to be used directly, but rather via the *Environment* methods *db_transaction* and *db_query*.

class trac.db.api.**QueryContextManager**(*env*)
Bases: *trac.db.api.DbContextManager*

Database Context Manager for retrieving a read-only *ConnectionWrapper*.

class trac.db.api.**TransactionContextManager**(*env*)
Bases: *trac.db.api.DbContextManager*

Transactioned Database Context Manager for retrieving a *ConnectionWrapper*.

The outermost such context manager will perform a commit upon normal exit or a rollback after an exception.

The above are both subclasses of *DbContextManager*:

class trac.db.api.**DbContextManager**(*env*)
Bases: object

Database Context Manager

The outermost *DbContextManager* will close the connection.

execute(*query*, *params=None*)
Shortcut for directly executing a query.

executemany(*query*, *params=None*)
Shortcut for directly calling "executemany" on a query.

The API of database backend specific connection classes (like SQLiteConnection) is specified and documented in a base class, the *ConnectionBase*.

class trac.db.api.**ConnectionBase**
Bases: object

Abstract base class for database connection classes.

cast(*column*, *type*)
Returns a clause casting column as type.

concat(**args*)
Returns a clause concatenating the sequence args.

drop_column(*table*, *column*)
Drops the column from table.

drop_table(*table*)
Drops the table.

get_column_names(*table*)
Returns the list of the column names in table.

get_last_id(*cursor*, *table*, *column='id'*)
Returns the current value of the primary key sequence for table. The column of the primary key may be specified, which defaults to id.

get_sequence_names()
Returns a list of the sequence names.

get_table_names()
Returns a list of the table names.

has_table(*table*)
Returns whether the table exists.

like()
Returns a case-insensitive LIKE clause.

like_escape(*text*)
Returns text escaped for use in a LIKE clause.

prefix_match()
Return a case sensitive prefix-matching operator.

prefix_match_value(*prefix*)
Return a value for case sensitive prefix-matching operator.

quote(*identifier*)
Returns the quoted identifier.

reset_tables()
Deletes all data from the tables and resets autoincrement indexes.

Returns list of names of the tables that were reset.

update_sequence(*cursor*, *table*, *column='id'*)
Updates the current value of the primary key sequence for `table`. The `column` of the primary key may be specified, which defaults to `id`.

Components

class `trac.db.api.`**DatabaseManager**
Bases: *trac.core.Component*

Component used to manage the *IDatabaseConnector* implementations.

backup(*dest=None*)
Save a backup of the database.

Parameters **dest** – base filename to write to.

Returns the file actually written.

backup_dir
Database backup location

connection_uri
Database connection [wiki:TracEnvironment#DatabaseConnectionStrings string] for this project

connectors
List of components that implement *IDatabaseConnector*

create_tables(*schema*)
Create the specified tables.

Parameters **schema** – an iterable of table objects.

Since version 1.0.2

debug_sql
Show the SQL queries in the Trac log, at DEBUG level.

drop_columns(*table*, *columns*)
Drops the specified columns from table.

Since version 1.2

drop_tables(*schema*)
Drop the specified tables.

Parameters **schema** – an iterable of `Table` objects or table names.

Since version 1.0.2

get_column_names(*table*)
Returns a list of the column names for `table`.

Parameters **table** – a `Table` object or table name.

Since 1.2

get_connection(*readonly=False*)
Get a database connection from the pool.

If `readonly` is `True`, the returned connection will purposely lack the `rollback` and `commit` methods.

get_database_version(*name='database_version'*)
Returns the database version from the SYSTEM table as an int, or `False` if the entry is not found.

Parameters **name** – The name of the entry that contains the database version in the SYSTEM table. Defaults to `database_version`, which contains the database version for Trac.

get_sequence_names()
Returns a list of the sequence names.

Since 1.3.2

get_table_names()
Returns a list of the table names.

Since 1.1.6

has_table(*table*)
Returns whether the table exists.

insert_into_tables(*data_or_callable*)
Insert data into existing tables.

Parameters **data_or_callable** – Nested tuples of table names, column names and row data:

```
(table1,
 (column1, column2),
 ((row1col1, row1col2),
  (row2col1, row2col2)),
 table2, ...)
```

or a callable that takes a single parameter `db` and returns the aforementioned nested tuple.

Since version 1.1.3

needs_upgrade(*version*, *name='database_version'*)
Checks the database version to determine if an upgrade is needed.

Parameters

- **version** – the expected integer database version.
- **name** – the name of the entry in the SYSTEM table that contains the database version. Defaults to `database_version`, which contains the database version for Trac.

Returns `True` if the stored version is less than the expected version, `False` if it is equal to the expected version.

Raises ***TracError*** – if the stored version is greater than the expected version.

reset_tables()
Deletes all data from the tables and resets autoincrement indexes.

Returns list of names of the tables that were reset.

Since version 1.1.3

set_database_version(*version*, *name='database_version'*)
Sets the database version in the SYSTEM table.

Parameters

- **version** – an integer database version.
- **name** – The name of the entry that contains the database version in the SYSTEM table. Defaults to `database_version`, which contains the database version for Trac.

timeout
Timeout value for database connection, in seconds. Use '0' to specify ''no timeout''.

upgrade(*version*, *name='database_version'*, *pkg=None*)
Invokes `do_upgrade(env, version, cursor)` in module `"%s/db%i.py" % (pkg, version)`, for each required version upgrade.

Parameters

- **version** – the expected integer database version.
- **name** – the name of the entry in the SYSTEM table that contains the database version. Defaults to `database_version`, which contains the database version for Trac.
- **pkg** – the package containing the upgrade modules.

Raises ***TracError*** – if the package or module doesn't exist.

upgrade_tables(*new_schema*)
Upgrade table schema to `new_schema`, preserving data in columns that exist in the current schema and `new_schema`.

Parameters **new_schema** – tuple or list of `Table` objects

Since version 1.2

Functions

trac.db.api.**get_column_names**(*cursor*)
Retrieve column names from a cursor, if possible.

trac.db.api.**parse_connection_uri**(*db_str*)
Parse the database connection string.

The database connection string for an environment is specified through the `database` option in the `[trac]` section of trac.ini.

Returns a tuple containing the scheme and a dictionary of attributes: `user`, `password`, `host`, `port`, `path`, `params`.

Since 1.1.3

See also

wiki/TracDev/DatabaseApi

trac.db.mysql_backend

class trac.db.mysql_backend.**MySQLConnection**(*path*, *log*, *user=None*, *password=None*, *host=None*, *port=None*, *params={}*)
Bases: *trac.db.api.ConnectionBase*, *trac.db.util.ConnectionWrapper*

Connection wrapper for MySQL.

class trac.db.mysql_backend.**MySQLConnector**
Bases: *trac.core.Component*

Database connector for MySQL version 4.1 and greater.

Database URLs should be of the form:

```
{{{
mysql://user[:password]@host[:port]/database[?param1=value&param2=value]
}}}
```

The following parameters are supported:

- `compress`: Enable compression (0 or 1)
- `init_command`: Command to run once the connection is created
- `named_pipe`: Use a named pipe to connect on Windows (0 or 1)
- `read_default_file`: Read default client values from the given file
- `read_default_group`: Configuration group to use from the default file
- `unix_socket`: Use a Unix socket at the given path to connect

alter_column_types(*table*, *columns*)
Yield SQL statements altering the type of one or more columns of a table.

Type changes are specified as a `columns` dict mapping column names to `(from, to)` SQL type tuples.

mysqldump_path
Location of mysqldump for MySQL database backups

trac.db.pool

class trac.db.pool.**ConnectionPoolBackend**(*maxsize*)
Bases: object

A process-wide LRU-based connection pool.

shutdown(*tid=None*)
Close pooled connections not used in a while

class trac.db.pool.**PooledConnection**(*pool*, *cnx*, *key*, *tid*, *log=None*)
Bases: *trac.db.util.ConnectionWrapper*

A database connection that can be pooled. When closed, it gets returned to the pool.

exception trac.db.pool.**TimeoutError**(*message*, *title=None*, *show_traceback=False*)
Bases: *trac.core.TracError*

Exception raised by the connection pool when no connection has become available after a given timeout.

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

trac.db.postgres_backend

class trac.db.postgres_backend.**PostgreSQLConnection**(*path*, *log=None*, *user=None*, *password=None*, *host=None*, *port=None*, *params={}*)
Bases: *trac.db.api.ConnectionBase*, *trac.db.util.ConnectionWrapper*

Connection wrapper for PostgreSQL.

class trac.db.postgres_backend.**PostgreSQLConnector**
Bases: *trac.core.Component*

Database connector for PostgreSQL.

Database URLs should be of the form: {{{ postgres://user[:password]@host[:port]/database[?schema=my_schema] }}}

alter_column_types(*table*, *columns*)
Yield SQL statements altering the type of one or more columns of a table.

Type changes are specified as a `columns` dict mapping column names to `(from, to)` SQL type tuples.

pg_dump_path
Location of pg_dump for Postgres database backups

trac.db.postgres_backend.**assemble_pg_dsn**(*path*, *user=None*, *password=None*, *host=None*, *port=None*)
Quote the parameters and assemble the DSN.

trac.db.schema

class trac.db.schema.**Column**(*name*, *type='text'*, *size=None*, *key_size=None*, *auto_increment=False*)
Bases: object

Declare a table column in a database schema.

class trac.db.schema.**Index**(*columns*, *unique=False*)
Bases: object

Declare an index for a database schema.

class trac.db.schema.**Table**(*name*, *key=[]*)
Bases: object

Declare a table in a database schema.

remove_columns(*column_names*)
Remove columns specified in the list or tuple `column_names`.

trac.db.sqlite_backend

class trac.db.sqlite_backend.**SQLiteConnection**(*path*, *log=None*, *params={}*)
Bases: *trac.db.api.ConnectionBase*, *trac.db.util.ConnectionWrapper*

Connection wrapper for SQLite.

class trac.db.sqlite_backend.**SQLiteConnector**
Bases: *trac.core.Component*

Database connector for SQLite.

Database URLs should be of the form: {{{ sqlite:path/to/trac.db }}}

alter_column_types(*table*, *columns*)
Yield SQL statements altering the type of one or more columns of a table.

Type changes are specified as a `columns` dict mapping column names to `(from, to)` SQL type tuples.

backup(*dest_file*)
Simple SQLite-specific backup of the database.

Parameters **dest_file** – Destination file basename

extensions
Paths to [https://sqlite.org/loadext.html sqlite extensions]. The paths may be absolute or relative to the Trac environment.

trac.db.utils – Trac DB utilities

Utilities for the Trac DB abstraction layer.

Classes

The following classes are not meant to be used directly. In particular, the *ConnectionWrapper* is what the *DbContextManager* context managers will return.

For example:

```
>>> with env.db_query as db:
...     for name, value in db.execute("SELECT name, value FROM system"):
...         print("row: [{name}, {value}]".format(name=name, value=value))
...
row: [database_version, 29]
```

Here db is a *ConnectionWrapper*.

class trac.db.util.**ConnectionWrapper**(*cnx*, *log=None*, *readonly=False*)
Bases: object

Generic wrapper around connection objects.

Since 0.12 This wrapper no longer makes cursors produced by the connection iterable using *IterableCursor*.

Since 1.0 added a 'readonly' flag preventing the forwarding of commit and rollback

All of the following methods but *execute*, *executemany* and *check_select* need to be implemented by the backend-specific subclass.

In addition, the standard methods from **PEP 0249** *Connection Objects* are also available.

cast(*self*, *column*, *type*)
Local SQL dialect for type casting.

Parameters

- **column** – name of the column
- **type** – generic type (int, int64, text)

concat(self, *args):
Local SQL dialect for string concatenation.

Parameters **args** – values to be concatenated specified as multiple parameters.

like(self):
Local SQL dialect for a case-insensitive LIKE clause.

like_escape(self, text):
Local SQL dialect for searching for litteral text in a LIKE clause.

quote(self, identifier):
Local SQL dialect for quoting an identifier.

get_last_id(self, cursor, table, column='id'):
Local SQL dialect for retrieving the last value of an auto-increment column, immediately following an INSERT clause.

Parameters

- **cursor** – the cursor in which the INSERT was executed
- **table** – the name of the table in which the insertion happened
- **column** – the name of the auto-increment column

Some backends, like PostgreSQL, support that feature natively indirectly via sequences.

update_sequence(self, cursor, table, column='id'):
Local SQL dialect for resetting a sequence.

Same parameters as for `get_last_id`.

This can be used whenever rows were created *with an explicit value for the auto-increment column*, as it could happen during a database upgrade and the recreation of a table. See #8575 for details.

check_select(*query*)
Verify if the query is compatible according to the readonly nature of the wrapped Connection.

Returns `True` if this is a SELECT

Raise `ValueError` if this is not a SELECT and the wrapped Connection is read-only.

execute(*query*, *params=None*)
Execute an SQL `query`

The optional `params` is a tuple containing the parameter values expected by the query.

If the query is a SELECT, return all the rows ("fetchall"). When more control is needed, use `cursor()`.

executemany(*query*, *params=None*)
Execute an SQL `query`, on a sequence of tuples ("executemany").

The optional `params` is a sequence of tuples containing the parameter values expected by the query.

If the query is a SELECT, return all the rows ("fetchall"). When more control is needed, use `cursor()`.

Also, all the *ConnectionWrapper* subclasses (`SQLiteConnection`, `PostgreSQLConnection` and `MySQLConnection`) have a reimplemented `cursor()` method which returns an *IterableCursor*.

class `trac.db.util.`**IterableCursor**(*cursor*, *log=None*)
Bases: `object`

Wrapper for DB-API cursor objects that makes the cursor iterable and escapes all "%"s used inside literal strings with parameterized queries.

Iteration will generate the rows of a SELECT query one by one.

trac.dist

Extra commands for setup.py.

We provide a few extra command classes in `l10n_cmdclass` for localization tasks. We also modify the standard commands `distutils.command.build` and `setuptools.command.install_lib` classes in order to call the l10n commands for compiling catalogs at the right time during install.

class trac.dist.**check_catalog**(*dist*)
Bases: distutils.cmd.Command

Check message catalog command for use setup.py scripts.

Create and initialize a new Command object. Most importantly, invokes the 'initialize_options()' method, which is the real initializer and depends on the actual command being instantiated.

trac.dist.**check_markup**(*catalog*, *message*)
Verify markups in the translation.

trac.dist.**extract_html**(*fileobj*, *keywords*, *comment_tags*, *options*, *text=False*)
Extracts translatable texts from templates.

We simplify white-space found in translatable texts collected via the gettext function (which is what the trans directives use) and for text in the legacy Genshi templates, otherwise we would have near duplicates (e.g. admin.html, prefs.html).

We assume the template function gettext will do the same before trying to fetch the translation from the catalog.

trac.dist.**extract_javascript_script**(*fileobj*, *keywords*, *comment_tags*, *options*)
Extract messages from Javascript embedded in <script> tags.

Select <script type="javascript/text"> tags and delegate to extract_javascript.

trac.dist.**extract_python**(*fileobj*, *keywords*, *comment_tags*, *options*)
Extract messages from Python source code, This is patched extract_python from Babel to support keyword argument mapping.

kwargs_maps option: names of keyword arguments will be mapping to index of messages array.

cleandoc_keywords option: a list of keywords to clean up the extracted messages with cleandoc.

trac.dist.**extract_text**(*fileobj*, *keywords*, *comment_tags*, *options*)
Extract messages from Genshi or Jinja2 text templates.

This is only needed as an interim measure, as long as we have both.

class trac.dist.**generate_messages_js**(*dist*)
Bases: distutils.cmd.Command

Generating message javascripts command for use setup.py scripts.

Create and initialize a new Command object. Most importantly, invokes the 'initialize_options()' method, which is the real initializer and depends on the actual command being instantiated.

trac.dist.**simplify_message**(*message*)
Transforms an extracted messsage (string or tuple) into one in which the repeated white-space has been simplified to a single space.

trac.env – Trac Environment model and APIs

Interfaces

class trac.env.**IEnvironmentSetupParticipant**
Bases: *trac.core.Interface*

Extension point interface for components that need to participate in the creation and upgrading of Trac environments, for example to create additional database tables.

Please note that *IEnvironmentSetupParticipant* instances are called in arbitrary order. If your upgrades must be ordered consistently, please implement the ordering in a single *IEnvironmentSetupParticipant*. See the database upgrade infrastructure in Trac core for an example.

See also trac.env.IEnvironmentSetupParticipant extension point

environment_created()
Called when a new Trac environment is created.

environment_needs_upgrade()
Called when Trac checks whether the environment needs to be upgraded.

Should return `True` if this participant needs an upgrade to be performed, `False` otherwise.

upgrade_environment()
Actually perform an environment upgrade.

Implementations of this method don't need to commit any database transactions. This is done implicitly for each participant if the upgrade succeeds without an error being raised.

However, if the *upgrade_environment* consists of small, restartable, steps of upgrade, it can decide to commit on its own after each successful step.

class trac.env.**ISystemInfoProvider**
Bases: *trac.core.Interface*

Provider of system information, displayed in the "About Trac" page and in internal error reports.

See also trac.env.ISystemInfoProvider extension point

get_system_info()
Yield a sequence of `(name, version)` tuples describing the name and version information of external packages used by a component.

Components

The *Environment* is special in the sense it is not only a `Component`, but also a *trac.core.ComponentManager*.

class trac.env.**Environment**(*path*, *create=False*, *options=[]*)
Bases: *trac.core.Component*, *trac.core.ComponentManager*

Trac environment manager.

Trac stores project information in a Trac environment. It consists of a directory structure containing among other things:

- a configuration file,
- project-specific templates and plugins,
- the wiki and ticket attachments files,
- the SQLite database file (stores tickets, wiki pages...) in case the database backend is sqlite

Initialize the Trac environment.

Parameters

- **path** – the absolute path to the Trac environment
- **create** – if `True`, the environment is created and populated with default data; otherwise, the environment is expected to already exist.

- **options** – A list of `(section, name, value)` tuples that define configuration options

abs_href
The application URL

attachments_dir
Absolute path to the attachments directory.

Since 1.3.1

backup(*dest=None*)
Create a backup of the database.

Parameters **dest** – Destination file; if not specified, the backup is stored in a file called db_name.trac_version.bak

base_url
Reference URL for the Trac deployment.

This is the base URL that will be used when producing documents that will be used outside of the web browsing context, like for example when inserting URLs pointing to Trac resources in notification e-mails.

base_url_for_redirect
Optionally use `[trac] base_url` for redirects.

In some configurations, usually involving running Trac behind a HTTP proxy, Trac can't automatically reconstruct the URL that is used to access it. You may need to use this option to force Trac to use the *base_url* setting also for redirects. This introduces the obvious limitation that this environment will only be usable when accessible from that URL, as redirects are frequently used.

component_activated(*component*)
Initialize additional member variables for components.

Every component activated through the *Environment* object gets three member variables: *env* (the environment object), `config` (the environment configuration) and `log` (a logger object).

component_guard(**args*, ***kwds*)
Traps any runtime exception raised when working with a component and logs the error.

Parameters

- **component** – the component responsible for any error that could happen inside the context
- **reraise** – if `True`, an error is logged but not suppressed. By default, errors are suppressed.

components_section
This section is used to enable or disable components provided by plugins, as well as by Trac itself. The component to enable/disable is specified via the name of the option. Whether its enabled is determined by the option value; setting the value to `enabled` or `on` will enable the component, any other value (typically `disabled` or `off`) will disable the component.

The option name is either the fully qualified name of the components or the module/package prefix of the component. The former enables/disables a specific component, while the latter enables/disables any component in the specified package/module.

Consider the following configuration snippet: {{{ [components] trac.ticket.report.ReportModule = disabled acct_mgr.* = enabled }}}

The first option tells Trac to disable the [wiki:TracReports report module]. The second option instructs Trac to enable all components in the `acct_mgr` package. Note that the trailing wildcard is required for module/package matching.

To view the list of active components, go to the ''Plugins" page on ''About Trac" (requires `CONFIG_VIEW` [wiki:TracPermissions permissions]).

See also: TracPlugins

conf_dir
Absolute path to the conf directory.

Since 1.0.11

config_file_path
Path of the trac.ini file.

create(*options=[]*)
Create the basic directory structure of the environment, initialize the database and populate the configuration file with default values.

If options contains ('inherit', 'file'), default values will not be loaded; they are expected to be provided by that file or other options.

Raises

- ***TracError*** – if the base directory of `path` does not exist.
- ***TracError*** – if `path` exists and is not empty.

database_initial_version
Returns the version of the database at the time of creation.

In practice, for database created before 0.11, this will return `False` which is "older" than any db version number.

Since 1.0.2

database_version
Returns the current version of the database.

Since 1.0.2

db_exc
Return an object (typically a module) containing all the backend-specific exception types as attributes, named according to the Python Database API (http://www.python.org/dev/peps/pep-0249/).

To catch a database exception, use the following pattern:

```
try:
    with env.db_transaction as db:
        ...
except env.db_exc.IntegrityError as e:
    ...
```

db_query
Return a context manager (*QueryContextManager*) which can be used to obtain a read-only database connection.

Example:

```
with env.db_query as db:
    cursor = db.cursor()
    cursor.execute("SELECT ...")
```

```
    for row in cursor.fetchall():
        ...
```

Note that a connection retrieved this way can be "called" directly in order to execute a query:

```
with env.db_query as db:
    for row in db("SELECT ..."):
        ...
```

Warning after a `with env.db_query as db` block, though the `db` variable is still defined, you shouldn't use it as it might have been closed when exiting the context, if this context was the outermost context (*db_query* or *db_transaction*).

If you don't need to manipulate the connection itself, this can even be simplified to:

```
for row in env.db_query("SELECT ..."):
    ...
```

db_transaction

Return a context manager (*TransactionContextManager*) which can be used to obtain a writable database connection.

Example:

```
with env.db_transaction as db:
    cursor = db.cursor()
    cursor.execute("UPDATE ...")
```

Upon successful exit of the context, the context manager will commit the transaction. In case of nested contexts, only the outermost context performs a commit. However, should an exception happen, any context manager will perform a rollback. You should *not* call `commit()` yourself within such block, as this will force a commit even if that transaction is part of a larger transaction.

Like for its read-only counterpart, you can directly execute a DML query on the `db`:

```
with env.db_transaction as db:
    db("UPDATE ...")
```

Warning after a `with env.db_transaction` as db‘ block, though the `db` variable is still available, you shouldn't use it as it might have been closed when exiting the context, if this context was the outermost context (*db_query* or *db_transaction*).

If you don't need to manipulate the connection itself, this can also be simplified to:

```
env.db_transaction("UPDATE ...")
```

enable_component(*cls*)

Enable a component or module.

env

Property returning the *Environment* object, which is often required for functions and methods that take a `Component` instance.

files_dir

Absolute path to the files directory.

Since 1.3.2

get_known_users(*as_dict=False*)
Returns information about all known users, i.e. users that have logged in to this Trac environment and possibly set their name and email.

By default this function returns a iterator that yields one tuple for every user, of the form (username, name, email), ordered alpha-numerically by username. When `as_dict` is `True` the function returns a dictionary mapping username to a (name, email) tuple.

Since 1.2 the `as_dict` parameter is available.

get_systeminfo()
Return a list of `(name, version)` tuples describing the name and version information of external packages used by Trac and plugins.

Since 1.3.1 deprecated and will be removed in 1.5.1. Use system_info property instead.

href
The application root path

htdocs_dir
Absolute path to the htdocs directory.

Since 1.0.11

invalidate_known_users_cache()
Clear the known_users cache.

is_component_enabled(*cls*)
Implemented to only allow activation of components that are not disabled in the configuration.

This is called by the `ComponentManager` base class when a component is about to be activated. If this method returns `False`, the component does not get activated. If it returns `None`, the component only gets activated if it is located in the `plugins` directory of the environment.

log_dir
Absolute path to the log directory.

Since 1.0.11

log_file
If *log_type* is `file`, this should be a path to the log-file. Relative paths are resolved relative to the `log` directory of the environment.

log_file_path
Path to the log file.

log_format
Custom logging format.

If nothing is set, the following will be used:

```
Trac[$(module)s] $(levelname)s:  $(message)s
```

In addition to regular key names supported by the [http://docs.python.org/library/logging.html Python logger library] one could use:

- `$(path)s` the path for the current environment
- `$(basename)s` the last path component of the current environment
- `$(project)s` the project name

Note the usage of `$(...)s` instead of `%(...)s` as the latter form would be interpreted by the !ConfigParser itself.

Example: `($(thread)d) Trac[$(basename)s:$(module)s] $(levelname)s: $(message)s`

log_level
Level of verbosity in log.

Should be one of (`CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`).

log_type
Logging facility to use.

Should be one of (`none`, `file`, `stderr`, `syslog`, `winlog`).

name
The environment name.

Since 1.2

needs_upgrade()
Return whether the environment needs to be upgraded.

plugins_dir
Absolute path to the plugins directory.

Since 1.0.11

project_admin
E-Mail address of the project's administrator.

project_admin_trac_url
Base URL of a Trac instance where errors in this Trac should be reported.

This can be an absolute or relative URL, or '.' to reference this Trac instance. An empty value will disable the reporting buttons.

project_description
Short description of the project.

project_footer
Page footer text (right-aligned).

project_icon
URL of the icon of the project.

project_name
Name of the project.

project_url
URL of the main project web site, usually the website in which the *base_url* resides. This is used in notification e-mails.

secure_cookies
Restrict cookies to HTTPS connections.

When true, set the `secure` flag on all cookies so that they are only sent to the server on HTTPS connections. Use this if your Trac instance is only accessible through HTTPS.

setup_config()
Load the configuration file.

setup_log()
Initialize the logging sub-system.

setup_participants
List of components that implement `IEnvironmentSetupParticipant`

shared_plugins_dir
Path to the //shared plugins directory//.

Plugins in that directory are loaded in addition to those in the directory of the environment `plugins`, with this one taking precedence.

shutdown(*tid=None*)
Close the environment.

system_info
List of `(name, version)` tuples describing the name and version information of external packages used by Trac and plugins.

system_info_providers
List of components that implement `ISystemInfoProvider`

templates_dir
Absolute path to the templates directory.

Since 1.0.11

trac_version
Returns the version of Trac. :since: 1.2

upgrade(*backup=False*, *backup_dest=None*)
Upgrade database.

Parameters

- **backup** – whether or not to backup before upgrading
- **backup_dest** – name of the backup file

Returns whether the upgrade was performed

verify()
Verify that the provided path points to a valid Trac environment directory.

class trac.env.**EnvironmentSetup**
Bases: *trac.core.Component*

Manage automatic environment upgrades.

environment_created()
Insert default data into the database.

class trac.env.**EnvironmentAdmin**
Bases: *trac.core.Component*

trac-admin command provider for environment administration.

Functions

trac.env.**open_environment**(*env_path=None*, *use_cache=False*)
Open an existing environment object, and verify that the database is up to date.

Parameters

- **env_path** – absolute path to the environment directory; if omitted, the value of the `TRAC_ENV` environment variable is used
- **use_cache** – whether the environment should be cached for subsequent invocations of this function

Returns the *Environment* object

Exceptions

exception trac.env.**BackupError**
Bases: *trac.core.TracBaseError*, exceptions.RuntimeError

Exception raised during an upgrade when the DB backup fails.

trac.loader

trac.loader.**load_components**(*env*, *extra_path=None*, *loaders=(<function _load_eggs>, <function _load_py_files>)*)
Load all plugin components found on the given search path.

trac.loader.**get_plugin_info**(*env*, *include_core=False*)
Return package information about Trac core and installed plugins.

trac.loader.**load_eggs**(*entry_point_name*)
Loader that loads any eggs on the search path and sys.path.

trac.loader.**load_py_files**()
Loader that look for Python source files in the plugins directories, which simply get imported, thereby registering them with the component manager if they define any components.

trac.loader.**match_plugins_to_frames**(*plugins*, *frames*)
Add a frame_idx element to plugin information as returned by *get_plugin_info()*, containing the index of the highest frame in the list that was located in the plugin.

trac.log

trac.mimeview.api – Trac content transformation APIs

Interfaces

class trac.mimeview.api.**IHTMLPreviewRenderer**
Bases: *trac.core.Interface*

Extension point interface for components that add HTML renderers of specific content types to the *Mimeview* component.

Note: This interface will be merged with IContentConverter, as conversion to text/html will simply be a particular content conversion.

Note however that the IHTMLPreviewRenderer will still be supported for a while through an adapter, whereas the IContentConverter interface itself will be changed.

So if all you want to do is convert to HTML and don't feel like following the API changes, you should rather implement this interface for the time being.

See also trac.mimeview.api.IHTMLPreviewRenderer extension point

get_extra_mimetypes()
Augment the Mimeview system with new mimetypes associations.

This is an optional method. Not implementing the method or returning nothing is fine, the component will still be asked via *get_quality_ratio* if it supports a known mimetype. But implementing it can be useful when the component knows about additional mimetypes which may augment the list of already mimetype to keywords associations.

Generate `(mimetype, keywords)` pairs for each additional mimetype, with `keywords` being a list of keywords or extensions that can be used as aliases for the mimetype (typically file suffixes or Wiki processor keys).

New in version 1.0.

get_quality_ratio(*mimetype*)
Return the level of support this renderer provides for the `content` of the specified MIME type. The return value must be a number between 0 and 9, where 0 means no support and 9 means "perfect" support.

render(*context*, *mimetype*, *content*, *filename=None*, *url=None*)
Render an XHTML preview of the raw `content` in a *RenderingContext*.

The `content` might be:

- a `str` object
- an `unicode` string
- any object with a `read` method, returning one of the above

It is assumed that the content will correspond to the given `mimetype`.

Besides the `content` value, the same content may eventually be available through the `filename` or `url` parameters. This is useful for renderers that embed objects, using <object> or <img> instead of including the content inline.

Can return the generated XHTML text as a single string or as an iterable that yields strings. In the latter case, the list will be considered to correspond to lines of text in the original content.

class `trac.mimeview.api.`**IHTMLPreviewAnnotator**
Bases: *trac.core.Interface*

Extension point interface for components that can annotate an XHTML representation of file contents with additional information.

See also trac.mimeview.api.IHTMLPreviewAnnotator extension point

annotate_row(*context*, *row*, *number*, *line*, *data*)
Return the XHTML markup for the table cell that contains the annotation data.

`context` is the context corresponding to the content being annotated, `row` is the tr Element being built, `number` is the line number being processed and `line` is the line's actual content. `data` is whatever additional data the *get_annotation_data* method decided to provide.

get_annotation_data(*context*)
Return some metadata to be used by the *annotate_row* method below.

This will be called only once, before lines are processed. If this raises an error, that annotator won't be used.

get_annotation_type()
Return a (type, label, description) tuple that defines the type of annotation and provides human readable names. The `type` element should be unique to the annotator. The `label` element is used as column heading for the table, while `description` is used as a display name to let the user toggle the appearance of the annotation type.

class `trac.mimeview.api.`**IContentConverter**
Bases: *trac.core.Interface*

An extension point interface for generic MIME based content conversion.

