Hello World

'Hello World' Tutorial for Tiki Developers

Introduction - A Historical Context

Luis Argerich (the founder of Tiki) wrote:

> *One of my goals behind Tiki was to make it very easy to any kind of programmer to be able to contribute, if you make a "genius" framework using object oriented code, MVC and other gadgets you need "genius" programmers. If a new PHP coder can't contribute without learning all sorts of things something is wrong with the framework specially for a community-driven project.*
>
> *I always thought that adding one PHP file and one template file and putting your code in the PHP and your design in the template was the simplest way to add features to Tiki. Furthermore features can cooperate sharing the database but are completely independent, you remove the code and nothing happens to the rest of Tiki. It is true that the code inside each feature can be not very clear but it is code that is highly cohesive because it does just one thing.*
>
> *I believe in functionality through simplicity, if you start simple you can always evolve if that is good for the project, and if you ever need it. If you start like the NASA you can realize too late that you made some ridiculous choices and there is no way back.*

mose subsequently on the devel mailing list wrote:

> *is there any internal development documentation of Tiki project. I*
> *think some architecture description, described database schema and it's*
> *relations, internal rules, object relations.*
>
> *I found only documentation for adding new feature, but I think it must*
> *be something more, because of huge number of developers working on this*
> *project.*
>
> *Is internal documentation something available on demand or it simply*
> *doesn't exist?*
>
> *It just doesn't exist. And here are the reasons:*
>
> *The initial design of Tiki by Luis Argerich was especially focused on having code with very small abstractions. His purpose, as far as I can imagine, was to make the coding easy so to get a wider population of contributors, and an easy 'hackability'. Then, the source code was initially simple enough and didn't require more documentation that the db structure and the existing source code. There is very few and simple relations in the database, they usually are explicit enough. Many files were using code duplication rather than code abstraction.*
>
> *That made a ground for a large number of contributions, then the code has been evolving organically from the existing basis. Quite fast on some aspects. Maintaining a code documentation was an effort that nobody felt necessary. Well, personally, I never needed it even in early times.*
>
> *Technically speaking, the Tiki development is quite far from the corporate habits. Low-design, fast evolution cycles, organic selection over time of features, it's actually like if we were talking about the wiki-way, but in the coding field (validation of code is posterior to its commit and not prior, like in a wiki).*
>
> *Of course such outfit brings drawbacks from the industry-oriented point of view that deals with*

> *time in much more a mechanical way than what our organic evolution proposes.*
>
> *But I think that's what attracted many users and contributors of Tiki, and disgusted others.*

Marc Laporte (project admin) wrote:

> *Tiki has an all-in-one approach to features. Think of it as a suite of applications like Open Office. Tiki has more built-in features than any other open source Web app (if you know of one, please let me know). Some people are concerned about all the features and that "it can't work" or "it's not the way to do it". Since Tiki has a different model than the "conventional wisdom", I feel it's important to explain how it works and why it works.*

Please read about the Tiki model.

Alain Désilets more recently wrote:

> *I know this is all wrong, but fear it might be right - Alain Désilets*

Nelson Ko (another project admin) wrote:

> *Keeping it simple is a well-enshrined principle of Tiki. However, being simple does not mean shunning abstraction and/or documentation. It is accepted that abstraction and documentation are good things, but too often too many have too much of a good thing simply because they are introduced for their own sakes.*
>
> *It is important to recognize that the ultimate goal of abstraction and documentation is to make something simple even simpler, and the usefulness of such things increases with the volume of code. But in no way should these be used as crutches to excuse complicated design .*
>
> *Increasing abstraction is a natural progression in the progress of all engineering. However, abstract too early and you might get locked in too early to a particular design. Abstract too late and things become less tractable. Ultimately, the demand and supply of this in an open community like ours should be organic. It will self-adjust over time, just as even PHP itself is evolving over time.*

- How CMS architecture affects dev communities - A case study of Drupal, Tiki Wiki and XOOPS

1.2. Glossary

Things are sometimes named a bit differently than in other Web Applications. Please see Glossary

1.3. Who does what?

Tiki Wiki CMS Groupware is a vast project. It's tricky for new people to know how the Tiki Community is organized, to find the right person to contact and to start to contribute. What may I expect from others within Tiki Community, and what do others expect from me? Who does what? Who should I talk to? How does it work? This page is intended to help.

Important: everyone is a volunteer. Things get done because someone, **like you** decides to take the time to make it better.

Please see:
http://tiki.org/WhoWhat

1.4. Smarty/PHP

Tiki uses the Smarty template engine. Its purpose is to separate the presentation layer from the programming layer.

The PHP interfaces with the database, does the computation and assigns the Smarty variables. The Smarty templates use these variables to generate the actual HTML.

For instance:

```
<?php // Smarty use sample $user='admin'; $smarty->assign('user', $user);
```

This PHP code assigns the Smarty variable *$user*.

```
<h1>admin</h1>
```

In a Smarty template, you can loop over an array of data and do some simple computation. To preserve separation and optimization, no PHP code must be inserted in a template.

1.5. The 'Hello World' page

How to display "hello world" in the central column of a Tiki page:

Create a PHP file in the Tiki root directory (where *tiki-install.php* is).
hello_world.php:

```
<?php // hello_world displayed in a Tiki center column include_once('tiki-setup.php');
$smarty->assign('mid', 'hello_world.tpl'); $smarty->display('tiki.tpl');
```

Create a template located in the sub directory *templates/* (http://www.domain.com/templates/) or in *templates/styles/your_style/*
hello_world.tpl:

```
<p>Hello World</p>
```

To test it, type the URL in your browser : http://my.domain.com/hello_world.php (where http://my.domain.com links to your Tiki root).

Notice the use of *include_once('tiki-setup.php')* in the PHP code. This include initializes the context to be able to use smarty. Later we will see it is doing more.
tiki.tpl is the Tiki global template file. It is the (almost) only template that contains a complete html page.
A simplified working *tiki.tpl* file would be:

```
{* top *} {include file="header.tpl"} {* middle *} {* left column *} {section name=ix
loop=$left_modules} {$left_modules[ix].data} {/section} {* middle column *} {include file=$mid} {*
right column *} {section name=ix loop=$right_modules} {$right_modules[ix].data} {/section} {*
bottom*} {include file="footer.tpl"}
```

You can see the top section *header.tpl*, the bottom section *footer.tpl* and the 3 columns. The left column

displays all the left modules through a loop. Idem for the right column. The central part is an include of the *$mid* file. (Notice: in Tiki 1.10, it is simply {$mid} and not {include file=$mid})
In our hello_world example, the $mid part is the template *hello_world.tpl* and will display the string *hello world*.

The templates *hello_world.tpl* can be created in the sub-directory *templates/* or the sub-directory *templates/styles/your_style/* (where your_style is the name of the theme you are currently using (A theme name is the css file name without extension). If it exists, the template in your current theme directory overrides the one in the default directory. The advantage of putting your templates in your local theme directory is to better identify what you have changed from the original Tiki. (Notice: if you are using SVN and are handy with SVN conflict resolution, it may be better to directly modify the templates in the default directory to be able to update/merge the Tiki modification with yours)

The *$left_modules* and *$right_modules* smarty variables are set up in tiki-modules.php. You will see later in the module section how the modules are set.

