Table of Contents

Table of Contents

Table of Contents

Foswiki::Func

package Foswiki::Func

Interface for Foswiki extensions developers

This module defines the main interfaces that extensions can use to interact with the Foswiki engine and content.

Refer to `lib/Foswiki/Plugins/EmptyPlugin.pm` for a template Plugin and starter documentation on how to write a Plugin.

Plugins should **only** call methods in packages documented in DevelopingPlugins. If you use functions in other Foswiki libraries you risk creating a security hole, and you will probably need to change your plugin when you upgrade Foswiki.

On this page:

        - eachGroupMember($group) -> $iterator
        - checkAccessPermission($type, $id, $text, $topic, $web, $meta) -> $boolean
- Webs, Topics and Attachments
        - getListOfWebs($filter [, $web]) -> @webs
        - webExists($web) -> $boolean
        - createWeb($newWeb, $baseWeb, $opts)
        - moveWeb($oldName, $newName)
        - eachChangeSince($web, $time) -> $iterator
        - getTopicList($web) -> @topics
        - topicExists($web, $topic) -> $boolean
        - checkTopicEditLock($web, $topic, $script) -> ($oopsUrl, $loginName, $unlockTime)
        - setTopicEditLock($web, $topic, $lock)
        - saveTopic($web, $topic, $meta, $text, $options)
        - saveTopicText($web, $topic, $text, $ignorePermissions, $dontNotify) -> $oopsUrl
        - moveTopic($web, $topic, $newWeb, $newTopic)
        - getRevisionInfo($web, $topic, $rev, $attachment) -> ($date, $user, $rev, $comment)
        - getRevisionAtTime($web, $topic, $time) -> $rev
        - readTopic($web, $topic, $rev) -> ($meta, $text)
        - readTopicText($web, $topic, $rev, $ignorePermissions) -> $text
        - attachmentExists($web, $topic, $attachment) -> $boolean
        - readAttachment($web, $topic, $name, $rev) -> $data
        - saveAttachment($web, $topic, $attachment, \%opts)
        - moveAttachment($web, $topic, $attachment, $newWeb, $newTopic, $newAttachment)
- Assembling Pages
        - readTemplate($name, $skin) -> $text
        - loadTemplate ($name, $skin, $web) -> $text
        - expandTemplate($def) -> $string
        - writeHeader()
        - redirectCgiQuery($query, $url, $passthru)
        - addToHEAD($id, $header, $requires)
        - expandCommonVariables($text, $topic, $web, $meta) -> $text
        - renderText($text, $web) -> $text
        - internalLink($pre, $web, $topic, $label, $anchor, $createLink) -> $text
- E-mail
        - sendEmail ($text, $retries) -> $error
- Creating New Topics
        - expandVariablesOnTopicCreation ($text) -> $text
- Special handlers
        - registerTagHandler($var, \&fn, $syntax)
        - registerRESTHandler($alias, \&fn, %options)
                - Example
                - Calling REST handlers from the command-line
        - decodeFormatTokens($str) -> $unencodedString
- Searching
        - searchInWebContent($searchString, $web, \@topics, \%options) -> \%map
- Plugin-specific file handling
        - getWorkArea($pluginName) -> $directorypath
        - readFile($filename) -> $text
        - saveFile($filename, $text)
- General Utilities
        - normalizeWebTopicName($web, $topic) -> ($web, $topic)
        - StaticMethod sanitizeAttachmentName($fname) -> ($fileName, $origName)
        - spaceOutWikiWord($word, $sep) -> $text
        - writeWarning($text)

        - writeDebug($text)
        - isTrue($value, $default) -> $boolean
        - isValidWikiWord ($text) -> $boolean
        - isValidWebName($name [, $system]) -> $boolean
    - StaticMethod isValidTopicName($name [, $allowNonWW]) -> $boolean
        - extractParameters($attr) -> %params
        - extractNameValuePair($attr, $name) -> $value
    - Deprecated functions
        - getRegularExpression($name) -> $expr
        - getScriptUrlPath( ) -> $path
        - getWikiToolName( ) -> $name
        - getMainWebname( ) -> $name
        - getTwikiWebname( ) -> $name
        - getOopsUrl($web, $topic, $template, $param1, $param2, $param3, $param4) -> $url
        - wikiToEmail($wikiName) -> $email
        - permissionsSet($web) -> $boolean
        - getPublicWebList( ) -> @webs
        - formatTime($time, $format, $timezone) -> $text
        - formatGmTime($time, $format) -> $text
        - getDataDir( ) -> $dir
        - getPubDir( ) -> $dir

API version $Date: 2009-12-02 22:28:39 +0100 (Wed, 02 Dec 2009) $ (revision $Rev: 6075 (2010-01-17) $)

Since *date* indicates where functions or parameters have been added since the baseline of the API (TWiki release 4.2.3). The *date* indicates the earliest date of a Foswiki release that will support that function or parameter.