Note: This api will likely change in the future (see #3332)

See also trac.mimeview.api.IContentConverter extension point

convert_content(*req*, *mimetype*, *content*, *key*)
Convert the given content from mimetype to the output MIME type represented by key. Returns a tuple in the form (content, output_mime_type) or None if conversion is not possible.

content must be a `str` instance or an iterable instance which iterates `str` instances.

get_supported_conversions()
Return an iterable of tuples in the form (key, name, extension, in_mimetype, out_mimetype, quality) representing the MIME conversions supported and the quality ratio of the conversion in the range 0 to 9, where 0 means no support and 9 means "perfect" support. eg. ('latex', 'LaTeX', 'tex', 'text/x-trac-wiki', 'text/plain', 8)

Components

class `trac.mimeview.api.`**Mimeview**
Bases: *trac.core.Component*

Generic HTML renderer for data, typically source code.

annotators
List of components that implement *IHTMLPreviewAnnotator*

configured_modes_mapping(*renderer*)
Return a MIME type to `(mode,quality)` mapping for given `option`

convert_content(*req*, *mimetype*, *content*, *key*, *filename=None*, *url=None*, *iterable=False*)
Convert the given content to the target MIME type represented by `key`, which can be either a MIME type or a key. Returns a tuple of (content, output_mime_type, extension).

converters
List of components that implement *IContentConverter*

default_charset
Charset to be used when in doubt.

get_annotation_types()
Generator that returns all available annotation types.

get_charset(*content=''*, *mimetype=None*)
Infer the character encoding from the `content` or the `mimetype`.

`content` is either a `str` or an `unicode` object.

The charset will be determined using this order:

- from the charset information present in the `mimetype` argument
- auto-detection of the charset from the `content`
- the configured *default_charset*

get_mimetype(*filename*, *content=None*)
Infer the MIME type from the `filename` or the `content`.

`content` is either a `str` or an `unicode` object.

Return the detected MIME type, augmented by the charset information (i.e. "<mimetype>; charset=..."), or `None` if detection failed.

get_supported_conversions(*mimetype*)
Return a list of target MIME types as instances of the `namedtuple` `MimeConversion`. Output is ordered from best to worst quality.

The `MimeConversion` `namedtuple` has fields: key, name, extension, in_mimetype, out_mimetype, quality, converter.

is_binary(*mimetype=None*, *filename=None*, *content=None*)
Check if a file must be considered as binary.

max_preview_size
Maximum file size for HTML preview.

preview_data(*context*, *content*, *length*, *mimetype*, *filename*, *url=None*, *annotations=None*, *force_source=False*)
Prepares a rendered preview of the given `content`.

Note: `content` will usually be an object with a `read` method.

render(*context*, *mimetype*, *content*, *filename=None*, *url=None*, *annotations=None*, *force_source=False*)
Render an XHTML preview of the given `content`.

`content` is the same as an *IHTMLPreviewRenderer.render*'s `content` argument.

The specified `mimetype` will be used to select the most appropriate *IHTMLPreviewRenderer* implementation available for this MIME type. If not given, the MIME type will be infered from the filename or the content.

Return a string containing the XHTML text.

When rendering with an *IHTMLPreviewRenderer* fails, a warning is added to the request associated with the context (if any), unless the `disable_warnings` hint is set to `True`.

renderers
List of components that implement *IHTMLPreviewRenderer*

send_converted(*req*, *in_type*, *content*, *selector*, *filename='file'*)
Helper method for converting `content` and sending it directly.

`selector` can be either a key or a MIME Type.

tab_width
Displayed tab width in file preview.

to_unicode(*content*, *mimetype=None*, *charset=None*)
Convert `content` (an encoded `str` object) to an `unicode` object.

This calls `trac.util.to_unicode` with the `charset` provided, or the one obtained by *Mimeview.get_charset()*.

treat_as_binary
Comma-separated list of MIME types that should be treated as binary data.

class `trac.mimeview.api.`**ImageRenderer**
Bases: *trac.core.Component*

Inline image display.

This component doesn't need the `content` at all.

class trac.mimeview.api.**LineNumberAnnotator**
Bases: *trac.core.Component*

Text annotator that adds a column with line numbers.

class trac.mimeview.api.**PlainTextRenderer**
Bases: *trac.core.Component*

HTML preview renderer for plain text, and fallback for any kind of text for which no more specific renderer is available.

class trac.mimeview.api.**WikiTextRenderer**
Bases: *trac.core.Component*

HTML renderer for files containing Trac's own Wiki formatting markup.

Helper classes

class trac.mimeview.api.**RenderingContext**(*resource*, *href=None*, *perm=None*)
Bases: object

A rendering context specifies ''how" the content should be rendered.

It holds together all the needed contextual information that will be needed by individual renderer components.

To that end, a context keeps track of the Href instance (`.href`) which should be used as a base for building URLs.

It also provides a `PermissionCache` (`.perm`) which can be used to restrict the output so that only the authorized information is shown.

A rendering context may also be associated to some Trac resource which will be used as the implicit reference when rendering relative links or for retrieving relative content and can be used to retrieve related metadata.

Rendering contexts can be nested, and a new context can be created from an existing context using the call syntax. The previous context can be retrieved using the `.parent` attribute.

For example, when rendering a wiki text of a wiki page, the context will be associated to a resource identifying that wiki page.

If that wiki text contains a `[[TicketQuery]]` wiki macro, the macro will set up nested contexts for each matching ticket that will be used for rendering the ticket descriptions.

Since version 1.0

Directly create a *RenderingContext*.

Parameters

- **resource** (`Resource`) – the associated resource
- **href** – an `Href` object suitable for creating URLs
- **perm** – a `PermissionCache` object used for restricting the generated output to "authorized" information only.

The actual *perm* attribute of the rendering context will be bound to the given `resource` so that fine-grained permission checks will apply to that.

child(*resource=None*, *id=False*, *version=False*, *parent=False*)
Create a nested rendering context.

`self` will be the parent for the new nested context.

Parameters

- **resource** – either a `Resource` object or the realm string for a resource specification to be associated to the new context. If `None`, the resource will be the same as the resource of the parent context.
- **id** – the identifier part of the resource specification
- **version** – the version of the resource specification

Returns the new context object

Return type *RenderingContext*

```
>>> context = RenderingContext('wiki', 'WikiStart')
>>> ticket1 = Resource('ticket', 1)
>>> context.child('ticket', 1).resource == ticket1
True
>>> context.child(ticket1).resource is ticket1
True
>>> context.child(ticket1)().resource is ticket1
True
```

get_hint(*hint*, *default=None*)

Retrieve a rendering hint from this context or an ancestor context.

```
>>> ctx = RenderingContext('timeline')
>>> ctx.set_hints(wiki_flavor='oneliner')
>>> t_ctx = ctx('ticket', 1)
>>> t_ctx.get_hint('wiki_flavor')
'oneliner'
>>> t_ctx.get_hint('preserve_newlines', True)
True
```

has_hint(*hint*)

Test whether a rendering hint is defined in this context or in some ancestor context.

```
>>> ctx = RenderingContext('timeline')
>>> ctx.set_hints(wiki_flavor='oneliner')
>>> t_ctx = ctx('ticket', 1)
>>> t_ctx.has_hint('wiki_flavor')
True
>>> t_ctx.has_hint('preserve_newlines')
False
```

set_hints(***keyvalues*)

Set rendering hints for this rendering context.

```
>>> ctx = RenderingContext('timeline')
>>> ctx.set_hints(wiki_flavor='oneliner', shorten_lines=True)
>>> t_ctx = ctx('ticket', 1)
>>> t_ctx.set_hints(wiki_flavor='html', preserve_newlines=True)
>>> (t_ctx.get_hint('wiki_flavor'), t_ctx.get_hint('shorten_lines'),
↪    t_ctx.get_hint('preserve_newlines'))
('html', True, True)
>>> (ctx.get_hint('wiki_flavor'), ctx.get_hint('shorten_lines'),
↪ctx.get_hint('preserve_newlines'))
('oneliner', True, None)
```

class `trac.mimeview.api.`**Content**(*input*, *max_size*)

Bases: `object`

A lazy file-like object that only reads `input` if necessary.

Functions

`trac.mimeview.api.`**`get_mimetype`**(*filename*, *content=None*, *mime_map=MIME_MAP*)
Guess the most probable MIME type of a file with the given name.

Parameters

- **`filename`** – is either a filename (the lookup will then use the suffix) or some arbitrary keyword.
- **`content`** – is either a `str` or an `unicode` string.

`trac.mimeview.api.`**`ct_mimetype`**(*content_type*)
Return the mimetype part of a content type.

`trac.mimeview.api.`**`is_binary`**(*data*)
Detect binary content by checking the first thousand bytes for zeroes.

Operate on either `str` or `unicode` strings.

`trac.mimeview.api.`**`detect_unicode`**(*data*)
Detect different unicode charsets by looking for BOMs (Byte Order Mark).

Operate obviously only on `str` objects.

`trac.mimeview.api.`**`content_to_unicode`**(*env*, *content*, *mimetype*)
Retrieve an `unicode` object from a `content` to be previewed.

In case the raw content had an unicode BOM, we remove it.

```
>>> from trac.test import EnvironmentStub
>>> env = EnvironmentStub()
>>> content_to_unicode(env, u"\ufeffNo BOM! h\u00e9 !", '')
u'No BOM! h\xe9 !'
>>> content_to_unicode(env, "No BOM! hé !", '')
u'No BOM! h\xe9 !'
```

Sub-modules

class `trac.mimeview.patch.`**`PatchRenderer`**
Bases: *trac.core.Component*

HTML renderer for patches in unified diff format.

This uses the same layout as in the wiki diff view or the changeset view.

class `trac.mimeview.pygments.`**`PygmentsRenderer`**
Bases: *trac.core.Component*

HTML renderer for syntax highlighting based on Pygments.

`default_style`
The default style to use for Pygments syntax highlighting.

`pygments_lexer_options`
Configure Pygments [%(url)s lexer] options.

For example, to set the [%(url)s#lexers-for-php-and-related-languages PhpLexer] options `startinline` and `funcnamehighlighting`: {{{#!ini [pygments-lexer] php.startinline = True php.funcnamehighlighting = True }}}

The lexer name is derived from the class name, with `Lexer` stripped from the end. The lexer //short names// can also be used in place of the lexer name.

pygments_modes
List of additional MIME types known by Pygments.

For each, a tuple `mimetype:mode:quality` has to be specified, where `mimetype` is the MIME type, `mode` is the corresponding Pygments mode to be used for the conversion and `quality` is the quality ratio associated to this conversion. That can also be used to override the default quality ratio used by the Pygments render.

class trac.mimeview.rst.**ReStructuredTextRenderer**
Bases: *trac.core.Component*

HTML renderer for plain text in reStructuredText format.

trac.mimeview.rst.**code_block_directive**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)
Create a code-block directive for docutils.

Usage: .. code-block:: language

If the language can be syntax highlighted it will be.

trac.mimeview.rst.**trac_directive**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)
Inserts a `reference` node into the document for a given TracLink, based on the content of the arguments.

Usage:

```
.. trac:: target [text]
```

`target` may be any TracLink, provided it doesn't embed a space character (e.g. wiki:"..." notation won't work).

`[text]` is optional. If not given, `target` is used as the reference text.

Trac support for Textile See also: https://github.com/textile/python-textile

class trac.mimeview.txtl.**TextileRenderer**
Bases: *trac.core.Component*

Renders plain text in Textile format as HTML.

trac.notification

trac.perm – the Trac permission system

Interfaces

class trac.perm.**IPermissionRequestor**
Bases: *trac.core.Interface*

Extension point interface for components that define actions.

get_permission_actions()
Return a list of actions defined by this component.

The items in the list may either be simple strings, or `(string, sequence)` tuples. The latter are considered to be "meta permissions" that group several simple actions under one name for convenience, adding to it if another component already defined that name.

class `trac.perm.`**IPermissionStore**
Bases: *trac.core.Interface*

Extension point interface for components that provide storage and management of permissions.

get_all_permissions()
Return all permissions for all users.

The permissions are returned as a list of (subject, action) formatted tuples.

get_user_permissions(*username*)
Return all permissions for the user with the specified name.

The permissions are returned as a dictionary where the key is the name of the permission, and the value is either `True` for granted permissions or `False` for explicitly denied permissions.

get_users_with_permissions(*permissions*)
Retrieve a list of users that have any of the specified permissions.

Users are returned as a list of usernames.

grant_permission(*username*, *action*)
Grant a user permission to perform an action.

revoke_permission(*username*, *action*)
Revokes the permission of the given user to perform an action.

class `trac.perm.`**IPermissionGroupProvider**
Bases: *trac.core.Interface*

Extension point interface for components that provide information about user groups.

get_permission_groups(*username*)
Return a list of names of the groups that the user with the specified name is a member of.

class `trac.perm.`**IPermissionPolicy**
Bases: *trac.core.Interface*

A security policy provider used for fine grained permission checks.

check_permission(*action*, *username*, *resource*, *perm*)
Check that the action can be performed by username on the resource

Parameters

- **action** – the name of the permission
- **username** – the username string or 'anonymous' if there's no authenticated user
- **resource** – the resource on which the check applies. Will be `None`, if the check is a global one and not made on a resource in particular
- **perm** – the permission cache for that username and resource, which can be used for doing secondary checks on other permissions. Care must be taken to avoid recursion.

Returns `True` if action is allowed, `False` if action is denied, or `None` if indifferent. If `None` is returned, the next policy in the chain will be used, and so on.

Note that when checking a permission on a realm resource (i.e. when `id` is `None`), this usually corresponds to some preliminary check done before making a fine-grained check on some resource. Therefore the *IPermissionPolicy* should be conservative and return:

- `True` if the action *can* be allowed for some resources in that realm. Later, for specific resource, the policy will be able to return `True` (allow), `False` (deny) or `None` (don't decide).
- `None` if the action *can not* be performed for *some* resources. This corresponds to situation where the policy is only interested in returning `False` or `None` on specific resources.
- `False` if the action *can not* be performed for *any* resource in that realm (that's a very strong decision as that will usually prevent any fine-grained check to even happen).

Note that performing permission checks on realm resources may seem redundant for now as the action name itself contains the realm, but this will probably change in the future (e.g. `'VIEW' in ...`).

Components

class `trac.perm.`**PermissionSystem**
Bases: *trac.core.Component*

Permission management sub-system.

check_permission(*action*, *username=None*, *resource=None*, *perm=None*)
Return True if permission to perform action for the given resource is allowed.

expand_actions(*actions*)
Helper method for expanding all meta actions.

get_actions(*skip=None*)
Get a list of all actions defined by permission requestors.

get_actions_dict()
Get all actions from permission requestors as a `dict`.

The keys are the action names. The values are the additional actions granted by each action. For simple actions, this is an empty list. For meta actions, this is the list of actions covered by the action.

get_all_permissions()
Return all permissions for all users.

The permissions are returned as a list of (subject, action) formatted tuples.

get_groups_dict()
Get all groups as a `dict`.

The keys are the group names. The values are the group members.

Since 1.1.3

get_permission_actions()
Implement the global `TRAC_ADMIN` meta permission.

get_user_permissions(*username=None*, *undefined=False*)
Return the permissions of the specified user.

The return value is a dictionary containing all the actions granted to the user mapped to `True`.

Parameters **undefined** – if `True`, include actions that are not defined.

Since 1.3.1 added the `undefined` parameter.

get_users_dict()
Get all users as a `dict`.

The keys are the user names. The values are the actions possessed by the user.

Since 1.1.3

get_users_with_permission(*permission*)
Return all users that have the specified permission.

Users are returned as a list of user names.

grant_permission(*username*, *action*)
Grant the user with the given name permission to perform to specified action.

Raises ***PermissionExistsError*** – if user already has the permission or is a member of the group.

Since 1.3.1 raises PermissionExistsError rather than IntegrityError

policies
List of components implementing *IPermissionPolicy*, in the order in which they will be applied. These components manage fine-grained access control to Trac resources.

requestors
List of components that implement *IPermissionRequestor*

revoke_permission(*username*, *action*)
Revokes the permission of the specified user to perform an action.

store
Name of the component implementing *IPermissionStore*, which is used for managing user and group permissions.

class trac.perm.**DefaultPermissionGroupProvider**
Bases: *trac.core.Component*

Permission group provider providing the basic builtin permission groups 'anonymous' and 'authenticated'.

class trac.perm.**DefaultPermissionPolicy**
Bases: *trac.core.Component*

Default permission policy using the IPermissionStore system.

class trac.perm.**DefaultPermissionStore**
Bases: *trac.core.Component*

Default implementation of permission storage and group management.

This component uses the `permission` table in the database to store both permissions and groups.

get_all_permissions()
Return all permissions for all users.

The permissions are returned as a list of (subject, action) formatted tuples.

get_user_permissions(*username*)
Retrieve a list of permissions for the given user.

The permissions are stored in the database as (username, action) records. There's simple support for groups by using lowercase names for the action column: such a record represents a group and not an actual permission, and declares that the user is part of that group.

get_users_with_permissions(*permissions*)
Retrieve a list of users that have any of the specified permissions

Users are returned as a list of usernames.

grant_permission(*username*, *action*)
Grants a user the permission to perform the specified action.

group_providers
List of components that implement *IPermissionGroupProvider*

revoke_permission(*username*, *action*)
Revokes a users' permission to perform the specified action.

class trac.perm.**PermissionAdmin**
Bases: *trac.core.Component*

trac-admin command provider for permission system administration.

Exceptions

exception trac.perm.**PermissionError**(*action=None*, *resource=None*, *env=None*, *msg=None*)
Bases: exceptions.StandardError, *trac.core.TracBaseError*

Insufficient permissions to perform the operation.

exception trac.perm.**PermissionExistsError**(*message*, *title=None*, *show_traceback=False*)
Bases: *trac.core.TracError*

Thrown when a unique key constraint is violated.

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

Miscellaneous

class trac.perm.**PermissionCache**(*env*, *username=None*, *resource=None*, *cache=None*, *groups=None*)
Bases: object

Cache that maintains the permissions of a single user.

Permissions are usually checked using the following syntax:

'WIKI_MODIFY' in perm

One can also apply more fine grained permission checks and specify a specific resource for which the permission should be available:

'WIKI_MODIFY' in perm('wiki', 'WikiStart')

If there's already a page object available, the check is simply:

'WIKI_MODIFY' in perm(page.resource)

If instead of a check, one wants to assert that a given permission is available, the following form should be used:

perm.require('WIKI_MODIFY')

or

perm('wiki', 'WikiStart').require('WIKI_MODIFY')

or

perm(page.resource).require('WIKI_MODIFY')

When using require, a *PermissionError* exception is raised if the permission is missing.

trac.prefs.api – Trac Administration panels

Primary interface for managing preference panels (tabs).

Interfaces

class trac.prefs.api.**IPreferencePanelProvider**
Bases: *trac.core.Interface*

Provides panels for managing user preferences.

See also trac.prefs.api.IPreferencePanelProvider extension point

get_preference_panels(*req*)
Return a list of available preference panels.

The items returned by this function must be tuple of the form `(panel, label)`, or `(panel, label, parent_panel)` for child panels.

render_preference_panel(*req*, *panel*)
Process a request for a preference panel.

This function should return a tuple of the form `(template, data)`, where `template` is the name of the template to use and `data` is the data used when rendering the template.

Note: When a plugin wants to use a legacy Genshi template instead of a Jinja2 template, it needs to return instead a *triple* of the form `(template, data, None)`, similar to what *IRequestHandler.process_request* does.

---

trac.prefs.web_ui

class trac.prefs.web_ui.**PreferencesModule**
Bases: *trac.core.Component*

Displays the preference panels and dispatch control to the individual panels

panel_providers
List of components that implement *IPreferencePanelProvider*

trac.resource

class trac.resource.**Resource**
Bases: object

Resource identifier.

This specifies as precisely as possible *which* resource from a Trac environment is manipulated.

A resource is identified by:

- a `realm` (a string like `'wiki'` or `'ticket'`)
- an `id`, which uniquely identifies a resource within its realm. If the `id` information is not set, then the resource represents the realm as a whole.
- an optional `version` information. If `version` is `None`, this refers by convention to the latest version of the resource.

Some generic and commonly used rendering methods are associated as well to the Resource object. Those properties and methods actually delegate the real work to the Resource's manager.

Create a new Resource object from a specification.

Parameters

- **resource_or_realm** – this can be either: - a *Resource*, which is then used as a base for making a copy - a basestring, used to specify a realm
- **id** – the resource identifier
- **version** – the version or None for indicating the latest version

```
>>> main = Resource('wiki', 'WikiStart')
>>> repr(main)
"<Resource u'wiki:WikiStart'>"
```

```
>>> Resource(main) is main
True
```

```
>>> main3 = Resource(main, version=3)
>>> repr(main3)
"<Resource u'wiki:WikiStart@3'>"
```

```
>>> main0 = main3(version=0)
>>> repr(main0)
"<Resource u'wiki:WikiStart@0'>"
```

In a copy, if id is overridden, then the original version value will not be reused.

```
>>> repr(Resource(main3, id="WikiEnd"))
"<Resource u'wiki:WikiEnd'>"
```

```
>>> repr(Resource(None))
"<Resource ''>"
```

child(*realm*, *id=False*, *version=False*)

Retrieve a child resource for a secondary realm.

Same as __call__, except that this one sets the parent to self.

```
>>> repr(Resource(None).child('attachment', 'file.txt'))
"<Resource u', attachment:file.txt'>"
```

exception trac.resource.**ResourceExistsError**(*message*, *title=None*, *show_traceback=False*)

Bases: *trac.core.TracError*

Thrown when attempting to insert an existing resource.

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

exception trac.resource.**ResourceNotFound**(*message*, *title=None*, *show_traceback=False*)

Bases: *trac.core.TracError*

Thrown when a non-existent resource is requested

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

class trac.resource.**ResourceSystem**
Bases: *trac.core.Component*

Resource identification and description manager.

This component makes the link between *Resource* identifiers and their corresponding manager Component.

get_known_realms()
Return a list of all the realm names of resource managers.

get_resource_manager(*realm*)
Return the component responsible for resources in the given realm

Parameters **realm** – the realm name

Returns a Component implementing IResourceManager or None

resource_managers
List of components that implement IResourceManager

trac.resource.**get_relative_resource**(*resource*, *path=''*)
Build a Resource relative to a reference resource.

Parameters **path** – path leading to another resource within the same realm.

trac.resource.**get_relative_url**(*env*, *resource*, *href*, *path=''*, ***kwargs*)
Build an URL relative to a resource given as reference.

Parameters **path** – path leading to another resource within the same realm.

```
>>> from trac.test import EnvironmentStub
>>> env = EnvironmentStub()
>>> from trac.web.href import Href
>>> href = Href('/trac.cgi')
>>> main = Resource('wiki', 'Main', version=3)
```

Without parameters, return the canonical URL for the resource, like *get_resource_url* does.

```
>>> get_relative_url(env, main, href)
'/trac.cgi/wiki/Main?version=3'
```

Paths are relative to the given resource:

```
>>> get_relative_url(env, main, href, '.')
'/trac.cgi/wiki/Main?version=3'
```

```
>>> get_relative_url(env, main, href, './Sub')
'/trac.cgi/wiki/Main/Sub'
```

```
>>> get_relative_url(env, main, href, './Sub/Infra')
'/trac.cgi/wiki/Main/Sub/Infra'
```

```
>>> get_relative_url(env, main, href, './Sub/')
'/trac.cgi/wiki/Main/Sub'
```

```
>>> mainsub = main(id='Main/Sub')
>>> get_relative_url(env, mainsub, href, '..')
'/trac.cgi/wiki/Main'
```

```
>>> get_relative_url(env, main, href, '../Other')
'/trac.cgi/wiki/Other'
```

References always stay within the current resource realm:

```
>>> get_relative_url(env, mainsub, href, '../..')
'/trac.cgi/wiki'
```

```
>>> get_relative_url(env, mainsub, href, '../../..')
'/trac.cgi/wiki'
```

```
>>> get_relative_url(env, mainsub, href, '/toplevel')
'/trac.cgi/wiki/toplevel'
```

Extra keyword arguments are forwarded as query parameters:

```
>>> get_relative_url(env, main, href, action='diff')
'/trac.cgi/wiki/Main?action=diff&version=3'
```

trac.resource.**get_resource_description**(*env*, *resource*, *format='default'*, ***kwargs*)

Retrieve a standardized description for the given resource.

This function delegates the work to the resource manager for that resource if it implements a *get_resource_description* method, otherwise reverts to simple presentation of the realm and identifier information.

Parameters

- **env** – the Environment where IResourceManager components live
- **resource** – the *Resource* object specifying the Trac resource
- **format** – which formats to use for the description

Additional keyword arguments can be provided and will be propagated to resource manager that might make use of them (typically, a context parameter for creating context dependent output).

```
>>> from trac.test import EnvironmentStub
>>> env = EnvironmentStub()
>>> main = Resource('generic', 'Main')
>>> get_resource_description(env, main)
u'generic:Main'
```

```
>>> get_resource_description(env, main(version=3))
u'generic:Main'
```

```
>>> get_resource_description(env, main(version=3), format='summary')
u'generic:Main at version 3'
```

trac.resource.**get_resource_url**(*env*, *resource*, *href*, ***kwargs*)

Retrieve the canonical URL for the given resource.

This function delegates the work to the resource manager for that resource if it implements a *get_resource_url* method, otherwise reverts to simple '/realm/identifier' style URLs.

Parameters

- **env** – the `Environment` where `IResourceManager` components live
- **resource** – the *Resource* object specifying the Trac resource
- **href** – an `Href` object used for building the URL

Additional keyword arguments are translated as query parameters in the URL.

```
>>> from trac.test import EnvironmentStub
>>> from trac.web.href import Href
>>> env = EnvironmentStub()
>>> href = Href('/trac.cgi')
>>> main = Resource('generic', 'Main')
>>> get_resource_url(env, main, href)
'/trac.cgi/generic/Main'
```

```
>>> get_resource_url(env, main(version=3), href)
'/trac.cgi/generic/Main?version=3'
```

```
>>> get_resource_url(env, main(version=3), href)
'/trac.cgi/generic/Main?version=3'
```

```
>>> get_resource_url(env, main(version=3), href, action='diff')
'/trac.cgi/generic/Main?action=diff&version=3'
```

```
>>> get_resource_url(env, main(version=3), href, action='diff', version=5)
'/trac.cgi/generic/Main?action=diff&version=5'
```

`trac.resource.`**render_resource_link**(*env*, *context*, *resource*, *format='default'*)

Utility for generating a link `Element` to the given resource.

Some component manager may directly use an extra `context` parameter in order to directly generate rich content. Otherwise, the textual output is wrapped in a link to the resource.

`trac.resource.`**resource_exists**(*env*, *resource*)

Checks for resource existence without actually instantiating a model.

Returns `True` if the resource exists, `False` if it doesn't and `None` in case no conclusion could be made (i.e. when `IResourceManager.resource_exists` is not implemented).

```
>>> from trac.test import EnvironmentStub
>>> env = EnvironmentStub()
```

```
>>> resource_exists(env, Resource('dummy-realm', 'dummy-id')) is None
True
>>> resource_exists(env, Resource('dummy-realm'))
False
```

trac.search.api

class `trac.search.api.`**ISearchSource**

Bases: *trac.core.Interface*

Extension point interface for adding search sources to the search system.

get_search_filters(*req*)
Return a list of filters that this search source supports.

Each filter must be a `(name, label[, default])` tuple, where `name` is the internal name, `label` is a human-readable name for display and `default` is an optional boolean for determining whether this filter is searchable by default.

get_search_results(*req*, *terms*, *filters*)
Return a list of search results matching each search term in `terms`.

The `filters` parameters is a list of the enabled filters, each item being the name of the tuples returned by `get_search_events`.

The events returned by this function must be tuples of the form `(href, title, date, author, excerpt)`.

trac.search.api.**search_to_regexps**(*terms*)
Convert search query terms into regular expressions.

trac.search.api.**search_to_sql**(*db*, *columns*, *terms*)
Convert a search query into an SQL WHERE clause and corresponding parameters.

The result is returned as an `(sql, params)` tuple.

trac.search.web_ui

class trac.search.web_ui.**SearchModule**
Bases: *trac.core.Component*

Controller for the search sub-system

default_disabled_filters
Specifies which search filters should be disabled by default on the search page. This will also restrict the filters for the quick search function. The filter names defined by default components are: `wiki`, `ticket`, `milestone` and `changeset`. For plugins, look for their implementation of the ISearchSource interface, in the `get_search_filters()` method, the first member of returned tuple. Once disabled, search filters can still be manually enabled by the user on the search page.

min_query_length
Minimum length of query string allowed when performing a search.

search_sources
List of components that implement *ISearchSource*

trac.ticket.admin

class trac.ticket.admin.**TicketAdmin**
Bases: *trac.core.Component*

trac-admin command provider for ticket administration.

trac.ticket.api

class trac.ticket.api.**IMilestoneChangeListener**
Bases: *trac.core.Interface*

Extension point interface for components that require notification when milestones are created, modified, or deleted.

milestone_changed(*milestone*, *old_values*)
Called when a milestone is modified.

`old_values` is a dictionary containing the previous values of the milestone properties that changed. Currently those properties can be 'name', 'due', 'completed', or 'description'.

milestone_created(*milestone*)
Called when a milestone is created.

milestone_deleted(*milestone*)
Called when a milestone is deleted.

class `trac.ticket.api.`**ITicketActionController**
Bases: *trac.core.Interface*

Extension point interface for components willing to participate in the ticket workflow.

This is mainly about controlling the changes to the ticket ''status'', though not restricted to it.

apply_action_side_effects(*req*, *ticket*, *action*)
Perform side effects once all changes have been made to the ticket.

Multiple controllers might be involved, so the apply side-effects offers a chance to trigger a side-effect based on the given `action` after the new state of the ticket has been saved.

This method will only be called if the controller claimed to handle the given `action` in the call to *get_ticket_actions*.

get_all_status()
Returns an iterable of all the possible values for the ''status" field this action controller knows about.

This will be used to populate the query options and the like. It is assumed that the initial status of a ticket is 'new' and the terminal status of a ticket is 'closed'.

get_ticket_actions(*req*, *ticket*)
Return an iterable of `(weight, action)` tuples corresponding to the actions that are contributed by this component. The list is dependent on the current state of the ticket and the actual request parameter.

`action` is a key used to identify that particular action. (note that 'history' and 'diff' are reserved and should not be used by plugins)

The actions will be presented on the page in descending order of the integer weight. The first action in the list is used as the default action.

When in doubt, use a weight of 0.

get_ticket_changes(*req*, *ticket*, *action*)
Return a dictionary of ticket field changes.