1.6. About templates

More about: Templates Best Practices

1.7. tiki-setup.php

This include does everything you need to have a Tiki page. It does a lot more that initializing smarty. By including this file in your PHP, you:

- establish the connection with the database.
- read all the system preferences, user preferences, permissions from the database.
- check if the user is logged via a cookie.
- computes all the modules (in Tiki1.9). In Tiki 1.10 this is done in $smarty->display
- check up for some security points
- ...

A couple of interesting variables are also set there

- user (user login)
- group (default group name)
- language
- all preferences
- ...

All these variables are set for PHP and for smarty.

To see your username in the hello world program, modify your template:

```
<p>Hello World, Dear {$user|escape}</p>
```

1.8. How to debug and to see the smarty variables

See also: Kint

If you want to see all the smarty variables that are set, you can use the smarty function {debug} in your templates. Smarty will display a popup window.

```
{debug} <p>Hello World</p>
```

You can also use the PHP function echo or print_r to see some PHP variable for debugging.
And you can use the function debug_backtrace to see the php trace.

```
<?php function test() { echo "<pre>"; print_r(debug_backtrace()); echo "</pre>"; } include_once('tiki-setup.php'); echo "<pre>user:$user</pre>"; test(); $smarty->assign('mid', 'hello_world.tpl'); $smarty->display('tiki.tpl');
```

1.9. To execute some queries in the PHP

Tiki uses the ADODB database abstraction library. A good description about how to code the queries can be found at DbAbstractionDev

A typical list routine without permission check.
In *lib/contribution/contributionlib.php*

```
// List a subset of contribution // $offset = will return the objects from there to $offset + $maxRecords (if 0 = from the beginning) // $maxRecords = max number to return if = -1 returns all the objects // $sort_mode = order criterium // $find = string to search function list_contributions($offset=0, $maxRecords=-1, $sort_mode='name_asc', $find='') { $bindvars = array(); $mid = ''; if ($find) { $mid .= " where (`name` like ?)"; $bindvars[] = "%$find%"; } $query = "select * from `tiki_contributions` $mid order by ".$this->convert_sortmode($sort_mode); $result = $this->query($query, $bindvars, $maxRecords, $offset); $ret = array(); while ($res = $result->fetchRow()) { $ret[] = $res; } $retval = array(); $retval['data'] = $ret; if ($maxRecords > 0) { $query_cant = "select count(*) from `tiki_contributions` $mid"; $retval['cant'] = $this->getOne($query_cant, $bindvars); } else { $retval['cant'] = count($ret); } return $retval; }
```

First, you have to notice that all table names and column names must be back quoted.
The *$sort_mode* is the concatenation of the column name, the underscore and asc or desc. It can be any column.

In the main php, *tiki-contribution.php*, you will have

```
include_once('lib/contribution/contributionlib.php'); $contributions = $contributionlib->list_contributions(); $smarty->assign('contributions', $contributions['data']);
```

and in the template *templates/tiki-contribution.tpl*, you will have

```
{cycle values="even,odd" print=false} {section name=ix loop=$contributions} <tr> <td class="{cycle advance=false}">{$contributions["ix"].name|escape}</td> <td
```

```
class="{cycle}">{$contributions["ix"].hits}</td> </tr> {/section}
```

Notice: There is not yet a good method to check the perms in this kind of routine. You have to get all the objects and checked them.

If there is a problem with the query, Tiki will output a message saying that a query failed. This message is usually not detailed enough to allow you to figure out what's wrong.

To debug the query:

- Put a traces printing the content of the query and its arguments.
- Try to execute the query manually in PHPMyAdmin.
  - Go to PHPMyAdmin > Databases
  - Click on the name of the DB you are using for testing and debugging.
  - Click on SQL
  - In the Run SQL query/queries on database text field, paste the query that was output by your trace.
  - This query probably ends with something like this: (?, ?, ?). Replace the question marks by the arguments that are passed to the $this->query() method (you should have output them too in the trace). Make sure you put the arguments between backquotes.
  - Click on **Go** button, and see what error message is displayed.
  - Note: In your SQL query, the table and field names must be between backquotes (`), but the values must be between single quotes (').

1.10. To introduce a new feature or a new preference

Each feature is optional. Thus, there is a preference to activate. See Create a new preference

1.11. To modify the database schema

If you need to add, delete or modify the structure of a table you should look at: Database Schema Upgrade.

1.12. To get/set a user preference

To get the value of a user preference, you can use in a function :

```
global $tikilib, $someuser; $foo = $tikilib->get_user_preference($someuser, $pref_name,
$default_pref_value);
```

$someuser is the user name.
$pref_name can be *theme*, *allowMsgs*
The default for $default_pref_value is ""
The effect of the above code is to populate the global array variable $user_preferences[someuser], which is also returned in the variable $foo.
Notice: *global $tikilib, $someuser;* is not necessary if you add these lines in the same file that the include tiki-setup.php (the hello_world.php) but are needed in you add these lines in a function.
Notice: To access the prefs of the current user (user), it is not necessary to use the above. These are always available as variables. In version 1.10, they are available in the $prefs array.

To set a user preference

```
global $tikilib, $someuser; $tikilib->set_user_preference($someuser, $pref_name, $pref_value);
```

Notice: we don't include lib/tikilib.php because it is always included in tiki-setup.php. Meanwhile, the other libraries must be included like this:

```
global $categlib; include_once('lib/categories/categlib.php');
```

Notice the use of global. Global is necessary because your library can have been included in another function that is out of scope of your new function.
Notice that these two commands actually work together as a single operation. This is why putting both on the same line actually helps code review, and is an accepted exception to the usual coding standards.

You can have a list of user preferences in the SQL table tiki_user_preferences. Same remark than for tiki_preferences, the list can be incomplete.

1.13. To create a menu or to introduce a menu option

Tiki has a couple of ways to introduce menus
- one by creating the menu and the menu options through sql (1)
- one by using the tiki API (2)
- one by using the templating / module feature (3)
- one based on phplayers (4)

There is not really a best method to create a menu. It depends on if your menu can be dynamically updated, you will not use the template. If your menu is fixed, perhaps a template is the best.
The phplayers method is not described here as there is no current API. (phplayers menu are interesting because they can have more than 2 levels (Means an option in an option in a section)

1.14. Menu created by sql

```
INSERT INTO tiki_menus (menuId,name,description,type) VALUES (100,'My menu','My very own menu','d' INSERT INTO tiki_menu_options (menuId,type,name,url,position,section,perm,groupname) VALUES (100,'o','My option','tiki-my_option.php',10,'','',''); INSERT INTO tiki_menu_options (menuId,type,name,url,position,section,perm,groupname) VALUES (100,'o','My other option','tiki-my_other_option.php',15,'','','');
```

A menu affects 2 tables. One for the menu by itself, and the other one for the menu options. In the previous example, we created a menu ˜*My menu* with 2 options.
A menu type can be
- d (dynamic collapsed)
- e (dynamic extended)
- f (fixed)

an option type can be:
- s (section)
- r (sorted section)
- o (option)
- - (separator)

The section parameter corresponds the feature. If the feature is not activated, the option will not show up.
The groupname parameter is a filter on the group.