Deprecated *date* indicates where a function or parameters has been deprecated. Deprecated functions will still work, though they should *not* be called in new plugins and should be replaced in older plugins as soon as possible. Deprecated parameters are simply ignored in Foswiki releases after *date*.

Until *date* indicates where a function or parameter has been removed. The *date* indicates the latest date at which Foswiki releases still supported the function or parameter.

Environment

getSkin( ) -> $skin

Get the skin path, set by the `SKIN` and `COVER` preferences variables or the `skin` and `cover` CGI parameters

Return: `$skin` Comma-separated list of skins, e.g. `'gnu,tartan'`. Empty string if none.

getUrlHost( ) -> $host

Get protocol, domain and optional port of script URL

Return: `$host` URL host, e.g. `"http://example.com:80"`

getScriptUrl($web, $topic, $script, ...) -> $url

Compose fully qualified URL

- `$web` - Web name, e.g. `'Main'`
- `$topic` - Topic name, e.g. `'WebNotify'`
- `$script` - Script name, e.g. `'view'`
- `...` - an arbitrary number of name=>value parameter pairs that will be url-encoded and added to the url. The special parameter name '#' is reserved for specifying an anchor. e.g. `getScriptUrl('x','y','view','#'=>'XXX',a=>1,b=>2)` will give `.../view/x/y?a=1&b=2#XXX`

Return: `$url` URL, e.g. `"http://example.com:80/cgi-bin/view.pl/Main/WebNotify"`

getViewUrl($web, $topic) -> $url

Compose fully qualified view URL

- `$web` - Web name, e.g. `'Main'`. The current web is taken if empty
- `$topic` - Topic name, e.g. `'WebNotify'`

Return: `$url` URL, e.g. `"http://example.com:80/cgi-bin/view.pl/Main/WebNotify"`

getPubUrlPath( ) -> $path

Get pub URL path

Return: `$path` URL path of pub directory, e.g. `"/pub"`

getExternalResource($url) -> $response

Get whatever is at the other end of a URL (using an HTTP GET request). Will only work for encrypted protocols such as `https` if the `LWP` CPAN module is installed.

Note that the `$url` may have an optional user and password, as specified by the relevant RFC. Any proxy set in `configure` is honoured.

The `$response` is an object that is known to implement the following subset of the methods of `LWP::Response`. It may in fact be an `LWP::Response` object, but it may also not be if `LWP` is not available, so callers may only assume the following subset of methods is available:

`code()`
`message()`
`header($field)`
`content()`
`is_error()`
`is_redirect()`

Note that if LWP is **not** available, this function:

1. can only really be trusted for HTTP/1.0 urls. If HTTP/1.1 or another protocol is required, you are **strongly** recommended to `require LWP`.
2. Will not parse multipart content

In the event of the server returning an error, then `is_error()` will return true, `code()` will return a valid HTTP status code as specified in RFC 2616 and RFC 2518, and `message()` will return the message that was received from the server. In the event of a client-side error (e.g. an unparseable URL) then `is_error()` will return true and `message()` will return an explanatory message. `code()` will return 400 (BAD

REQUEST).

Note: Callers can easily check the availability of other HTTP::Response methods as follows:

```
my $response = Foswiki::Func::getExternalResource($url);
if (!$response->is_error() && $response->isa('HTTP::Response')) {
    ... other methods of HTTP::Response may be called
} else {
    ... only the methods listed above may be called
}
```

getCgiQuery( ) -> $query

Get CGI query object. Important: Plugins cannot assume that scripts run under CGI, Plugins must always test if the CGI query object is set

Return: `$query` CGI query object; or 0 if script is called as a shell script

getSessionKeys() -> @keys

Get a list of all the names of session variables. The list is unsorted.

Session keys are stored and retrieved using `setSessionValue` and `getSessionValue`.

getSessionValue($key) -> $value

Get a session value from the client session module

- `$key` - Session key

Return: `$value` Value associated with key; empty string if not set

setSessionValue($key, $value) -> $boolean

Set a session value.

- `$key` - Session key
- `$value` - Value associated with key

Return: true if function succeeded

clearSessionValue($key) -> $boolean

Clear a session value that was set using `setSessionValue`.

- `$key` - name of value stored in session to be cleared. Note that you **cannot** clear `AUTHUSER`.

Return: true if the session value was cleared

getContext() -> \%hash

Get a hash of context identifiers representing the currently active context.