This method must not have any side-effects because it will also be called in preview mode (`req.args['preview']` will be set, then). See *apply_action_side_effects* for that. If the latter indeed triggers some side-effects, it is advised to emit a warning (`trac.web.chrome.add_warning(req, reason)`) when this method is called in preview mode.

This method will only be called if the controller claimed to handle the given `action` in the call to *get_ticket_actions*.

render_ticket_action_control(*req*, *ticket*, *action*)
Return a tuple in the form of `(label, control, hint)`

`label` is a short text that will be used when listing the action, `control` is the markup for the action control and `hint` should explain what will happen if this action is taken.

This method will only be called if the controller claimed to handle the given `action` in the call to *get_ticket_actions*.

Note that the radio button for the action has an `id` of `"action_%s" % action`. Any `id`s used in `control` need to be made unique. The method used in the default ITicketActionController is to use `"action_%s_something" % action`.

class trac.ticket.api.**ITicketChangeListener**
Bases: *trac.core.Interface*

Extension point interface for components that require notification when tickets are created, modified, or deleted.

ticket_change_deleted(*ticket*, *cdate*, *changes*)
Called when a ticket change is deleted.

`changes` is a dictionary of tuple `(oldvalue, newvalue)` containing the ticket change of the fields that have changed.

ticket_changed(*ticket*, *comment*, *author*, *old_values*)
Called when a ticket is modified.

`old_values` is a dictionary containing the previous values of the fields that have changed.

ticket_comment_modified(*ticket*, *cdate*, *author*, *comment*, *old_comment*)
Called when a ticket comment is modified.

ticket_created(*ticket*)
Called when a ticket is created.

ticket_deleted(*ticket*)
Called when a ticket is deleted.

class trac.ticket.api.**ITicketManipulator**
Bases: *trac.core.Interface*

Miscellaneous manipulation of ticket workflow features.

prepare_ticket(*req*, *ticket*, *fields*, *actions*)
Not currently called, but should be provided for future compatibility.

validate_comment(*comment*)
Validate ticket comment.

Must return a list of messages, one for each problem detected. The return value `[]` indicates no problems.

Since 1.3.2

validate_ticket(*req*, *ticket*)
Validate a ticket after it's been populated from user input.

Must return a list of `(field, message)` tuples, one for each problem detected. `field` can be `None` to indicate an overall problem with the ticket. Therefore, a return value of `[]` means everything is OK.

class trac.ticket.api.**TicketFieldList**(**args*)
Bases: `list`

Improved ticket field list, allowing access by name.

trac.ticket.batch

class trac.ticket.batch.**BatchModifyModule**
Bases: *trac.core.Component*

Ticket batch modification module.

This component allows multiple tickets to be modified in one request from the custom query page. For users with the TICKET_BATCH_MODIFY permission it will add a [TracBatchModify batch modify] section underneath custom query results. Users can choose which tickets and fields they wish to modify.

ticket_manipulators
List of components that implement *ITicketManipulator*

trac.ticket.batch.**datetime_now**()
[tz] -> new datetime with tz's local day and time.

trac.ticket.default_workflow

class trac.ticket.default_workflow.**ConfigurableTicketWorkflow**
Bases: *trac.core.Component*

Ticket action controller which provides actions according to a workflow defined in trac.ini.

The workflow is defined in the `[ticket-workflow]` section of the [wiki:TracIni#ticket-workflow-section trac.ini] configuration file.

environment_created()
When an environment is created, we provide the basic-workflow, unless a ticket-workflow section already exists.

get_actions_by_operation(*operation*)
Return a list of all actions with a given operation (for use in the controller's get_all_status())

get_actions_by_operation_for_req(*req*, *ticket*, *operation*)
Return list of all actions with a given operation that are valid in the given state for the controller's get_ticket_actions().

If state='*' (the default), all actions with the given operation are returned.

get_all_status()
Return a list of all states described by the configuration.

get_allowed_owners(*req*, *ticket*, *action*)
Returns users listed in the `set_owner` field of the action or possessing the `TICKET_MODIFY` permission if `set_owner` is not specified.

This method can be overridden in a subclasses in order to customize the list of users that populate the assign-to select box.

Since 1.3.2

get_ticket_actions(*req*, *ticket*)
Returns a list of (weight, action) tuples that are valid for this request and this ticket.

ticket_workflow_section
The workflow for tickets is controlled by plugins. By default, there's only a *ConfigurableTicketWorkflow* component in charge. That component allows the workflow to be configured via this section in the `trac.ini` file. See TracWorkflow for more details.

trac.ticket.default_workflow.**get_workflow_config**(*config*)
Usually passed self.config, this will return the parsed ticket-workflow section.

trac.ticket.default_workflow.**load_workflow_config_snippet**(*config*, *filename*)
Loads the ticket-workflow section from the given file (expected to be in the 'workflows' tree) into the provided config.

trac.ticket.default_workflow.**parse_workflow_config**(*rawactions*)
Given a list of options from [ticket-workflow]

trac.ticket.model

Components

class trac.ticket.model.**MilestoneCache**
Bases: *trac.core.Component*

Cache for milestone data and factory for 'milestone' resources.

factory(*name_due_completed_description_tuple*, *milestone_component=None*)
Build a Milestone object from milestone data.

That instance remains *private*, i.e. can't be retrieved by name by other processes or even by other threads in the same process, until its insert method gets called with success.

fetchall()
Iterator on all milestones.

fetchone(*name*, *milestone=None*)
Retrieve an existing milestone having the given name.

If milestone is specified, fill that instance instead of creating a fresh one.

Returns None if no such milestone exists

milestones
Dictionary containing milestone data, indexed by name.

Milestone data consist of a tuple containing the name, the datetime objects for due and completed dates and the description.

Miscellaneous

trac.ticket.model.**simplify_whitespace**(*name*)
Strip spaces and remove duplicate spaces within names

trac.ticket.notification

class trac.ticket.notification.**BatchTicketChangeEvent**(*targets*, *time*, *author*, *comment*, *new_values*, *action*)
Bases: trac.notification.api.NotificationEvent

Represent a ticket batch modify NotificationEvent.

class trac.ticket.notification.**CarbonCopySubscriber**
Bases: *trac.core.Component*

Carbon copy subscriber for cc ticket field.

class trac.ticket.notification.**TicketAttachmentNotifier**
Bases: *trac.core.Component*

Sends notification on attachment change.

class trac.ticket.notification.**TicketChangeEvent**(*category*, *target*, *time*, *author*, *comment=None*, *changes=None*, *attachment=None*)
Bases: trac.notification.api.NotificationEvent

Represent a ticket change NotificationEvent.

class trac.ticket.notification.**TicketFormatter**
Bases: *trac.core.Component*

Format *TicketChangeEvent* notifications.

ambiguous_char_width
Width of ambiguous characters that should be used in the table of the notification mail.

If single, the same width as characters in US-ASCII. This is expected by most users. If double, twice the width of US-ASCII characters. This is expected by CJK users.

batch_subject_template
Like *ticket_subject_template* but for batch modifications. (''since 1.0'')

ticket_subject_template
A Jinja2 text template snippet used to get the notification subject.

The template variables are documented on the [TracNotification#Customizingthee-mailsubject TracNotification] page.

class trac.ticket.notification.**TicketOwnerSubscriber**
Bases: *trac.core.Component*

Allows ticket owners to subscribe to their tickets.

class trac.ticket.notification.**TicketPreviousUpdatersSubscriber**
Bases: *trac.core.Component*

Allows subscribing to future changes simply by updating a ticket.

class trac.ticket.notification.**TicketReporterSubscriber**
Bases: *trac.core.Component*

Allows the users to subscribe to tickets that they report.

class trac.ticket.notification.**TicketUpdaterSubscriber**
Bases: *trac.core.Component*

Allows updaters to subscribe to their own updates.

trac.ticket.query

exception trac.ticket.query.**QuerySyntaxError**(*message*, *title=None*, *show_traceback=False*)
Bases: *trac.core.TracError*

Exception raised when a ticket query cannot be parsed from a string.

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

exception trac.ticket.query.**QueryValueError**(*errors*)
Bases: *trac.core.TracError*

Exception raised when a ticket query has bad constraint values.

trac.ticket.query.**datetime_now**()
[tz] -> new datetime with tz's local day and time.

trac.ticket.report

trac.ticket.report.**cell_value**(*v*)
Normalize a cell value for display. >>> (cell_value(None), cell_value(0), cell_value(1), cell_value('v')) ('', '0', u'1', u'v')

trac.ticket.report.**split_sql**(*sql*, *clause_re*, *skel=None*)
Split an SQL query according to a toplevel clause regexp.

We assume there's only one such clause present in the outer query.

```
>>> split_sql('''SELECT a FROM x  ORDER            BY u, v''', _order_by_re)
('SELECT a FROM x  ', ' u, v')
```

trac.ticket.report.**sql_skeleton**(*sql*)
Strip an SQL query to leave only its toplevel structure.

This is probably not 100% robust but should be enough for most needs.

```
>>> re.sub('\s+', lambda m: '<%d>' % len(m.group(0)), sql_skeleton(''' \n
↪SELECT a FROM (SELECT x FROM z ORDER BY COALESCE(u, ')/*')) ORDER \n
↪* SELECT a FROM (SELECT x /* FROM z                                \n
↪          ORDER BY */ COALESCE(u, '\)X')) ORDER */          \n
↪(SELECT s FROM f WHERE v in ('ORDER BY', '(\')')          \n
↪ORDER BY (1), '') -- LIMIT                                \n
'<10>SELECT<1>a<1>FROM<48>ORDER<164>BY<1>c,<144>'
```

trac.ticket.roadmap – The Roadmap and Milestone modules

The component responsible for the *Roadmap* feature in Trac is the *RoadmapModule*. It provides an overview of the milestones and the progress in each of these milestones.

The component responsible for interacting with each milestone is the *MilestoneModule*. A milestone also provides an overview of the progress in terms of tickets processed.

The grouping of tickets in each progress bar is governed by the use of another component implementing the *ITicketGroupStatsProvider* interface. By default, this is the *DefaultTicketGroupStatsProvider* (for both the *RoadmapModule* and the *MilestoneModule*), which provides a configurable way to specify how tickets are grouped.

Interfaces

class trac.ticket.roadmap.**ITicketGroupStatsProvider**
Bases: *trac.core.Interface*

See also trac.ticket.roadmap.ITicketGroupStatsProvider extension point

get_ticket_group_stats(*ticket_ids*)
Gather statistics on a group of tickets.

This method returns a valid *TicketGroupStats* object.

class trac.ticket.roadmap.**TicketGroupStats**(*title*, *unit*)
Bases: object

Encapsulates statistics on a group of tickets.

Parameters

- **title** – the display name of this group of stats (e.g. 'ticket status')
- **unit** – is the units for these stats in plural form, e.g. _('hours')

add_interval(*title*, *count*, *qry_args*, *css_class*, *overall_completion=None*)
Adds a division to this stats' group's progress bar.

Parameters

- **title** – the display name (e.g. 'closed', 'spent effort') of this interval that will be displayed in front of the unit name
- **count** – the number of units in the interval
- **qry_args** – a dict of extra params that will yield the subset of tickets in this interval on a query.
- **css_class** – is the css class that will be used to display the division
- **overall_completion** – can be set to true to make this interval count towards overall completion of this group of tickets.

Changed in version 0.12: deprecated countsToProg argument was removed, use overall_completion instead

Components

class trac.ticket.roadmap.**MilestoneModule**
Bases: *trac.core.Component*

View and edit individual milestones.

default_group_by
Default field to use for grouping tickets in the grouped progress bar. (''since 1.2'')

get_default_due(*req*)
Returns a datetime object representing the default due date in the user's timezone. The default due time is 18:00 in the user's time zone.

stats_provider
Name of the component implementing *ITicketGroupStatsProvider*, which is used to collect statistics on groups of tickets for display in the milestone views.

class trac.ticket.roadmap.**RoadmapModule**
Bases: *trac.core.Component*

Give an overview over all the milestones.

stats_provider
Name of the component implementing *ITicketGroupStatsProvider*, which is used to collect statistics on groups of tickets for display in the roadmap views.

class trac.ticket.roadmap.**DefaultTicketGroupStatsProvider**
Bases: *trac.core.Component*

Configurable ticket group statistics provider.

See TracIni#milestone-groups-section for a detailed example configuration.

Helper Functions

trac.ticket.roadmap.**apply_ticket_permissions**(*env*, *req*, *tickets*)
Apply permissions to a set of milestone tickets as returned by *get_tickets_for_milestone()*.

trac.ticket.roadmap.**get_tickets_for_milestone**(*env*, *milestone=None*, *field='component'*)
Retrieve all tickets associated with the given `milestone`.

trac.ticket.roadmap.**grouped_stats_data**(*env*, *stats_provider*, *tickets*, *by*, *per_group_stats_data*)
Get the `tickets` stats data grouped by ticket field `by`.

`per_group_stats_data(gstat, group_name)` should return a data dict to include for the group with field value `group_name`.

trac.ticket.roadmap.**get_num_tickets_for_milestone**(*env*, *milestone*, *exclude_closed=False*)
Returns the number of tickets associated with the milestone.

Parameters

- **milestone** – name of a milestone or a Milestone instance.
- **exclude_closed** – whether tickets with status 'closed' should be excluded from the count. Defaults to False.

Since 1.2

trac.ticket.roadmap.**group_milestones**(*milestones*, *include_completed*)
Group milestones into "open with due date", "open with no due date", and possibly "completed". Return a list of (label, milestones) tuples.

trac.ticket.web_ui

class trac.ticket.web_ui.**DefaultTicketPolicy**
Bases: *trac.core.Component*

Default permission policy for the ticket system.

Authenticated users with `TICKET_APPEND` can edit their own ticket comments. Authenticated users with `TICKET_APPEND` or `TICKET_CHGPROP` can edit the description of a ticket they reported.

exception trac.ticket.web_ui.**InvalidTicket**(*message*, *title=None*, *show_traceback=False*)
Bases: *trac.core.TracError*

Exception raised when a ticket fails validation.

Since 1.3.2 deprecated and will be removed in 1.5.1

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

trac.ticket.web_ui.**datetime_now**()
[tz] -> new datetime with tz's local day and time.

trac.timeline.api

class trac.timeline.api.**ITimelineEventProvider**
Bases: *trac.core.Interface*

Extension point interface for adding sources for timed events to the timeline.

get_timeline_events(*req*, *start*, *stop*, *filters*)
Return a list of events in the time range given by the `start` and `stop` parameters.

The `filters` parameters is a list of the enabled filters, each item being the name of the tuples returned by *get_timeline_filters*.

The events are `(kind, date, author, data)` tuples, where `kind` is a string used for categorizing the event, `date` is a `datetime` object, `author` is a string and `data` is some private data that the component will reuse when rendering the event.

When the event has been created indirectly by another module, like this happens when calling `AttachmentModule.get_timeline_events()` the tuple can also specify explicitly the provider by returning tuples of the following form: `(kind, date, author, data, provider)`.

get_timeline_filters(*req*)
Return a list of filters that this event provider supports.

Each filter must be a (name, label) tuple, where `name` is the internal name, and `label` is a human-readable name for display.

Optionally, the tuple can contain a third element, `checked`. If `checked` is omitted or True, the filter is active by default, otherwise it will be inactive.

render_timeline_event(*context*, *field*, *event*)
Display the title of the event in the given context.

Parameters

- **context** – the `RenderingContext` object that can be used for rendering
- **field** – what specific part information from the event should be rendered: can be the 'title', the 'description' or the 'url'
- **event** – the event tuple, as returned by *get_timeline_events*

trac.timeline.web_ui

trac.timeline.web_ui.**datetime_now**()
[tz] -> new datetime with tz's local day and time.

trac.util – General purpose utilities

The *trac.util* package is a hodgepodge of various categories of utilities. If a category contains enough code in itself, it earns a sub-module on its own, like the following ones:

trac.util.autoreload

trac.util.autoreload.**main**(*func*, *modification_callback*, **args*, ***kwargs*)
Run the given function and restart any time modules are changed.

trac.util.compat

Various classes and functions to provide some backwards-compatibility with previous versions of Python from 2.6 onward.

class trac.util.compat.**Popen**(*args*, *bufsize=0*, *executable=None*, *stdin=None*, *stdout=None*, *stderr=None*, *preexec_fn=None*, *close_fds=False*, *shell=False*, *cwd=None*, *env=None*, *universal_newlines=False*, *startupinfo=None*, *creationflags=0*)

Bases: subprocess.Popen

Popen objects are supported as context managers starting in Python 3.2. This code was taken from Python 3.5 and can be removed when support for Python < 3.2 is dropped.

Create new Popen instance.

trac.util.compat.**wait_for_file_mtime_change**(*filename*)

This function is typically called before a file save operation, waiting if necessary for the file modification time to change. The purpose is to avoid successive file updates going undetected by the caching mechanism that depends on a change in the file modification time to know when the file should be reparsed.

trac.util.concurrency

class trac.util.concurrency.**ThreadLocal**(***kwargs*)

Bases: thread._local

A thread-local storage allowing to set default values on construction.

trac.util.daemon

trac.util.daemon.**daemonize**(*pidfile=None*, *progname=None*, *stdin='/dev/null'*, *stdout='/dev/null'*, *stderr='/dev/null'*, *umask=18*)

Fork a daemon process.

trac.util.daemon.**handle_signal**(*signum*, *frame*)

Handle signals sent to the daemonized process.

trac.util.datefmt – Date and Time manipulation

Since version 0.10, Trac mainly uses datetime.datetime objects for handling date and time values. This enables us to properly deal with timezones so that time can be shown in the user's own local time.

Current time

trac.util.datefmt.**datetime_now**()

[tz] -> new datetime with tz's local day and time.

trac.util.datefmt.**time_now**()

time() -> floating point number

Return the current time in seconds since the Epoch. Fractions of a second may be present if the system clock provides them.

Conversion

From "anything" to a datetime:

trac.util.datefmt.**to_datetime**(*t*, *tzinfo=None*)
Convert t into a datetime object in the tzinfo timezone.

If no tzinfo is given, the local timezone localtz will be used.

t is converted using the following rules:

- If t is already a datetime object,
- if it is timezone-"naive", it is localized to tzinfo
- if it is already timezone-aware, t is mapped to the given timezone (datetime.datetime.astimezone)
- If t is None, the current time will be used.
- If t is a number, it is interpreted as a timestamp.

Any other input will trigger a TypeError.

All returned datetime instances are timezone aware and normalized.

A datetime can be converted to milliseconds and microseconds timestamps. The latter is the preferred representation for dates and times values for storing them in the database, since Trac 0.12.

trac.util.datefmt.**to_timestamp**(*dt*)
Return the corresponding POSIX timestamp

trac.util.datefmt.**to_utimestamp**(*dt*)
Return a microsecond POSIX timestamp for the given datetime.

Besides *to_datetime*, there's a specialized conversion from microseconds timestamps to datetime:

trac.util.datefmt.**from_utimestamp**(*ts*)
Return the datetime for the given microsecond POSIX timestamp.

Parsing

trac.util.datefmt.**parse_date**(*text*, *tzinfo=None*, *locale=None*, *hint='date'*)

Formatting

trac.util.datefmt.**pretty_timedelta**(*time1*, *time2=None*, *resolution=None*)
Calculate time delta between two datetime objects. (the result is somewhat imprecise, only use for prettyprinting).

If either time1 or time2 is None, the current time will be used instead.

trac.util.datefmt.**format_datetime**(*t=None*, *format='%x %X'*, *tzinfo=None*, *locale=None*)
Format the datetime object t into an unicode string

If t is None, the current time will be used.

The formatting will be done using the given format, which consist of conventional strftime keys. In addition the format can be 'iso8601' to specify the international date format (compliant with RFC 3339).

tzinfo will default to the local timezone if left to None.

Derivatives:

trac.util.datefmt.**format_date**(*t=None*, *format='%x'*, *tzinfo=None*, *locale=None*)
Convenience method for formatting the date part of a datetime object. See *format_datetime* for more details.

trac.util.datefmt.**format_time**(*t=None*, *format='%X'*, *tzinfo=None*, *locale=None*)
Convenience method for formatting the time part of a datetime object. See *format_datetime* for more details.

Propose suggestion for date/time input format:

trac.util.datefmt.**get_date_format_hint**(*locale=None*)
Present the default format used by *format_date* in a human readable form. This is a format that will be recognized by *parse_date* when reading a date.

trac.util.datefmt.**get_datetime_format_hint**(*locale=None*)
Present the default format used by *format_datetime* in a human readable form. This is a format that will be recognized by *parse_date* when reading a date.

trac.util.datefmt.**http_date**(*t=None*)
Format datetime object t as a rfc822 timestamp

trac.util.datefmt.**is_24_hours**(*locale*)
Returns True for 24 hour time formats.

Formatting and parsing according to user preferences:

trac.util.datefmt.**user_time**(*req*, *func*, **args*, ***kwargs*)
A helper function which passes to tzinfo and locale keyword arguments of func using req parameter. It is expected to be used with format_* and *parse_date* methods in *trac.util.datefmt* package.

Parameters

- **req** – a instance of Request
- **func** – a function which must accept tzinfo and locale keyword arguments
- **args** – arguments which pass to func function
- **kwargs** – keyword arguments which pass to func function

jQuery UI datepicker helpers

trac.util.datefmt.**get_date_format_jquery_ui**(*locale*)
Get the date format for the jQuery UI datepicker library.

trac.util.datefmt.**get_time_format_jquery_ui**(*locale*)
Get the time format for the jQuery UI timepicker addon.

trac.util.datefmt.**get_day_names_jquery_ui**(*req*)
Get the day names for the jQuery UI datepicker library

trac.util.datefmt.**get_first_week_day_jquery_ui**(*req*)
Get first week day for jQuery date picker

trac.util.datefmt.**get_month_names_jquery_ui**(*req*)
Get the month names for the jQuery UI datepicker library

trac.util.datefmt.**get_timezone_list_jquery_ui**(*t=None*)
Get timezone list for jQuery timepicker addon

Timezone utilities

trac.util.datefmt.**localtz**
A global *LocalTimezone* instance.

class trac.util.datefmt.**LocalTimezone**(*offset=None*)
Bases: datetime.tzinfo

A 'local' time zone implementation

trac.util.datefmt.**all_timezones**

List of all available timezones. If pytz is installed, this corresponds to a rich variety of "official" timezones, otherwise this corresponds to *FixedOffset* instances, ranging from GMT -12:00 to GMT +13:00.

trac.util.datefmt.**timezone**(*tzname*)
Fetch timezone instance by name or raise KeyError

trac.util.datefmt.**get_timezone**(*tzname*)
Fetch timezone instance by name or return None

class trac.util.datefmt.**FixedOffset**(*offset*, *name*)
Bases: datetime.tzinfo

Fixed offset in minutes east from UTC.

trac.util.html – HTML transformations

Building HTML programmatically

With the introduction of the Jinja2 template engine in Trac 1.3.x, the (X)HTML content is produced either using Jinja2 snippet templates (see jinja2template) or using the builder API defined in this module. This builder API closely matches the Genshi genshi.builder API on the surface.

The builder API

The *tag* builder has some knowledge about generating HTML content, like knowing which elements are "void" elements, how attributes should be written when given a boolean value, etc.

trac.util.html.**tag**
An *ElementFactory*.

class trac.util.html.**ElementFactory**
Bases: *trac.util.html.XMLElementFactory*

An element factory can be used to build Fragments and Elements for arbitrary tag names.

class trac.util.html.**Element**(*tag*, **args*, ***kwargs*)
Bases: *trac.util.html.XMLElement*

An element represents an HTML element, with a tag name, attributes and content.

Some elements and attributes are rendered specially, according to the HTML5 specification (or going there...)

class trac.util.html.**Fragment**(**args*)
Bases: object

A fragment represents a sequence of strings or elements.

Note that the *Element* relies on the following lower-level API for generating the HTML attributes.

trac.util.html.**html_attribute**(*key*, *val*)
Returns the actual value for the attribute `key`, for the given `value`.

This follows the rules described in the HTML5 spec (Double-quoted attribute value syntax).

In addition, it treats the `'class'` and the `'style'` attributes in a special way, as it processes them through *classes* and *styles*.

Return type a `Markup` object containing the escaped attribute value, but it can also be `None` to indicate that the attribute should be omitted from the output

trac.util.html.**classes**(**args*, ***kwargs*)
Helper function for dynamically assembling a list of CSS class names in templates.

Any positional arguments are added to the list of class names. All positional arguments must be strings:

```
>>> classes('foo', 'bar')
u'foo bar'
```

In addition, the names of any supplied keyword arguments are added if they have a truth value:

```
>>> classes('foo', bar=True)
u'foo bar'
>>> classes('foo', bar=False)
u'foo'
```

If none of the arguments are added to the list, this function returns `''`:

```
>>> classes(bar=False)
u''
```

trac.util.html.**styles**(**args*, ***kwargs*)
Helper function for dynamically assembling a list of CSS style name and values in templates.

Any positional arguments are added to the list of styles. All positional arguments must be strings or dicts:

```
>>> styles('foo: bar', 'fu: baz', {'bottom-right': '1em'})
u'foo: bar; fu: baz; bottom-right: 1em'
```

In addition, the names of any supplied keyword arguments are added if they have a string value:

```
>>> styles(foo='bar', fu='baz')
u'foo: bar; fu: baz'
>>> styles(foo='bar', bar=False)
u'foo: bar'
```

If none of the arguments are added to the list, this function returns `''`:

```
>>> styles(bar=False)
u''
```

This HTML-specific behavior can be a hindrance to writing generic XML. In that case, better use the *xml* builder.

trac.util.html.**xml**
An *XMLElementFactory*.

class trac.util.html.**XMLElementFactory**
Bases: `object`

An XML element factory can be used to build Fragments and XMLElements for arbitrary tag names.

class trac.util.html.**XMLElement**(*tag*, **args*, ***kwargs*)
Bases: *trac.util.html.Fragment*

An element represents an XML element, with a tag name, attributes and content.

Building HTML from strings

It is also possible to mark an arbitrary string as containing HTML content, so that it will not be HTML-escaped by the template engine.

For this, use the `Markup` class, taken from the `markupsafe` package (itself a dependency of the Jinja2 package).

The `Markup` class should be imported from the present module:

```
from trac.util.html import Markup
```

HTML clean-up and sanitization

class trac.util.html.**TracHTMLSanitizer**(*safe_schemes=frozenset(['mailto', 'ftp', 'http', 'file', 'https', None])*, *safe_css=frozenset(['counter-reset', 'counter-increment', 'min-height', 'quotes', 'border-top', 'font', 'list-style-image', 'outline-width', 'border-right', 'border-radius', 'border-bottom', 'border-spacing', 'background', 'list-style-type', 'text-align', 'page-break-inside', 'orphans', 'page-break-before', 'border-bottom-right-radius', 'line-height', 'padding-left', 'font-size', 'right', 'word-spacing', 'padding-top', 'outline-style', 'bottom', 'content', 'border-right-style', 'padding-right', 'border-left-style', 'background-color', 'border-bottom-color', 'outline-color', 'unicode-bidi', 'max-width', 'font-family', 'caption-side', 'text-transform', 'border-right-width', 'border-top-style', 'color', 'border-collapse', 'border-bottom-width', 'float', 'height', 'max-height', 'margin-right', 'border-top-width', 'border-bottom-left-radius', 'top', 'border-width', 'min-width', 'width', 'font-variant', 'border-top-color', 'background-position', 'empty-cells', 'direction', 'border-left', 'visibility', 'padding', 'border-style', 'background-attachment', 'overflow', 'border-bottom-style', 'cursor', 'margin', 'display', 'border-left-width', 'letter-spacing', 'border-top-left-radius', 'vertical-align', 'clip', 'border-color', 'list-style', 'padding-bottom', 'margin-left', 'widows', 'border', 'font-style', 'border-left-color', 'background-repeat', 'table-layout', 'margin-bottom', 'border-top-right-radius', 'font-weight', 'opacity', 'border-right-color', 'page-break-after', 'white-space', 'text-indent', 'background-image', 'outline', 'clear', 'z-index', 'text-decoration', 'margin-top', 'position', 'left', 'list-style-position']), safe_tags=frozenset(['em', 'pre', 'code', 'p', 'h2', 'h3', 'h1', 'h6', 'h4', 'h5', 'table', 'font', 'u', 'select', 'kbd', 'strong', 'span', 'sub', 'img', 'area', 'menu', 'tt', 'tr', 'tbody', 'label', 'hr', 'dfn', 'tfoot', 'th', 'sup', 'strike', 'input', 'td', 'samp', 'cite', 'thead', 'map', 'dl', 'blockquote', 'fieldset', 'option', 'form', 'acronym', 'big', 'dd', 'var', 'ol', 'abbr', 'br', 'address', 'optgroup', 'li', 'dt', 'ins', 'legend', 'a', 'b', 'center', 'textarea', 'colgroup', 'i', 'button', 'q', 'caption', 's', 'del', 'small', 'div', 'col', 'dir', 'ul']), safe_attrs=frozenset(['rev', 'prompt', 'color', 'colspan', 'accesskey', 'usemap', 'cols', 'accept', 'datetime', 'char', 'accept-charset', 'shape', 'href', 'hreflang', 'selected', 'frame', 'type', 'alt', 'nowrap', 'border', 'id', 'axis', 'compact', 'style', 'rows', 'checked', 'for', 'start', 'hspace', 'charset', 'ismap', 'label', 'target', 'bgcolor', 'readonly', 'rel', 'valign', 'scope', 'size', 'cellspacing', 'cite', 'media', 'multiple', 'src', 'rules', 'nohref', 'action', 'rowspan', 'abbr', 'span', 'method', 'height', 'class', 'enctype', 'lang', 'disabled', 'name', 'charoff', 'clear', 'summary', 'value', 'longdesc', 'headers', 'vspace', 'noshade', 'coords', 'width', 'maxlength', 'cellpadding', 'title', 'align', 'dir', 'tabindex']), uri_attrs=frozenset(['src', 'lowsrc', 'href', 'dynsrc', 'background', 'action']), safe_origins=frozenset(['data:']))*

Bases: object

Sanitize HTML constructions which are potentially vector of phishing or XSS attacks, in user-supplied HTML.

The usual way to use the sanitizer is to call the *sanitize* method on some potentially unsafe HTML content.

Note that for backward compatibility, the TracHTMLSanitizer still behaves as a Genshi filter.

See also genshi.HTMLSanitizer from which the TracHTMLSanitizer has evolved.

Note: safe_schemes and safe_css have to remain the first parameters, for backward-compatibility purpose.

is_safe_css(*prop*, *value*)
Determine whether the given css property declaration is to be considered safe for inclusion in the output.

is_safe_elem(*tag*, *attrs*)
Determine whether the given element should be considered safe for inclusion in the output.

Parameters

- **tag** (*QName or basestring*) – the tag name of the element
- **attrs** (*Attrs or list*) – the element attributes

Returns whether the element should be considered safe

Return type bool

is_safe_uri(*uri*)
Determine whether the given URI is to be considered safe for inclusion in the output.