Notice: the menuId 42 is reserved for the application menu (the default Tiki menu).
Trick: we usually don't give a consecutive number to menu options to be able later to add additional options easier.
If an option can be see with 2 perms, you need to add 2 options with the same position and different perm. Idem if it can be see in 2 sections or 2 groups.
Some documentation can be found at Custom Menus
Notice: there are some bugs about the expand/collapse javascript in tw1.9.2.

To insert this menu in a column, you need to create a user menu that uses it (adminTo insert this menu in a column, you need to create a user menu that uses it (adminTo insert this menu in a column, you need to create a user menu that uses it (adminTo insert this menu in a column, you need to create a user menu that uses it (adminTo insert this menu in a column, you need to create a user menu that uses it (adminTo insert this menu in a column, you need to create a user menu that uses it (adminTo insert this menu in a column, you need to create a user menu that uses it (adminTo insert this menu in a column, you need to create a user menu that uses it (adminTo insert this menu in a column, you need to create a user menu that uses it (admin->modules->create a new user menu) and then assign this user menu (admin->modules->assign new module)

1.14.1. Menu created with the Tiki API

```
global $menulib; include_once(˜'lib/menubuilder/menulib.php'); $menulib->replace_menu($menuId, $name, $description, $type); foreach ('optionmenu') { $menulib $menulib->replace_menu_option($menuId, $optionId, $name, $url, $type, $position, $section, $perm, $groupname); }
```

The parameter values are the same than described previously.
Notice: you can notice the use of global $menulib. This is very useful when you use this part of code in a function. The include can have been done somewhere else, and the global gives you the opportunity to catch the variable $menulib.

If you don't know the $menuId you want to use, you need to

```
$menulib->replace_menu(0, $name, $description, $type); $query = 'select max(`menuId`) from `tiki_menus`'; $menuId = $tikilib->getOne($query); foreach (optionmenu) { $menulib->replace_menu_option($menuId, $optionId, $name, $url, $type, $position, $section, $perm, $groupname); }
```

Next, you have to create a user menu and assign it

```
global $modlib; include_once ('lib/modules/modlib.php'); $modlib->replace_user_module($name, $title, "{menu id=$menuId}"); $modlib->assign_module($name, $title, $position, $order, 0, $rows , serialize($groups), $params, $type);
```

$position is *l* (left column) or *r* (right column)
If you assign 2 modules with the same order in the same column, they will appear both in a random order.
$groups is an array of groups (ex: array("Registered", "Anonymous"))
$params is the value you put in the param field in admin->modules->assign module. (ex: 'max=10')
$type is 'D' to assign to everybody.

1.15. Menu created with a smarty template

You can also create a menu with a template module. A good example is templates/modules/mod-application_menu.tpl. This menu is the old menu that has been replaced by a database menu to be able to use the mods.

You only need to create a new template in templates/modules/ with a filename beginning with mod-. This new module will automatically appear in the admin->modules scrolling list.

1.16. To introduce a new permission

1.16.1. Before permission revamp (sometime between Tiki 6 and Tiki 9)

[+]

1.16.2. Current way to add permissions

You know you are after the permission revamp when yous MySQL database *users_permissions* table is empty.

You need to edit **function get_raw_permissions()** in file **lib/userslib.php** and add your permission at the correct place like this:

Sample permission definition

```
array( 'name' => 'tiki_p_blog_post', 'description' => tra('Can post to a blog'), 'level' => 'registered',
'type' => 'blogs', 'admin' => false, 'prefs' => array('feature_blogs'), 'scope' => 'object', ),
```

level	can be editors, registered, admin (seems mostly deprecated)
type	is the group of perms (wiki, cms, blog..)
admin	is true for the perm that admins the whole type (note naming convention tiki_p_type_admin)
prefs	array of features which need to be active for the permission to be active
scope	??? TODO: explain
apply_to	??? TODO: explain

Then you need to *Clear Tiki Caches* and the new permission will show up in tiki-objectpermissions.php admin interface.
You can create a new group of perms there. There is no other requirement for a group of perms.
NOTE: This is the order they appear in tiki-objectpermissions.php admin interface and it's important to keep them grouped by 'type'

1.16.3. Usage

In a php, the perm is used like this:

```
if ($tiki_p_whatever == 'y')
```

Don't forget to compare to a value; a common error is to write only

```
if ($tiki_p_whatever)//wrong code
```

We usually name a perm with a name beginning by tiki_p_

1.17. To check a permission

1.17.1. Check perms before permission revamp (sometime between Tiki 6 and Tiki 9)

[+]

1.17.2. Current way to check permissions

Introduced in version 4: Permission Revamp
Permission Cleanup has been merged into trunk for release in Tiki 4.0, as part of the Workspace. Note that this page is being included dynamically from the Hello World wiki page.

It was previously in branches/experimental/perms-take2. Implementation details can be found in the code, especially in Perms.php

The objective of the clean-up is to provide an homogeneous interface to access permissions in a way that is simple and efficient. The interface used should not reflect how the permissions are stored. The previous interfaces used multiple functions across different libraries that were confusing and caused frequent WYSIWYCA problems in addition to being inefficient when filtering lists of objects.

The permissions used on categories were confusing and lacked the customizability expected in TikiWiki. The new layer uses exactly the same permissions on objects, categories and global permissions.

The redesign offers fully tested code with guaranteed behaviors.

From old to new

Validate a global permission

Previous code

```
<?php // $tiki_p_* variables made available in tiki-setup.php if( $tiki_p_edit_article == 'y' ) { // ... } ?>
```

Replacement

```
<?php // Available globally, by tiki-setup.php $globalperms = Perms::get(); if(
$globalperms->edit_article ) { // ... } ?>
```

The previous method caused a lot of pollution of the global scope and forced the usage of multiple global variables inside functions. The new API allows to obtain a permission accessor from anywhere. The global permissions obtained will be the true global permissions, not those that may be overridden by other calls in the page.

Validate an object permission

Previous code

```
<?php // $tikilib and $user defined in tiki-setup.php if( $tikilib->user_has_perm_on_object( $user,
'HomePage', 'wiki page', 'tiki_p_view', 'tiki_p_view_categorized' ) ) { // ... } ?>
```

Replacement

```
<?php $objectperms = Perms::get( array( 'type' => 'wiki page', 'object' => 'HomePage' ) ); if( $objectperms->view ) { // ... } ?>
```

Obtaining permissions on an object now follows the exact same pattern as obtaining global permissions. No need to remember which function had to be called and on which library it was located, what is the order of the parameters and which permission applies if it has to be resolved through categories.

Because the global permissions are loaded at the beginning of the script, verifying permissions on an object will take at most 3 additional queries.

- One query to verify object permission
- On miss, one query to obtain the categories on the object
- One query to obtain permissions on those categories, if applicable and not previously encountered.

The previous amount of queries qualifies as *hard to tell* due to caching schemes and complex control flows. However, it was likely to be up to 5 in normal cases, plus checks on parent categories.

Filter a list of objects

This code is not exactly accurate because the actual filter code was moved inside list_pages() between 2.0 and 3.0. However, the concept remains and the current code within list_pages() will be updated to benefit from the new API.