The context is a set of identifiers that are set during specific phases of processing. For example, each of the standard scripts in the 'bin' directory each has a context identifier - the view script has 'view', the edit script has 'edit' etc. So you can easily tell what 'type' of script your Plugin is being called within. The core context identifiers are listed in the IfStatements topic. Please be careful not to overwrite any of these identifiers!

Context identifiers can be used to communicate between Plugins, and between Plugins and templates. For example, in FirstPlugin?.pm, you might write:

```
sub initPlugin {
   Foswiki::Func::getContext()->{'MyID'} = 1;
   ...
```

This can be used in SecondPlugin.pm like this:

```
sub initPlugin {
   if( Foswiki::Func::getContext()->{'MyID'} ) {
      ...
   }
   ...
```

or in a template, like this:

```
%TMPL:DEF{"ON"}% Not off %TMPL:END%
%TMPL:DEF{"OFF"}% Not on %TMPL:END%
%TMPL:P{context="MyID" then="ON" else="OFF"}%
```

or in a topic:

```
%IF{"context MyID" then="MyID is ON" else="MyID is OFF"}%
```

Note: **all** plugins have an **automatically generated** context identifier if they are installed and initialised. For example, if the FirstPlugin? is working, the context ID 'FirstPlugin' will be set.

pushTopicContext($web, $topic)

- `$web` - new web
- `$topic` - new topic

Change the Foswiki context so it behaves as if it was processing `$web.$topic` from now on. All the preferences will be reset to those of the new topic. Note that if the new topic is not readable by the logged in user due to access control considerations, there will **not** be an exception. It is the duty of the caller to check access permissions before changing the topic.

It is the duty of the caller to restore the original context by calling `popTopicContext`.

Note that this call does **not** re-initialise plugins, so if you have used global variables to remember the web and topic in `initPlugin`, then those values will be unchanged.

popTopicContext()

Returns the Foswiki context to the state it was in before the `pushTopicContext` was called.

Preferences

getPreferencesValue($key, $web) -> $value

Get a preferences value for the currently requested context, from the currently request topic, its web and the site.

- `$key` - Preference name
- `$web` - Name of web, optional. if defined, we shortcircuit to the WebPreferences (and its Sitewide defaults)

Return: `$value` Preferences value; empty string if not set

- Example for preferences setting:
  - WebPreferences topic has: `* Set WEBBGCOLOR = #FFFFC0`
  - `my $webColor = Foswiki::Func::getPreferencesValue( 'WEBBGCOLOR', 'Sandbox' );`

- Example for MyPlugin? setting:
  - if the MyPlugin? topic has: `* Set COLOR = red`
  - Use `"MYPLUGIN_COLOR"` for `$key`
  - `my $color = Foswiki::Func::getPreferencesValue( "MYPLUGIN_COLOR" );`

NOTE: If `$NO_PREFS_IN_TOPIC` is enabled in the plugin, then preferences set in the plugin topic will be ignored.

getPluginPreferencesValue($key) -> $value

Get a preferences value from your Plugin

- `$key` - Plugin Preferences key w/o PLUGINNAME_ prefix.

Return: `$value` Preferences value; empty string if not set

Note: This function will will **only** work when called from the Plugin.pm file itself. it **will not work** if called from a sub-package (e.g. Foswiki::Plugins::MyPlugin::MyModule)

NOTE: If `$NO_PREFS_IN_TOPIC` is enabled in the plugin, then preferences set in the plugin topic will be ignored.

getPreferencesFlag($key, $web) -> $value

Get a preferences flag from Foswiki or from a Plugin

- `$key` - Preferences key
- `$web` - Name of web, optional. Current web if not specified; does not apply to settings of Plugin topics

Return: `$value` Preferences flag `'1'` (if set), or `"0"` (for preferences values `"off"`, `"no"` and `"0"`)

- Example for Plugin setting:
  - MyPlugin? topic has: `* Set SHOWHELP = off`
  - Use `"MYPLUGIN_SHOWHELP"` for `$key`
  - `my $showHelp = Foswiki::Func::getPreferencesFlag( "MYPLUGIN_SHOWHELP" );`

NOTE: If `$NO_PREFS_IN_TOPIC` is enabled in the plugin, then preferences set in the plugin topic will be ignored.

getPluginPreferencesFlag($key) -> $boolean

Get a preferences flag from your Plugin

- `$key` - Plugin Preferences key w/o PLUGINNAME_ prefix.

Return: false for preferences values `"off"`, `"no"` and `"0"`, or values not set at all. True otherwise.

Note: This function will will **only** work when called from the Plugin.pm file itself. it **will not work** if called from a sub-package (e.g. Foswiki::Plugins::MyPlugin::MyModule)

NOTE: If `$NO_PREFS_IN_TOPIC` is enabled in the plugin, then preferences set in the plugin topic will be ignored.

setPreferencesValue($name, $val)

Set the preferences value so that future calls to getPreferencesValue will return this value, and `%$name%` will expand to the preference when used in future variable expansions.

