

## **Wikiprint Book**

**Title: Fine grained permissions**

**Subject: Tibisay - TracFineGrainedPermissions**

**Version: 3**

**Date: 17/05/24 09:22:09**

## Table of Contents

<b>Fine grained permissions</b>	<b>3</b>
Permission Policies	3
AuthzPolicy	3
Configuration	3
Usage Notes	4
Missing Features	6
AuthzSourcePolicy (mod_authz_svn-like permission policy)	6
Trac Configuration	6
Subversion Configuration	7
ReadonlyWikiPolicy?	7
Debugging permissions	8

## Fine grained permissions

Before Trac 0.11, it was only possible to define fine-grained permissions checks on the repository browser sub-system.

Since 0.11, there's a general mechanism in place that allows custom **permission policy plugins** to grant or deny any action on any kind of Trac resources, even at the level of specific versions of such resources.

Note that for Trac 0.12, `authz_policy` has been integrated as an optional module (in `tracopt.perm.authz_policy.*`), so it's installed by default and can simply be activated via the *Plugins* panel in the Trac administration module.

### Permission Policies

A great diversity of permission policies can be implemented, and Trac comes with a few examples.

Which policies are currently active is determined by a configuration setting in [TracIni](#): e.g.

```
[trac]
permission_policies = ReadonlyWikiPolicy, DefaultPermissionPolicy, LegacyAttachmentPolicy
```

This lists the [#ReadonlyWikiPolicy](#) which controls readonly access to wiki pages, followed by the `DefaultPermissionPolicy` which checks for the traditional coarse grained style permissions described in [TracPermissions](#), and the `LegacyAttachmentPolicy` which knows how to use the coarse grained permissions for checking the permissions available on attachments.

Among the possible optional choices, there is [#AuthzPolicy](#), a very generic permission policy, based on an Authz-style system. See [#authz\\_policy.py](#) for details.

Another popular permission policy [#AuthzSourcePolicy](#), re-implements the pre-0.12 support for checking fine-grained permissions limited to Subversion repositories in terms of the new system.

See also [#sample-plugins/permissions](#) for more examples.

### AuthzPolicy

#### Configuration

- Install [#ConfigObj](#) (still needed for 0.12 and later).
- Copy [/tracopt/perm/authz\\_policy.py](#) to your environment's plugins directory (only for Trac 0.11).
- Put a [#authzpolicy.conf](#) file somewhere, preferably on a secured location on the server, not readable for others than the webuser. If the file contains non-ASCII characters, the UTF-8 encoding should be used.

Update your `trac.ini`:

modify the [permission\\_policies](#) entry in the `[trac]` section

```
[trac]
...
permission_policies = AuthzPolicy, ReadonlyWikiPolicy, DefaultPermissionPolicy, LegacyAttachmentPolicy
```

add a new `[authz_policy]` section

```
[authz_policy]
authz_file = /some/trac/env/conf/authzpolicy.conf
```

enable the plugin through [WebAdmin](#) or by editing the `[components]` section

```
[components]
...
# Trac 0.12
tracopt.perm.authz_policy.* = enabled
# for Trac 0.11 use this
#authz_policy.* = enabled
```

## Usage Notes

Note that the order in which permission policies are specified is quite critical, as policies will be examined in the sequence provided.

A policy will return either `True`, `False` or `None` for a given permission check. `True` is returned if the policy explicitly grants the permission. `False` is returned if the policy explicitly denies the permission. `None` is returned if the policy is unable to either grant or deny the permission.

NOTE: Only if the return value is `None` will the *next* permission policy be consulted. If none of the policies explicitly grants the permission, the final result will be `False` (i.e. permission denied).

The `authzpolicy.conf` file is a `.ini` style configuration file:

```
[wiki:PrivatePage@*]
john = WIKI_VIEW, !WIKI_MODIFY
jack = WIKI_VIEW
* =
```

- 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 explicitly, 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 descriptor **IN ORDER** of appearance in the configuration file. **ORDER IS CRITICAL**. Once a section matches, the current username is matched against the keys (usernames) of the section, **IN ORDER**.

- If a key (username) is prefixed with a `@`, it is treated as a group.
- If a value (permission) 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.