Previous code

```
<?php $pages = $tikilib->list_pages(); $filtered = array(); foreach( $pages as $page ) { if( $tikilib->user_has_perm_on_object( $user, $page['pageName'], 'wiki page', 'tiki_p_view', 'tiki_p_view_categorized' ) ) { $filtered[] = $page; } } ?>
```

Replacement

```
<?php $pages = $tikilib->list_pages(); $filtered = Perms::filter( array( 'type' => 'wiki page' ), 'object', $pages, array( 'object' => 'pageName' ), 'view' ); ?>
```

The previous code had to verify the permissions for each object individually. Considering an average of 3 queries per object, which is likely to be on the lower end of the spectrum, filtering a list of 30 objects would have required 90 queries.

The replacement benefits from bulk loading, also accessible through Perms::bulk() if the objective is not list filtering, and will filter the entire list in 3 queries or less, just like it would for a single object.

Key features

- Handling of lists in a constant amount of queries
- Re-use of loaded rules when possible
- Less reliance on global variables
- Removal of redundant tiki_p_ permission prefix
- Single point of definition of rules
- Extensible rule system hidden behind a facade allowing to update rules without affecting the rest of the code.

- Use different rules in unit tests to validate code without affecting the database.

Practical usage

List of object types

From a look at lib/tikilib.php, it seems that the available object types are:

- 'wiki page'
- 'tracker'
- 'blog'
- 'map'
- 'forum'
- 'file gallery'
- 'image gallery'
- 'topic'
- 'calendar'
- 'comments'
- 'map_changed'
- 'category_changed'

Extensibility

- Composable rules. Add or remove resolution rules from configuration. For example, removing a single line can disable object permissions. They can even be changed at runtime.
- Creation of new rules to lookup permissions differently. Global, Category and Object fit in this layer.
- Smarter resolvers. Want to introduce dependencies in permissions?
- Fallback resolvers to handle special cases, like creator permissions, admin permissions.
  - Direct : Default check on exact permission
  - Indirect : Check an alternate, like parent, permission instead
  - Creator : Check an alternate permission when the active user is the creator of the object (must be included in the context)
  - *More?*

Because Resolver is an interface and verifying permissions only rely on it, everything can be changed without affecting the rest of the application, as long as permission names remain the same. However, a compatibility layer can be added even to that.

ResolverFactories generate the resolvers. These look up in the data source. More can be created and composed in different ways.

Set-up

```
<?php $groups = array( ... ); $perms = new Perms; $perms->setGroups( $groups );
$perms->setResolverFactories( array( new Perms_ResolverFactory_ObjectFactory, new
Perms_ResolverFactory_CategoryFactory, new Perms_ResolverFactory_GlobalFactory, ) );
$perms->setCheckSequence( array( new Perms_Check_Direct, new Perms_Check_Indirect( array(
'view' => 'wiki_admin', // ... ) ), new Perms_Check_Creator( $user ), ) ); // Activate Perms::set( $perms );
?>
```

Additional considerations

Related to these changes are the changes to categories. To remain efficient, category lookup on objects has to be limited to one query and so does the permission lookup on categories. The current adjacent node model typically requires multiple queries to dig down the hierarchy. At this time, assigning permissions on a root category can optionally copy the permissions down to every node. The maintenance cost is a little higher, but it allows for faster lookup fitting the performance requirements.

For this case, which is the current implementation, only the direct categories are considered in the category lookup.

An alternative keeping permissions on the top level nodes only would be the nested set model.

Common problems

Mix up between global and local permissions. In multiple files, the object permissions overwrite the global permission variables. Effectively, this ignored object-level permissions. This can work correctly as part of a transition. However, the override must be done **before** any permissions are checked. TikiLib::get_perm_object($id, $type) is the recommended method to override global variables.

Custom code to validate permissions. Most of it must have been removed by now to be replaced with get_perm_object, but a lot of copy-pasted code contained logic to verify permissions that was outdated in many cases. Any chunk of code containing object or category permission logic should be removed from tiki-*.php should be replaced with the appropriate calls.

Listings require global permissions in many cases. The correct behavior would be to allow if any object can be listed. Because of the permissions on object and categories, this may be hard to determine. At this time, the permissions should be granted globally and restricted locally by adding category permissions. In the future, as part of the improved listing filtering, it should be possible to solve this one correctly. However, this is not expected before 5.0.

Troubleshooting

Because of the dynamic nature of the code, attempting to print values randomly from within the objects will not be very helpful. The behavior of the code is based on **object composition** rather than procedural code. The libraries must be treated as black boxes. Their correctness is demonstrated by the extensive unit test suite.

An important thing to know is that once an accessor is built, through Perms::get(...), it contains all the information it needs to resolve permissions for any set of groups that may be provided to it. They are also completely independent. Modifying their state will not affect the rest of the system. Thus, a simple **print_r** or **var_dump** on the accessor will show all there is to know.

Here is a simple output and how to interpret it.

Sample print_r of an accessor

```
Perms_Accessor Object ( [resolver:private] => Perms_Resolver_Static Object ( [known:private] => Array ( [Anonymous] => Array ( [view] => 1 [forum_read] => 1 [forum_post] => 1 [forum_post_topic] => 1 [post_comments] => 1 [read_comments] => 1 [wiki_view_comments] => 1 ) [Admins] => Array ( [admin] => 1 ) ) ) [prefix:private] => tiki_p_ [context:private] => Array ( ) [groups:private] => Array ( [0] => Admins [1] => Registered ) [checkSequence:private] => Array ( [0] => Perms_Check_Alternate Object ( [permission:private] => admin [resolver:private] => Perms_Resolver_Static Object (
[known:private] => Array ( [Anonymous] => Array ( [view] => 1 [forum_read] => 1 [forum_post] => 1
[forum_post_topic] => 1 [post_comments] => 1 [read_comments] => 1 [wiki_view_comments] => 1 )
[Admins] => Array ( [admin] => 1 ) ) ) ) ) [1] => Perms_Check_Direct Object ( ) [2] =>
Perms_Check_Indirect Object ( [map:private] => Array ( [ws_view] => ws_admin [ws_removews] =>
ws_admin [ws_adminws] => ws_admin // ... removed for readability ... [admin_integrator] => admin
[admin_dynamic] => admin [admin_banning] => admin [admin_banners] => admin [add_object] =>
admin_categories [access_closed_site] => admin ) ) [3] => Perms_Check_Creator Object (
[user:private] => admin [key:private] => creator [suffix:private] => _own ) ) )
```

At the very top, the **resolvers** property contains the set of rules that apply for the requested object. In this case, global permissions were obtained. However, there are no differences with object or category permissions. The ResolverFactories always obtain the permissions and provide them as a static resolver. The content is simply a map between groups and the permissions that are granted to them.

The prefix is an optional **prefix** that can be used when checking permissions. $perms->view or $perms->tiki_p_view will provide the exact same permission. This was used for backward compatibility.

The **context** is the array that was used to obtain the permissions. In this case, the context is empty because it was fetched for the global scope. However, the object may also contain an object type, id and creator. Other properties could be used in the future if needed. The context is essentially used by the resolver factories to identify which one applies and where to obtain the information from. It is kept in the accessor as a reference, but can also be used by the check rules.