The default implementation checks whether the scheme of the URI is in the set of allowed URIs (`safe_schemes`).

```
>>> sanitizer = TracHTMLSanitizer()
>>> sanitizer.is_safe_uri('http://example.org/')
True
>>> sanitizer.is_safe_uri('javascript:alert(document.cookie)')
False
```

Parameters **uri** – the URI to check

Returns `True` if the URI can be considered safe, `False` otherwise

Return type `bool`

sanitize(*html*)
Transforms the incoming HTML by removing anything's that deemed unsafe.

Parameters **html** – the input HTML

Type basestring

Returns the sanitized content

Return type Markup

sanitize_attrs(*tag*, *attrs*)
Remove potentially dangerous attributes and sanitize the style attribute .

Parameters **tag** – the tag name of the element

Returns a dict containing only safe or sanitized attributes

Return type dict

sanitize_css(*text*)

Remove potentially dangerous property declarations from CSS code.

In particular, properties using the CSS `url()` function with a scheme that is not considered safe are removed:

```
>>> sanitizer = TracHTMLSanitizer()
>>> sanitizer.sanitize_css(u'''
...   background: url(javascript:alert("foo"));
...   color: #000;
... ''')
[u'color: #000']
```

Also, the proprietary Internet Explorer function `expression()` is always stripped:

```
>>> sanitizer.sanitize_css(u'''
...   background: #fff;
...   color: #000;
...   width: e/**/xpression(alert("F"));
... ''')
[u'background: #fff', u'color: #000', u'width: e xpression(alert("F"))']
```

Parameters **text** – the CSS text; this is expected to be `unicode` and to not contain any character or numeric references

Returns a list of declarations that are considered safe

Return type `list`

class trac.util.html.**Deuglifier**

Bases: `object`

Help base class used for cleaning up HTML riddled with `<FONT COLOR=...>` tags and replace them with appropriate `<span class="...">`.

The subclass must define a `rules()` static method returning a list of regular expression fragments, each defining a capture group in which the name will be reused for the span's class. Two special group names, `font` and `endfont` are used to emit `<span>` and `</span>`, respectively.

trac.util.html.**escape**(*str*, *quotes=True*)

Create a Markup instance from a string and escape special characters it may contain (<, >, & and ").

Parameters

- **text** – the string to escape; if not a string, it is assumed that the input can be converted to a string
- **quotes** – if `True`, double quote characters are escaped in addition to the other special characters

```
>>> escape('"1 < 2"')
Markup(u'&#34;1 &lt; 2&#34;')
```

```
>>> escape(['"1 < 2"'])
Markup(u"['&#34;1 &lt; 2&#34;']")
```

If the `quotes` parameter is set to `False`, the " character is left as is. Escaping quotes is generally only required for strings that are to be used in attribute values.

```
>>> escape('"1 < 2"', quotes=False)
Markup(u'"1 &lt; 2"')
```

```
>>> escape(['"1 < 2"'], quotes=False)
Markup(u'["1 &lt; 2"]')
```

However, *escape* behaves slightly differently with Markup and *Fragment* behave instances, as they are passed through unmodified.

```
>>> escape(Markup('"1 < 2 &#39;"'))
Markup(u'"1 < 2 &#39;"')
```

```
>>> escape(Markup('"1 < 2 &#39;"'), quotes=False)
Markup(u'"1 < 2 &#39;"')
```

```
>>> escape(tag.b('"1 < 2"'))
Markup(u'<b>"1 &lt; 2"</b>')
```

```
>>> escape(tag.b('"1 < 2"'), quotes=False)
Markup(u'<b>"1 &lt; 2"</b>')
```

Returns the escaped Markup string

Return type Markup

trac.util.html.**unescape**(*text*)

Reverse-escapes &, <, >, and " and returns a unicode object.

```
>>> unescape(Markup('1 &lt; 2'))
u'1 < 2'
```

If the provided text object is not a Markup instance, it is returned unchanged.

```
>>> unescape('1 &lt; 2')
'1 &lt; 2'
```

Parameters **text** – the text to unescape

Returns the unescsaped string

Return type unicode

trac.util.html.**stripentities**(*text*, *keepxmlentities=False*)

Return a copy of the given text with any character or numeric entities replaced by the equivalent UTF-8 characters.

```
>>> stripentities('1 &lt; 2')
Markup(u'1 < 2')
>>> stripentities('more &hellip;')
Markup(u'more \u2026')
>>> stripentities('&#8230;')
Markup(u'\u2026')
>>> stripentities('&#x2026;')
Markup(u'\u2026')
>>> stripentities(Markup(u'\u2026'))
Markup(u'\u2026')
```

If the `keepxmlentities` parameter is provided and is a truth value, the core XML entities (&, ', >, < and ") are left intact.

```
>>> stripentities('1 &lt; 2 &hellip;', keepxmlentities=True)
Markup(u'1 &lt; 2 \u2026')
```

Returns a `Markup` instance with entities removed

Return type `Markup`

`trac.util.html.`**`striptags`**(*text*)

Return a copy of the text with any XML/HTML tags removed.

```
>>> striptags('<span>Foo</span> bar')
Markup(u'Foo bar')
>>> striptags('<span class="bar">Foo</span>')
Markup(u'Foo')
>>> striptags('Foo<br />')
Markup(u'Foo')
```

HTML/XML comments are stripped, too:

```
>>> striptags('<!-- <blub>hehe</blah> -->test')
Markup(u'test')
```

Parameters **`text`** – the string to remove tags from

Returns a `Markup` instance with all tags removed

Return type `Markup`

`trac.util.html.`**`plaintext`**(*text*, *keeplinebreaks=True*)

Extract the text elements from (X)HTML content

Parameters

- **`text`** – `unicode` or *`Fragment`*
- **`keeplinebreaks`** – optionally keep linebreaks

class `trac.util.html.`**`FormTokenInjector`**(*form_token*, *out*)

Bases: *`trac.util.html.HTMLTransform`*

Identify and protect forms from CSRF attacks.

This filter works by adding a input type=hidden field to POST forms.

class `trac.util.html.`**`HTMLTransform`**(*out*)

Bases: `HTMLParser.HTMLParser`

Convenience base class for writing HTMLParsers.

The default implementation of the HTMLParser `handle_*` methods do nothing, while in our case we try to rewrite the incoming document unmodified.

class `trac.util.html.`**`HTMLSanitization`**(*sanitizer*, *out*)

Bases: *`trac.util.html.HTMLTransform`*

Sanitize parsed HTML using TracHTMLSanitizer.

Misc. HTML processing

trac.util.html.**find_element**(*frag*, *attr=None*, *cls=None*, *tag=None*)
Return the first element in the fragment having the given attribute, class or tag, using a preorder depth-first search.

trac.util.html.**to_fragment**(*input*)
Convert input to a *Fragment* object.

trac.util.html.**valid_html_bytes**(*bytes*)

trac.util.html.**to_fragment**(*input*)
Convert input to a *Fragment* object.

Kept for backward compatibility purposes:

trac.util.html.**expand_markup**(*stream*, *ctxt=None*)
A Genshi stream filter for expanding `genshi.Markup` events.

Note: Expansion may not be possible if the fragment is badly formed, or partial.

`trac.util.presentation` – Utilities for dynamic content generation

trac.util.presentation.**jinja2_update**(*jenv*)
Augment a Jinja2 environment with filters, tests and global functions defined in this module.

We define a few Jinja2 custom filters.

trac.util.presentation.**flatten_filter**(*value*)
Combine incoming sequences in one.

trac.util.presentation.**groupattr_filter**(*_eval_ctx*, *iterable*, *num*, *attr*, **args*, ***kwargs*)
Similar to *group*, but as an attribute filter.

trac.util.presentation.**htmlattr_filter**(*_eval_ctx*, *d*, *autospace=True*)
Create an SGML/XML attribute string based on the items in a dict.

If the dict itself is `none` or `undefined`, it returns the empty string. `d` can also be an iterable or a mapping, in which case it will be converted to a `dict`.

All values that are neither `none` nor `undefined` are automatically escaped.

For HTML attributes like `'checked'` and `'selected'`, a truth value will be converted to the key value itself. For others it will be `'true'` or `'on'`. For `'class'`, the `classes` processing will be applied.

Example:

```
<ul${{'class': {'my': 1, 'list': True, 'empty': False},
      'missing': none, 'checked': 1, 'selected': False,
      'autocomplete': True, 'id': 'list-%d'|format(variable),
      'style': {'border-radius': '3px' if rounded,
                'background': '#f7f7f7'}
     }|htmlattr}>
...
</ul>
```

Results in something like this:

```
<ul class="my list" id="list-42" checked="checked" autocomplete="on"
    style="border-radius: 3px; background: #f7f7f7">
```

```
...
</ul>
```

As you can see it automatically prepends a space in front of the item if the filter returned something unless the second parameter is false.

Adapted from Jinja2's builtin `do_xmlattr` filter.

`trac.util.presentation.`**`max_filter`**(*seq*, *default=None*)
Returns the max value from the sequence.

`trac.util.presentation.`**`min_filter`**(*seq*, *default=None*)
Returns the min value from the sequence.

`trac.util.presentation.`**`trim_filter`**(*value*, *what=None*)
Strip leading and trailing whitespace or other specified character.

Adapted from Jinja2's builtin `trim` filter.

We also define a few Jinja2 custom tests.

`trac.util.presentation.`**`is_greaterthan`**(*a*, *b*)

`trac.util.presentation.`**`is_greaterthanorequal`**(*a*, *b*)

`trac.util.presentation.`**`is_lessthan`**(*a*, *b*)

`trac.util.presentation.`**`is_lessthanorequal`**(*a*, *b*)

`trac.util.presentation.`**`is_not_equalto`**(*a*, *b*)

`trac.util.presentation.`**`is_not_in`**(*a*, *b*)

`trac.util.presentation.`**`istext`**(*text*)
`True` for text (`unicode` and `str`), but `False` for `Markup`.

The following utilities are all available within Jinja2 templates.

`trac.util.presentation.`**`captioned_button`**(*req*, *symbol*, *text*)
Return symbol and text or only symbol, according to user preferences.

`trac.util.presentation.`**`first_last`**(*idx*, *seq*)
Generate `first` or `last` or both, according to the position `idx` in sequence `seq`.

In Jinja2 templates, rather use:

```
<li ${{'class': {'first': loop.first, 'last': loop.last}}|htmlattr}>
```

This is less error prone, as the sequence remains implicit and therefore can't be wrong.

`trac.util.presentation.`**`group`**(*iterable*, *num*, *predicate=None*)
Combines the elements produced by the given iterable so that every `n` items are returned as a tuple.

```
>>> items = [1, 2, 3, 4]
>>> for item in group(items, 2):
...     print(item)
(1, 2)
(3, 4)
```

The last tuple is padded with `None` values if its' length is smaller than `num`.

```
>>> items = [1, 2, 3, 4, 5]
>>> for item in group(items, 2):
...     print(item)
```

```
(1, 2)
(3, 4)
(5, None)
```

The optional `predicate` parameter can be used to flag elements that should not be packed together with other items. Only those elements where the predicate function returns True are grouped with other elements, otherwise they are returned as a tuple of length 1:

```
>>> items = [1, 2, 3, 4]
>>> for item in group(items, 2, lambda x: x != 3):
...     print(item)
(1, 2)
(3,)
(4, None)
```

`trac.util.presentation.`**`istext`**(*text*)

`True` for text (`unicode` and `str`), but `False` for `Markup`.

`trac.util.presentation.`**`paginate`**(*items*, *page=0*, *max_per_page=10*)

Simple generic pagination.

Given an iterable, this function returns:

- the slice of objects on the requested page,
- the total number of items, and
- the total number of pages.

The `items` parameter can be a list, tuple, or iterator:

```
>>> items = list(xrange(12))
>>> items
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>> paginate(items)
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 12, 2)
>>> paginate(items, page=1)
([10, 11], 12, 2)
>>> paginate(iter(items))
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 12, 2)
>>> paginate(iter(items), page=1)
([10, 11], 12, 2)
```

This function also works with generators:

```
>>> def generate():
...     for idx in xrange(12):
...         yield idx
>>> paginate(generate())
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 12, 2)
>>> paginate(generate(), page=1)
([10, 11], 12, 2)
```

The `max_per_page` parameter can be used to set the number of items that should be displayed per page:

```
>>> items = xrange(12)
>>> paginate(items, page=0, max_per_page=6)
([0, 1, 2, 3, 4, 5], 12, 2)
>>> paginate(items, page=1, max_per_page=6)
([6, 7, 8, 9, 10, 11], 12, 2)
```

Raises ***TracError*** – if `page` is out of the range of the paginated results.

`trac.util.presentation.`**`separated`**(*items*, *sep=', '*, *last=None*)
Yield `(item, sep)` tuples, one for each element in `items`.

The separator after the last item is specified by the `last` parameter, which defaults to `None`. (Since 1.1.3)

```
>>> list(separated([1, 2]))
[(1, ','), (2, None)]
```

```
>>> list(separated([1]))
[(1, None)]
```

```
>>> list(separated('abc', ':'))
[('a', ':'), ('b', ':'), ('c', None)]
```

```
>>> list(separated((1, 2, 3), sep=';', last='.'))
[(1, ';'), (2, ';'), (3, '.')]
```

`trac.util.presentation.`**`to_json`**(*value*)
Encode `value` to JSON.

Modules generating paginated output will be happy to use a rich pagination controller. See *Query*, *Report* and *Search* modules for example usage.

class `trac.util.presentation.`**`Paginator`**(*items*, *page=0*, *max_per_page=10*, *num_items=None*)
Bases: `object`

Pagination controller

`trac.util.text` – Text manipulation

The Jinja2 template engine

As Jinja2 is mainly a text template engine, the low-level helper functions dealing with this package are placed here.

`trac.util.text.`**`jinja2env`**(***kwargs*)
Creates a Jinja2 `Environment` configured with Trac conventions.

All default parameters can optionally be overriden. The `loader` parameter is not set by default, so unless it is set by the caller, only inline templates can be created from the environment.

Return type `jinja.Environment`

`trac.util.text.`**`jinja2template`**(*template*, *text=False*)
Creates a Jinja2 `Template` from inlined source.

Parameters

- **`template`** – the template content
- **`text`** – if set to `False`, the result of the variable expansion will be XML/HTML escaped

The Unicode toolbox

Trac internals are almost exclusively dealing with Unicode text, represented by `unicode` objects. The main advantage of using `unicode` over UTF-8 encoded `str` (as this used to be the case before version 0.10), is that text transformation functions in the present module will operate in a safe way on individual characters, and won't risk to cut a multi-byte sequence in the middle. Similar issues with Python string handling routines are avoided as well. For example, did you know that "Priorità" is encoded as `'Priorit\xc3\x0a'` in UTF-8? Calling `strip()` on this value in some locales can cut away the trailing `\x0a` and it's no longer valid UTF-8...

The drawback is that most of the outside world, while eventually "Unicode", is definitely not `unicode`. This is why we need to convert back and forth between `str` and `unicode` at the boundaries of the system. And more often than not we even have to guess which encoding is used in the incoming `str` strings.

Encoding `unicode` to `str` is usually directly performed by calling `encode()` on the `unicode` instance, while decoding is preferably left to the *`to_unicode`* helper function, which converts `str` to `unicode` in a robust and guaranteed successful way.

`trac.util.text.`**`to_unicode`**(*text*, *charset=None*)
Convert input to an `unicode` object.

For a `str` object, we'll first try to decode the bytes using the given `charset` encoding (or UTF-8 if none is specified), then we fall back to the latin1 encoding which might be correct or not, but at least preserves the original byte sequence by mapping each byte to the corresponding unicode code point in the range U+0000 to U+00FF.

For anything else, a simple `unicode()` conversion is attempted, with special care taken with `Exception` objects.

`trac.util.text.`**`exception_to_unicode`**(*e*, *traceback=False*)
Convert an `Exception` to an `unicode` object.

In addition to *`to_unicode`*, this representation of the exception also contains the class name and optionally the traceback.

Web utilities

`trac.util.text.`**`unicode_quote`**(*value*, *safe='/'*)
A unicode aware version of `urllib.quote`

Parameters

- **`value`** – anything that converts to a `str`. If `unicode` input is given, it will be UTF-8 encoded.
- **`safe`** – as in `quote`, the characters that would otherwise be quoted but shouldn't here (defaults to '/')

`trac.util.text.`**`unicode_quote_plus`**(*value*, *safe=''*)
A unicode aware version of `urllib.quote_plus`.

Parameters

- **`value`** – anything that converts to a `str`. If `unicode` input is given, it will be UTF-8 encoded.
- **`safe`** – as in `quote_plus`, the characters that would otherwise be quoted but shouldn't here (defaults to '/')

`trac.util.text.`**`unicode_unquote`**(*value*)
A unicode aware version of `urllib.unquote`.

Parameters **str** – UTF-8 encoded `str` value (for example, as obtained by *unicode_quote*).

Return type `unicode`

`trac.util.text.`**unicode_urlencode**(*params*, *safe=''*)
A unicode aware version of `urllib.urlencode`.

Values set to *empty* are converted to the key alone, without the equal sign.

`trac.util.text.`**quote_query_string**(*text*)
Quote strings for query string

`trac.util.text.`**javascript_quote**(*text*)
Quote strings for inclusion in single or double quote delimited Javascript strings

`trac.util.text.`**to_js_string**(*text*)
Embed the given string in a double quote delimited Javascript string (conform to the JSON spec)

Console and file system

`trac.util.text.`**getpreferredencoding**()
Return the encoding, which is retrieved on ahead, according to user preference.

We should use this instead of `locale.getpreferredencoding()` which is not thread-safe.

`trac.util.text.`**path_to_unicode**(*path*)
Convert a filesystem path to unicode, using the filesystem encoding.

`trac.util.text.`**stream_encoding**(*stream*)
Return the appropriate encoding for the given stream.

`trac.util.text.`**console_print**(*out*, **args*, ***kwargs*)
Output the given arguments to the console, encoding the output as appropriate.

Parameters **kwargs** – `newline` controls whether a newline will be appended (defaults to `True`)

`trac.util.text.`**printout**(**args*, ***kwargs*)
Do a *console_print* on `sys.stdout`.

`trac.util.text.`**printerr**(**args*, ***kwargs*)
Do a *console_print* on `sys.stderr`.

`trac.util.text.`**raw_input**(*prompt*)
Input one line from the console and converts it to unicode as appropriate.

Miscellaneous

`trac.util.text.`**empty**
A special tag object evaluating to the empty string, used as marker for missing value (as opposed to a present but empty value).

class `trac.util.text.`**unicode_passwd**
Bases: `unicode`

Conceal the actual content of the string when `repr` is called.

`trac.util.text.`**cleandoc**(*message*)
Removes uniform indentation and leading/trailing whitespace.

`trac.util.text.`**levenshtein_distance**(*lhs*, *rhs*)
Return the Levenshtein distance between two strings.

trac.util.text.**sub_vars**(*text*, *args*)
Substitute $XYZ-style variables in a string with provided values.

Parameters

- **text** – string containing variables to substitute.
- **args** – dictionary with keys matching the variables to be substituted. The keys should not be prefixed with the $ character.

trac.util.text.**getpreferredencoding**()
Return the encoding, which is retrieved on ahead, according to user preference.

We should use this instead of `locale.getpreferredencoding()` which is not thread-safe.

Text formatting

trac.util.text.**pretty_size**(*size*, *format='%.1f'*)
Pretty print content size information with appropriate unit.

Parameters

- **size** – number of bytes
- **format** – can be used to adjust the precision shown

trac.util.text.**breakable_path**(*path*)
Make a path breakable after path separators, and conversely, avoid breaking at spaces.

trac.util.text.**normalize_whitespace**(*text*, *to_space=u'\xa0'*, *remove=u'\u200b'*)
Normalize whitespace in a string, by replacing special spaces by normal spaces and removing zero-width spaces.

trac.util.text.**unquote_label**(*txt*)
Remove (one level of) enclosing single or double quotes.

New in version 1.0.

trac.util.text.**fix_eol**(*text*, *eol*)
Fix end-of-lines in a text.

trac.util.text.**expandtabs**(*s*, *tabstop=8*, *ignoring=None*)
Expand tab characters `'\t'` into spaces.

Parameters

- **tabstop** – number of space characters per tab (defaults to the canonical 8)
- **ignoring** – if not `None`, the expansion will be "smart" and go from one tabstop to the next. In addition, this parameter lists characters which can be ignored when computing the indent.

trac.util.text.**is_obfuscated**(*word*)
Returns `True` if the `word` looks like an obfuscated e-mail address.

Since 1.2

trac.util.text.**obfuscate_email_address**(*address*)
Replace anything looking like an e-mail address (`'@something'`) with a trailing ellipsis (`'@...'`)

trac.util.text.**text_width**(*text*, *ambiwidth=1*)
Determine the column width of `text` in Unicode characters.

The characters in the East Asian Fullwidth (F) or East Asian Wide (W) have a column width of 2. The other characters in the East Asian Halfwidth (H) or East Asian Narrow (Na) have a column width of 1.

That `ambiwidth` parameter is used for the column width of the East Asian Ambiguous (A). If `1`, the same width as characters in US-ASCII. This is expected by most users. If `2`, twice the width of US-ASCII characters. This is expected by CJK users.

cf. http://www.unicode.org/reports/tr11/.

`trac.util.text.`**`print_table`**(*data*, *headers=None*, *sep=' '*, *out=None*, *ambiwidth=None*)
Print data according to a tabular layout.

Parameters

- **`data`** – a sequence of rows; assume all rows are of equal length.
- **`headers`** – an optional row containing column headers; must be of the same length as each row in `data`.
- **`sep`** – column separator
- **`out`** – output file descriptor (`None` means use `sys.stdout`)
- **`ambiwidth`** – column width of the East Asian Ambiguous (A). If None, detect ambiwidth with the locale settings. If others, pass to the `ambiwidth` parameter of *`text_width`*.

`trac.util.text.`**`shorten_line`**(*text*, *maxlen=75*)
Truncates `text` to length less than or equal to `maxlen` characters.

This tries to be (a bit) clever and attempts to find a proper word boundary for doing so.

`trac.util.text.`**`stripws`**(*text*, *leading=True*, *trailing=True*)
Strips unicode white-spaces and ZWSPs from `text`.

Parameters

- **`leading`** – strips leading spaces from `text` unless `leading` is `False`.
- **`trailing`** – strips trailing spaces from `text` unless `trailing` is `False`.

`trac.util.text.`**`strip_line_ws`**(*text*, *leading=True*, *trailing=True*)
Strips unicode white-spaces and ZWSPs from each line of `text`.

Parameters

- **`leading`** – strips leading spaces from `text` unless `leading` is `False`.
- **`trailing`** – strips trailing spaces from `text` unless `trailing` is `False`.

`trac.util.text.`**`strip_line_ws`**(*text*, *leading=True*, *trailing=True*)
Strips unicode white-spaces and ZWSPs from each line of `text`.

Parameters

- **`leading`** – strips leading spaces from `text` unless `leading` is `False`.
- **`trailing`** – strips trailing spaces from `text` unless `trailing` is `False`.

`trac.util.text.`**`wrap`**(*t*, *cols=75*, *initial_indent=''*, *subsequent_indent=''*, *linesep='\n'*, *ambiwidth=1*)
Wraps the single paragraph in `t`, which contains unicode characters. The every line is at most `cols` characters long.

That `ambiwidth` parameter is used for the column width of the East Asian Ambiguous (A). If `1`, the same width as characters in US-ASCII. This is expected by most users. If `2`, twice the width of US-ASCII characters. This is expected by CJK users.

`trac.util.text.`**`cleandoc`**(*message*)
Removes uniform indentation and leading/trailing whitespace.

trac.util.text.**sub_vars**(*text*, *args*)
Substitute $XYZ-style variables in a string with provided values.

Parameters

- **text** – string containing variables to substitute.
- **args** – dictionary with keys matching the variables to be substituted. The keys should not be prefixed with the $ character.

Conversion utilities

trac.util.text.**unicode_to_base64**(*text*, *strip_newlines=True*)
Safe conversion of `text` to base64 representation using utf-8 bytes.

Strips newlines from output unless `strip_newlines` is `False`.

trac.util.text.**unicode_from_base64**(*text*)
Safe conversion of `text` to unicode based on utf-8 bytes.

trac.util.text.**to_utf8**(*text*, *charset='latin1'*)
Convert input to a UTF-8 `str` object.

If the input is not an `unicode` object, we assume the encoding is already UTF-8, ISO Latin-1, or as specified by the optional *charset* parameter.

trac.util.translation

Utilities for text translation with gettext.

Functions

trac.util.translation.**deactivate**()
Deactivate translations. :return: the current Translations, if any

trac.util.translation.**reactivate**(*t*)
Reactivate previously deactivated translations. :param t: the Translations, as returned by *deactivate*

trac.util.translation.**make_activable**(*get_locale*, *env_path=None*)
Defer activation of translations. :param get_locale: a callable returning a Babel Locale object :param env_path: the environment to use for looking up catalogs

trac.util.translation.**get_available_locales**()
Return a list of locale identifiers of the locales for which translations are available.

trac.util.translation.**domain_functions**(*domain*, **symbols*)
Prepare partial instantiations of domain translation functions.

Parameters

- **domain** – domain used for partial instantiation
- **symbols** – remaining parameters are the name of commonly used translation function which will be bound to the domain

Note: the symbols can also be given as an iterable in the 2nd argument.

trac.util.translation.**s_dgettext**(*domain*, *msgid*, ***kwargs*)
Retrieves translations for "squeezed" messages, in a domain.

See *s_gettext* for additional details.

trac.util.translation.**s_gettext**(*msgid*, ***kwargs*)
Retrieves translations for "squeezed" messages (in default domain).

Squeezed messages are text blocks in which white-space has been simplified during extraction (see *trac.dist.extract_html*). The catalog contain msgid with minimal whitespace. As a consequence, the msgid have to be normalized as well at retrieval time (i.e. here).

This typically happens for trans blocks and gettext functions in Jinja2 templates, as well as all the text extracted from legacy Genshi templates.

Internals

class trac.util.translation.**NullTranslationsBabel**(*fp=None*)
Bases: gettext.NullTranslations

NullTranslations doesn't have the domain related methods.

class trac.util.translation.**TranslationsProxy**
Bases: object

Delegate Translations calls to the currently active Translations.

If there's none, wrap those calls in LazyProxy objects.

Activation is controlled by `activate` and *deactivate* methods. However, if retrieving the locale information is costly, it's also possible to enable activation on demand only, by providing a callable to *make_activable*.

Otherwise, the functions are direct members of the *trac.util* package (i.e. placed in the "__init__.py" file).

Web related utilities

trac.util.**get_reporter_id**(*req*, *arg_name=None*)
Get most informative "reporter" identity out of a request.

That's the `Request`'s authname if not 'anonymous', or a `Request` argument, or the session name and e-mail, or only the name or only the e-mail, or 'anonymous' as last resort.

Parameters

- **req** – a *trac.web.api.Request*
- **arg_name** – if given, a `Request` argument which may contain the id for non-authentified users

trac.util.**content_disposition**(*type=None*, *filename=None*)
Generate a properly escaped Content-Disposition header.

OS related utilies

trac.util.**copytree**(*src*, *dst*, *symlinks=False*, *skip=[]*, *overwrite=False*)
Recursively copy a directory tree using copy2() (from shutil.copytree.)

Added a `skip` parameter consisting of absolute paths which we don't want to copy.

trac.util.**create_file**(*path*, *data=''*, *mode='w'*)
Create a new file with the given data.

Data string or iterable of strings.

trac.util.**create_unique_file**(*path*)
Create a new file. An index is added if the path exists

trac.util.**getuser**()
Retrieve the identity of the process owner

trac.util.**is_path_below**(*path*, *parent*)
Return True iff `path` is equal to parent or is located below `parent` at any level.

trac.util.**makedirs**(*path*, *overwrite=False*)
Create as many directories as necessary to make `path` exist.

If `overwrite` is `True`, don't raise an exception in case `path` already exists.

trac.util.**native_path**(*path*)
Converts a Windows-style or POSIX-style path to the native style.

i.e. on Windows, convert POSIX path to Windows path, and in a POSIX system, convert Windows path to POSIX path.

Parameters **path** – the input path

Returns the path converted to native style

trac.util.**read_file**(*path*, *mode='r'*)
Read a file and return its content.

trac.util.**rename**(*old*, *new*)
Rename a file or directory.

trac.util.**terminate**(*process*)
Terminate the process.

If the process has already finished and has not been waited for, the function does not raise OSError and WindowsError exceptions unlike a terminate method of `subprocess.Popen`.

Parameters **process** – the integer id (`pid`) of the process.

trac.util.**touch_file**(*filename*)
Update modified time of the given file. The file is created if missing.

class trac.util.**AtomicFile**(*path*, *mode='w'*, *bufsize=-1*)
Bases: `object`

A file that appears atomically with its full content.

This file-like object writes to a temporary file in the same directory as the final file. If the file is committed, the temporary file is renamed atomically (on Unix, at least) to its final name. If it is rolled back, the temporary file is removed.

class trac.util.**NaivePopen**(*command*, *input=None*, *capturestderr=None*)
Bases: `object`

This is a deadlock-safe version of popen that returns an object with errorlevel, out (a string) and err (a string).

The optional `input`, which must be a `str` object, is first written to a temporary file from which the process will read.

(`capturestderr` may not work under Windows 9x.)

Example:

```
print(Popen3('grep spam','\n\nhere spam\n\n').out)
```

exception trac.util.**WindowsError**
Bases: exceptions.OSError

Dummy exception replacing WindowsError on non-Windows platforms

Also defined on non-Windows systems (by a dummy OSError subclass).

class trac.util.**file_or_std**(*filename*, *mode='r'*, *bufsize=-1*)
Bases: object

Context manager for opening a file or using a standard stream

If filename is non-empty, open the file and close it when exiting the block. Otherwise, use sys.stdin if opening for reading, or sys.stdout if opening for writing or appending.

trac.util.**urandom**
The standard os.urandom when available, otherwise a reasonable replacement.

Python "system" utilities

Complements the inspect, traceback and sys modules.

trac.util.**fq_class_name**(*obj*)
Return the fully qualified class name of given object.

trac.util.**arity**(*f*)
Return the number of arguments expected by the given function, unbound or bound method.

trac.util.**get_last_traceback**()
Retrieve the last traceback as an unicode string.

trac.util.**get_lines_from_file**(*filename*, *lineno*, *context=0*, *globals=None*)
Return content number of lines before and after the specified lineno from the (source code) file identified by filename.

Returns a (lines_before, line, lines_after) tuple.

trac.util.**get_frame_info**(*tb*)
Return frame information for a traceback.

trac.util.**import_namespace**(*globals_dict*, *module_name*)
Import the namespace of a module into a globals dict.