The preference only persists for the rest of this request. Finalised preferences cannot be redefined using this function.

Returns 1 if the preference was defined, and 0 otherwise.

User Handling and Access Control

getDefaultUserName( ) -> $loginName

Get default user name as defined in the configuration as `DefaultUserLogin`

Return: `$loginName` Default user name, e.g. `'guest'`

getCanonicalUserID($user) -> $cUID

- `$user` can be a login, wikiname or web.wikiname

Return the cUID of the specified user. A cUID is a unique identifier which is assigned by Foswiki for each user. BEWARE: While the default TopicUserMapping? uses a cUID that looks like a user's LoginName, some characters are modified to make them compatible with rcs. Other usermappings may use other conventions - the JoomlaUserMapping for example, has cUIDs like 'JoomlaeUserMapping_1234'.

If $user is undefined, it assumes the currently logged-in user.

Return: `$cUID`, an internal unique and portable escaped identifier for registered users. This may be autogenerated for an authenticated but unregistered user.

getWikiName($user) -> $wikiName

return the WikiName of the specified user if $user is undefined Get Wiki name of logged in user

- $user can be a cUID, login, wikiname or web.wikiname

Return: `$wikiName` Wiki Name, e.g. `'JohnDoe'`

getWikiUserName($user) -> $wikiName

return the userWeb.WikiName of the specified user if $user is undefined Get Wiki name of logged in user

- $user can be a cUID, login, wikiname or web.wikiname

Return: `$wikiName` Wiki Name, e.g. `"Main.JohnDoe"`

wikiToUserName($id) -> $loginName

Translate a Wiki name to a login name.

- `$id` - Wiki name, e.g. `'Main.JohnDoe'` or `'JohnDoe'`. $id may also be a login name. This will normally be transparent, but should be borne in mind if you have login names that are also legal wiki names.

Return: `$loginName` Login name of user, e.g. `'jdoe'`, or undef if not matched.

Note that it is possible for several login names to map to the same wikiname. This function will only return the **first** login name that maps to the wikiname.

returns undef if the WikiName is not found.

userToWikiName($loginName, $dontAddWeb) -> $wikiName

Translate a login name to a Wiki name

- `$loginName` - Login name, e.g. `'jdoe'`. This may also be a wiki name. This will normally be transparent, but may be relevant if you have login names that are also valid wiki names.
- `$dontAddWeb` - Do not add web prefix if `"1"`

Return: `$wikiName` Wiki name of user, e.g. `'Main.JohnDoe'` or `'JohnDoe'`

userToWikiName will always return a name. If the user does not exist in the mapping, the $loginName parameter is returned. (backward compatibility)

emailToWikiNames($email, $dontAddWeb) -> @wikiNames

- `$email` - email address to look up
- `$dontAddWeb` - Do not add web prefix if `"1"`

Find the wikinames of all users who have the given email address as their registered address. Since several users could register with the same email address, this returns a list of wikinames rather than a single wikiname.

wikinameToEmails($user) -> @emails

- `$user` - wikiname of user to look up

Returns the registered email addresses of the named user. If $user is undef, returns the registered email addresses for the logged-in user.

$user may also be a group.

isGuest( ) -> $boolean

Test if logged in user is a guest (WikiGuest?)

isAnAdmin($id) -> $boolean

Find out if the user is an admin or not. If the user is not given, the currently logged-in user is assumed.

- $id can be either a login name or a WikiName

isGroupMember($group, $id) -> $boolean

Find out if $id is in the named group. e.g.

```
if( Foswiki::Func::isGroupMember( "HesperionXXGroup", "jordi" )) {
    ...
}
```

If `$user` is `undef`, it defaults to the currently logged-in user.

- $id can be a login name or a WikiName

eachUser() -> $iterator

Get an iterator over the list of all the registered users **not** including groups. The iterator will return each wiki name in turn (e.g. 'FredBloggs').

Use it as follows:

```
my $iterator = Foswiki::Func::eachUser();
while ($it->hasNext()) {
    my $user = $it->next();
    # $user is a wikiname
}
```

WARNING on large sites, this could be a long list!

eachMembership($id) -> $iterator

- `$id` - WikiName or login name of the user. If `$id` is `undef`, defaults to the currently logged-in user.

Get an iterator over the names of all groups that the user is a member of.

eachGroup() -> $iterator

Get an iterator over all groups.