The **groups** property simply contains the list of groups currently being verified. These can be changed using setGroups(). By default, they are set to the groups of the current user by Perms::get() for convenience. Except for a few rare cases, permissions are checked for the currently active session.

The **check sequence** determines how the permissions will be verified. As the name indicates, they are verified sequentially. The verification stops as soon as one of them provides a positive response. In this sequence, there are 4 steps:

- The alternate check first verifies if the groups verified have the global admin privilege. To perform this verification, the check contains a reference to the global resolver. This check does not use the context's resolver at all. By being the first check made, it insures that the global admins will always be allowed to perform any action.
- The direct check verifies if the permission is granted directly by the resolver.
- The indirect check in this case is configured to verify the admin privilege related to the privilege currently validated. A user with admin_wiki will be granted all rights related to the wiki feature. These rules are loaded from the users_permissions table.
- The creator check is not widely used at this time, but would allow for a standardized _own permission naming convention. It verifies an alternate permission if the current user matches the creator in the context.

Responsibilities

The Perms component was designed separate the concerns of the user land and the internal rules. As part of normal usage, one only needs to know of Perms::get() to obtain the set of rules they should use and of the property accessing technique to verify individual permissions. More advanced usage may include list filtering, as mentioned earlier.

The accessors provided by Perms::get() are meant to be self-contained.

Typical usage

```
<?php $perms = Perms::get( array( 'type' => 'wiki page', 'object' => $page ) ); if( isset($_POST['save'])
&& $perms->edit ) { // Do stuff }
```

The Perms class acts mainly as a facade to the subsystem. It holds the rules and the default values to be assigned in order to keep the code simple in the rest of the system. It's primary purpose is to build accessors. Most of the values required, like the default groups to assign and the check sequence to be performed, are provided as configurations. However, the central component in the accessor is the resolver. The resolvers are independent rules that apply for a given object, or context. These resolvers are obtained through the resolver factories.

When queried, the resolver factories inspect the context and identify if they need to make a verification on it. Each factory verifies a very specific set of rules. As they are validated in sequence, the first one to provide a set of rules will be the ones that are used. In a typical installation, the sequence is the following:

- object to verify if permissions were assigned directly on the context object,
- category to verify if any of the categories in which the object is in contain any permission,
- global as a fallback.

The resolvers are also responsible of the efficient fetching of their resources, especially when requested in bulk. As an example, the category factory first fetches the list of all categories that apply to the requested objects. It then verifies which of those categories it has not yet fetched the permissions for and fetch the permissions for all of those categories in a single query and index them. Afterwards, it gathers the permissions from it's internal cache to build the list of the individual objects.

1.17.3. Special case 1: File Gallery Owners

Some information to come

1.17.4. Special case 2: Tracker Field permissions

Tracker fields can have their own permissions with the following options:
Visibility:

- n: Visible by all
- r: Visible by all but not in RSS feeds
- y: Visible after creation by administrators only
- p: Editable by administrators only
- c: Editable by administrators and creator only
- i: Immutable after creation

Visible by - comma separated list of groups whose members can view the field and its contents.
Editable by - comma separated list of groups whose members can edit the field contents.

Code that handles these permission checks is located in Tracker_Item: *canViewField* and *canEditField*.
Implementing these permisions outside the Tracker_Item class requires understanding on how exactly they are enforced in that class.
Some of the permissions give access only to item owners. See next section for retrieving the tracker item owners.

1.17.5. Special case 3: Tracker Item Owners (User Selector)

Item owner or owners are the user(s) selected from any User Selector field which is given the Item Owner option. *Tracker_Definition:getItemOwnerFields* is used to retrieve these fields. Their contents are csv-formatted list of user logins. There are convenience methods for retrieving item owners:

Tracker_Definition:getItemUsers and *TrackerLib:get_item_creators*. Tracker_Item automatically retrieves the owners at initialization time.

1.17.6. Special case 4: User Pages

Some information to come

1.17.7. Special case 5: Parent-child level permissions

Since Tiki 18, permission subsystem has been extended to support parent-child permission relations. As of Tiki 18, only Tracker->Item relation is supported but there is a work in progress to add others. Parent-child permission relations work by extending the default resolver factories that are checked when resolving object permissions. The default chain is Object Level -> Category Level -> Global Level. This means that if a specific object permission is not set, category level permission is checked. If it is not set as well, global level one is looked up. Introduction of parent tracker permission check extends this chain: Object Level -> Category Level -> Parent Object Level -> Parent Category Level -> Global Level. This means that tracker-level permissions will be applied to individual tracker items if those items don't have specific categorization and permissions on those categories exist. Similarly, if tracker is categorized and permissions exist on that category, tracker items will inherit these permissions unless they have their own object or category level permissions.
Example:

```
Perms::get(array('type' => 'trackeritem', 'object' => ID))
```

Returns permissions in this order:
1. TrackerItem Object permissions (if any).
2. TrackerItem Category permissions (if item is categorized and permissions are available)
3. Parent Tracker Object permissions (if any).
4. Parent Tracker Category permissions (if tracker is categorized and there are any permissions)
5. Global permissions.

1.18. To wiki parse a textarea

Imagine you have a textarea and you want this textarea to accept wiki format.
To parse the textarea, you will have to call

```
$tikilib->parse_data($text)
```

Usually if you have a choice like â€œit is htmlâ€, you need to add an additional parameter

```
$tikilib->parse($test, $is_html)
```

PS: some tikwiki1.9 have bugs around the is_html parameter. There are patched by replacing $test with htmlspecialchars($test);

If you want to add some quicktags to the textarea

```
include_once ('lib/quicktags/quicktagslib.php'); $quicktags =
$quicktagslib->list_quicktags(0,-1,'taglabel_desc','','wiki'); $smarty->assign_by_ref('quicktags',
$quicktags["data"]); $smarty->assign('quicktagscant', $quicktags["cant"]);
```

In the templates in order to use the quicktags, you have to add in the template where you want the quicktags to appear:

```
{include file="tiki-edit_help_tool.tpl"}
```

PS: only one quicktag zone per page is allowed

If you want to add the resize buttoms, you have to add in the php:

```
include_once("textareasize.php");
```

And in your tpl, you have to add something like this:

```
<form id='editpageform'> {include file="textareasize.tpl" area_name='editwiki'
formId='editpageform'} <textarea id='editwiki' rows="{$rows}"
cols="{$cols}">{$pagedata|escape}</textarea> <input type="hidden" name="rows"
value="{$rows}"/> <input type="hidden" name="cols" value="{$cols}"/> </form>
```

1.19. Categories

How to make your feature use categories.

Get an object's array of categories

```
$categlib = TikiLib::lib('categ'); $objectCategoryIds =
$categlib->get_object_categories($objectType,$objectId);
```

more docs need to be written

1.20. Visual themes

Tiki, integrating the Bootstrap CSS Framework, fully and flexibly supports theme stylesheets for page element layout and styling which can be applied in a number of ways - one theme specified by admin for the site globally, the option of allowing users to select a theme to use, theme stylesheets being applied separately to various site features, categories, pages, and so on, as well as other implementation options.