This function is used in stub modules to import all symbols defined in another module into the global namespace of the stub, usually for backward compatibility.

trac.util.**safe__import__**(*module_name*)
Safe imports: rollback after a failed import.

Initially inspired from the RollbackImporter in PyUnit, but it's now much simpler and works better for our needs.

See http://pyunit.sourceforge.net/notes/reloading.html

trac.util.**safe_repr**(*x*)
repr replacement which "never" breaks.

Make sure we always get a representation of the input x without risking to trigger an exception (e.g. from a buggy x.__repr__).

New in version 1.0.

trac.util.**get_doc**(*obj*)
Return the docstring of an object as a tuple `(summary, description)`, where `summary` is the first paragraph and `description` is the remaining text.

Setuptools utilities

trac.util.**get_module_path**(*module*)
Return the base path the given module is imported from

trac.util.**get_sources**(*path*)
Return a dictionary mapping Python module source paths to the distributions that contain them.

trac.util.**get_pkginfo**(*dist*)
Get a dictionary containing package information for a package

`dist` can be either a Distribution instance or, as a shortcut, directly the module instance, if one can safely infer a Distribution instance from it.

Always returns a dictionary but it will be empty if no Distribution instance can be created for the given module.

Cryptographic related utilities

trac.util.**hex_entropy**(*digits=32*)
Generate `digits` number of hex digits of entropy.

trac.util.**md5crypt**(*password*, *salt*, *magic='1'*)
Based on FreeBSD src/lib/libcrypt/crypt.c 1.2

Parameters

- **password** – the plain text password to crypt
- **salt** – the raw salt
- **magic** – our magic string

trac.util.**salt**(*length=2*)
Returns a string of `length` random letters and numbers.

Data structures which don't fit anywhere else

class trac.util.**Ranges**(*r=None*, *reorder=False*)
Bases: `object`

Holds information about ranges parsed from a string

Author Tim Hatch

```
>>> x = Ranges("1,2,9-15")
>>> 1 in x
True
>>> 5 in x
False
>>> 10 in x
True
>>> 16 in x
False
>>> [i for i in xrange(20) if i in x]
[1, 2, 9, 10, 11, 12, 13, 14, 15]
```

Also supports iteration, which makes that last example a bit simpler:

```
>>> list(x)
[1, 2, 9, 10, 11, 12, 13, 14, 15]
```

Note that it automatically reduces the list and short-circuits when the desired ranges are a relatively small portion of the entire set:

```
>>> x = Ranges("99")
>>> 1 in x # really fast
False
>>> x = Ranges("1, 2, 1-2, 2") # reduces this to 1-2
>>> x.pairs
[(1, 2)]
>>> x = Ranges("1-9,2-4") # handle ranges that completely overlap
>>> list(x)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The members 'a' and 'b' refer to the min and max value of the range, and are None if the range is empty:

```
>>> x.a
1
>>> x.b
9
>>> e = Ranges()
>>> e.a, e.b
(None, None)
```

Empty ranges are ok, and ranges can be constructed in pieces, if you so choose:

```
>>> x = Ranges()
>>> x.appendrange("1, 2, 3")
>>> x.appendrange("5-9")
>>> x.appendrange("2-3") # reduce'd away
>>> list(x)
[1, 2, 3, 5, 6, 7, 8, 9]
```

Reversed ranges are ignored, unless the Ranges has the `reorder` property set.

```
>>> str(Ranges("20-10"))
''
>>> str(Ranges("20-10", reorder=True))
'10-20'
```

As rendered ranges are often using u',u200b' (comma + Zero-width space) to enable wrapping, we also support reading such ranges, as they can be copy/pasted back.

```
>>> str(Ranges(u'1,\u200b3,\u200b5,\u200b6,\u200b7,\u200b9'))
'1,3,5-7,9'
```

appendrange(*r*)
Add ranges to the current one.

A range is specified as a string of the form "low-high", and `r` can be a list of such strings, a string containing comma-separated ranges, or `None`.

truncate(*max*)
Truncate the Ranges by setting a maximal allowed value.

Note that this `max` can be a value in a gap, so the only guarantee is that `self.b` will be lesser than or equal to `max`.

```
>>> r = Ranges("10-20,25-45")
>>> str(r.truncate(30))
'10-20,25-30'
```

```
>>> str(r.truncate(22))
'10-20'
```

```
>>> str(r.truncate(10))
'10'
```

trac.util.**create_zipinfo**(*filename*, *mtime=None*, *dir=False*, *executable=False*, *symlink=False*, *comment=None*)

Create a instance of `ZipInfo`.

Parameters

- **filename** – file name of the entry
- **mtime** – modified time of the entry
- **dir** – if `True`, the entry is a directory
- **executable** – if `True`, the entry is a executable file
- **symlink** – if `True`, the entry is a symbolic link
- **comment** – comment of the entry

trac.util.**to_ranges**(*revs*)

Converts a list of revisions to a minimal set of ranges.

```
>>> to_ranges([2, 12, 3, 6, 9, 1, 5, 11])
'1-3,5-6,9,11-12'
>>> to_ranges([])
''
```

trac.util.**to_list**(*splittable*, *sep=', '*)

Split a string at `sep` and return a list without any empty items.

```
>>> to_list('1,2, 3,4 ')
['1', '2', '3', '4']
>>> to_list('1;2; 3;4 ', sep=';')
['1', '2', '3', '4']
>>> to_list('')
[]
>>> to_list(None)
[]
>>> to_list([])
[]
```

class trac.util.**lazy**(*fn*)

Bases: `object`

A lazily-evaluated attribute.

Since 1.0

Algorithmic utilities

trac.util.**embedded_numbers**(*s*)
Comparison function for natural order sorting based on http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/214202.

trac.util.**partition**(*iterable*, *order=None*)

```
>>> partition([(1, "a"), (2, "b"), (3, "a")])
{'a': [1, 3], 'b': [2]}
>>> partition([(1, "a"), (2, "b"), (3, "a")], "ab")
[[1, 3], [2]]
```

trac.util.**as_int**(*s*, *default*, *min=None*, *max=None*)
Convert s to an int and limit it to the given range, or return default if unsuccessful.

trac.util.**as_bool**(*value*, *default=False*)
Convert the given value to a `bool`.

If `value` is a string, return `True` for any of "yes", "true", "enabled", "on" or non-zero numbers, ignoring case. For non-string arguments, return the argument converted to a `bool`, or `default` if the conversion fails.

Since 1.2 the `default` argument can be specified.

trac.util.**pathjoin**(**args*)
Strip `/` from the arguments and join them with a single `/`.

trac.util.**sub_val**(*the_list*, *item_to_remove*, *item_to_add*)
Substitute an item if the item is found in a list, otherwise leave the list unmodified.

trac.versioncontrol.admin

class trac.versioncontrol.admin.**RepositoryAdminPanel**
Bases: *trac.core.Component*

Web admin panel for repository administration.

allowed_repository_dir_prefixes
Comma-separated list of allowed prefixes for repository directories when adding and editing repositories in the repository admin panel. If the list is empty, all repository directories are allowed.

class trac.versioncontrol.admin.**VersionControlAdmin**
Bases: *trac.core.Component*

trac-admin command provider for version control administration.

environment_created()
Index the repositories.

trac.versioncontrol.api – Trac Version Control APIs

This module implements an abstraction layer over different kind of version control systems and the mechanism to access several heterogeneous repositories under a single "virtual" hierarchy.

This abstraction was derived from the original model built around the Subversion system (versioned tree, changesets). It gradually became more general, now aiming at supporting distributed version control systems (DVCS).

Interfaces

class trac.versioncontrol.api.**IRepositoryConnector**
Bases: *trac.core.Interface*

Provide support for a specific version control system.

See also trac.versioncontrol.api.IRepositoryConnector extension point

get_repository(*repos_type*, *repos_dir*, *params*)
Return a Repository instance for the given repository type and dir.

get_supported_types()
Return the types of version control systems that are supported.

Yields `(repotype, priority)` pairs, where `repotype` is used to match against the repository's `type` attribute.

If multiple provider match a given type, the `priority` is used to choose between them (highest number is highest priority).

If the `priority` returned is negative, this indicates that the connector for the given `repotype` indeed exists but can't be used for some reason. The `error` property can then be used to store an error message or exception relevant to the problem detected.

class trac.versioncontrol.api.**IRepositoryProvider**
Bases: *trac.core.Interface*

Provide known named instances of Repository.

See also trac.versioncontrol.api.IRepositoryProvider extension point

get_repositories()
Generate repository information for known repositories.

Repository information is a key,value pair, where the value is a dictionary which must contain at the very least either of the following entries:

- **`'dir'`: the repository directory which can be used by the** connector to create a *Repository* instance. This defines a "real" repository.
- **`'alias'`: the name of another repository. This defines an** alias to another (real) repository.

Optional entries:

- **`'type'`: the type of the repository (if not given, the** default repository type will be used).
- **`'description'`: a description of the repository (can** contain WikiFormatting).
- **`'hidden'`: if set to `'true'`, the repository is hidden** from the repository index (default: `'false'`).
- **`'sync_per_request'`: if set to `'true'`, the repository will be** synchronized on every request (default: `'false'`).
- `'url'`: the base URL for checking out the repository.

class trac.versioncontrol.api.**IRepositoryChangeListener**
Bases: *trac.core.Interface*

Listen for changes in repositories.

See also trac.versioncontrol.api.IRepositoryChangeListener extension point

changeset_added(*repos*, *changeset*)
Called after a changeset has been added to a repository.

changeset_modified(*repos*, *changeset*, *old_changeset*)
Called after a changeset has been modified in a repository.

The `old_changeset` argument contains the metadata of the changeset prior to the modification. It is `None` if the old metadata cannot be retrieved.

Components

class `trac.versioncontrol.api.`**RepositoryManager**
Bases: *trac.core.Component*

Version control system manager.

change_listeners
List of components that implement *IRepositoryChangeListener*

connectors
List of components that implement *IRepositoryConnector*

default_repository_type
Default repository connector type.

This is used as the default repository type for repositories defined in the [TracIni#repositories-section repositories] section or using the "Repositories" admin panel.

get_all_repositories()
Return a dictionary of repository information, indexed by name.

get_default_repository(*context*)
Recover the appropriate repository from the current context.

Lookup the closest source or changeset resource in the context hierarchy and return the name of its associated repository.

get_real_repositories()
Return a sorted list of all real repositories (i.e. excluding aliases).

get_repositories()
Retrieve repositories specified in TracIni.

The `[repositories]` section can be used to specify a list of repositories.

get_repositories_by_dir(*directory*)
Retrieve the repositories based on the given directory.

Parameters **directory** – the key for identifying the repositories.

Returns list of *Repository* instances.

get_repository(*reponame*)
Retrieve the appropriate *Repository* for the given repository name.

Parameters **reponame** – the key for specifying the repository. If no name is given, take the default repository.

Returns if no corresponding repository was defined, simply return `None`.

Raises

- ***InvalidConnector*** – if the repository connector cannot be opened.
- ***InvalidRepository*** – if the repository cannot be opened.

get_repository_by_path(*path*)
Retrieve a matching *Repository* for the given `path`.

Parameters **path** – the eventually scoped repository-scoped path

Returns a `(reponame, repos, path)` triple, where `path` is the remaining part of `path` once the `reponame` has been truncated, if needed.

get_repository_id(*reponame*)
Return a unique id for the given repository name.

This will create and save a new id if none is found.

Note: this should probably be renamed as we're dealing exclusively with *db* repository ids here.

get_supported_types()
Return the list of supported repository types.

notify(*event*, *reponame*, *revs*)
Notify repositories and change listeners about repository events.

The supported events are the names of the methods defined in the *IRepositoryChangeListener* interface.

providers
List of components that implement *IRepositoryProvider*

read_file_by_path(*path*)
Read the file specified by `path`

Parameters **path** – the repository-scoped path. The repository revision may specified by appending `@` followed by the revision, otherwise the HEAD revision is assumed.

Returns the file content as a unicode string. `None` is returned if the file is not found.

Since 1.2.2

reload_repositories()
Reload the repositories from the providers.

repositories_section
One of the methods for registering repositories is to populate the `[repositories]` section of `trac.ini`.

This is especially suited for setting up aliases, using a [TracIni#GlobalConfiguration shared configuration], or specifying repositories at the time of environment creation.

See [TracRepositoryAdmin#ReposTracIni TracRepositoryAdmin] for details on the format of this section, and look elsewhere on the page for information on other repository providers.

shutdown(*tid=None*)
Free *Repository* instances bound to a given thread identifier

class `trac.versioncontrol.api.`**DbRepositoryProvider**
Bases: *trac.core.Component*

Component providing repositories registered in the DB.

add_alias(*reponame*, *target*)
Create an alias repository.

add_repository(*reponame*, *dir*, *type_=None*)
Add a repository.

get_repositories()
Retrieve repositories specified in the repository DB table.

modify_repository(*reponame*, *changes*)
Modify attributes of a repository.

remove_repository(*reponame*)
Remove a repository.

Exceptions

exception trac.versioncontrol.api.**InvalidConnector**(*message*, *title=None*, *show_traceback=False*)
Bases: *trac.core.TracError*

Exception raised when a repository connector is invalid.

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

exception trac.versioncontrol.api.**InvalidRepository**(*message*, *title=None*, *show_traceback=False*)
Bases: *trac.core.TracError*

Exception raised when a repository is invalid.

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

Subclasses of ResourceNotFound.

exception trac.versioncontrol.api.**NoSuchChangeset**(*rev*)
Bases: *trac.resource.ResourceNotFound*

exception trac.versioncontrol.api.**NoSuchNode**(*path*, *rev*, *msg=None*)
Bases: *trac.resource.ResourceNotFound*

Abstract classes

class trac.versioncontrol.api.**Repository**(*name*, *params*, *log*)
Bases: object

Base class for a repository provided by a version control system.

Initialize a repository.

Parameters

- **name** – a unique name identifying the repository, usually a type-specific prefix followed by the path to the repository.
- **params** – a dict of parameters for the repository. Contains the name of the repository under the key "name" and the surrogate key that identifies the repository in the database under the key "id".
- **log** – a logger instance.

Raises *InvalidRepository* – if the repository cannot be opened.

can_view(*perm*)
Return True if view permission is granted on the repository.

clear(*youngest_rev=None*)
Clear any data that may have been cached in instance properties.

`youngest_rev` can be specified as a way to force the value of the `youngest_rev` property (''will change in 0.12'').

close()
Close the connection to the repository.

display_rev(*rev*)
Return a string representation of a revision in the repos for displaying to the user.

This can be a shortened revision string, e.g. for repositories using long hashes.

Raises ***NoSuchChangeset*** – If the given `rev` isn't found.

Since 1.2 Always returns a string or `None`.

get_base()
Return the name of the base repository for this repository.

This function returns the name of the base repository to which scoped repositories belong. For non-scoped repositories, it returns the repository name.

get_changes(*old_path*, *old_rev*, *new_path*, *new_rev*, *ignore_ancestry=1*)
Generates changes corresponding to generalized diffs.

Generator that yields change tuples (old_node, new_node, kind, change) for each node change between the two arbitrary (path,rev) pairs.

The old_node is assumed to be None when the change is an ADD, the new_node is assumed to be None when the change is a DELETE.

get_changeset(*rev*)
Retrieve a Changeset corresponding to the given revision `rev`.

get_changeset_uid(*rev*)
Return a globally unique identifier for the ''rev" changeset.

Two changesets from different repositories can sometimes refer to the ''very same" changeset (e.g. the repositories are clones).

get_changesets(*start*, *stop*)
Generate Changeset belonging to the given time period (start, stop).

get_node(*path*, *rev=None*)
Retrieve a Node from the repository at the given path.

A Node represents a directory or a file at a given revision in the repository. If the `rev` parameter is specified, the Node corresponding to that revision is returned, otherwise the Node corresponding to the youngest revision is returned.

get_oldest_rev()
Return the oldest revision stored in the repository.

get_path_history(*path*, *rev=None*, *limit=None*)
Retrieve all the revisions containing this path.

If given, `rev` is used as a starting point (i.e. no revision ''newer" than `rev` should be returned). The result format should be the same as the one of Node.get_history()

get_path_url(*path*, *rev*)
Return the repository URL for the given path and revision.

The returned URL can be `None`, meaning that no URL has been specified for the repository, an absolute URL, or a scheme-relative URL starting with `//`, in which case the scheme of the request should be prepended.

get_quickjump_entries(*rev*)
Generate a list of interesting places in the repository.

`rev` might be used to restrict the list of available locations, but in general it's best to produce all known locations.

The generated results must be of the form (category, name, path, rev).

get_youngest_rev()
Return the youngest revision in the repository.

has_node(*path*, *rev=None*)
Tell if there's a node at the specified (path,rev) combination.

When `rev` is `None`, the latest revision is implied.

is_viewable(*perm*)
Return True if view permission is granted on the repository.

next_rev(*rev*, *path=''*)
Return the revision immediately following the specified revision.

If `path` is given, filter out descendant revisions having no changes below `path`.

In presence of multiple children, this follows the first child.

normalize_path(*path*)
Return a canonical representation of path in the repos.

normalize_rev(*rev*)
Return a (unique) canonical representation of a revision.

It's up to the backend to decide which string values of `rev` (usually provided by the user) should be accepted, and how they should be normalized. Some backends may for instance want to match against known tags or branch names.

In addition, if `rev` is `None` or '', the youngest revision should be returned.

Raises ***NoSuchChangeset*** – If the given `rev` isn't found.

parent_revs(*rev*)
Return a list of parents of the specified revision.

previous_rev(*rev*, *path=''*)
Return the revision immediately preceding the specified revision.

If `path` is given, filter out ancestor revisions having no changes below `path`.

In presence of multiple parents, this follows the first parent.

rev_older_than(*rev1*, *rev2*)
Provides a total order over revisions.

Return `True` if `rev1` is an ancestor of `rev2`.

short_rev(*rev*)
Return a compact string representation of a revision in the repos.

Raises ***NoSuchChangeset*** – If the given `rev` isn't found.

Since 1.2 Always returns a string or `None`.

sync(*rev_callback=None*, *clean=False*)
Perform a sync of the repository cache, if relevant.

If given, `rev_callback` must be a callable taking a `rev` parameter. The backend will call this function for each `rev` it decided to synchronize, once the synchronization changes are committed to the cache. When `clean` is `True`, the cache is cleaned first.

sync_changeset(*rev*)
Resync the repository cache for the given `rev`, if relevant.

Returns a "metadata-only" changeset containing the metadata prior to the resync, or `None` if the old values cannot be retrieved (typically when the repository is not cached).

class `trac.versioncontrol.api.`**Node**(*repos*, *path*, *rev*, *kind*)
Bases: `object`

Represents a directory or file in the repository at a given revision.

can_view(*perm*)
Return True if view permission is granted on the node.

get_annotations()
Provide detailed backward history for the content of this Node.

Retrieve an array of revisions, one `rev` for each line of content for that node. Only expected to work on (text) FILE nodes, of course.

get_content()
Return a stream for reading the content of the node.

This method will return `None` for directories. The returned object must support a `read([len])` method.

get_content_length()
The length in bytes of the content.

Will be `None` for a directory.

get_content_type()
The MIME type corresponding to the content, if known.

Will be `None` for a directory.

get_entries()
Generator that yields the immediate child entries of a directory.

The entries are returned in no particular order. If the node is a file, this method returns `None`.

get_history(*limit=None*)
Provide backward history for this Node.

Generator that yields `(path, rev, chg)` tuples, one for each revision in which the node was changed. This generator will follow copies and moves of a node (if the underlying version control system supports that), which will be indicated by the first element of the tuple (i.e. the path) changing. Starts with an entry for the current revision.

Parameters **limit** – if given, yield at most `limit` results.

get_previous()
Return the change event corresponding to the previous revision.

This returns a `(path, rev, chg)` tuple.

get_processed_content(*keyword_substitution=True*, *eol_hint=None*)
Return a stream for reading the content of the node, with some standard processing applied.

Parameters

- **keyword_substitution** – if `True`, meta-data keywords present in the content like `$Rev$` are substituted (which keyword are substituted and how they are substituted is backend specific)
- **eol_hint** – which style of line ending is expected if `None` was explicitly specified for the file itself in the version control backend (for example in Subversion, if it was set to `'native'`). It can be `None`, `'LF'`, `'CR'` or `'CRLF'`.

get_properties()
Returns the properties (meta-data) of the node, as a dictionary.

The set of properties depends on the version control system.

is_viewable(*perm*)
Return True if view permission is granted on the node.

class trac.versioncontrol.api.**Changeset**(*repos*, *rev*, *message*, *author*, *date*)
Bases: `object`

Represents a set of changes committed at once in a repository.

can_view(*perm*)
Return True if view permission is granted on the changeset.

get_bookmarks()
Yield bookmarks associated with this changeset.

New in version 1.1.5.

get_branches()
Yield branches to which this changeset belong. Each branch is given as a pair `(name, head)`, where `name` is the branch name and `head` a flag set if the changeset is a head for this branch (i.e. if it has no children changeset).

get_changes()
Generator that produces a tuple for every change in the changeset.

The tuple will contain `(path, kind, change, base_path, base_rev)`, where `change` can be one of Changeset.ADD, Changeset.COPY, Changeset.DELETE, Changeset.EDIT or Changeset.MOVE, and `kind` is one of Node.FILE or Node.DIRECTORY. The `path` is the targeted path for the `change` (which is the ''deleted'' path for a DELETE change). The `base_path` and `base_rev` are the source path and rev for the action (`None` and `-1` in the case of an ADD change).

get_properties()
Returns the properties (meta-data) of the node, as a dictionary.

The set of properties depends on the version control system.

Warning: this used to yield 4-elements tuple (besides `name` and `text`, there were `wikiflag` and `htmlclass` values). This is now replaced by the usage of IPropertyRenderer (see #1601).

get_tags()
Yield tags associated with this changeset.

New in version 1.0.

is_viewable(*perm*)
Return True if view permission is granted on the changeset.

class trac.versioncontrol.api.**EmptyChangeset**(*repos*, *rev*, *message=None*, *author=None*, *date=None*)

Bases: *trac.versioncontrol.api.Changeset*

Changeset that contains no changes. This is typically used when the changeset can't be retrieved.

Helper Functions

trac.versioncontrol.api.**is_default**(*reponame*)

Check whether reponame is the default repository.

trac.versioncontrol.cache

trac.versioncontrol.diff – Utilities for generation of diffs

Synopsis

get_filtered_hunks, *get_hunks* are low-level wrappers for Python's difflib.SequenceMatcher, and they generate groups of opcodes corresponding to diff "hunks".

get_change_extent is a low-level utility used when marking intra-lines differences.

diff_blocks is used at a higher-level to fill the template data needed by the "diff_div.html" template.

unified_diff is also a higher-level function returning differences following the unified diff file format.

Finally, *get_diff_options* is an utility for retrieving user diff preferences from a *Request*.

Function Reference

trac.versioncontrol.diff.**get_change_extent**(*str1*, *str2*)

Determines the extent of differences between two strings.

Returns a pair containing the offset at which the changes start, and the negative offset at which the changes end.

If the two strings have neither a common prefix nor a common suffix, (0, 0) is returned.

trac.versioncontrol.diff.**get_filtered_hunks**(*fromlines*, *tolines*, *context=None*, *ignore_blank_lines=False*, *ignore_case=False*, *ignore_space_changes=False*)

Retrieve differences in the form of difflib.SequenceMatcher opcodes, grouped according to the context and ignore_* parameters.

Parameters

- **fromlines** – list of lines corresponding to the old content
- **tolines** – list of lines corresponding to the new content
- **ignore_blank_lines** – differences about empty lines only are ignored
- **ignore_case** – upper case / lower case only differences are ignored
- **ignore_space_changes** – differences in amount of spaces are ignored
- **context** – the number of "equal" lines kept for representing the context of the change

Returns generator of grouped difflib.SequenceMatcher opcodes

If none of the `ignore_*` parameters is `True`, there's nothing to filter out the results will come straight from the SequenceMatcher.

`trac.versioncontrol.diff.`**`get_hunks`**(*fromlines*, *tolines*, *context=None*)
Generator yielding grouped opcodes describing differences .

See *get_filtered_hunks* for the parameter descriptions.

`trac.versioncontrol.diff.`**`diff_blocks`**(*fromlines*, *tolines*, *context=None*, *tabwidth=8*, *ignore_blank_lines=0*, *ignore_case=0*, *ignore_space_changes=0*)
Return an array that is adequate for adding to the data dictionary

See *get_filtered_hunks* for the parameter descriptions.

See also the diff_div.html template.

`trac.versioncontrol.diff.`**`unified_diff`**(*fromlines*, *tolines*, *context=None*, *ignore_blank_lines=0*, *ignore_case=0*, *ignore_space_changes=0*)
Generator producing lines corresponding to a textual diff.

See *get_filtered_hunks* for the parameter descriptions.

`trac.versioncontrol.diff.`**`get_diff_options`**(*req*)
Retrieve user preferences for diffs.

Returns

`(style, options, data)` triple.

`style` can be `'inline'` or `'sidebyside'`,

`options` a sequence of "diff" flags,

`data` the style and options information represented as key/value pairs in dictionaries, for example:

```
{'style': u'sidebyside',
 'options': {'contextall': 1, 'contextlines': 2,
             'ignorecase': 0,  'ignoreblanklines': 0,
             'ignorewhitespace': 1}}
```

`trac.versioncontrol.diff.`**`filter_ignorable_lines`**(*hunks*, *fromlines*, *tolines*, *context*, *ignore_blank_lines*, *ignore_case*, *ignore_space_changes*)
Detect line changes that should be ignored and emits them as tagged as "equal", possibly joined with the preceding and/or following "equal" block.

See *get_filtered_hunks* for the parameter descriptions.

trac.versioncontrol.svn_authz

class `trac.versioncontrol.svn_authz.`**`AuthzSourcePolicy`**
Bases: *trac.core.Component*

Permission policy for `source:` and `changeset:` resources using a Subversion authz file.

`FILE_VIEW` and `BROWSER_VIEW` permissions are granted as specified in the authz file.

`CHANGESET_VIEW` permission is granted for changesets where `FILE_VIEW` is granted on at least one modified file, as well as for empty changesets.

authz_file
The path to the Subversion [%(svnbook)s authorization (authz) file]. To enable authz permission checking, the *AuthzSourcePolicy* permission policy must be added to `[trac] permission_policies`. Non-absolute paths are relative to the Environment `conf` directory.

authz_module_name
The module prefix used in the *authz_file* for the default repository. If left empty, the global section is used.

`trac.versioncontrol.svn_authz.`**parse**(*authz_file*, *modules*)
Parse a Subversion authorization file.

Return a dict of modules, each containing a dict of paths, each containing a dict mapping users to permissions. Only modules contained in `modules` are retained.

trac.versioncontrol.web_ui.browser

class `trac.versioncontrol.web_ui.browser.`**DefaultPropertyRenderer**
Bases: *trac.core.Component*

Default version control property renderer.

class `trac.versioncontrol.web_ui.browser.`**IPropertyRenderer**
Bases: *trac.core.Interface*

Render node properties in TracBrowser and TracChangeset views.

match_property(*name*, *mode*)
Indicate whether this renderer can treat the given property

mode is the current rendering context, which can be:

- 'browser' rendered in the browser view
- 'changeset' rendered in the changeset view as a node property
- 'revprop' rendered in the changeset view as a revision property

Other identifiers might be used by plugins, so it's advised to simply ignore unknown modes.

Returns a quality number, ranging from 0 (unsupported) to 9 (''perfect" match).

render_property(*name*, *mode*, *context*, *props*)
Render the given property.

`name` is the property name as given to `match()`, `mode` is the same as for *match_property*, `context` is the context for the node being render (useful when the rendering depends on the node kind) and `props` is the collection of the corresponding properties (i.e. the `node.get_properties()`).

The rendered result can be one of the following:

- `None`: the property will be skipped
- an `unicode` value: the property will be displayed as text
- a `RenderedProperty` instance: the property will only be displayed using the instance's `content` attribute, and the other attributes will also be used in some display contexts (like `revprop`)
- `Markup` or `Fragment`: the property will be displayed normally, using that content as a block-level markup

class trac.versioncontrol.web_ui.browser.**WikiPropertyRenderer**
Bases: *trac.core.Component*

Wiki text property renderer.

oneliner_properties
Comma-separated list of version control properties to render as oneliner wiki content in the repository browser.

wiki_properties
Comma-separated list of version control properties to render as wiki content in the repository browser.

trac.versioncontrol.web_ui.browser.**datetime_now**()
[tz] -> new datetime with tz's local day and time.

trac.versioncontrol.web_ui.changeset

class trac.versioncontrol.web_ui.changeset.**ChangesetModule**
Bases: *trac.core.Component*

Renderer providing flexible functionality for showing sets of differences.

If the differences shown are coming from a specific changeset, then that changeset information can be shown too.

In addition, it is possible to show only a subset of the changeset: Only the changes affecting a given path will be shown. This is called the ''restricted" changeset.

But the differences can also be computed in a more general way, between two arbitrary paths and/or between two arbitrary revisions. In that case, there's no changeset information displayed.

max_diff_bytes
Maximum total size in bytes of the modified files (their old size plus their new size) for which the changeset view will attempt to show the diffs inlined.

max_diff_files
Maximum number of modified files for which the changeset view will attempt to show the diffs inlined.

process_request(*req*)
The appropriate mode of operation is inferred from the request parameters:

- If `new_path` and `old_path` are equal (or `old_path` is omitted) and `new` and `old` are equal (or `old` is omitted), then we're about to view a revision Changeset: `chgset` is True. Furthermore, if the path is not the root, the changeset is ''restricted" to that path (only the changes affecting that path, its children or its ancestor directories will be shown).
- In any other case, the set of changes corresponds to arbitrary differences between path@rev pairs. If `new_path` and `old_path` are equal, the ''restricted" flag will also be set, meaning in this case that the differences between two revisions are restricted to those occurring on that path.

In any case, either path@rev pairs must exist.

property_diff_renderers
List of components that implement *IPropertyDiffRenderer*

render_property_diff(*name*, *old_node*, *old_props*, *new_node*, *new_props*, *options*)
Renders diffs of a node property to HTML.

timeline_collapse
Whether consecutive changesets from the same author having exactly the same message should be presented as one event. That event will link to the range of changesets in the log view.

timeline_long_messages
Whether wiki-formatted changeset messages should be multiline or not.

If this option is not specified or is false and *wiki_format_messages* is set to true, changeset messages will be single line only, losing some formatting (bullet points, etc).

timeline_show_files
Number of files to show (`-1` for unlimited, `0` to disable).

This can also be `location`, for showing the common prefix for the changed files.

wiki_format_messages
Whether wiki formatting should be applied to changeset messages.

If this option is disabled, changeset messages will be rendered as pre-formatted text.

class trac.versioncontrol.web_ui.changeset.**DefaultPropertyDiffRenderer**
Bases: *trac.core.Component*

Default version control property difference renderer.

class trac.versioncontrol.web_ui.changeset.**IPropertyDiffRenderer**
Bases: *trac.core.Interface*

Render node properties in TracBrowser and TracChangeset views.

match_property_diff(*name*)
Indicate whether this renderer can treat the given property diffs

Returns a quality number, ranging from 0 (unsupported) to 9 (''perfect" match).

render_property_diff(*name*, *old_context*, *old_props*, *new_context*, *new_props*, *options*)
Render the given diff of property to HTML.

`name` is the property name as given to *match_property_diff()*, `old_context` corresponds to the old node being render (useful when the rendering depends on the node kind) and `old_props` is the corresponding collection of all properties. Same for `new_node` and `new_props`. `options` are the current diffs options.

The rendered result can be one of the following:

- `None`: the property change will be shown the normal way (''changed from `old` to `new`'')
- an `unicode` value: the change will be shown as textual content
- `Markup` or `Fragment`: the change will shown as block markup

trac.versioncontrol.web_ui.log

trac.versioncontrol.web_ui.util

trac.versioncontrol.web_ui.util.**make_log_graph**(*repos*, *revs*)
Generate graph information for the given revisions.

Returns a tuple `(threads, vertices, columns)`, where:

- `threads`: List of paint command lists `[(type, column, line)]`, where `type` is either 0 for "move to" or 1 for "line to", and `column` and `line` are coordinates.
- `vertices`: List of `(column, thread_index)` tuples, where the `i`th item specifies the column in which to draw the dot in line `i` and the corresponding thread.
- `columns`: Maximum width of the graph.

trac.versioncontrol.web_ui.util.**render_zip**(*req*, *filename*, *repos*, *root_node*, *iter_nodes*)
Send a ZIP file containing the data corresponding to the nodes iterable.

Parameters

- **root_node** (*Node*) – optional ancestor for all the *nodes*
- **iter_nodes** – callable taking the optional *root_node* as input and generating the *Node* for which the content should be added into the zip.

trac.web.api – Trac Web Request Handling

Primary interface for handling web requests.

Interfaces

The following interfaces allow components to interact at various stages of the web requests processing pipeline.

class trac.web.api.**IRequestHandler**
Bases: *trac.core.Interface*

Decide which *trac.core.Component* handles which *Request*, and how.

The boolean property *is_valid_default_handler* determines whether the *IRequestFilter* can be used as a default_handler and defaults to True. To be suitable as a default_handler, an *IRequestFilter* must return an HTML document and data dictionary for rendering the document, and must not require that *match_request* be called prior to *process_request*.

The boolean property jquery_noconflict determines whether jQuery's noConflict mode will be activated by the handler, and defaults to False.

See also trac.web.api.IRequestHandler extension point

match_request(*req*)
Return whether the handler wants to process the given request.

process_request(*req*)
Process the request.

Return a (template_name, data) pair, where data is a dictionary of substitutions for the Jinja2 template (the template context, in Jinja2 terms).

Optionally, the return value can also be a (template_name, data, metadata) triple, where metadata is a dict with hints for the template engine or the web front-end.

Keys supported are:

- 'content_type': the mimetype used for content delivery; "text/html" is assumed if the key is not present or the metadata was not specified
- 'text': a boolean value indicating whether the Jinja2 auto-escaping feature should be deactivated (text=True) or not (text=False); defaults to False, suitable for generating HTML or XML content
- 'fragment': a boolean value indicating whether the generated content will be used as part of another page (fragment=True) or as a stand-alone page (fragment=False), the default
- 'domain': a string value indicating the translation domain to which the translated strings in the template belong to

Note that if template processing should not occur, this method can simply send the response itself (see *Request* methods) and not return anything, as the *Request* methods raise a *RequestDone* exception.

Since 1.0 Clearsilver templates are no longer supported.

Since 1.1.2 the rendering `method` (xml, xhtml or text) may be returned as a fourth parameter in the tuple, but if not specified it will be inferred from the `content_type` when rendering the template.

Since 1.3.2 returns a pair, or a tuple in which the third element is a `dict` instead of a string like in the old API. Note that the old API (`(template, data, content_type)` where `content_type` is a string or `None`) is still supported. When used, this means that `template` is a legacy Genshi template. This support for the old API will be removed in Trac 1.5.1, in which `metadata` will always be a `dict` or `None` when specified.

Note: The *IRequestHandler.process_request* method plays a major role during the compatibility period in which both the legacy Genshi templates and the new Jinja2 templates are supported by Trac.

The return type of `(template_name, data, content_type)` tuple is still supported, and when it is used, it is interpreted as an indication that the template is actually a legacy Genshi template, and not a Jinja2 template. For the same backward compatibility reasons, returning `(template, data, None)` is interpreted as specifying a `content_type` of `None` (i.e. ending up with the `"text/html"` default).

This support for legacy Genshi templates will be removed in Trac 1.5.1, where only the new API will be supported. At that point, if the third value in the returned tuple is `None`, this will have the same effect as returning only a `(template, data)` pair or `(template, data, {})` triple (i.e. an empty metadata dict).

---

class `trac.web.api.`**IRequestFilter**

Bases: *trac.core.Interface*

Enable components to interfere with the processing done by the main handler, either before and/or after it enters in action.

See also trac.web.api.IRequestFilter extension point

post_process_request(*req*, *template*, *data*, *metadata=None*, *method=None*)

Do any post-processing the request might need

This typically means adding values to the template `data` dictionary, or changing the Jinja2 template.

`data` and `metdata` may be updated in place.

Always returns a tuple of `(template, data)` or `(template, data, metadata)`, even if unchanged.

Be aware that returning a `(template, data, None)` triple will be interpreted as using the legacy API and will indicate that the template is a legacy Genshi template. The same will happen if the third value of the tuple is a string.

The `method` last parameter is deprecated and is now supposed to be passed as `'method'` in the `metadata` dict. Even better, use directly the `'text'` indication (`True` or `False`).

Note that `template`, `data`, `content_type` will be `None` if:

- called when processing an error page
- the default request handler did not return any result

Since 1.0 Clearsilver templates are no longer supported.

Since 1.1.2 the rendering `method` will be passed if it is returned by the request handler, otherwise `method` will be `None`. For backward compatibility, the parameter is optional in the implementation's signature.

Since 1.3.2 Genshi templates are still supported, and if `process_request` used the old API (`(template, data, content_type)`), the `metadata` parameter passed to *`post_process_request`* will actually be the `content_type` value (`String` or `None`). This support for the old API will be removed in Trac 1.5.1, in which `metadata` will always be a `dict` or `None`.

pre_process_request(*req*, *handler*)

Called after initial handler selection, and can be used to change the selected handler or redirect request.

Always returns the request handler, even if unchanged.

For how the main content itself can be generated, see *`trac.web.chrome`*.

class trac.web.api.**ITemplateStreamFilter**

Bases: *`trac.core.Interface`*

Transform the generated content by filtering the Genshi event stream generated by the template, prior to its serialization.

Deprecated the Genshi template filtering concept doesn't apply anymore to Jinja2 templates, please consider converting your plugins to perform browser-side modifications of the rendered page using JavaScript. This interface will be removed in Trac 1.5.1.

See TracDev/PortingFromGenshiToJinja#ReplacingITemplateStreamFilter for details.

See also trac.web.api.ITemplateStreamFilter extension point

filter_stream(*req*, *method*, *filename*, *stream*, *data*)

Return a filtered Genshi event stream, or the original unfiltered stream if no match.

`req` is the current request object, `method` is the Genshi render method (xml, xhtml or text), `filename` is the filename of the template to be rendered, `stream` is the event stream and `data` is the data for the current template.

See the Genshi documentation for more information.

class trac.web.api.**IAuthenticator**

Bases: *`trac.core.Interface`*

Extension point interface for components that can provide the name of the remote user.

See also trac.web.api.IAuthenticator extension point

authenticate(*req*)

Return the name of the remote user, or `None` if the identity of the user is unknown.

Classes

class trac.web.api.**Request**(*environ*, *start_response*)

Bases: `object`

Represents a HTTP request/response pair.

This class provides a convenience API over WSGI.

Create the request wrapper.

Parameters

- **environ** – The WSGI environment dict
- **start_response** – The WSGI callback for starting the response
- **callbacks** – A dictionary of functions that are used to lazily evaluate attribute lookups

authname
The name associated with the user after authentification or `'anonymous'` if no authentification took place.

This corresponds to the *remote_user* when the request is targeted to an area requiring authentication, otherwise the authname is retrieved from the `trac_auth` cookie.

href
An *Href* instance for generating *relative* URLs pointing to resources within the current Trac environment.

abs_href
An *Href* instance for generating *absolute* URLs pointing to resources within the current Trac environment.

add_redirect_listener(*listener*)
Add a callable to be called prior to executing a redirect.

The callable is passed the arguments to the *redirect()* call.

base_path
The root path of the application

check_modified(*datetime*, *extra=''*)
Check the request "If-None-Match" header against an entity tag.

The entity tag is generated from the specified last modified time (`datetime`), optionally appending an `extra` string to indicate variants of the requested resource.

That `extra` parameter can also be a list, in which case the MD5 sum of the list content will be used.

If the generated tag matches the "If-None-Match" header of the request, this method sends a "304 Not Modified" response to the client. Otherwise, it adds the entity tag as an "ETag" header to the response so that consecutive requests can be cached.

end_headers()
Must be called after all headers have been sent and before the actual content is written.

get_header(*name*)
Return the value of the specified HTTP header, or `None` if there's no such header in the request.

is_authenticated
Returns `True` if *authname* is not `anonymous`.

Since 1.3.2

is_xhr
Returns `True` if the request is an `XMLHttpRequest`.

Since 1.1.6

method
The HTTP method of the request

path_info
Path inside the application

query_string
Query part of the request

read(*size=None*)
Read the specified number of bytes from the request body.

redirect(*url*, *permanent=False*)
Send a redirect to the client, forwarding to the specified URL.

The `url` may be relative or absolute, relative URLs will be translated appropriately.

remote_addr
IP address of the remote user

remote_user
Name of the remote user.

Will be `None` if the user has not logged in using HTTP authentication.

scheme
The scheme of the request URL

send_file(*path*, *mimetype=None*)
Send a local file to the browser.

This method includes the "Last-Modified", "Content-Type" and "Content-Length" headers in the response, corresponding to the file attributes. It also checks the last modification time of the local file against the "If-Modified-Since" provided by the user agent, and sends a "304 Not Modified" response if it matches.

send_header(*name*, *value*)
Send the response header with the specified name and value.

`value` must either be an `unicode` string or can be converted to one (e.g. numbers, ...)

send_response(*code=200*)
Set the status code of the response.

server_name
Name of the server

server_port
Port number the server is bound to

write(*data*)
Write the given data to the response body.

data **must** be a `str` string or an iterable instance which iterates `str` strings, encoded with the charset which has been specified in the `'Content-Type'` header or UTF-8 otherwise.

Note that when the `'Content-Length'` header is specified, its value either corresponds to the length of *data*, or, if there are multiple calls to *write*, to the cumulative length of the *data* arguments.

class `trac.web.api.`**RequestDone**(*iterable=None*)
Bases: *trac.core.TracBaseError*

Marker exception that indicates whether request processing has completed and a response was sent.

Helper Functions

`trac.web.api.`**arg_list_to_args**(*arg_list*)
Convert a list of `(name, value)` tuples into into a `_RequestArgs`.

`trac.web.api.`**parse_arg_list**(*query_string*)
Parse a query string into a list of `(name, value)` tuples.

Since 1.1.2 a leading `?` is stripped from `query_string`.

trac.web.api.**is_valid_default_handler**(*handler*)
Returns True if the handler is a valid default handler, as described in the *IRequestHandler* interface documentation.

Exceptions

exception trac.web.api.**TracNotImplementedError**(*message*, *title=None*, *show_traceback=False*)
Bases: *trac.core.TracError*, exceptions.NotImplementedError

Raised when a NotImplementedError is trapped.

This exception is for internal use and should not be raised by plugins. Plugins should raise NotImplementedError.

Since 1.0.11

If message is an Element object, everything up to the first <p> will be displayed in the red box, and everything after will be displayed below the red box. If title is given, it will be displayed as the large header above the error message.

exception trac.web.api.**HTTPBadGateway**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 502 Bad Gateway

Factory for HTTPException classes.

exception trac.web.api.**HTTPBadRequest**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 400 Bad Request

Factory for HTTPException classes.

exception trac.web.api.**HTTPConflict**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 409 Conflict

Factory for HTTPException classes.

exception trac.web.api.**HTTPExpectationFailed**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 417 Expectation Failed

Factory for HTTPException classes.

exception trac.web.api.**HTTPForbidden**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 403 Forbidden

Factory for HTTPException classes.

exception trac.web.api.**HTTPGatewayTimeout**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 504 Gateway Timeout

Factory for HTTPException classes.

exception trac.web.api.**HTTPGone**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 410 Gone

Factory for HTTPException classes.

exception trac.web.api.**HTTPLengthRequired**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 411 Length Required

Factory for HTTPException classes.

exception trac.web.api.**HTTPMethodNotAllowed**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 405 Method Not Allowed

Factory for HTTPException classes.

exception trac.web.api.**HTTPNotAcceptable**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 406 Not Acceptable

Factory for HTTPException classes.

exception trac.web.api.**HTTPNotFound**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 404 Not Found

Factory for HTTPException classes.

exception trac.web.api.**HTTPNotImplemented**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 501 Not Implemented

Factory for HTTPException classes.

exception trac.web.api.**HTTPPaymentRequired**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 402 Payment Required

Factory for HTTPException classes.

exception trac.web.api.**HTTPPreconditionFailed**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 412 Precondition Failed

Factory for HTTPException classes.

exception trac.web.api.**HTTPProxyAuthenticationRequired**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 407 Proxy Authentication Required

Factory for HTTPException classes.

exception trac.web.api.**HTTPRequestEntityTooLarge**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 413 Request Entity Too Large

Factory for HTTPException classes.

exception trac.web.api.**HTTPRequestTimeout**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 408 Request Timeout

Factory for HTTPException classes.

exception trac.web.api.**HTTPRequestUriTooLong**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 414 Request-Uri Too Long

Factory for HTTPException classes.

exception trac.web.api.**HTTPRequestedRangeNotSatisfiable**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 416 Requested Range Not Satisfiable

Factory for HTTPException classes.

exception trac.web.api.**HTTPServiceUnavailable**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 503 Service Unavailable

Factory for HTTPException classes.

exception trac.web.api.**HTTPUnauthorized**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 401 Unauthorized

Factory for HTTPException classes.

exception trac.web.api.**HTTPUnsupportedMediaType**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 415 Unsupported Media Type

Factory for HTTPException classes.

exception trac.web.api.**HTTPVersionNotSupported**(*detail*, **args*)
Bases: trac.web.api.HTTPException

Exception for HTTP 505 Http Version Not Supported

Factory for HTTPException classes.

trac.web.auth – Trac Authentication

This module deals with web request authentication, and provides the default implementation for the *IAuthenticator* interface.

Component

class trac.web.auth.**LoginModule**
Bases: *trac.core.Component*

User authentication manager.

This component implements user authentication based on HTTP authentication provided by the web-server, combined with cookies for communicating the login information across the whole site.

This mechanism expects that the web-server is setup so that a request to the path '/login' requires authentication (such as Basic or Digest). The login name is then stored in the database and associated with a unique key that gets passed back to the user agent using the 'trac_auth' cookie. This cookie is used to identify the user in subsequent requests to non-protected resources.

auth_cookie_domain
Auth cookie domain attribute.

The auth cookie can be shared among multiple subdomains by setting the value to the domain. (//since 1.2//)

auth_cookie_lifetime
Lifetime of the authentication cookie, in seconds.

This value determines how long the browser will cache authentication information, and therefore, after how much inactivity a user will have to log in again. The value of 0 makes the cookie expire at the end of the browsing session.

auth_cookie_path
Path for the authentication cookie. Set this to the common base path of several Trac instances if you want them to share the cookie.

check_ip
Whether the IP address of the user should be checked for authentication.

ignore_case
Whether login names should be converted to lower case.

Support Classes

A few classes are provided for directly computing the REMOTE_USER information from the HTTP headers for Basic or Digest authentication. This will be used by the `AuthenticationMiddleware`.

class `trac.web.auth.`**`BasicAuthentication`**(*htpasswd*, *realm*)
Bases: `trac.web.auth.PasswordFileAuthentication`

class `trac.web.auth.`**`DigestAuthentication`**(*htdigest*, *realm*)
Bases: `trac.web.auth.PasswordFileAuthentication`

A simple HTTP digest authentication implementation (**RFC 2617**).

load(*filename*)
Load account information from apache style htdigest files, only users from the specified realm are used

send_auth_request(*environ*, *start_response*, *stale='false'*)
Send a digest challange to the browser. Record used nonces to avoid replay attacks.

`trac.web.chrome` – Trac content generation for the Web

Interfaces

class `trac.web.chrome.`**`INavigationContributor`**
Bases: *`trac.core.Interface`*

Extension point interface for components that contribute items to the navigation.

See also trac.web.chrome.INavigationContributor extension point

get_active_navigation_item(*req*)
This method is only called for the `IRequestHandler` processing the request.

It should return the name of the navigation item to be highlighted as active/current.

get_navigation_items(*req*)
Should return an iterable object over the list of navigation items to add, each being a tuple in the form (category, name, text).

The category determines the location of the navigation item and can be `mainnav` or `metanav`. The name is a unique identifier that must match the string returned by get_active_navigation_item. The text is typically a link element with text that corresponds to the desired label for the navigation item, and an href.

class `trac.web.chrome.`**ITemplateProvider**
Bases: *trac.core.Interface*

Extension point interface for components that provide their own Jinja2 templates and/or accompanying static resources.

See also trac.web.chrome.ITemplateProvider extension point

get_htdocs_dirs()
Return a list of directories with static resources (such as style sheets, images, etc.)

Each item in the list must be a `(prefix, abspath)` tuple. The `prefix` part defines the path in the URL that requests to these resources are prefixed with.

The `abspath` is the absolute path to the directory containing the resources on the local file system.

get_templates_dirs()
Return a list of directories containing the provided template files.

Components

The *Chrome* component is in charge of generating the content of the pages, with the help of template engines. The default engine for Trac 1.4 is Jinja2, though we'll still support Genshi until the next development cycle begins (Trac 1.5.1).

The high-level API for generating content is the *render_template* method, which is paired with the output of the *process_request* method. As such, it accepts simple but versatile parameters, and generates output which can be directly sent to the web front-end.

There's an intermediate level API for generating *fragment* of content, either HTML or text. A fragment is typically an element of a web page, or the result for an XHR, or simply not a HTML at all but just some text. When the output of the fragment will be sent to the web front-end (typically when responding to XHR), use *generate_fragment*. Otherwise, when the output should be manipulated programmatically as a string (typically integrated in a *Fragment*), use *render_fragment*.

The low-level API for generating content with the Jinja2 template engine comprises *prepare_template*, which creates a `jinja2.Template`. More precisely, it combines *load_template* and *populate_data*. Such a `jinja2.Template` can then be passed to either *generate_template_stream* or *render_template_string*, depending on the desired kind of output.

For even lower-level access to the template engine, see the section *The Jinja2 template engine* related to Jinja2.

class `trac.web.chrome.`**Chrome**
Bases: *trac.core.Component*

Web site chrome assembly manager.

Chrome is everything that is not actual page content.

add_auto_preview(*req*)
Setup auto-preview for `<textarea>` fields.

add_jquery_ui(*req*)
Add a reference to the jQuery UI script and link the stylesheet.

add_textarea_grips(*req*)
Make `<textarea class="trac-resizable">` fields resizable if enabled by configuration.

add_wiki_toolbars(*req*)
Add wiki toolbars to `<textarea class="wikitext">` fields.

author_email(*author*, *email_map*)
Returns the author email from the `email_map` if `author` doesn't look like an email address.

authorinfo(*req*, *author*, *email_map=None*, *resource=None*)
Format a username to HTML.

Calls *Chrome.format_author* to format the username, and wraps the formatted username in a `span` with class `trac-author`, `trac-author-anonymous` or `trac-author-none`.

Parameters

- **req** – the `Request` object.
- **author** – the author string to be formatted.
- **email_map** – dictionary mapping usernames to email addresses.
- **resource** – optional `Resource` object for `EMAIL_VIEW` fine-grained permissions checks.

Since 1.1.6 accepts the optional `resource` keyword parameter.

auto_preview_timeout
Inactivity timeout in seconds after which the automatic wiki preview triggers an update. This option can contain floating-point values. The lower the setting, the more requests will be made to the server. Set this to 0 to disable automatic preview.

auto_reload
Automatically reload template files after modification.

cc_list(*cc_field*)
Split a CC: value in a list of addresses.

default_dateinfo_format
The date information format. Valid options are 'relative' for displaying relative format and 'absolute' for displaying absolute format. (''since 1.0'')

environment_created()
Create the environment templates directory.

format_author(*req*, *author*, *resource=None*, *show_email=None*)
Format a username in plain text.

If `[trac] show_email_addresses` is `False`, email addresses will be obfuscated when the user doesn't have `EMAIL_VIEW` (for the resource) and the optional parameter `show_email` is `None`. Returns translated `anonymous` or `none`, when the author string is `anonymous` or evaluates to `False`, respectively.

Parameters

- **req** – a `Request` or `RenderingContext` object.
- **author** – the author string to be formatted.

- **resource** – an optional `Resource` object for performing fine-grained permission checks for `EMAIL_VIEW`.
- **show_email** – an optional parameter that allows explicit control of e-mail obfuscation.

Since 1.1.6 accepts the optional `resource` keyword parameter.

Since 1.2 Full name is returned when `[trac] show_full_names` is `True`.

Since 1.2 Email addresses are obfuscated when *show_email_addresses* is False and `req` is Falsy. Previously email addresses would not be obfuscated whenever `req` was Falsy (typically `None`).

format_emails(*context*, *value*, *sep=', '*)

Normalize a list of e-mails and obfuscate them if needed.

Parameters

- **context** – the context in which the check for obfuscation should be done
- **value** – a string containing a comma-separated list of e-mails
- **sep** – the separator to use when rendering the list again

generate_fragment(*req*, *filename*, *data*, *text=False*, *domain=None*)

Produces content ready to be sent from the given template *filename* and input *data*, with minimal overhead.

It calls *prepare_template* to augment the *data* with the usual helper functions that can be expected in Trac templates, including the translation helper functions adapted to the given *domain*, except for some of the chrome "late" data.

If you don't need that and don't want the overhead, use *load_template* and *generate_template_stream* directly.

The generated output is suitable to pass directly to `Request.send`, see *generate_template_stream* for details.

See also *render_fragment*, which returns a string instead.

generate_template_stream(*template*, *data*, *text=False*, *iterable=None*)

Returns the rendered template in a form that can be "sent".

This will be either a single UTF-8 encoded `str` object, or an iterable made of chunks of the above.

Parameters

- **template** (`jinja2.Template`) – the Jinja2 template
- **text** – in text mode (`True`) the generated bytes will not be sanitized (see `valid_html_bytes`).
- **iterable** – determine whether the output should be generated in chunks or as a single `str`; if `None`, the *use_chunked_encoding* property will be used to determine this instead

Return type `str` or an iterable of `str`, depending on *iterable*

Turning off the XML/HTML auto-escape feature for variable expansions has to be disabled when loading the template (see *load_template*), so remember to stay consistent with the *text* parameter.

genshi_cache_size

The maximum number of templates that the template loader will cache in memory. You may want to choose a higher value if your site uses a larger number of templates, and you have enough memory to spare, or you can reduce it if you are short on memory.

(''deprecated, will be removed in Trac 1.5.1'')

get_all_templates_dirs()
Return a list of the names of all known templates directories.

get_email_map()
Get the email addresses of all known users.

get_interface_customization_files()
Returns a dictionary containing the lists of files present in the site and shared templates and htdocs directories.

get_permission_actions()
`EMAIL_VIEW` permission allows for showing email addresses even if `[trac] show_email_addresses` is `false`.

htdocs_location
Base URL for serving the core static resources below `/chrome/common/`.

It can be left empty, and Trac will simply serve those resources itself.

Advanced users can use this together with [TracAdmin trac-admin ... deploy <deploydir>] to allow serving the static resources for Trac directly from the web server. Note however that this only applies to the `<deploydir>/htdocs/common` directory, the other deployed resources (i.e. those from plugins) will not be made available this way and additional rewrite rules will be needed in the web server.

iterable_content(*stream*, *text=False*, ***kwargs*)
Generate an iterable object which iterates `str` instances from the given stream instance.

Parameters **text** – in text mode (`True`) XML/HTML auto-escape of variable expansion is disabled.

If text is a string, this corresponds to the old API and we assume to have a Genshi stream. This backward compatibility behavior will be removed in Trac 1.5.1.

jquery_location
Location of the jQuery !JavaScript library (version %(version)s).

An empty value loads jQuery from the copy bundled with Trac.

Alternatively, jQuery could be loaded from a CDN, for example: http://code.jquery.com/jquery-%(version)s.min.js, http://ajax.aspnetcdn.com/ajax/jQuery/jquery-%(version)s.min.js or https://ajax.googleapis.com/ajax/libs/jquery/%(version)s/jquery.min.js.

(''since 1.0'')

jquery_ui_location
Location of the jQuery UI !JavaScript library (version %(version)s).

An empty value loads jQuery UI from the copy bundled with Trac.

Alternatively, jQuery UI could be loaded from a CDN, for example: https://ajax.googleapis.com/ajax/libs/jqueryui/%(version)s/jquery-ui.min.js or http://ajax.aspnetcdn.com/ajax/jquery.ui/%(version)s/jquery-ui.min.js.

(''since 1.0'')

jquery_ui_theme_location
Location of the theme to be used with the jQuery UI !JavaScript library (version %(version)s).

An empty value loads the custom Trac jQuery UI theme from the copy bundled with Trac.

Alternatively, a jQuery UI theme could be loaded from a CDN, for example: https://ajax.googleapis.com/ajax/libs/jqueryui/%(version)s/themes/start/jquery-ui.css or http://ajax.aspnetcdn.com/ajax/jquery.ui/%(version)s/themes/start/jquery-ui.css.

(''since 1.0'')

load_template(*filename*, *text=False*)
Retrieves a template with the given name.

This simply loads the template. If you want to make use of the "standard" Trac API for templates, also call *populate_data*, or consider using the shortcut method *prepare_template* instead.

Parameters **text** – in text mode (`True`) XML/HTML auto-escape of variable expansion is disabled.

Note: If the `text` argument is set to a string instead of a boolean, this corresponds to the legacy API and it will be assumed that you want to load a Genshi template instead of a Jinja2 template. If *text* is `'text'`, a `NewTextTemplate` instance will be created. If it is set to `None` or to another string value, a `MarkupTemplate` instance will be created and returned.

This backward compatibility behavior will be removed in Trac 1.5.1.

---

logo_alt
Alternative text for the header logo.

logo_height
Height of the header logo image in pixels.

logo_link
URL to link to, from the header logo.

logo_src
URL of the image to use as header logo. It can be absolute, server relative or relative.

If relative, it is relative to one of the `/chrome` locations: `site/your-logo.png` if `your-logo.png` is located in the `htdocs` folder within your TracEnvironment; `common/your-logo.png` if `your-logo.png` is located in the folder mapped to the [#trac-section htdocs_location] URL. Only specifying `your-logo.png` is equivalent to the latter.

logo_width
Width of the header logo image in pixels.

mainnav
Configures the main navigation bar, which by default contains //Wiki//, //Timeline//, //Roadmap//, //Browse Source//, //View Tickets//, //New Ticket//, //Search// and //Admin//.

The `label`, `href`, and `order` attributes can be specified. Entries can be disabled by setting the value of the navigation item to `disabled`.

The following example renames the link to WikiStart to //Home//, links the //View Tickets// entry to a specific report and disables the //Search// entry. {{{#!ini [mainnav] wiki.label = Home tickets.href = /report/24 search = disabled }}}

See TracNavigation for more details.

metanav
Configures the meta navigation entries, which by default are //Login//, //Logout//, //Preferences//, ''!Help/Guide" and //About Trac//. The allowed attributes are the same as for `[mainnav]`. Additionally, a special entry is supported - `logout.redirect` is the page the user sees after hitting the logout button. For example:

{{{#!ini [metanav] logout.redirect = wiki/Logout }}}

See TracNavigation for more details.

navigation_contributors
List of components that implement *INavigationContributor*

never_obfuscate_mailto
Never obfuscate `mailto:` links explicitly written in the wiki, even if *show_email_addresses* is false or the user doesn't have EMAIL_VIEW permission.

populate_data(*req=None*, *data=None*, *d=None*)
Fills a dictionary with the standard set of fields expected by templates.

Parameters

- **req** – a `Request` object; if `None`, no request related fields will be set
- **data** – user-provided `dict`, which can be used to override the defaults; if `None`, the defaults will be returned
- **d** – `dict` which is populated with the defaults; if `None`, an empty `dict` will be used

prepare_request(*req*, *handler=None*)
Prepare the basic chrome data for the request.

Parameters

- **req** – the request object
- **handler** – the `IRequestHandler` instance that is processing the request

prepare_template(*req*, *filename*, *data*, *text=False*, *domain=None*)
Prepares the rendering of a Jinja2 template.

This loads the template and prepopulates a data `dict` with the "standard" Trac API for templates.

Parameters

- **req** – a `Request` instance (optional)
- **filename** – the name of a Jinja2 template, which must be found in one of the template directories (see `get_templates_dirs`)
- **data** – user specified data dictionary, used to override the default context set from the request *req* (see *populate_data*)
- **text** – in text mode (`True`) XML/HTML auto-escape of variable expansion is disabled.

Return type a pair of Jinja2 `Template` and a `dict`.

render_fragment(*req*, *filename*, *data*, *text=False*, *domain=None*)
Produces a string from given template *filename* and input *data*, with minimal overhead.

It calls *prepare_template* to augment the *data* with the usual helper functions that can be expected in Trac templates, including the translation helper functions adapted to the given *domain*, except for some of the chrome "late" data.

If you don't need that and don't want the overhead, use *load_template* and *render_template_string* directly.

Return type the generated output is an `unicode` string if *text* is `True`, or a `Markup` string otherwise.

See also *generate_fragment*, which produces an output suitable to pass directly to `Request.send` instead.

render_template(*req*, *filename*, *data*, *metadata=None*, *fragment=False*, *iterable=False*, *method=None*)
Renders the `filename` template using `data` for the context.

It attempts to load a Jinja2 template, augments the provided *data* with standard data, and renders it according to the options provided in *metadata*.

The `fragment`, `iterable` and `method` parameters are deprecated and will be removed in Trac 1.5.1. Instead, use the `metadata` dictionary with keys with the same name. The `method` key is itself deprecated, use `text=True` instead of `method='text'` to indicate that the template is a plain text one, with no need for HTML/XML escaping to take place.

When `fragment` is specified, or `method` is `'text'`, or `text` is `True`, we generate some content which does not need all of the chrome related data, typically HTML fragments, XML or plain text.

If `iterable` is set, we use *`generate_template_stream`* to produce the output (iterable of UTF-8 encoded bytes), otherwise we use *`render_template_string`* and UTF-8 encode the result.

Note: If the Jinja2 template is not found, this method will try to load a Genshi template instead.

Also if `metadata` is *not* a dictionary, we assume that it is the `content_type` value from the legacy API (a string or `None`). As a consequence, we assume the template will be a Genshi template and we'll use the legacy Genshi template engine for the rendering.

This backward compatibility behavior will be removed in Trac 1.5.1.

`render_template_string`(*template*, *data*, *text=False*)
Renders the template as an unicode or Markup string.

Parameters

- **`template`** (`jinja2.Template`) – the Jinja2 template
- **`text`** – in text mode (`True`) the generated string will not be wrapped in `Markup`

Return type `unicode` if *text* is `True`, `Markup` otherwise.

Turning off the XML/HTML auto-escape feature for variable expansions has to be disabled when loading the template (see *`load_template`*), so remember to stay consistent with the *text* parameter.

`resizable_textareas`
Make `<textarea>` fields resizable. Requires !JavaScript.

`shared_htdocs_dir`
Path to the //shared htdocs directory//.

Static resources in that directory are mapped to /chrome/shared under the environment URL, in addition to common and site locations.

This can be useful in site.html for common interface customization of multiple Trac environments.

(''since 1.0'')

`shared_templates_dir`
Path to the //shared templates directory//.

Templates in that directory are loaded in addition to those in the environments `templates` directory, but the latter take precedence.

`show_email_addresses`
Show email addresses instead of usernames. If false, email addresses are obfuscated for users that don't have EMAIL_VIEW permission.

`show_full_names`
Show full names instead of usernames. (//since 1.2//)

stream_filters
List of components that implement *ITemplateStreamFilter*

template_providers
List of components that implement *ITemplateProvider*

use_chunked_encoding
If enabled, send contents as chunked encoding in HTTP/1.1. Otherwise, send contents with `Content-Length` header after entire of the contents are rendered. (''since 1.0.6'')

wiki_toolbars
Add a simple toolbar on top of Wiki <textarea>s. (''since 1.0.2'')

Functions

Most of the helper functions are related to content generation, and in particular, (X)HTML content generation, in one way or another.

`trac.web.chrome.`**`web_context`**(*req*, *resource=None*, *id=False*, *version=False*, *parent=False*, *absurls=False*)
Create a rendering context from a request.

The `perm` and `href` properties of the context will be initialized from the corresponding properties of the request object.