Use it as follows:

```
my $iterator = Foswiki::Func::eachGroup();
```

```
while ($it->hasNext()) {
    my $group = $it->next();
    # $group is a group name e.g. AdminGroup
}
```

WARNING on large sites, this could be a long list!

isGroup($group) -> $boolean

Checks if `$group` is the name of a user group.

eachGroupMember($group) -> $iterator

Get an iterator over all the members of the named group. Returns undef if $group is not a valid group.

Use it as follows:

```
my $iterator = Foswiki::Func::eachGroupMember('RadioheadGroup');
while ($it->hasNext()) {
    my $user = $it->next();
    # $user is a wiki name e.g. 'TomYorke', 'PhilSelway'
}
```

WARNING on large sites, this could be a long list!

checkAccessPermission($type, $id, $text, $topic, $web, $meta) -> $boolean

Check access permission for a topic based on the System.AccessControl rules

- `$type` - Access type, required, e.g. `'VIEW'`, `'CHANGE'`.
- `$id` - WikiName of remote user, required, e.g. `"RickShaw"`. $id may also be a login name. If `$id` is '', 0 or `undef` then access is **always permitted**.
- `$text` - Topic text, optional. If 'perl false' (undef, 0 or ''), topic `$web.$topic` is consulted. `$text` may optionally contain embedded `%META:PREFERENCE` tags. Provide this parameter if:
    1. You are setting different access controls in the text to those defined in the stored topic,
    2. You already have the topic text in hand, and want to help avoid having to read it again,
    3. You are providing a `$meta` parameter.
- `$topic` - Topic name, required, e.g. `'PrivateStuff'`
- `$web` - Web name, required, e.g. `'Sandbox'`
- `$meta` - Meta-data object, as returned by `readTopic`. Optional. If `undef`, but `$text` is defined, then access controls will be parsed from `$text`. If defined, then metadata embedded in `$text` will be ignored. This parameter is always ignored if `$text` is undefined. Settings in `$meta` override `Set` settings in $text.

A perl true result indicates that access is permitted.

Note the weird parameter order is due to compatibility constraints with earlier releases.

Tip if you want, you can use this method to check your own access control types. For example, if you:

- Set ALLOWTOPICSPIN = IncyWincy?

in `ThatWeb.ThisTopic`, then a call to `checkAccessPermission('SPIN', 'IncyWincy', undef, 'ThisTopic', 'ThatWeb', undef)` will return `true`.

Webs, Topics and Attachments

getListOfWebs($filter [, $web]) -> @webs

- `$filter` - spec of web types to recover

Gets a list of webs, filtered according to the spec in the $filter, which may include one of:

1. 'user' (for only user webs)
2. 'template' (for only template webs i.e. those starting with "_")

`$filter` may also contain the word 'public' which will further filter out webs that have NOSEARCHALL set on them. 'allowed' filters out webs the current user can't read.

- `$web` - (since NextWiki? 1.0.0) name of web to get list of subwebs for. Defaults to the root.

For example, the deprecated getPublicWebList function can be duplicated as follows:

```
my @webs = Foswiki::Func::getListOfWebs( "user,public" );
```

webExists($web) -> $boolean

Test if web exists

- `$web` - Web name, required, e.g. `'Sandbox'`

createWeb($newWeb, $baseWeb, $opts)

- `$newWeb` is the name of the new web.
- `$baseWeb` is the name of an existing web (a template web). If the base web is a system web, all topics in it will be copied into the new web. If it is a normal web, only topics starting with 'Web' will be copied. If no base web is specified, an empty web (with no topics) will be created. If it is specified but does not exist, an error will be thrown.
- `$opts` is a ref to a hash that contains settings to be modified in

the web preferences topic in the new web.

```
use Error qw( :try );
use Foswiki::AccessControlException;

try {
    Foswiki::Func::createWeb( "Newweb" );
} catch Error::Simple with {
    my $e = shift;
    # see documentation on Error::Simple
} catch Foswiki::AccessControlException with {
    my $e = shift;
    # see documentation on Foswiki::AccessControlException
} otherwise {
    ...
};
```

moveWeb($oldName, $newName)

Move (rename) a web.

```
use Error qw( :try );
use Foswiki::AccessControlException;

try {
    Foswiki::Func::moveWeb( "Oldweb", "Newweb" );
} catch Error::Simple with {
    my $e = shift;
    # see documentation on Error::Simple
} catch Foswiki::AccessControlException with {
    my $e = shift;
    # see documentation on Foswiki::AccessControlException
} otherwise {
    ...
};
```

To delete a web, move it to a subweb of `Trash`

```
Foswiki::Func::moveWeb( "Deadweb", "Trash.Deadweb" );
```

eachChangeSince($web, $time) -> $iterator

Get an iterator over the list of all the changes in the given web between `$time` and now. $time is a time in seconds since 1st Jan 1970, and is not guaranteed to return any changes that occurred before (now - {Store}{RememberChangesFor}). {Store}{RememberChangesFor}) is a setting in `configure`. Changes are returned in **most-recent-first** order.