With a few feature-specific exceptions, Tiki's CSS stylesheets are **not** edited directly. They are compiled from relevant SCSS files, so to make a change in or addition to a theme stylesheet or other CSS file, the relevant SCSS file should be edited.

Related links:

- Using Less CSS with Tiki

- https://dev.tiki.org/PhpStorm#File_Watcher_Settings_using_scss_integrated_with_composer_
- https://themes.tiki.org/Updating-a-Tiki-theme-from-Bootstrap-3-to-4#Less_to_SCSS_Sass_
- https://getbootstrap.com/
- https://sass-lang.com/

1.21. Search

Tiki has 2 search features. One that is based on the mysql fulltext search feature and the other one that is Tiki feature. (see http://doc.Tiki.org/Search+Admin for more details)

1.22. To add a new object in a MySQL search

First, you have to add a fulltext index in the database.
For instance, you have a table

```
CREATE TABLE contribution ( title varchar(80) default NULL, body text ) TYPE=MyISAM;
```

If you want to be able to search on the title and the body field, you will need to add a fulltext index

```
CREATE TABLE contribution ( title varchar(80) default NULL, body text, hits int(8) default NULL, lastModif int(14) default NULL, FULLTEXT KEY ft (title ,body) ) TYPE=MyISAM;
```

Mysql will automatically takes care of the indexation in real time.
You can put as many fields as you want

Notice: if a field is in a wiki format and if you want to index what you see, you will need to have another field that contains the parsed field. Even in this case, you can have some difference (for instance if your field contains a last module format, only the result at indexation time will be indexed). Usually, in Tiki we index the field not parsed as.. in IMHO no choice is perfect.

Second, you have to add the search function in lib/searchlib.php

```
function find_contributions($words = '', $offset = 0, $maxRecords = -1, $fulltext = false) { static $search_contributions = array( 'from' => '`tiki_contributions` c, 'name' => 'c. `title`', 'data' => 'c. `body`', 'hits' => 'c. `hits`', 'lastModif' => 'c. `lastModif`', 'href' => 'tiki-view_contribution.php?contributionId=%d', 'id' => array('c. `contributionId`'), 'pageName' => 'c. `title`', 'search' => array('c.`title`', 'c.`body`'), 'permName' => 'tiki_p_view_contribution', 'objectType' => 'contribution', 'objectKey' => 'c.`title`', ); return $this->_find($contributions, $words, $offset, $maxRecords, $fulltext); }
```

'name' is the column that contains the name of the object
'search' must be the list of fields of the mysql fulltext index.

Third, you need to add a call to the function in templates/modules/mod-search_box.tpl

```
{if $feature_contribution eq 'y'} <option value="contributions">{tr}Contributions{/tr}</option> {/if}
```

Also in templates/tiki-searchresults.tpl

```
{if $feature_contribution eq 'y'} <a class="linkbut" href="tiki-searchresults.php?highlight={$words}&amp;where=contributions">{tr}Contributions{/tr}</a> {/if} ... <form class="forms" method="get" action="tiki-searchresults.php"> ... {if $feature_articles eq 'y'} <option value="contributions">{tr}Contributions{/tr}</option> {/if} </form>
```

And in tiki-searchresults.php

```
if ($where == 'contributions' && $feature_contribution != 'y') { $smarty->assign('msg', tra("This feature is disabled").": feature_contribution"); $smarty->display("error.tpl"); die; }
```

And you need to add the search on this object in the global search. In lib/searchlib.php, you need to update the function find_pages

```
function find_pages($words = '', $offset = 0, $maxRecords = -1, $fulltext = false) { ... if ($feature_contribution == 'y') { $rv = $this->find_contributions($words, $offset, $maxRecords, $fulltext); foreach ($rv['data'] as $a) { $a['type'] = tra('Contribution); array_push($data, $a); } $cant += $rv['cant']; } }
```

If you data are a little more sophisticated, you can do some join in the search. This is the faqs example

```
function find_faqs($words = '', $offset = 0, $maxRecords = -1, $fulltext = false) { static $search_faqs = array( 'from' => '`tiki_faqs` f , `tiki_faq_questions` q', 'name' => 'f. `title`', 'data' => 'f. `description`', 'hits' => 'f. `hits`', 'lastModif' => 'f. `created`', 'href' => 'tiki-view_faq.php?faqId=%d', 'id' => array('f. `faqId`'), 'pageName' => 'CONCAT(f. `title`, ": ", q. `question`)', 'search' => array('q. `question`', 'q. `answer`'), 'filter' => 'q. `faqId` = f. `faqId`', 'permName' => 'tiki_p_view_faqs', 'objectType' => 'faq', 'objectKey' => 'f.`title`', ); return $this->_find($search_faqs, $words, $offset, $maxRecords, $fulltext); }
```

NO additional perm checking is needed. The function does the perm checking.

1.23. To add a confirmation step

Notice: We consider here only the "Protect against CSRF with a ticket:" feature in admin->login. The "Protect against CSRF with a confirmation step:" was supposed to disappear.

```
if ($_REQUEST["action"] == 'delete') { $area = 'delete_object'; if ($feature_ticketlib2 != 'y' or (isset($_POST['daconfirm']) and isset($_SESSION["ticket_$area"]))) { key_check($area); ...delete... } else { key_get($area); } }
```

This works only if the delete parameters are passed directly in the URL. Key_get lists the parameters passed through a form input. In this case, you have to copy the parameters with the limit of URL size (it is why usually the multiple delete has no confirmation step

Here is an example from the multiple remove in admin->users

```
if ($_REQUEST["submit_mult"] == "remove_users") { $area = 'batchdeluser'; if ($feature_ticketlib2 == 'n' or (isset($_POST['daconfirm']) and isset($_SESSION["ticket_$area"]))) { key_check($area); foreach ($_REQUEST["checked"] as $deleteuser) { $userlib->remove_user($deleteuser); $tikifeedback[] = array('num'=>0,'mes'=> sprintf(tra("%s <b>%s</b> successfully deleted."),tra("user"),$deleteuser)); } } elseif ( $feature_ticketlib2 == 'y') { $ch = ""; foreach ($_REQUEST['checked'] as $c) { $ch .= "&amp;checked[]=".urlencode($c); } key_get($area, "", "tiki-adminusers.php?submit_mult=remove_users".$ch); } else { key_get($area); }
```

- To protect your application against sea surf(CSRF - XSS)

Each Tiki page must include a protection against sea surfing.
For instance tiki-action_object.php

```
<?php require_once('tiki-setup.php'); if ($_REQUEST['action_object']) { check_ticket('action_object'); // execute your action } ask_ticket('action_objet'); $smarty->assign('mid', 'tiki-action_object.tpl'); $smarty->display('tiki.tpl'); ?>
```

1.24. To create a new module

Modules are the boxes typically found on right & left sides of portal-style sites. They are great to re-use content. In Tiki, contrary to similar web applications, modules are typically built-in (bundled) in Tiki. You can pick from the list or you can create your own.

To create a new module, the minimum thing you need to do is to create a template file under the templates/modules/ (or templates/styles/your_style/modules/) with a filename beginning with mod- and a .tpl extension.
Example, a contribution module will be
templates/modules/mod-contribution.tpl

```
{tikimodule title="{tr}Contribution{/tr}" name="contribution" flip=$module_params.flip decorations=$module_params.decorations} <div align="center"> My contribution </div> {/tikimodule}
```

Notice the use of the smarty function tikimodule that will set up the box, the title and the box preferences.