```
>>> from trac.test import Mock, MockPerm
>>> req = Mock(href=Mock(), perm=MockPerm())
>>> context = web_context(req)
>>> context.href is req.href
True
>>> context.perm is req.perm
True
```

Parameters

- **req** – the HTTP request object
- **resource** – the `Resource` object or realm
- **id** – the resource identifier
- **version** – the resource version
- **absurls** – whether URLs generated by the `href` object should be absolute (including the protocol scheme and host name)

Returns a new rendering context

Return type `RenderingContext`

Since version 1.0

`trac.web.chrome.`**`add_meta`**(*req*, *content*, *http_equiv=None*, *name=None*, *scheme=None*, *lang=None*)
Add a `<meta>` tag into the `<head>` of the generated HTML.

Web resources

`trac.web.chrome.`**`add_stylesheet`**(*req*, *filename*, *mimetype='text/css'*, ***attrs*)
Add a link to a style sheet to the chrome info so that it gets included in the generated HTML page.

If `filename` is a network-path reference (i.e. starts with a protocol or `//`), the return value will not be modified. If `filename` is absolute (i.e. starts with `/`), the generated link will be based off the application root path. If it is relative, the link will be based off the `/chrome/` path.

`trac.web.chrome.`**`add_script`**(*req*, *filename*, *mimetype='text/javascript'*, *charset='utf-8'*, *ie_if=None*)
Add a reference to an external javascript file to the template.

If `filename` is a network-path reference (i.e. starts with a protocol or `//`), the return value will not be modified. If `filename` is absolute (i.e. starts with `/`), the generated link will be based off the application root path. If it is relative, the link will be based off the `/chrome/` path.

`trac.web.chrome.`**`add_script_data`**(*req*, *data={}*, ***kwargs*)
Add data to be made available in javascript scripts as global variables.

The keys in `data` and the keyword argument names provide the names of the global variables. The values are converted to JSON and assigned to the corresponding variables.

Page admonitions

`trac.web.chrome.`**`add_warning`**(*req*, *msg*, **args*)
Add a non-fatal warning to the request object.

When rendering pages, all warnings will be rendered to the user. Note that the message is escaped (and therefore converted to `Markup`) before it is stored in the request object.

`trac.web.chrome.`**`add_notice`**(*req*, *msg*, **args*)
Add an informational notice to the request object.

When rendering pages, all notices will be rendered to the user. Note that the message is escaped (and therefore converted to `Markup`) before it is stored in the request object.

Contextual Navigation

`trac.web.chrome.`**`add_link`**(*req*, *rel*, *href*, *title=None*, *mimetype=None*, *classname=None*, ***attrs*)
Add a link to the chrome info that will be inserted as <link> element in the <head> of the generated HTML

`trac.web.chrome.`**`add_ctxtnav`**(*req*, *elm_or_label*, *href=None*, *title=None*)
Add an entry to the current page's ctxtnav bar.

`trac.web.chrome.`**`prevnext_nav`**(*req*, *prev_label*, *next_label*, *up_label=None*)
Add Previous/Up/Next navigation links.

Parameters

- **`req`** – a `Request` object
- **`prev_label`** – the label to use for left (previous) link
- **`up_label`** – the label to use for the middle (up) link
- **`next_label`** – the label to use for right (next) link

Miscellaneous

`trac.web.chrome.`**`accesskey`**(*req*, *key*)
Helper function for creating accesskey HTML attribute according to preference values

trac.web.chrome.**auth_link**(*req*, *link*)
Return an "authenticated" link to `link` for authenticated users.

If the user is anonymous, returns `link` unchanged. For authenticated users, returns a link to `/login` that redirects to `link` after authentication.

Internals

trac.web.chrome.**chrome_info_script**(*req*, *use_late=None*)
Get script elements from chrome info of the request object during rendering template or after rendering.

Parameters

- **req** – the HTTP request object.
- **use_late** – if True, `late_links` will be used instead of `links`.

trac.web.chrome.**chrome_resource_path**(*req*, *filename*)
Get the path for a chrome resource given its `filename`.

If `filename` is a network-path reference (i.e. starts with a protocol or `//`), the return value will not be modified. If `filename` is absolute (i.e. starts with `/`), the generated link will be based off the application root path. If it is relative, the link will be based off the `/chrome/` path.

trac.web.href – Creation of URLs

This module mainly proposes the following class:

class trac.web.href.**Href**(*base*, *path_safe="/!~*'()"*, *query_safe="!~*'()"*)
Bases: object

Implements a callable that constructs URLs with the given base. The function can be called with any number of positional and keyword arguments which then are used to assemble the URL.

Positional arguments are appended as individual segments to the path of the URL:

```
>>> href = Href('/trac')
>>> repr(href)
"<Href '/trac'>"
>>> href('ticket', 540)
'/trac/ticket/540'
>>> href('ticket', 540, 'attachment', 'bugfix.patch')
'/trac/ticket/540/attachment/bugfix.patch'
>>> href('ticket', '540/attachment/bugfix.patch')
'/trac/ticket/540/attachment/bugfix.patch'
```

If a positional parameter evaluates to None, it will be skipped:

```
>>> href('ticket', 540, 'attachment', None)
'/trac/ticket/540/attachment'
```

The first path segment can also be specified by calling an attribute of the instance, as follows:

```
>>> href.ticket(540)
'/trac/ticket/540'
>>> href.changeset(42, format='diff')
'/trac/changeset/42?format=diff'
```

Simply calling the Href object with no arguments will return the base URL:

```
>>> href()
'/trac'
```

Keyword arguments are added to the query string, unless the value is None:

```
>>> href = Href('/trac')
>>> href('timeline', format='rss')
'/trac/timeline?format=rss'
>>> href('timeline', format=None)
'/trac/timeline'
>>> href('search', q='foo bar')
'/trac/search?q=foo+bar'
```

Multiple values for one parameter are specified using a sequence (a list or tuple) for the parameter:

```
>>> href('timeline', show=['ticket', 'wiki', 'changeset'])
'/trac/timeline?show=ticket&show=wiki&show=changeset'
```

Alternatively, query string parameters can be added by passing a dict or list as last positional argument:

```
>>> href('timeline', {'from': '02/24/05', 'daysback': 30})
'/trac/timeline?daysback=30&from=02%2F24%2F05'
>>> href('timeline', {})
'/trac/timeline'
>>> href('timeline', [('from', '02/24/05')])
'/trac/timeline?from=02%2F24%2F05'
>>> href('timeline', ()) == href('timeline', []) == href('timeline', {})
True
```

The usual way of quoting arguments that would otherwise be interpreted as Python keywords is supported too:

```
>>> href('timeline', from_='02/24/05', daysback=30)
'/trac/timeline?from=02%2F24%2F05&daysback=30'
```

If the order of query string parameters should be preserved, you may also pass a sequence of (name, value) tuples as last positional argument:

```
>>> href('query', (('group', 'component'), ('groupdesc', 1)))
'/trac/query?group=component&groupdesc=1'
```

```
>>> params = []
>>> params.append(('group', 'component'))
>>> params.append(('groupdesc', 1))
>>> href('query', params)
'/trac/query?group=component&groupdesc=1'
```

By specifying an absolute base, the function returned will also generate absolute URLs:

```
>>> href = Href('http://trac.edgewall.org')
>>> href('ticket', 540)
'http://trac.edgewall.org/ticket/540'
```

```
>>> href = Href('https://trac.edgewall.org')
>>> href('ticket', 540)
'https://trac.edgewall.org/ticket/540'
```

In common usage, it may improve readability to use the function-calling ability for the first component of the URL as mentioned earlier:

```
>>> href = Href('/trac')
>>> href.ticket(540)
'/trac/ticket/540'
>>> href.browser('/trunk/README.txt', format='txt')
'/trac/browser/trunk/README.txt?format=txt'
```

The `path_safe` argument specifies the characters that don't need to be quoted in the path arguments. Likewise, the `query_safe` argument specifies the characters that don't need to be quoted in the query string:

```
>>> href = Href('')
>>> href.milestone('<look,here>', param='<here,too>')
'/milestone/%3Clook%2Chere%3E?param=%3Chere%2Ctoo%3E'
```

```
>>> href = Href('', path_safe='/<,', query_safe=',>')
>>> href.milestone('<look,here>', param='<here,too>')
'/milestone/<look,here%3E?param=%3Chere,too>'
```

`trac.web.main` – Trac Web Entry Point

Entry point for dispatching web requests.

`trac.web.dispatch_request`

The WSGI compliant callable. It adapts the `environ` information passed from the WSGI gateway and retrieve the appropriate *`Environment`* from it, creates a *`Request`* instance and let the *`RequestDispatcher`* component forward it to the component implementing a matching *`IRequestHandler`*.

`trac.web.main.`**`dispatch_request`**(*environ*, *start_response*)

Main entry point for the Trac web interface.

Parameters

- **`environ`** – the WSGI environment dict
- **`start_response`** – the WSGI callback for starting the response

Components

class `trac.web.main.`**`RequestDispatcher`**

Bases: *`trac.core.Component`*

Web request dispatcher.

This component dispatches incoming requests to registered handlers. Besides, it also takes care of user authentication and request pre- and post-processing.

`authenticators`

List of components that implement *`IAuthenticator`*

`default_date_format`

The date format. Valid options are 'iso8601' for selecting ISO 8601 format, or leave it empty which means the default date format will be inferred from the browser's default language. (''since 1.0'')

default_handler
Name of the component that handles requests to the base URL.

Options include `TimelineModule`, `RoadmapModule`, `BrowserModule`, `QueryModule`, `ReportModule`, `TicketModule` and `WikiModule`.

The [/prefs/userinterface session preference] for default handler take precedence, when set.

default_language
The preferred language to use if no user preference has been set.

default_timezone
The default timezone to use

dispatch(*req*)
Find a registered handler that matches the request and let it process it.

In addition, this method initializes the data dictionary passed to the the template and adds the web site chrome.

filters
Ordered list of filters to apply to all requests.

handlers
List of components that implement *IRequestHandler*

set_default_callbacks(*req*)
Setup request callbacks for lazily-evaluated properties.

use_xsendfile
When true, send a `X-Sendfile` header and no content when sending files from the filesystem, so that the web server handles the content. This requires a web server that knows how to handle such a header, like Apache with `mod_xsendfile` or lighttpd. (''since 1.0'')

xsendfile_header
The header to use if *use_xsendfile* is enabled. If Nginx is used, set `X-Accel-Redirect`. (''since 1.0.6'')

Classes

class `trac.web.main.`**RequestWithSession**(*environ*, *start_response*)
Bases: *trac.web.api.Request*

A request that saves its associated session when sending the reply.

Create the request wrapper.

Parameters

- **environ** – The WSGI environment dict
- **start_response** – The WSGI callback for starting the response
- **callbacks** – A dictionary of functions that are used to lazily evaluate attribute lookups

Helper Functions

`trac.web.main.`**get_environments**(*environ*, *warn=False*)
Retrieve canonical environment name to path mapping.

The environments may not be all valid environments, but they are good candidates.

trac.web.main.**get_tracignore_patterns**(*env_parent_dir*)
Return the list of patterns from env_parent_dir/.tracignore or a default pattern of `".*"` if the file doesn't exist.

Miscellaneous

trac.web.main.**default_tracker = 'https://trac.edgewall.org'**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

trac.web.session

class trac.web.session.**Session**(*env*, *req*)
Bases: trac.web.session.DetachedSession

Basic session handling and per-session storage.

promote_session(*sid*)
Promotes an anonymous session to an authenticated session, if there is no preexisting session data for that user name.

class trac.web.session.**SessionAdmin**
Bases: *trac.core.Component*

trac-admin command provider for session management

request_handlers
List of components that implement *IRequestHandler*

trac.web.standalone

trac.web.wsgi

class trac.web.wsgi.**WSGIGateway**(*environ*, *stdin=<open file '<stdin>', mode 'r'>*, *stderr=<open file '<stderr>', mode 'w'>*)
Bases: object

Abstract base class for WSGI servers or gateways.

Initialize the gateway object.

run(*application*)
Start the gateway with the given WSGI application.

wsgi_file_wrapper
alias of _FileWrapper

trac.web.wsgi.**is_client_disconnect_exception**(*e*)
Determines whether the exception was caused by a disconnecting client.

Return type bool

trac.wiki.admin

class trac.wiki.admin.**WikiAdmin**
Bases: *trac.core.Component*

trac-admin command provider for wiki administration.

environment_created()
Add default wiki pages when environment is created.

trac.wiki.admin.**datetime_now**()
[tz] -> new datetime with tz's local day and time.

trac.wiki.api – The Wiki API

Interfaces

The wiki module presents several possibilities of extension, for interacting with the Wiki application and also for extending the Wiki syntax.

First, components can be notified of the changes happening in the wiki.

class trac.wiki.api.**IWikiChangeListener**
Bases: *trac.core.Interface*

Components that want to get notified about the creation, deletion and modification of wiki pages should implement that interface.

See also trac.wiki.api.IWikiChangeListener extension point.

wiki_page_added(*page*)
Called whenever a new Wiki page is added.

wiki_page_changed(*page*, *version*, *t*, *comment*, *author*)
Called when a page has been modified.

wiki_page_comment_modified(*page*, *old_comment*)
Called when a page comment has been modified.

wiki_page_deleted(*page*)
Called when a page has been deleted.

wiki_page_renamed(*page*, *old_name*)
Called when a page has been renamed.

wiki_page_version_deleted(*page*)
Called when a version of a page has been deleted.

Components can also interfere with the changes, before or after they're made.

class trac.wiki.api.**IWikiPageManipulator**
Bases: *trac.core.Interface*

Components that need to do specific pre- and post- processing of wiki page changes have to implement this interface.

Unlike change listeners, a manipulator can reject changes being committed to the database.

See also trac.wiki.api.IWikiPageManipulator extension point.

prepare_wiki_page(*req*, *page*, *fields*)
Validate a wiki page before rendering it.

Parameters

- **page** – is the WikiPage being viewed.

- **fields** – is a dictionary which contains the wiki `text` of the page, initially identical to `page.text` but it can eventually be transformed in place before being used as input to the formatter.

validate_wiki_page(*req*, *page*)
Validate a wiki page after it's been populated from user input.

Parameters **page** – is the `WikiPage` being edited.

Returns a list of `(field, message)` tuples, one for each problem detected. `field` can be `None` to indicate an overall problem with the page. Therefore, a return value of `[]` means everything is OK.

Then, the Wiki syntax itself can be extended. The first and less intrusive way is to provide new Wiki macros or Wiki processors. Those are basically the same thing, as they're implemented using the following interface. The difference comes from the invocation syntax used in the Wiki markup, which manifests itself in the `args` parameter of *IWikiMacroProvider.expand_macro()*.

class `trac.wiki.api.`**IWikiMacroProvider**
Bases: *trac.core.Interface*

Augment the Wiki markup with new Wiki macros.

Changed in version 0.12: new Wiki processors can also be added that way.

See also *WikiMacroBase* and wiki/WikiMacros#DevelopingCustomMacros and trac.wiki.api.IWikiMacroProvider extension point.

expand_macro(*formatter*, *name*, *content*, *args=None*)
Called by the formatter when rendering the parsed wiki text.

New in version 0.11.

Changed in version 0.12: added the `args` parameter

Parameters

- **formatter** – the wiki `Formatter` currently processing the wiki markup
- **name** – is the name by which the macro has been called; remember that via *get_macros*, multiple names could be associated to this macros. Note that the macro names are case sensitive.
- **content** – is the content of the macro call. When called using macro syntax (`[[Macro(content)]]`), this is the string contained between parentheses, usually containing macro arguments. When called using wiki processor syntax (`{{{!#Macro ...}}}`), it is the content of the processor block, that is, the text starting on the line following the macro name.
- **args** – will be a dictionary containing the named parameters passed when using the Wiki processor syntax.

  The named parameters can be specified when calling the macro using the wiki processor syntax:

```
{{{#!Macro arg1=value1 arg2="value 2"`
... some content ...
}}}
```

  In this example, `args` will be `{'arg1': 'value1', 'arg2': 'value 2'}` and `content` will be `"... some content ..."`.

  If no named parameters are given like in:

```
{{{#!Macro
...
}}}
```

then `args` will be `{}`. That makes it possible to differentiate the above situation from a call made using the macro syntax:

```
[[Macro(arg1=value1, arg2="value 2", ... some content...)]]
```

in which case `args` will always be `None`. Here `content` will be the `"arg1=value1, arg2="value 2", ... some content..."` string. If like in this example, `content` is expected to contain some arguments and named parameters, one can use the *parse_args* function to conveniently extract them.

get_macro_description(*name*)
Return a tuple of a domain name to translate and plain text description of the macro or only the description with the specified name.

Changed in version 1.0: *get_macro_description* can return a domain to translate the description.

get_macros()
Return an iterable that provides the names of the provided macros.

is_inline(*content*)
Return `True` if the content generated is an inline XHTML element.

New in version 1.0.

The Wiki syntax can also be extended by introducing new markup.

class `trac.wiki.api.`**IWikiSyntaxProvider**
Bases: *trac.core.Interface*

Enrich the Wiki syntax with new markup.

See also wiki:TracDev/IWikiSyntaxProviderExample and trac.wiki.api.IWikiSyntaxProvider extension point.

get_link_resolvers()
Return an iterable over `(namespace, formatter)` tuples.

Each formatter should be a function of the form:

```
def format(formatter, ns, target, label, fullmatch=None):
    pass
```

and should return some HTML fragment. The `label` is already HTML escaped, whereas the `target` is not. The `fullmatch` argument is optional, and is bound to the regexp match object for the link.

get_wiki_syntax()
Return an iterable that provides additional wiki syntax.

Additional wiki syntax correspond to a pair of `(regexp, cb)`, the `regexp` for the additional syntax and the callback `cb` which will be called if there's a match. That function is of the form `cb(formatter, ns, match)`.

The Wiki System

The wiki system provide an access to all the pages.

class trac.wiki.api.**WikiSystem**
Bases: *trac.core.Component*

Wiki system manager.

change_listeners
List of components that implement *IWikiChangeListener*

get_pages(*prefix=None*)
Iterate over the names of existing Wiki pages.

Parameters **prefix** – if given, only names that start with that prefix are included.

has_page(*pagename*)
Whether a page with the specified name exists.

ignore_missing_pages
Enable/disable highlighting CamelCase links to missing pages.

macro_providers
List of components that implement *IWikiMacroProvider*

make_label_from_target(*target*)
Create a label from a wiki target.

A trailing fragment and query string is stripped. Then, leading ./, ../ and / elements are stripped, except when this would lead to an empty label. Finally, if *split_page_names* is true, the label is split accordingly.

pages
Return the names of all existing wiki pages.

render_unsafe_content
Enable/disable the use of unsafe HTML tags such as `<script>` or `<embed>` with the HTML [wiki:WikiProcessors WikiProcessor].

For public sites where anonymous users can edit the wiki it is recommended to leave this option disabled.

resolve_relative_name(*pagename*, *referrer*)
Resolves a pagename relative to a referrer pagename.

safe_origins
List of URIs considered "safe cross-origin", that will be rendered as `img` element without `crossorigin="anonymous"` attribute or used in `url()` of inline style attribute even if `[wiki] render_unsafe_content` is `false` (''since 1.0.15'').

To make any origins safe, specify "*" in the list.

safe_schemes
List of URI schemes considered "safe", that will be rendered as external links even if `[wiki] render_unsafe_content` is `false`.

split_page_names
Enable/disable splitting the WikiPageNames with space characters.

syntax_providers
List of components that implement *IWikiSyntaxProvider*

Other Functions

trac.wiki.api.**parse_args**(*args*, *strict=True*)
Utility for parsing macro "content" and splitting them into arguments.

The content is split along commas, unless they are escaped with a backquote (see example below).

Parameters

- **args** – a string containing macros arguments
- **strict** – if `True`, only Python-like identifiers will be recognized as keyword arguments

Example usage:

```
>>> parse_args('')
([], {})
>>> parse_args('Some text')
(['Some text'], {})
>>> parse_args('Some text, mode= 3, some other arg\, with a comma.')
(['Some text', ' some other arg, with a comma.'], {'mode': ' 3'})
>>> parse_args('milestone=milestone1,status!=closed', strict=False)
([], {'status!': 'closed', 'milestone': 'milestone1'})
```

`trac.wiki.api.`**validate_page_name**(*pagename*)
Utility for validating wiki page name.

Parameters **pagename** – wiki page name to validate

trac.wiki.formatter

`trac.wiki.formatter.`**wiki_to_outline**(*wikitext*, *env*, *db=None*, *absurls=False*, *max_depth=None*, *min_depth=None*, *req=None*)

Deprecated will be removed in 1.0 and replaced by something else

class `trac.wiki.formatter.`**Formatter**(*env*, *context*)
Bases: `object`

Base Wiki formatter.

Parses and formats wiki text, in a given `RenderingContext`.

close_tag(*open_tag*, *close_tag=None*)
Open a inline style tag.

If *close_tag* is not specified, it's an indirect tag (0.12)

open_tag(*tag_open*, *tag_close=None*)
Open an inline style tag.

If `tag_close` is not specified, `tag_open` is an indirect tag (0.12)

replace(*fullmatch*)
Replace one match with its corresponding expansion

simple_tag_handler(*match*, *open_tag*, *close_tag*)
Generic handler for simple binary style tags

tag_open_p(*tag*)
Do we currently have any open tag with `tag` as end-tag?

`trac.wiki.formatter.`**split_url_into_path_query_fragment**(*target*)
Split a target along `?` and `#` in `(path, query, fragment)`.

```
>>> split_url_into_path_query_fragment('http://path?a=1&b=2#frag?ment')
('http://path', '?a=1&b=2', '#frag?ment')
>>> split_url_into_path_query_fragment('http://path#frag?ment')
```

```
('http://path', '', '#frag?ment')
>>> split_url_into_path_query_fragment('http://path?a=1&b=2')
('http://path', '?a=1&b=2', '')
>>> split_url_into_path_query_fragment('http://path')
('http://path', '', '')
```

trac.wiki.formatter.**concat_path_query_fragment**(*path*, *query*, *fragment=None*)
Assemble `path`, `query` and `fragment` into a proper URL.

Can be used to re-assemble an URL decomposed using *split_url_into_path_query_fragment* after modification.

```
>>> concat_path_query_fragment('/wiki/page', '?version=1')
'/wiki/page?version=1'
>>> concat_path_query_fragment('/wiki/page#a', '?version=1', '#b')
'/wiki/page?version=1#b'
>>> concat_path_query_fragment('/wiki/page?version=1#a', '?format=txt')
'/wiki/page?version=1&format=txt#a'
>>> concat_path_query_fragment('/wiki/page?version=1', '&format=txt')
'/wiki/page?version=1&format=txt'
>>> concat_path_query_fragment('/wiki/page?version=1', 'format=txt')
'/wiki/page?version=1&format=txt'
>>> concat_path_query_fragment('/wiki/page?version=1#a', '?format=txt', '#')
'/wiki/page?version=1&format=txt'
```

Components

class trac.wiki.formatter.**HtmlFormatter**(*env*, *context*, *wikidom*)
Bases: object

Format parsed wiki text to HTML

generate(*escape_newlines=False*)
Generate HTML elements.

newlines in the wikidom will be preserved if `escape_newlines` is set.

class trac.wiki.formatter.**InlineHtmlFormatter**(*env*, *context*, *wikidom*)
Bases: object

Format parsed wiki text to inline elements HTML.

Block level content will be discarded or compacted.

generate(*shorten=False*)
Generate HTML inline elements.

If `shorten` is set, the generation will stop once enough characters have been emitted.

class trac.wiki.formatter.**LinkFormatter**(*env*, *context*)
Bases: *trac.wiki.formatter.OutlineFormatter*

Special formatter that focuses on TracLinks.

match(*wikitext*)
Return the Wiki match found at the beginning of the `wikitext`

class trac.wiki.formatter.**OneLinerFormatter**(*env*, *context*)
Bases: *trac.wiki.formatter.Formatter*

A special version of the wiki formatter that only implement a subset of the wiki formatting functions. This version is useful for rendering short wiki-formatted messages on a single line

class trac.wiki.formatter.**OutlineFormatter**(*env*, *context*)
Bases: *trac.wiki.formatter.Formatter*

Special formatter that generates an outline of all the headings.

trac.wiki.intertrac

class trac.wiki.intertrac.**InterTracDispatcher**
Bases: *trac.core.Component*

InterTrac dispatcher.

intertrac_section
This section configures InterTrac prefixes. Option names in this section that contain a are of the format `<name>.`. Option names that don't contain a define an alias.

The *url* attribute is mandatory and is used for locating the other Trac. This can be a relative path when the other Trac environment is located on the same server.

The `title` attribute is used for generating a tooltip when the cursor is hovered over an InterTrac link.

Example configuration: {{{#!ini [intertrac] # – Example of setting up an alias: t = trac

– Link to an external Trac: genshi.title = Edgewall's Trac for Genshi genshi.url = http://genshi.edgewall.org }}}

trac.wiki.interwiki

class trac.wiki.interwiki.**InterWikiMap**
Bases: *trac.core.Component*

InterWiki map manager.

interwiki_map
Map from upper-cased namespaces to (namespace, prefix, title) values.

interwiki_section
Every option in the `[interwiki]` section defines one InterWiki prefix. The option name defines the prefix. The option value defines the URL, optionally followed by a description separated from the URL by whitespace. Parametric URLs are supported as well.

''Example:'' {{{ [interwiki] MeatBall = http://www.usemod.com/cgi-bin/mb.pl? PEP = http://www.python.org/peps/pep-\protect\T1\textdollar1.html Python Enhancement Proposal $1 tsvn = tsvn: Interact with TortoiseSvn }}}

url(*ns*, *target*)
Return `(url, title)` for the given InterWiki `ns`.

Expand the colon-separated `target` arguments.

trac.wiki.macros – The standard set of Wiki macros

The standard set of components corresponding to Wiki macros are not meant to be used directly from the API. You may study their implementation though, for getting inspiration. In particular, you'll see they all subclass the

WikiMacroBase class, which provides a convenient way to implement a new *IWikiMacroProvider* interface.

class trac.wiki.macros.**WikiMacroBase**
Bases: *trac.core.Component*

Abstract base class for wiki macros.

get_macro_description(*name*)
Return the subclass's gettext domain and macro description

get_macros()
Yield the name of the macro based on the class name.

See also wiki/WikiMacros#DevelopingCustomMacros.

trac.wiki.model

class trac.wiki.model.**WikiPage**(*env*, *name=None*, *version=None*)
Bases: object

Represents a wiki page (new or existing).

Create a new page object or retrieves an existing page.

Parameters

- **env** – an Environment object.
- **name** – the page name or a Resource object.
- **version** – the page version. The value takes precedence over the Resource version when both are specified.

delete(*version=None*)
Delete one or all versions of a page.

edit_comment(*new_comment*)
Edit comment of wiki page version in-place.

get_history()
Retrieve the edit history of a wiki page.

Returns a tuple containing the version, datetime, author and comment.

rename(*new_name*)
Rename wiki page in-place, keeping the history intact. Renaming a page this way will eventually leave dangling references to the old page - which literally doesn't exist anymore.

save(*author*, *comment*, *t=None*)
Save a new version of a page.

trac.wiki.model.**datetime_now**()
[tz] -> new datetime with tz's local day and time.

trac.wiki.parser

class trac.wiki.parser.**WikiParser**
Bases: *trac.core.Component*

Wiki text parser.

parse(*wikitext*)
Parse `wikitext` and produce a WikiDOM tree.

`trac.wiki.parser.`**parse_processor_args**(*processor_args*)
Parse a string containing parameter assignments, and return the corresponding dictionary.

Isolated keywords are interpreted as `bool` flags, `False` if the keyword is prefixed with "-", `True` otherwise.