Use it as follows:

```
my $iterator = Foswiki::Func::eachChangeSince(
    $web, time() - 7 * 24 * 60 * 60); # the last 7 days
while ($iterator->hasNext()) {
    my $change = $iterator->next();
    # $change is a perl hash that contains the following fields:
    # topic => topic name
    # user => wikiname - wikiname of user who made the change
    # time => time of the change
    # revision => revision number *after* the change
    # more => more info about the change (e.g. 'minor')
}
```

getTopicList($web) -> @topics

Get list of all topics in a web

- `$web` - Web name, required, e.g. `'Sandbox'`

Return: `@topics` Topic list, e.g. `( 'WebChanges', 'WebHome', 'WebIndex', 'WebNotify' )`

topicExists($web, $topic) -> $boolean

Test if topic exists

- `$web` - Web name, optional, e.g. `'Main'`.
- `$topic` - Topic name, required, e.g. `'TokyoOffice'`, or `"Main.TokyoOffice"`

$web and $topic are parsed as described in the documentation for `normalizeWebTopicName`. Specifically, the Main is used if $web is not specified and $topic has no web specifier. To get an expected

behaviour it is recommened to specify the current web for $web; don't leave it empty.

checkTopicEditLock($web, $topic, $script) -> ($oopsUrl, $loginName, $unlockTime)

Check if a lease has been taken by some other user.

- `$web` Web name, e.g. `"Main"`, or empty
- `$topic` Topic name, e.g. `"MyTopic"`, or `"Main.MyTopic"`

Return: `( $oopsUrl, $loginName, $unlockTime )` - The `$oopsUrl` for calling redirectCgiQuery(), user's `$loginName`, and estimated `$unlockTime` in minutes, or ('', '', 0) if no lease exists.

- `$script` The script to invoke when continuing with the edit

setTopicEditLock($web, $topic, $lock)

- `$web` Web name, e.g. `"Main"`, or empty
- `$topic` Topic name, e.g. `"MyTopic"`, or `"Main.MyTopic"`
- `$lock` 1 to lease the topic, 0 to clear an existing lease

Takes out a "lease" on the topic. The lease doesn't prevent anyone from editing and changing the topic, but it does redirect them to a warning screen, so this provides some protection. The `edit` script always takes out a lease.

It is **impossible** to fully lock a topic. Concurrent changes will be merged.

saveTopic($web, $topic, $meta, $text, $options)

- `$web` - web for the topic
- `$topic` - topic name
- `$meta` - reference to Foswiki::Meta object
- `$text` - text of the topic (without embedded meta-data!!!
- `\%options` - ref to hash of save options `\%options` may include:

`dontlog`	don't log this change in twiki log
`forcenewrevision`	force the save to increment the revision counter
`minor`	True if this is a minor change, and is not to be notified
`comment`	Comment relating to the save

For example,

```
my( $meta, $text ) = Foswiki::Func::readTopic( $web, $topic );
$text =~ s/APPLE/ORANGE/g;
Foswiki::Func::saveTopic( $web, $topic, $meta, $text, { forcenewrevision => 1 } );
```

Note: Plugins handlers (e.g. `beforeSaveHandler`) will be called as appropriate.

In the event of an error an exception will be thrown. Callers can elect to trap the exceptions thrown, or allow them to propagate to the calling environment. May throw Foswiki::OopsException, Foswiki::AccessControlException or Error::Simple.

saveTopicText($web, $topic, $text, $ignorePermissions, $dontNotify) -> $oopsUrl

Save topic text, typically obtained by readTopicText(). Topic data usually includes meta data; the file attachment meta data is replaced by the meta data from the topic file if it exists.

- `$web` - Web name, e.g. `'Main'`, or empty
- `$topic` - Topic name, e.g. `'MyTopic'`, or `"Main.MyTopic"`
- `$text` - Topic text to save, assumed to include meta data
- `$ignorePermissions` - Set to `"1"` if checkAccessPermission() is already performed and OK
- `$dontNotify` - Set to `"1"` if not to notify users of the change

Return: `$oopsUrl` Empty string if OK; the `$oopsUrl` for calling redirectCgiQuery() in case of error

This method is a lot less efficient and much more dangerous than `saveTopic`.

```
my $text = Foswiki::Func::readTopicText( $web, $topic );

# check for oops URL in case of error:
if( $text =~ /^http.*?\/oops/ ) {
    Foswiki::Func::redirectCgiQuery( $query, $text );
    return;
}
# do topic text manipulation like:
$text =~ s/old/new/g;
# do meta data manipulation like:
$text =~ s/(META\:FIELD.*?name\=\"TopicClassification\".*?value\=\")[^\"]*/$1BugResolved/;
$oopsUrl = Foswiki::Func::saveTopicText( $web, $topic, $text ); # save topic text
```

moveTopic($web, $topic, $newWeb, $newTopic)

- `$web` source web - required
- `$topic` source topic - required
- `$newWeb` dest web
- `$newTopic` dest topic

Renames the topic. Throws an exception if something went wrong. If $newWeb is undef, it defaults to $web. If $newTopic is undef, it defaults to $topic.