If you need to set up some variables, you need to create a php file
modules/mod-contribution.php

```
<?php $smarty->assign('my_contribution_text', 'hello world'); ?>
```

The template that uses this variable will look like

```
{tikimodule title="{tr}Contribution{/tr}" name="contribution flip=$module_params.flip decorations=$module_params.decorations} <div style="text-align: center"> {$my_contribution_text} </div> {/tikimodule}
```

Notice: This is a simplified version. This simple program allows only to have one occurrence of the module (there is only one variable $my_contribution_text. And in 1.9, the modules are computed at the beginning in tiki-setup.php that don't allow to use variable set in the main program.

If you want to use the parameters that are set up in the admin->modules-> param module line or in the MODULE plugin, you can use

---

```
<?php $smarty->assign('max', $module_params['max']); ?>
```

where max is either a param of MODULE or a param inserted in the admin->modules->param line.

1.25. To create a new plugin

Also see Code Howto Create a Wiki Plugin

Plugins are an extension to basic wiki syntax. Despite the name, in Tiki, contrary to similar web applications, plugins are typically built-in (bundled) in Tiki. You can pick from the list or you can create your own.

To create a new plugin, you need to create a file in `lib/wiki-plugins/` with a filename beginning with `wikiplugin_` and an extension `.php`. (while the template is in `templates/wiki-plugins/*.tpl`). The filename should contain only lower case letters.

Example:
`lib/wiki-plugins/wikiplugin_helloworld.php`

```
<?php <?php //the info function is needed for tw >= 3.0 function wikiplugin_helloworld_info() { return array( 'name' => tra('Helloworld'), 'documentation' => 'PluginHelloWorld', // will add help link to https://doc.tiki.org/PluginHelloWorld 'description' => tra('Test.'), 'format' => 'wiki', // html or wiki: must be lower case; not any other value accepted 'prefs' => array( 'wikiplugin_helloworld', ), // each plugin can have an admin setting(pref) to enable them // this is the default for wiki plugins. 'body' => tra('What is to be placed inside the plugin\'s body'), 'validate' =>'all', //add this line if each insertion of this plugin needs to be validated by an admin, // (because new or modified plugin body or plugin arguments is found). Possible values are: 'all', body' or 'arguments'. 'params' => array( 'title' => array( 'required' => false, 'name' => tra('Title'), 'description' => tra('Describe what title is'), 'filter' => ('text'), 'default' => (''), ), ), ); } function wikiplugin_helloworld($data, $params) { $title = '__'.$params['title'].'__'; return "Hello $title $data!"; }
```

In a wiki page use

```
{HELLOWORLD(title=Mr)}Tiki{HELLOWORLD}
```

The display will be

Hello World **Mr** Tiki!

Note: The best is to try it first on an otherwise empty wiki page. That guarantees that your new wiki-plugin will be recognized. When your plugin is displayed at first time, you might be asked if you wanted to enable the new plugin (if in admin mode).

A plugin consists of 2 functions. The name of the functions must be formatted. They begin with `wikiplugin_`, then the name of the plugin in lower case, then `_info` for the help/info function.

The info function provides a descriptor for the plugin. The descriptor is used to generate the documentation, the edit UI for the plugins and provides processing instructions explaining how to handle the input, process the output and execute the plugin.

The process function has 2 parameters, the text inside the tags data and the list of parameters param as a PHP array.

Note: Many plugins in Tiki use extract() on the $params array. When done, EXTR_SKIP must be used. However, using extract is discouraged.

By default, the value returned by the function will be wiki parsed. Chunks of HTML provided by the plugin must be enclosed in ~np~ ... ~/np~ to avoid further parsing. If the entire output is HTML, **'format' => 'html',** can be added in the description function to skip parsing of the output.

Alternatively, the plugins may return WikiParser_PluginOutput objects generated through the various factories available in `lib/core/lib/WikiParser/PluginOutput.php`. These objects will provide standard output for common errors.

If you have more than one parameter, the wiki-plugin use will be

```
{HELLOWORLD(title=Mr, name=Tiki)}How are you?{HELLOWORLD}
```

the extract function in the PHP code will set for you all the parameters and you will be able to use `$params['name']` in the same way you use `$params['title']` in the PHP code.

If you want to be clean - of course - you have to test the param has been set

```
... function wikiplugin_helloworld($data, $params) { if (!isset($params['title'])) { return tra("Missing
parameter title"); } return "Hello World {$params['title']} $data!"; ... }
```

Now if you want to do more pretty display with smarty, you will use such a code

```
<?php .... the help stuff (with format => html) function wikiplugin_helloworld($data, $params) { global
```

> $smarty; if (! isset($params['name'])) { return tra('Missing parameter name'); } $smarty->assign('name', $params['name']); return $smarty->fetch('wiki-plugins/wikiplugin_helloworld.tpl'); }

and your tpl file `templates/wiki-plugins/wikiplugin_helloworld.tpl`

> <b>Hello World</b> {$name}

The info function is required for all plugins.

> ... function wikiplugin_helloworld_info() { return array('name' => tra('HelloWorld'), 'description' => tra('Greets person'), 'prefs' => array('wikiplugin_helloworld'), 'params' => array('name' => 'the name of the person to greet"),); }

If you want the plugin to be enabled by default the prefs entry will require an appropriate addition in two locations:

1- in lib/setup/prefs.php

> ... 'wikiplugin_helloworld' => 'y',

And 2- according to instructions are Preferences.

These two elements shall enable the control of the feature from the admin panel.

Note: If you try a more complex plugin code and get a blank page, then the best is to look if all the brackets and bracelets match. Check also that all your variables have their $ prefix.

1.25.1. Requiring validation of plugin calls

If you need that an admin validates each new or modified plugin call, you can use the plugin parameter **validate** (see example above for the HelloWorld plugin).

Possible values are: 'arguments', body' or 'all' (to monitor changes in both arguments and body).

1.25.2. Plugin help UI

Please see: PluginUI

1.25.3. Webservice Plugin

Please see: Webservices

1.26. Plugin filters

In order to have more security on the input added by users on plugin parameters, you can add the filter on each plugin param code, so that its content added by a user is validated against some sanitation checks. The current filter parameters, as defined in ./lib/core/TikiFilter.php are:

Filter name	Comments	Since
alpha	Removes all but alphabetic characters	
alphaspace	Removes all but alphabetic characters and spaces	
alnum	Only alphabetic characters and digits. All other characters are suppressed. I18n support	
digits	Removes everything except digits eg. '12345 to 67890' returns 1234567890	
int	Transforms a sclar phrase into an integer. eg. '-4 is less than 0' returns -4	
username	Strips XML and HTML tags	
pagename		
topicname		
themename		
email		
url	it will allow also using wiki argument variables such as { {itemId} }	
text		
date		
time		
datetime		
striptags		
word		
xss		
purifier		
wikicontent		
rawhtml_unsafe	passthrough filter - no security at all, meaning it **must** be handled otherwise. Use with care (read: avoid).	
lang		
imgsize		
attribute_type		