```
>>> parse_processor_args('ab="c de -f gh=ij" -')
{'ab': 'c de -f gh=ij'}
```

```
>>> sorted(parse_processor_args('ab=c de -f gh="ij klmn" p=q-r,s').items())
[('ab', 'c'), ('de', True), ('f', False), ('gh', 'ij klmn'), ('p', 'q-r,s')]
```

```
>>> args = 'data-name=foo-bar data-true -data-false'
>>> sorted(parse_processor_args(args).items())
[('data-false', False), ('data-name', 'foo-bar'), ('data-true', True)]
```

trac.wiki.web_api

class `trac.wiki.web_api.`**WikiRenderer**
Bases: *trac.core.Component*

Wiki text renderer.

trac.wiki.web_ui

class `trac.wiki.web_ui.`**DefaultWikiPolicy**
Bases: *trac.core.Component*

Default permission policy for the wiki system.

Wiki pages with the read-only attribute require `WIKI_ADMIN` to delete, modify or rename the page.

tracopt.perm.authz_policy

class `tracopt.perm.authz_policy.`**AuthzPolicy**
Bases: *trac.core.Component*

Permission policy using an authz-like configuration file.

Refer to SVN documentation for syntax of the authz file. Groups are supported.

As the fine-grained permissions brought by this permission policy are often used in complement of the other permission policies (like the `DefaultPermissionPolicy`), there's no need to redefine all the permissions here. Only additional rights or restrictions should be added.

=== Installation === Enabling this policy requires listing it in `trac.ini`:

```
{{{
[trac]
permission_policies = AuthzPolicy, DefaultPermissionPolicy

[authz_policy]
authz_file = conf/authzpolicy.conf
}}}
```

This means that the *AuthzPolicy* permissions will be checked first, and only if no rule is found will the DefaultPermissionPolicy be used.

=== Configuration === The authzpolicy.conf file is a ini style configuration file.

- Each section of the config is a glob pattern used to match against a Trac resource descriptor. These descriptors are in the form:

```
{{{
<realm>:<id>@<version>[/<realm>:<id>@<version> ...]
}}}
```

  Resources are ordered left to right, from parent to child. If any component is inapplicable, * is substituted. If the version pattern is not specified explicitely, all versions (@*) is added implicitly

  Example: Match the WikiStart page:

```
{{{
[wiki:*]
[wiki:WikiStart*]
[wiki:WikiStart@*]
[wiki:WikiStart]
}}}
```

  Example: Match the attachment wiki:WikiStart@117/attachment/FOO.JPG@* on WikiStart:

```
{{{
[wiki:*]
[wiki:WikiStart*]
[wiki:WikiStart@*]
[wiki:WikiStart@*/attachment/*]
[wiki:WikiStart@117/attachment/FOO.JPG]
}}}
```

- Sections are checked against the current Trac resource ''IN ORDER'' of appearance in the configuration file. ''ORDER IS CRITICAL''.
- Once a section matches, the current username is matched, ''IN ORDER'', against the keys of the section. If a key is prefixed with a @, it is treated as a group. If a key is prefixed with a , the permission is denied rather than granted. The username will match any of 'anonymous', 'authenticated', <username> or '*', using normal Trac permission rules.

Example configuration:

```
{{{
[groups]
administrators = athomas

[*/attachment:*]
* = WIKI_VIEW, TICKET_VIEW

[wiki:WikiStart@*]
@administrators = WIKI_ADMIN
anonymous = WIKI_VIEW
* = WIKI_VIEW

# Deny access to page templates
[wiki:PageTemplates/*]
* =
```

```
# Match everything else
[*]
@administrators = TRAC_ADMIN
anonymous = BROWSER_VIEW, CHANGESET_VIEW, FILE_VIEW, LOG_VIEW,
    MILESTONE_VIEW, POLL_VIEW, REPORT_SQL_VIEW, REPORT_VIEW,
    ROADMAP_VIEW, SEARCH_VIEW, TICKET_CREATE, TICKET_MODIFY,
    TICKET_VIEW, TIMELINE_VIEW,
    WIKI_CREATE, WIKI_MODIFY, WIKI_VIEW
# Give authenticated users some extra permissions
authenticated = REPO_SEARCH, XML_RPC
}}}
```

authz_file
Location of authz policy configuration file. Non-absolute paths are relative to the Environment `conf` directory.

tracopt.perm.config_perm_provider

class tracopt.perm.config_perm_provider.**ExtraPermissionsProvider**
Bases: *trac.core.Component*

Define arbitrary permissions.

Documentation can be found on the [wiki:TracIni#extra-permissions-section] page after enabling the component.

extra_permissions_section
This section provides a way to add arbitrary permissions to a Trac environment. This can be useful for adding new permissions to use for workflow actions, for example.

To add new permissions, create a new section `[extra-permissions]` in your `trac.ini`. Every entry in that section defines a meta-permission and a comma-separated list of permissions. For example: {{{#!ini [extra-permissions] EXTRA_ADMIN = EXTRA_VIEW, EXTRA_MODIFY, EXTRA_DELETE }}} This entry will define three new permissions `EXTRA_VIEW`, `EXTRA_MODIFY` and `EXTRA_DELETE`, as well as a meta-permissions `EXTRA_ADMIN` that grants all three permissions.

The permissions are created in upper-case characters regardless of the casing of the definitions in `trac.ini`. For example, the definition `extra_view` would create the permission `EXTRA_VIEW`.

If you don't want a meta-permission, start the meta-name with an underscore (_): {{{#!ini [extra-permissions] _perms = EXTRA_VIEW, EXTRA_MODIFY }}}

tracopt.ticket.clone

class tracopt.ticket.clone.**TicketCloneButton**
Bases: *trac.core.Component*

Add a ''Clone" button in the ticket box and in ticket comments.

This button is located next to the 'Reply' to description button, and pressing it will send a request for creating a new ticket which will be based on the cloned one.

tracopt.ticket.commit_updater

class tracopt.ticket.commit_updater.**CommitTicketUpdater**
Bases: *trac.core.Component*

Update tickets based on commit messages.

This component hooks into changeset notifications and searches commit messages for text in the form of: {{{ command #1 command #1, #2 command #1 & #2 command #1 and #2 }}}

Instead of the short-hand syntax "#1", "ticket:1" can be used as well, e.g.: {{{ command ticket:1 command ticket:1, ticket:2 command ticket:1 & ticket:2 command ticket:1 and ticket:2 }}}

Using the long-form syntax allows a comment to be included in the reference, e.g.: {{{ command ticket:1#comment:1 command ticket:1#comment:description }}}

In addition, the ':' character can be omitted and issue or bug can be used instead of ticket.

You can have more than one command in a message. The following commands are supported. There is more than one spelling for each command, to make this as user-friendly as possible.

close, closed, closes, fix, fixed, fixes:: The specified tickets are closed, and the commit message is added to them as a comment.

references, refs, addresses, re, see:: The specified tickets are left in their current status, and the commit message is added to them as a comment.

A fairly complicated example of what you can do is with a commit message of:

Changed blah and foo to do this or that. Fixes #10 and #12, and refs #12.

This will close #10 and #12, and add a note to #12.

check_perms
Check that the committer has permission to perform the requested operations on the referenced tickets.

This requires that the user names be the same for Trac and repository operations.

commands_close
Commands that close tickets, as a space-separated list.

commands_refs
Commands that add a reference, as a space-separated list.

If set to the special value `<ALL>`, all tickets referenced by the message will get a reference to the changeset.

envelope
Require commands to be enclosed in an envelope.

Must be empty or contain two characters. For example, if set to `[]`, then commands must be in the form of `[closes #4]`.

make_ticket_comment(*repos*, *changeset*)
Create the ticket comment from the changeset data.

notify
Send ticket change notification when updating a ticket.

tracopt.ticket.commit_updater.**datetime_now**()
[tz] -> new datetime with tz's local day and time.

tracopt.ticket.deleter

class tracopt.ticket.deleter.**TicketDeleter**
Bases: *trac.core.Component*

Ticket and ticket comment deleter.

This component allows deleting ticket comments and complete tickets. For users having TICKET_ADMIN permission, it adds a "Delete" button next to each "Reply" button on the page. The button in the ticket description requests deletion of the complete ticket, and the buttons in the change history request deletion of a single comment.

''Comment and ticket deletion are irreversible (and therefore ''dangerous'') operations.'' For that reason, a confirmation step is requested. The confirmation page shows the ticket box (in the case of a ticket deletion) or the ticket change (in the case of a comment deletion).

tracopt.versioncontrol.git.PyGIT

class tracopt.versioncontrol.git.PyGIT.**Storage**(*git_dir*, *log*, *git_bin='git'*, *git_fs_encoding=None*, *rev_cache=None*)

Bases: object

High-level wrapper around GitCore with in-memory caching

Initialize PyGit.Storage instance

git_dir: path to .git folder; this setting is not affected by the git_fs_encoding setting

log: logger instance

git_bin: path to executable this setting is not affected by the git_fs_encoding setting

git_fs_encoding: encoding used for paths stored in git repository; if None, no implicit decoding/encoding to/from unicode objects is performed, and bytestrings are returned instead

children_recursive(*sha*, *rev_dict=None*)
Recursively traverse children in breadth-first order

diff_tree(*tree1*, *tree2*, *path=''*, *find_renames=False*)
calls git diff-tree and returns tuples of the kind (mode1,mode2,obj1,obj2,action,path1,path2)

fullrev(*srev*)
try to reverse shortrev()

get_branch_contains(*sha*, *resolve=False*)
return list of reachable head sha ids or (names, sha) pairs if resolve is true

see also get_branches()

get_branches()
returns list of (local) branches, with active (= HEAD) one being the first item

head()
get current HEAD commit id

rev_cache
Retrieve revision cache

may rebuild cache on the fly if required

returns RevCache tuple

rev_is_anchestor_of(*rev1*, *rev2*)
return True if rev2 is successor of rev1

shortrev(*rev*, *min_len=7*)
try to shorten sha id

verifyrev(*rev*)
verify/lookup given revision object and return a sha id or None if lookup failed

tracopt.versioncontrol.git.PyGIT.**parse_commit**(*raw*)
Parse the raw content of a commit (as given by `git cat-file -p`).

Return the commit message and a dict of properties.

class tracopt.versioncontrol.git.PyGIT.**GitCore**(*git_dir=None*, *git_bin='git'*, *log=None*, *fs_encoding=None*)
Bases: object

Low-level wrapper around git executable

classmethod is_sha(*sha*)
returns whether sha is a potential sha id (i.e. proper hexstring between 4 and 40 characters)

class tracopt.versioncontrol.git.PyGIT.**SizedDict**(*max_size=0*)
Bases: dict

Size-bounded dictionary with FIFO replacement strategy

tracopt.versioncontrol.git.git_fs

class tracopt.versioncontrol.git.git_fs.**GitCachedChangeset**(*repos*, *rev*, *env*)
Bases: trac.versioncontrol.cache.CachedChangeset

Git-specific cached changeset.

class tracopt.versioncontrol.git.git_fs.**GitCachedRepository**(*env*, *repos*, *log*)
Bases: trac.versioncontrol.cache.CachedRepository

Git-specific cached repository.

class tracopt.versioncontrol.git.git_fs.**GitChangeset**(*repos*, *sha*)
Bases: *trac.versioncontrol.api.Changeset*

A Git changeset in the Git repository.

Corresponds to a Git commit blob.

class tracopt.versioncontrol.git.git_fs.**GitRepository**(*env*, *path*, *params*, *log*, *persistent_cache=False*, *git_bin='git'*, *git_fs_encoding='utf-8'*, *shortrev_len=7*, *rlookup_uid=<function <lambda>>*, *use_committer_id=False*, *use_committer_time=False*)
Bases: *trac.versioncontrol.api.Repository*

Git repository

get_changeset(*rev*)
GitChangeset factory method

tracopt.versioncontrol.git.git_fs.**intersperse**(*sep*, *iterable*)
The 'intersperse' generator takes an element and an iterable and intersperses that element between the elements of the iterable.

inspired by Haskell's `Data.List.intersperse`

tracopt.versioncontrol.svn.svn_fs – Subversion backend for Trac

This module can be considered to be private. However, it can serve as an example implementation of a version control backend.

Speaking of Subversion, we use its `svn.fs` layer mainly, which means we need direct (read) access to the repository content.

Though there's no documentation for the Python API per se, the doxygen documentation for the C libraries are usually enough. Another possible source of inspiration are the examples and the helper classes in the bindings themselves.

Note about Unicode

The Subversion bindings are not unicode-aware and they expect to receive UTF-8 encoded `string` parameters,

On the other hand, all paths manipulated by Trac are `unicode` objects.

Therefore:

- before being handed out to SVN, the Trac paths have to be encoded to UTF-8, using `_to_svn()`
- before being handed out to Trac, a SVN path has to be decoded from UTF-8, using `_from_svn()`

Whenever a value has to be stored as utf8, we explicitly mark the variable name with "_utf8", in order to avoid any possible confusion.

Warning: *SubversionNode.get_content()* returns an object from which one can read a stream of bytes. NO guarantees can be given about what that stream of bytes represents. It might be some text, encoded in some way or another. SVN properties *might* give some hints about the content, but they actually only reflect the beliefs of whomever set those properties...

class tracopt.versioncontrol.svn.svn_fs.**Pool**(*parent_pool=None*)
Bases: `object`

A Pythonic memory pool object

Create a new memory pool

assert_valid()
Assert that this memory_pool is still valid.

clear()
Clear embedded memory pool. Invalidate all subpools.

destroy()
Destroy embedded memory pool. If you do not destroy the memory pool manually, Python will destroy it automatically.

valid()
Check whether this memory pool and its parents are still valid

class tracopt.versioncontrol.svn.svn_fs.**SubversionRepository**(*path*, *params*, *log*)
Bases: *trac.versioncontrol.api.Repository*

Repository implementation based on the svn.fs API.

clear(*youngest_rev=None*)
Reset notion of `youngest` and `oldest`

close()
Dispose of low-level resources associated to this repository.

get_base()
Retrieve the base path corresponding to the Subversion repository itself.

This is the same as the `path` property minus the intra-repository scope, if one was specified.

get_changes(*old_path*, *old_rev*, *new_path*, *new_rev*, *ignore_ancestry=0*)
Determine differences between two arbitrary pairs of paths and revisions.

(wraps `repos.svn_repos_dir_delta`)

get_changeset(*rev*)
Produce a *SubversionChangeset* from given revision specification

get_changeset_uid(*rev*)
Build a value identifying the `rev` in this repository.

get_node(*path*, *rev=None*)
Produce a *SubversionNode* from given path and optionally revision specifications. No revision given means use the latest.

get_oldest_rev()
Gives an approximation of the oldest revision.

get_path_history(*path*, *rev=None*, *limit=None*)
Retrieve creation and deletion events that happened on given `path`.

get_path_url(*path*, *rev*)
Retrieve the "native" URL from which this repository is reachable from Subversion clients.

get_quickjump_entries(*rev*)
Retrieve known branches, as (name, id) pairs.

Purposedly ignores `rev` and always takes the last revision.

get_youngest_rev()
Retrieve the latest revision in the repository.

(wraps `fs.youngest_rev`)

has_node(*path*, *rev=None*, *pool=None*)
Check if `path` exists at `rev` (or latest if unspecified)

next_rev(*rev*, *path=''*, *find_initial_rev=False*)
Return revision immediately following `rev`, eventually below given `path` or globally.

normalize_path(*path*)
Take any path specification and produce a path suitable for the rest of the API

normalize_rev(*rev*)
Take any revision specification and produce a revision suitable for the rest of the API

previous_rev(*rev*, *path=''*)
Return revision immediately preceeding `rev`, eventually below given `path` or globally.

rev_older_than(*rev1*, *rev2*)
Check relative order between two revision specifications.

class `tracopt.versioncontrol.svn.svn_fs.`**SvnCachedRepository**(*env*, *repos*, *log*)
Bases: `trac.versioncontrol.cache.CachedRepository`

Subversion-specific cached repository, zero-pads revision numbers in the cache tables.

Components

class tracopt.versioncontrol.svn.svn_fs.**SubversionConnector**
Bases: *trac.core.Component*

branches
Comma separated list of paths categorized as branches. If a path ends with '*', then all the directory entries found below that path will be included. Example: `/trunk, /branches/*, /projectAlpha/trunk, /sandbox/*`

eol_style
End-of-Line character sequences when `svn:eol-style` property is `native`.

If `native`, substitute with the native EOL marker on the server. Otherwise, if `LF`, `CRLF` or `CR`, substitute with the specified EOL marker.

(''since 1.0.2'')

get_repository(*type*, *dir*, *params*)
Return a *SubversionRepository*.

The repository is wrapped in a `CachedRepository`, unless `type` is 'direct-svnfs'.

tags
Comma separated list of paths categorized as tags.

If a path ends with '*', then all the directory entries found below that path will be included. Example: `/tags/*, /projectAlpha/tags/A-1.0, /projectAlpha/tags/A-v1.1`

Concrete classes

class tracopt.versioncontrol.svn.svn_fs.**SubversionRepository**(*path*, *params*, *log*)
Bases: *trac.versioncontrol.api.Repository*

Repository implementation based on the svn.fs API.

clear(*youngest_rev=None*)
Reset notion of `youngest` and `oldest`

close()
Dispose of low-level resources associated to this repository.

get_base()
Retrieve the base path corresponding to the Subversion repository itself.

This is the same as the `path` property minus the intra-repository scope, if one was specified.

get_changes(*old_path*, *old_rev*, *new_path*, *new_rev*, *ignore_ancestry=0*)
Determine differences between two arbitrary pairs of paths and revisions.

(wraps `repos.svn_repos_dir_delta`)

get_changeset(*rev*)
Produce a *SubversionChangeset* from given revision specification

get_changeset_uid(*rev*)
Build a value identifying the `rev` in this repository.

get_node(*path*, *rev=None*)
Produce a *SubversionNode* from given path and optionally revision specifications. No revision given means use the latest.

get_oldest_rev()
Gives an approximation of the oldest revision.

get_path_history(*path*, *rev=None*, *limit=None*)
Retrieve creation and deletion events that happened on given `path`.

get_path_url(*path*, *rev*)
Retrieve the "native" URL from which this repository is reachable from Subversion clients.

get_quickjump_entries(*rev*)
Retrieve known branches, as (name, id) pairs.

Purposedly ignores `rev` and always takes the last revision.

get_youngest_rev()
Retrieve the latest revision in the repository.

(wraps `fs.youngest_rev`)

has_node(*path*, *rev=None*, *pool=None*)
Check if `path` exists at `rev` (or latest if unspecified)

next_rev(*rev*, *path=''*, *find_initial_rev=False*)
Return revision immediately following `rev`, eventually below given `path` or globally.

normalize_path(*path*)
Take any path specification and produce a path suitable for the rest of the API

normalize_rev(*rev*)
Take any revision specification and produce a revision suitable for the rest of the API

previous_rev(*rev*, *path=''*)
Return revision immediately preceeding `rev`, eventually below given `path` or globally.

rev_older_than(*rev1*, *rev2*)
Check relative order between two revision specifications.

class `tracopt.versioncontrol.svn.svn_fs.`**SubversionNode**(*path*, *rev*, *repos*, *pool=None*, *parent_root=None*)
Bases: *trac.versioncontrol.api.Node*

get_annotations()
Return a list the last changed revision for each line. (wraps `client.blame2`)

get_branch_origin()
Return the revision in which the node's path was created.

(wraps `fs.revision_root_revision(fs.closest_copy)`)

get_content()
Retrieve raw content as a "read()"able object.

get_content_length()
Retrieve byte size of a file.

Return `None` for a folder. (wraps `fs.file_length`)

get_content_type()
Retrieve mime-type property of a file.

Return `None` for a folder. (wraps `fs.revision_prop`)

get_copy_ancestry()
Retrieve the list of `(path,rev)` copy ancestors of this node. Most recent ancestor first. Each ancestor

(path, rev) corresponds to the path and revision of the source at the time the copy or move operation was performed.

get_entries()
Yield *SubversionNode* corresponding to entries in this directory.

(wraps fs.dir_entries)

get_history(*limit=None*)
Yield change events that happened on this path

get_last_modified()
Retrieve timestamp of last modification, in micro-seconds.

(wraps fs.revision_prop)

get_processed_content(*keyword_substitution=True*, *eol_hint=None*)
Retrieve processed content as a "read()"able object.

get_properties()
Return dict of node properties at current revision.

(wraps fs.node_proplist)

class tracopt.versioncontrol.svn.svn_fs.**SubversionChangeset**(*repos*, *rev*, *scope*, *pool=None*)
Bases: *trac.versioncontrol.api.Changeset*

get_changes()
Retrieve file changes for a given revision.

(wraps repos.svn_repos_replay)

get_properties()
Retrieve dict of Subversion properties for this revision (revprops)

Miscellaneous

class tracopt.versioncontrol.svn.svn_fs.**Pool**(*parent_pool=None*)
Bases: object

A Pythonic memory pool object

Create a new memory pool

assert_valid()
Assert that this memory_pool is still valid.

clear()
Clear embedded memory pool. Invalidate all subpools.

destroy()
Destroy embedded memory pool. If you do not destroy the memory pool manually, Python will destroy it automatically.

valid()
Check whether this memory pool and its parents are still valid

class tracopt.versioncontrol.svn.svn_fs.**SvnCachedRepository**(*env*, *repos*, *log*)
Bases: trac.versioncontrol.cache.CachedRepository

Subversion-specific cached repository, zero-pads revision numbers in the cache tables.

tracopt.versioncontrol.svn.svn_prop

Testing in Trac

So, you'd like to make sure Trac works in your configuration. Excellent. Here's what you need to know.

Running the tests

Prerequisites

Beyond the standard installation prereqs, you also need:

- Pygments (0.8+)
- Twill (0.9+)
- Coverage (for coverage)
- Figleaf (0.6.1+, alternative for coverage)

Additionally, if you're on Windows, you need to get fcrypt. See *Prerequisites on Windows* below for more information.

Invoking the tests

Just run **`make test`** in the Trac tree once you have everything installed. This will run the unit tests first, then the functional tests (if you have the dependencies) against SQLite. On a reasonably fast machine, the former takes 10 seconds and the latter a couple of minutes.

A few environment variables will influence the way tests are executed:

`TRAC_TEST_DB_URI`

Use another database backend than the default in-memory SQLite database. See *Using an alternate database backend* for more.

`TRAC_TEST_TRACD_OPTIONS`

Provide additional options to the standalone **`tracd`** server used for the functional tests.

`TRAC_TEST_ENV_PATH`

Use the specified path for the test environment directory.

`TRAC_TEST_PORT`

Use the specified port for running the standalone **`tracd`** server.

The `Makefile` is actually written in a way that allow you to get more control, if you want.

Other possible usages:

```
make test=trac/tests/allwiki.py # run all the Wiki formatter tests

make unit-test db=postgres # run only the unit tests with PostgreSQL

make functional-test db=mysql # run only the functional tests with MySQL

make test python=24 # run all the tests using Python 2.4
```

If you're running the tests on Windows and don't have cygwin, you'll need to manually run the tests using **`python trac\test.py`**, but this will run all the tests interleaved.

Understanding failures

Functional test failures can happen a few different ways.

Running trac-admin fails every time Make sure the prereqs are met. In particular, that new enough Genshi is available and has **`python setup.py egg_info`** run.

Repo creation fails Subversion is required for the tests; they are not designed to run without it.

Repo creation works, other repo access fails Probably a mismatch in svn bindings versus the **`svn`** binary.

Twill errors which save to HTML Check the html and see if there's a traceback contained in it. Chances are it has an obvious traceback with an error – these are triggered on the server, not the tester, so they're difficult for us to show in the failure itself.

If you can't decipher what the problem is from viewing the HTML, run the server manually and see what state that particular page is in.

Random weird platform issues Please report them.

Can't remove files on Windows Ugh. Please report them.

Reload tests fail Chances are, you're on a Windows VM that has an unstable clock and FAT32 filesystem (which has a granularity of several seconds). If that's not the case, report it.

Coverage doesn't work with functional tests Know issue, patches welcome...

Prerequisites on Windows

- You have to install fcrypt
- You may install pywin32 (optional, improve `subprocess` performance)

Writing Tests for Core

Where tests belong

If it's a regression, it belongs in `trac/tests/functional/testcases.py` for now. Module-specific tests generally already have a `trac/$MOD/tests/functional.py` which you can add to. The environment is brought up as few times as possible, for speed, and your test order is guaranteed to be run in the order it's added to the suite, at the end of the file.

Using Twill

The definitive guide for Twill commands is the Command Reference, but 90% of what you need is contained by convenience methods in `FunctionalTester.go_to_*()` and the following few commands:

tc.find Looks for a regex on the current page. If it isn't found, raise an exception.

Example:

```
tc.find("\bPreferences\b")
```

tc.notfind Like find, but raises an exception if it *is* there.

Example:

```
tc.find(r"\bPreferences\b")
```

tc.follow Find a link matching the regex, and simulate clicking it.

Example:

```
tc.follow("Login")
```

tc.fv Short for `formvalue`, fill in a field.

Example:

```
tc.fv("searchform", "q", "ponies")
```

tc.submit Submit the active form.

Example:

```
tc.submit()
```

Example

This is how you might construct a test that verifies that admin users can see detailed version information.

Start with the navigation. You shouldn't rely on the browser being in any specific state, so begin with `FunctionalTester.go_to_*()`.

```
def test_about_page(self):
    self._tester.logout()        # Begin by logging out.
    self._tester.go_to_front()   # The homepage has a link we want
    tc.follow("About")           # Follow the link with "About" in it

    tc.find("Trac is a web-based software")
    tc.notfind("Version Info")

    self._tester.login("admin")
    self._tester.go_to_front()
    tc.follow("About")

    tc.find("Trac is a web-based software")
    tc.find("Version Info")
```

Test Environment Helpers

Functional Test Environment

Functional Tester

Using an alternate database backend

The unit tests don't really touch the db. The functional tests will, however, but if you're not using sqlite you need to setup the database yourself. Once it's set up, just set *TRAC_TEST_DB_URI* to the connection string you would use

for an **trac-admin inittenv** and run the tests.

Postgres

Testing against Postgres requires you to setup a postgres database and user for testing, then setting an environment variable. The test scripts will create a schema within the database, and on consecutive runs remove the schema.

> **Warning:** Do not run this against a live Trac db schema, the schema *will* be removed if it exists.

On OS X and Linux, you can run the following to create the test database:

```
$ sudo -u postgres createuser -S -D -r -P -e tracuser
$ sudo -u postgres createdb -O tracuser trac
```

Windows:

```
> createuser -U postgres -S -D -r -P -e tracuser
> createdb -U postgres -O tracuser trac
```

Prior to running the tests, set the *TRAC_TEST_DB_URI* variable. If you do not include a schema in the URI, the schema `tractest` will be used.

OS X and Linux:

```
$ export TRAC_TEST_DB_URI=postgres://tracuser:password@localhost:5432/trac?
↪schema=tractest
$ make test
```

Windows:

```
set TRAC_TEST_DB_URI=postgres://tracuser:password@localhost:5432/trac?schema=tractest
```

Finally, run the tests as usual. Note that if you have already a test environment set up from a previous run, thesettings in testenv/trac/conf/trac.ini will be used. In particular, they will take precedence over the TRAC_TEST_DB_URI variable. Simply edit that trac.ini file or even remove the whole testenv folder if this gets in the way.

If in some cases the tests go wrong and you can't run the tests again because the schema is already there, you can drop the schema manually like this:

OS X and Linux:

```
> echo 'drop schema "tractest" cascade' | psql trac tracuser
```

Windows:

```
> echo drop schema "tractest" cascade | psql trac tracuser
```

If you later want to remove the test user and database, use the following:

On OS X and Linux, you can run the following to create the test database:

```
$ sudo -u postgres dropdb tractest
$ sudo -u postgres dropuser tractest
```

Windows:

```
> dropdb -U postgres trac
> dropuser -U postgres tracuser
```

MySQL

Create the database and user as you normally would. See the MySqlDb page for more information.

Example:

```
$ mysql -u root
CREATE DATABASE trac DEFAULT CHARACTER SET utf8 COLLATE utf8_bin;
CREATE USER tracuser IDENTIFIED BY 'password';
GRANT ALL ON trac.* TO tracuser;
FLUSH PRIVILEGES;
^D
$ export TRAC_TEST_DB_URI=mysql://tracuser:password@localhost/trac
$ make test
...
$ mysql -u root
DROP DATABASE trac
DROP USER tracuser
^D
```

If you have better ideas on automating this, please contact us.

Troubleshooting

If you hit the following error message:

```
trac.core.TracError: The Trac Environment needs to be upgraded.
```

This is because the test environment clean-up stopped half-way: the testenv/trac environment is still there, but the testenv/trac/conf/trac.ini file has already been removed. The default ticket workflow then requests an environment upgrade. Simply remove manually the whole testenv folder and, when using Postgres, remove the tractest schema manually as explained above.

Writing Tests for Plugins

Testing a VCS backend

You'll need to make several subclasses to get this working in the current test infrastructure. But first, we start with some imports. These are pretty much required for all plugin tests:

```
from trac.tests.functional import (FunctionalTestSuite,
                                   FunctionalTestCaseSetup,
                                   FunctionalTwillTestCaseSetup, tc)
from trac.tests.functional import testenv
```

Now subclass `FunctionalTestEnvironment`. This allows you to override methods that you need to set up your repo instead of the default Subversion one:

```
class GitFunctionalTestEnvironment(testenv.FunctionalTestEnvironment):
    repotype = 'git'

    def create_repo(self):
        os.mkdir(self.repodir)
        self.call_in_repo(["git", "init"])
        self.call_in_repo(["git", "config", "user.name", "Test User"])
        self.call_in_repo(["git", "config", "user.email", "test@example.com"])

    def get_enabled_components(self):
        return ['tracext.git.*']

    def get_repourl(self):
        return self.repodir + '/.git'
    repourl = property(get_repourl)
```

Now you need a bit of glue that sets up a test suite specifically for your plugin's repo type. Any testcases within this test suite will use the same environment. No other changes are generally necessary on the test suite:

```
class GitFunctionalTestSuite(FunctionalTestSuite):
    env_class = GitFunctionalTestEnvironment
```

Your test cases can call functions on either the *tester* or *twill commands* to do their job. Here's one that just verifies we were able to sync the repo without issue:

```
class EmptyRepoTestCase(FunctionalTwillTestCaseSetup):
    def runTest(self):
        self._tester.go_to_timeline()
        tc.notfind('Unsupported version control system')
```

Lastly, there's some boilerplate needed for the end of your test file, so it can be run from the command line:

```
def suite():
    # Here you need to create an instance of your subclass
    suite = GitFunctionalTestSuite()
    suite.addTest(EmptyRepoTestCase())
    # ...
    suite.addTest(AnotherRepoTestCase())
    return suite

if __name__ == '__main__':
    unittest.main(defaultTest='suite')
```

Todo

write more, with testing-specific steps.

Glossary

VCS Version Control System, what you use for versioning your source code

Documentation TODO

Todo

write more, with testing-specific steps.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/trac/checkouts/latest/doc/dev/testing.rst, line 16.)

CHAPTER 2

Indices and tables

- General Index
- modindex
- search
- *Glossary*
- *Documentation TODO*

Python Module Index

V

W

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

S

T

U

V

W

X