The destination topic must not already exist.

Rename a topic to the $Foswiki::cfg{TrashWebName} to delete it.

```
use Error qw( :try );

try {
    moveTopic( "Work", "TokyoOffice", "Trash", "ClosedOffice" );
} catch Error::Simple with {
    my $e = shift;
    # see documentation on Error::Simple
} catch Foswiki::AccessControlException with {
    my $e = shift;
    # see documentation on Foswiki::AccessControlException
} otherwise {
    ...
};
```

getRevisionInfo($web, $topic, $rev, $attachment) -> ($date, $user, $rev, $comment)

Get revision info of a topic or attachment

- `$web` - Web name, optional, e.g. `'Main'`
- `$topic` - Topic name, required, e.g. `'TokyoOffice'`
- `$rev` - revsion number, or tag name (can be in the format 1.2, or just the minor number)
- `$attachment` -attachment filename

Return: `( $date, $user, $rev, $comment )` List with: (last update date, login name of last user, minor part of top revision number, comment of attachment if attachment), e.g. `( 1234561, 'phoeny', "5", )`

$date	in epochSec
$user	Wiki name of the author (**not** login name)
$rev	actual rev number
$comment	comment given for uploaded attachment

NOTE: if you are trying to get revision info for a topic, use `$meta->getRevisionInfo` instead if you can - it is significantly more efficient.

getRevisionAtTime($web, $topic, $time) -> $rev

Get the revision number of a topic at a specific time.

- `$web` - web for topic
- `$topic` - topic
- `$time` - time (in epoch secs) for the rev

Return: Single-digit revision number, or undef if it couldn't be determined (either because the topic isn't that old, or there was a problem)

readTopic($web, $topic, $rev) -> ($meta, $text)

Read topic text and meta data, regardless of access permissions.

- `$web` - Web name, required, e.g. `'Main'`
- `$topic` - Topic name, required, e.g. `'TokyoOffice'`
- `$rev` - revision to read (default latest)

Return: `( $meta, $text )` Meta data object and topic text

`$meta` is a perl 'object' of class `Foswiki::Meta`. This class is fully documented in the source code documentation shipped with the release, or can be inspected in the `lib/Foswiki/Meta.pm` file.

This method **ignores** topic access permissions. You should be careful to use `checkAccessPermission` to ensure the current user has read access to the topic.

readTopicText($web, $topic, $rev, $ignorePermissions) -> $text

Read topic text, including meta data

- `$web` - Web name, e.g. `'Main'`, or empty

- $topic - Topic name, e.g. 'MyTopic', or "Main.MyTopic"
- $rev - Topic revision to read, optional. Specify the minor part of the revision, e.g. "5", not "1.5"; the top revision is returned if omitted or empty.
- $ignorePermissions - Set to "1" if checkAccessPermission() is already performed and OK; an oops URL is returned if user has no permission

Return: $text Topic text with embedded meta data; an oops URL for calling redirectCgiQuery() is returned in case of an error

This method is more efficient than readTopic, but returns meta-data embedded in the text. Plugins authors must be very careful to avoid damaging meta-data. You are recommended to use readTopic instead, which is a lot safer.

attachmentExists($web, $topic, $attachment) -> $boolean

Test if attachment exists

- $web - Web name, optional, e.g. Main.
- $topic - Topic name, required, e.g. TokyoOffice, or Main.TokyoOffice
- $attachment - attachment name, e.g.=logo.gif=

$web and $topic are parsed as described in the documentation for normalizeWebTopicName.

readAttachment($web, $topic, $name, $rev) -> $data

- $web - web for topic
- $topic - topic
- $name - attachment name
- $rev - revision to read (default latest)

Read an attachment from the store for a topic, and return it as a string. The names of attachments on a topic can be recovered from the meta-data returned by readTopic. If the attachment does not exist, or cannot be read, undef will be returned. If the revision is not specified, the latest version will be returned.