1.27. A select all checkbox

You have a table and each line is identified by an id
Put if not the table in a form.
You may adapt $objects, $ix to your need

```
<form> ..... <table> ....... {section name=ix loop=$objects} .... <input type="checkbox"
name="checked[]" value="{$objects[ix].id|escape}" {if $smarty.request.checked and
```

in_array($objects[ix].id,$smarty.request.checked)}checked="checked"{/if} /> .... {/section} <xxxxx type="text/javascript"> /* <![CDATA[*/ document.write('<tr><td colspan="..."> <input type="checkbox" id="clickall" yyyyy="switchCheckboxes(this.form,\'checked[]\',this.checked)"/>'); document.write('<label for="clickall">{tr}select all{/tr}</label> </td></tr>'); /*]]> */</xxxxx> ... </table> <div> {tr}Perform action with checked:{/tr} <input type="image" name="delsel" src='img/icons/cross.png' alt={tr}delete{/tr}' title='{tr}delete{/tr}' /> </div> </form>

replace xxxx with script and yyyy with onclick
in Tiki<9.0, the icon was img/icons/cross.png

In php you test

```
if (isset($_REQUEST['delsel_x']) && isset($_REQUEST['checked'])) { ....
foreach($_REQUEST['checked'] as $id) { ... } }
```

1.28. To do something specific in a template (ex.: tiki.tpl) conditional to the current item being in a category. Ex.: different header picture.

This is an example how to display in the top page something specific to the categories of the current object
This function is not completly functionnal. It needs that object set a php variable $objectId or $objectid or $objId and $objecType or $section. But if you need you can easily add a $objectId = $forumId for instance in the .php
Tiki>=3.0 this function has been introduced in tiki code and is called if the feature admin->category->'Categories used in tpl' has been set

```
{if !isset($objectCategoryIds)} {php} global $smarty, $section, $objId, $objectId, $objectid, $objectType, $cat_objid, $cat_type, $objectCategoryIds; global $tikilib; include_once('lib/tikilib.php'); global $categlib; include_once('lib/categories/categlib.php'); if (empty($objectType)) { if (!empty($cat_type)) { $objectType = $cat_type; } elseif (!empty($section)) { $objectType = $section; } if ($objectType == 'wiki') { $objectType = 'wiki page'; } elseif ($objectType == 'cms') { $objectType = 'article'; } elseif ($objectType == 'file_galleries') { $objectType = 'file gallery'; } elseif ($objectType == 'trackers') { $objectType = 'tracker'; } elseif ($objectType == 'forums') { $objectType = 'forum'; } } if (!empty($objectType) && empty($objectId)) { if (!empty($objId)) { $objectId = $objId; } elseif (!empty($objectid)) { $objectId = $objectid; } elseif (!empty($cat_objid)) { $objectId = $cat_objid; } elseif (!empty($_REQUEST['galleryId']) && $objectType == 'file gallery') { $objectId = $_REQUEST['galleryId']; } elseif (!empty($_REQUEST['trackerId']) && $objectType == 'tracker') { $objectId = $_REQUEST['trackerd']; } elseif (!empty($_REQUEST['forumId']) && $objectType == 'forum') { $objectId = $_REQUEST['forumd']; } elseif (!empty($_REQUEST['page']) && $objectType == 'wiki page') { $objectId = $_REQUEST['page']; } } if (!empty($objectId) && !empty($objectType)) { $objectCategoryIds = $categlib->get_object_categories($objectType,$objectId); $smarty->assign('objectCategoryIds', $objectCategoryIds); } {/php} {/if} {*OTHER_TPL {if empty($objectCategoryIds)}{php}global $smarty, $objectCategoryIds; if (!empty($objectCategoryIds)) $smarty->assign('objectCategoryIds', $objectCategoryIds);{/php}{/if} *} {if (empty($objectCategoryIds) or empty($objectCategoryIds[0]))} ...... {elseif $objectCategoryIds[0] eq '13'} ...... {/if}
```

If you are not using the last part in the same template, to uncomment the {*OTHER_TPL *}

1.29. Smarty variable you can use in a tpl

{$user}
{$default_group}
{$section}
{$page}
{$prefs} for tiki>=2.0

1.30. Escape in smarty

- In an option we need to escape all textual values like this

```
<option value={$name|escape}>{$name|escape}</option>
```

- In an url, we need to escape all the textual values like this

```
<a href="xx.php?name={$name|escape:'url'}>{$name|escape}</a>
```

Self link in templates

tiki >=2.0
If you want to write in a template a link to the same page but with an additional parameter or replace the value of the parameter, you can use

```
<a href="{$smarty.server.PHP_SELF}?{query archive="y"}">LINK</a>
```

1.31. Useful functions to write a script

First begin your script with

```
include_once('tiki-setup.php');
```

Then some useful functions - only the minimum of params are given - see the libraries fiels for more option

```
// create a new group global $userlib; include_once('lib/userslib.php');
$userlib->add_group($groupName, $description); (in tw <= 1.9, you need a 3rd param
$homePageName that can be '') //and include them $userslib->group_inclusion($groupName,
$includeGroupName); //(where $groupName='Registered' and $includeGroupName='Anonymous' for
example) // create a new user global $userlib; include_once('lib/userslib.php');
$userlib->add_user($userName, $password, $email); // create a new forum global $commentslib;
include_once('lib/commentslib.php'); $forumId = $commentslib->replace_forum(0, $forumName,
$description); (in tw <=2.0, the list of parameters is longer replace_forum(0, $forumName='',
$description='', 'n', 120, 'admin', '', 'n', 'n', 2592000, 'n', 259200, 10, 'lastPost_desc', '', '', 'y', 'y', 'n', 'y',
'y', 'n', 'n', '', 110, '', '', '', 'n', 'n', '', 'n', 'n', 'y', 'y', 'n', 'n', 'n', 'n', 'all_posted', '', '', 'n', 'att_no', 'db', '',
1000000, 0) // create a new calendar global $calendarlib; include_once ('lib/calendar/calendarlib.php');
$calendarId = $calendarlib->set_calendar(0, $creatorUser, $calendarName, $description); (in tw
<=1.9, one more parameter is needed $calendarId = $calendarlib->set_calendar(0, $creatorUser,
$calendarName, $description, array('custompriorities'=>'n')); // assign a user to a group global
```

$userlib; include_once('lib/userslib.php'); $userlib->assign_user_to_group($userName, $groupName); // assign perm to an object global $userlib; include_once('lib/userslib.php'); $userlib->assign_object_permission($groupName, $objectId, $objectType, $permName); with couple like ($forumId, 'forum'), ($calendarId, 'calendar') and with permName like 'tiki_p_view'

1.32. Naming convention

- Naming Convention

1.33. Composer and managing dependencies

See:
https://dev.tiki.org/Composer#Managing_dependencies

1.34. Related links

- Code Maps and Howtos
- Modularity
- TimeZones
- performance
- Security
- WebCodingStandards
- Mass spelling correction
- External Libraries
- The Twelve-Factor App: A methodology for building modern, scalable, maintainable software-as-a-service apps

1.35. Credits

Author: sylvie g

Sponsors: Gerhard Brehm, Marc Laporte