**Note:** Other groups which are created by user (e.g. by 'adding subjects to groups' on web interface page *Admin / Permissions*) cannot be used. See [■#5648](#) for details about this missing feature

For example, if the `authz_file` contains:

```
[wiki:WikiStart@*]
* = WIKI_VIEW

[wiki:PrivatePage@*]
john = WIKI_VIEW
* = !WIKI_VIEW
```

and the default permissions are set like this:

```
john          WIKI_VIEW
jack          WIKI_VIEW
# anonymous has no WIKI_VIEW
```

Then:

- All versions of [WikiStart](#) will be viewable by everybody (including anonymous)
- PrivatePage will be viewable only by john
- other pages will be viewable only by john and jack

Groups:

```
[groups]
admins = john, jack
devs = alice, bob

[wiki:Dev@*]
@admin = TRAC_ADMIN
@devs = WIKI_VIEW
* =

[*]
@admin = TRAC_ADMIN
* =
```

Then:

- everything is blocked (whitelist approach), but
- admins get all TRAC\_ADMIN everywhere and
- devs can view wiki pages.

Some repository examples (Browse Source specific):

```
# A single repository:
[repository:test_repo@*]
john = BROWSER_VIEW, FILE_VIEW
# John has BROWSER_VIEW and FILE_VIEW for the entire test_repo

# The default repository (requires Trac 1.0.2 or later):
[repository:@*]
john = BROWSER_VIEW, FILE_VIEW
# John has BROWSER_VIEW and FILE_VIEW for the entire default repository

# All repositories:
[repository:*@*]
jack = BROWSER_VIEW, FILE_VIEW
# Jack has BROWSER_VIEW and FILE_VIEW for all repositories
```

Very fine grain repository access:

```
# John has BROWSER_VIEW and FILE_VIEW access to trunk/src/some/location/ only
[repository:test_repo@*/source:trunk/src/some/location/*@*]
john = BROWSER_VIEW, FILE_VIEW

# John has BROWSER_VIEW and FILE_VIEW access to only revision 1 of all files at trunk/src/some/location only
[repository:test_repo@*/source:trunk/src/some/location/*@1]
john = BROWSER_VIEW, FILE_VIEW
```

```
# John has BROWSER_VIEW and FILE_VIEW access to all revisions of 'somefile' at trunk/src/some/location only
[repository:test_repo*/source:trunk/src/some/location/somefile@*]
john = BROWSER_VIEW, FILE_VIEW
```

```
# John has BROWSER_VIEW and FILE_VIEW access to only revision 1 of 'somefile' at trunk/src/some/location only
[repository:test_repo*/source:trunk/src/some/location/somefile@1]
john = BROWSER_VIEW, FILE_VIEW
```

Note: In order for Timeline to work/visible for John, we must add CHANGESET\_VIEW to the above permission list.

### Missing Features

Although possible with the DefaultPermissionPolicy handling (see Admin panel), fine-grained permissions still miss those grouping features (see [■#9573](#), [■#5648](#)). Patches are partially available, see forgotten authz\_policy.2.patch part of [■#6680](#).

You cannot do the following:

```
[groups]
team1 = a, b, c
team2 = d, e, f
team3 = g, h, i
departmentA = team1, team2
```

Permission groups are not supported either. You cannot do the following:

```
[groups]
permission_level_1 = WIKI_VIEW, TICKET_VIEW
permission_level_2 = permission_level_1, WIKI_MODIFY, TICKET_MODIFY
[*]
@team1 = permission_level_1
@team2 = permission_level_2
@team3 = permission_level_2, TICKET_CREATE
```

### AuthzSourcePolicy (mod\_authz\_svn-like permission policy)

At the time of this writing, the old fine grained permissions system from Trac 0.11 and before used for restricting access to the repository has been converted to a permission policy component, but from the user point of view, this makes little if no difference.

That kind of fine-grained permission control needs a definition file, which is the one used by Subversion's mod\_authz\_svn. More information about this file format and about its usage in Subversion is available in the [■Path-Based Authorization](#) section in the Server Configuration chapter of the svn book.