View permission on the topic is required for the read to be successful. Access control violations are flagged by a Foswiki::AccessControlException. Permissions are checked for the current user.

```
my( $meta, $text ) = Foswiki::Func::readTopic( $web, $topic );
my @attachments = $meta->find( 'FILEATTACHMENT' );
foreach my $a ( @attachments ) {
   try {
       my $data = Foswiki::Func::readAttachment( $web, $topic, $a->{name} );
       ...
   } catch Foswiki::AccessControlException with {
   };
}
```

saveAttachment($web, $topic, $attachment, \%opts)

- $web - web for topic
- $topic - topic to atach to
- $attachment - name of the attachment
- \%opts - Ref to hash of options

\%opts may include:

dontlog	don't log this change in twiki log
comment	comment for save
hide	if the attachment is to be hidden in normal topic view
stream	Stream of file to upload
file	Name of a file to use for the attachment data. ignored if stream is set. Local file on the server.
filepath	Client path to file
filesize	Size of uploaded data
filedate	Date

Save an attachment to the store for a topic. On success, returns undef. If there is an error, an exception will be thrown.

```
 try {
     Foswiki::Func::saveAttachment( $web, $topic, 'image.gif',
                                    { file => 'image.gif',
                                      comment => 'Picture of Health',
                                      hide => 1 } );
} catch Error::Simple with {
   # see documentation on Error
} otherwise {
   ...
};
```

moveAttachment($web, $topic, $attachment, $newWeb, $newTopic, $newAttachment)

- $web source web - required
- $topic source topic - required
- $attachment source attachment - required
- $newWeb dest web
- $newTopic dest topic
- $newAttachment dest attachment

Renames the topic. Throws an exception on error or access violation. If $newWeb is undef, it defaults to $web. If $newTopic is undef, it defaults to $topic. If $newAttachment is undef, it defaults to $attachment. If all of $newWeb, $newTopic and $newAttachment are undef, it is an error.

The destination topic must already exist, but the destination attachment must **not** exist.

Rename an attachment to $Foswiki::cfg{TrashWebName}.TrashAttament to delete it.

```
use Error qw( :try );

try {
   # move attachment between topics
   moveAttachment( "Countries", "Germany", "AlsaceLorraine.dat",
                     "Countries", "France" );
   # Note destination attachment name is defaulted to the same as source
} catch Foswiki::AccessControlException with {
   my $e = shift;
   # see documentation on Foswiki::AccessControlException
} catch Error::Simple with {
   my $e = shift;
   # see documentation on Error::Simple
};
```

Assembling Pages

readTemplate($name, $skin) -> $text

Read a template or skin. Embedded template directives get expanded

- `$name` - Template name, e.g. `'view'`
- `$skin` - Comma-separated list of skin names, optional, e.g. `'print'`

Return: `$text` Template text

loadTemplate ($name, $skin, $web) -> $text

- `$name` - template file name
- `$skin` - comma-separated list of skins to use (default: current skin)
- `$web` - the web to look in for topics that contain templates (default: current web)

Return: expanded template text (what's left after removal of all %TMPL:DEF% statements)

Reads a template and extracts template definitions, adding them to the list of loaded templates, overwriting any previous definition.

How Foswiki searches for templates is described in SkinTemplates.

If template text is found, extracts include statements and fully expands them.

expandTemplate($def) -> $string

Do a , only expanding the template (not expanding any variables other than %TMPL)

- `$def` - template name

Return: the text of the expanded template

A template is defined using a %TMPL:DEF% statement in a template file. See the documentation on Foswiki templates for more information.

writeHeader()

Prints a basic content-type HTML header for text/html to standard out.

redirectCgiQuery($query, $url, $passthru)

Redirect to URL

- `$query` - CGI query object. Ignored, only there for compatibility. The session CGI query object is used instead.
- `$url` - URL to redirect to
- `$passthru` - enable passthrough.

Return: none

Print output to STDOUT that will cause a 302 redirect to a new URL. Nothing more should be printed to STDOUT after this method has been called.

The `$passthru` parameter allows you to pass the parameters that were passed to the current query on to the target URL, as long as it is another URL on the same installation. If `$passthru` is set to a true value, then Foswiki will save the current URL parameters, and then try to restore them on the other side of the redirect. Parameters are stored on the server in a cache file.

Note that if `$passthru` is set, then any parameters in `$url` will be lost when the old parameters are restored. if you want to change any parameter values, you will need to do that in the current CGI query before redirecting e.g.

```
my $query = Foswiki::Func::getCgiQuery();
$query->param(-name => 'text', -value => 'Different text');
Foswiki::Func::redirectCgiQuery(
  undef, Foswiki::Func::getScriptUrl($web, $topic, 'edit'), 1);
```

`$passthru` does nothing if `$url` does not point to a script in the current Foswiki installation.

addToHEAD($id, $header, $requires)

Adds `$header` to the HTML header (the