Example:

```
[/]
* = r

[/branches/calc/bug-142]
harry = rw
sally = r

[/branches/calc/bug-142/secret]
harry =
```

- */ = Everyone has read access by default*
- */branches/calc/bug-142 = harry has read/write access, sally read only*
- */branches/calc/bug-142/secret = harry has no access, sally has read access (inherited as a sub folder permission)*

### Trac Configuration

To activate fine grained permissions you must specify the `authz_file` option in the `[trac]` section of `trac.ini`. If this option is set to null or not specified the permissions will not be used.

```
[trac]
authz_file = /path/to/svnaccessfile
```

If you want to support the use of the `[modulename: /some/path]` syntax within the `authz_file`, add

```
authz_module_name = modulename
```

where *modulename* refers to the same repository indicated by the `repository_dir` entry in the `[trac]` section. As an example, if the `repository_dir` entry in the `[trac]` section is `/srv/active/svn/blahblah`, that would yield the following:

```
[trac]
authz_file = /path/to/svnaccessfile
authz_module_name = blahblah
...
repository_dir = /srv/active/svn/blahblah
```

where the svn access file, `/path/to/svnaccessfile`, contains entries such as `[blahblah:/some/path]`.

**Note:** Usernames inside the Authz file must be the same as those used inside trac.

As of version 0.12, make sure you have *AuthzSourcePolicy* included in the `permission_policies` list in `trac.ini`, otherwise the `authz` permissions file will be ignored.

```
[trac]
permission_policies = AuthzSourcePolicy, ReadonlyWikiPolicy, DefaultPermissionPolicy, LegacyAttachmentPolicy
```

## Subversion Configuration

The same access file is typically applied to the corresponding Subversion repository using an Apache directive like this:

```
<Location /repos>
  DAV svn
  SVNParentPath /usr/local/svn

  # our access control policy
  AuthzSVNAccessFile /path/to/svnaccessfile
</Location>
```

For information about how to restrict access to entire projects in a multiple project environment see [wiki:TracMultipleProjectsSVNAccess](#)

## ReadonlyWikiPolicy?

Since 1.1.2, the read-only attribute of wiki pages is enabled and enforced when `ReadonlyWikiPolicy` is in the list of active permission policies. The default for new Trac installations in 1.1.2 and later is:

```
[trac]
permission_policies = ReadonlyWikiPolicy,
DefaultPermissionPolicy,
LegacyAttachmentPolicy
```

When upgrading from earlier versions of Trac, `ReadonlyWikiPolicy` will be appended to the list of `permission_policies` when upgrading the environment, provided that `permission_policies` has the default value. If any non-default `permission_policies` are active, `ReadonlyWikiPolicy` **will need to be manually added** to the list. A message will be echoed to the console when upgrading the environment, indicating if any action needs to be taken.

**ReadonlyWikiPolicy must be listed before DefaultPermissionPolicy.** The latter returns `True` to allow modify, delete or rename actions when the user has the respective `WIKI_*` permission, without consideration for the read-only attribute.

The `ReadonlyWikiPolicy` returns `False` to deny modify, delete and rename actions on wiki pages when the page has the read-only attribute set and the user does not have `WIKI_ADMIN`, regardless of `WIKI_MODIFY`, `WIKI_DELETE` and `WIKI_RENAME` permissions. It returns `None` for all other cases.

When active, the [#AuthzPolicy](#) should therefore come before `ReadonlyWikiPolicy`, allowing it to grant or deny the actions on individual resources, which is the usual ordering for `AuthzPolicy` in the `permission_policies` list.

```
[trac]
permission_policies = AuthzPolicy,
ReadonlyWikiPolicy,
DefaultPermissionPolicy,
LegacyAttachmentPolicy
```

The placement of [#AuthzSourcePolicy](#) relative to `ReadonlyWikiPolicy` does not matter since they don't perform checks on the same realms.

For all other permission policies, the user will need to decide the proper ordering. Generally, if the permission policy should be capable of overriding the check performed by `ReadonlyWikiPolicy`, it should come before `ReadonlyWikiPolicy` in the list. If the `ReadonlyWikiPolicy` should override the check performed by another permission policy, as is the case for `DefaultPermissionPolicy`, then `ReadonlyWikiPolicy` should come first.

## Debugging permissions

In `trac.ini` set:

```
[logging]
log_file = trac.log
log_level = DEBUG
log_type = file
```

And watch:

```
tail -n 0 -f log/trac.log | egrep '\[perm\]|\[authz_policy\]'
```

to understand what checks are being performed. See the sourced documentation of the plugin for more info.

See also: [TracPermissions](#), [TracHacks:FineGrainedPageAuthzEditorPlugin](#) for a simple editor plugin.