DaytonTikiCore

Note: Posting on behalf of Dayton. This is the entire email that Dayton sent to the dev list

Folks,

I thinks it's a little rude to post a large message to an entire list. So apologize. But I discovered as I was trying to do "important" stuff this morning that I couldn't do anything until I wrote this. And being a novice at wiki (and tiki) I figured that making a wiki page would add a couple of hours to the task. Again I apologize for side-stepping the (good) tiki custom of writing to the website.

The following is a somewhat detailed presentation of what I've done while implementing my extension to tiki (called handin). I've read some (if not all) the wiki pages on the core. This is not meant to address them point by point but it does cover many of the issues raised in them.

Well, I'm off to do "important" things. I'll return in several hours and try to get on the IRC channel.

dayton

1) There will be a directory in the tiki root (I'll use tiki) called 'tiki/ext'. And a database, 'tiki_extensions'. The database will include fields covering:

o Is this extension active and for what tiki groups.
o The extension handle (i.e. the subdirectory name, see below).
o The installed version of the extension.
o Required tiki version.
o Other dependencies (how this is specified I don't know)
o Person responsible locally for maintenance of the extension w/contact information.
o Contact information for the developer or global maintainers.
o Documentation url.

2) Code for the extension resides in a subdirectory of tiki/ext with the name of it's handle from the database. The extension I'm working on is called 'handin' so I'll use that for examples.

In tiki/ext/handin is a script 'handin-install.php'. This script checks if the extension is installed and if so what version is installed. The script then handles the installation or upgrade as appropriate. I suppose that the default for a fresh install is that the extension is only active for the admin group.

This way, installation is basically a matter of putting the code in place and invoking handin-install.php. Note that installation may involve creating modules and menus and other stuff in addition to

putting it's entry in tiki_extensions.

Maybe it should be the responsiblity of handin-install.php to check for dependencies and fail if things aren't right. This is cleaner than specifying in the database and trying to maintain dependencies on the tiki-core level. I like this better.

I suppose there should also be scripts, handin-disable.php, handin-enable.php, and handin-uninstall.php which do the appropriate thing.

3) The API for extensions is via callouts. There is a database 'tiki_callouts' with the following fields (excues me, I should say columns):

o 'extension' is the handle of the extension which put this in the database.

o 'location' specifies where the callout should be invoked. The core will have statements like

$tikiCallouts->doCallouts('post-setup');

where $tikiCallouts is an object of class Callouts created early in the setup process. The string 'post-setup' specifies the location of the callout. This particular example is at the end of tiki-setup.php.

o 'action' which is a text string of upto 255 characters. This is either the name of a file to include or a php command.

o 'sequence' which is an integer that specifies the order actions should be invoked for this extension if the extension specifies more than one action for a location. Maybe this provides too much flexibility.

o There might be other columns like 'installation-date' and columns for statistics like how many times the callout is invoked.

So now when $tikiCallouts->doCallouts('post-setup') is invoked it searches tiki_callouts for all rows with `location` = "post-setup". For each row found, the 'action' is performed or included as appropriate.

Order is important here and I'm not sure how to specify the order between extensions. Perhaps each extension should have an 'order number' which specifies its relationship to other extensions, but who sets it? What about cycles? (i.e., extA callouts should precede extB callouts which should precede extC callouts which should precede extA callouts). For a start we can say that the order is static and the extension's install script and/or the local admin is responsible for setting it.

At the end of this message I've included my code for the class Callouts. Note that my current implementations differs some from what I've described above.

4) This is an idea that came to me last night (I have dreams about tiki!) so I haven't had a chance to test it at all.

A script, tiki-ext.php, is used to invoke extension scripts. So that:

http://my.host/tiki/tiki-ext.php?e=handin;s=handinHomePage.php

would invoke tiki-ext.php which would in the end

require(ext/handin/handinHomePage.php);

This appeals to me for several reasons:

o tiki-ext.php can handle the setup of the environment (tiki-setup.php etc.) so the extension script need not do this.

o The extension scripts can be concealed from the web directly via .htaccess or similar techniques.

o tiki-ext.php can provide sanity checking, permissons checking, security, and statistics gathering. This and the above could make the site a little less dependent on the security of the extension's code.

Well that's about it. I can already think of modifications and objections (e.g., group permissions should be on the per callout basis and not a per extension basis.) But it's something to consider.

Now some code. Here's a summary of what I've done for the handin extension so far. My existing code varies from what I've described above, but similar enough to give you the feeling for what I've done.

In tiki-init.php (invoked at basically the same time as db/local.php, I like this scheme but am not married to it). I have

```
/* How were we invoked.
*/
$uri = $_SERVER'REQUEST_URI';

if"^/personal", $uri
{
$tikidomain = 'dayton';
}
elseif"^/cis26/03fall", $uri
{
$tikidomain = 'cis26-03fall';
$tikiCallouts->addExtension("handin");
```

}
else
{
trigger_error("Unknown tiki ($uri).", E_USER_ERROR);
}

Most of this doesn't is not related to extensions but my use of multiple tiki's based on the invoking document path. Note that for each tiki $tikidomain is set. This is used later to determine what database to use and what cache directories to use.

Notice that for the cis26-03fall domain (one of the classes I'm teaching this fall) $tikiCallouts->addExtension("handin") is invoked which enables the handin extension.

In the tiki_callouts database there are two rows:

extension location action
=== == ==
handin post-setup handin/handinSetup.php
handin tiki-index handin/callout-tiki-index.php

(Note that the handin directory is directly under the tiki root and not in tiki/ext.)

At the end of tiki-setup.php there is

$tikiCallouts->doCallouts('post-setup');

which causes handin/handinSetup.php to be included. In handin/handinSetup.php I create an object, $handin, which provides basic capabilities for the extension. Primarily, this involves connecting to the handin database (distinct from the tiki database) and providing methods for querying the database.

In the beginning of tiki-index.php (right after the initial includes) I have

$tikiCallouts->doCallouts('tiki-index');

which causes handin/callout-tiki-index.php to be included. This script checks to see if the user who logged in has filled out my form to get additional information (real name, college id, and email). If they have I let tiki-index.php continue. If not, a form pops up to gather and validate the info. Once the script is satisfied, it re-invokes tiki-index.

(Note that I pre-register my students based on info I get from the registrar so that tiki knows their usernames but not the email addresses, that's why I must collect them 'by hand').

This has passed my limited tests nicely.

enjoy

dayton

Here's the code for the class Callouts.

```
<?php
/* tiki-callouts.php
*/
class Callouts
{
var $_extensions;

function doCallouts($location)
{
if$this->_extensions
return;

if(!is_array($this->_extensions))
{
trigger_error("Callout::doCallouts(): \$_extensions is not an array.",
E_USER_ERROR);
return;
}

foreach($this->_extensions as $extension)
{
$actions = Callouts::_getActions($extension, $location);
Callouts::_doActions($actions);
}
}

function addExtension($extension)
{
if($this->_extensions == null)
{
$this->_extensions = array($extension);
}
else if(is_array($this->_extensions))
{
$this->_extensions[] = $extension;
}
else
{
trigger_error("Callouts::addExtension(): \$_extensions is not an array.",
E_USER_ERROR);
}
```

```
}

function _getActions($extension, $location)
{
global $dbTiki;

$sql = 'SELECT `action` ';
$sql .= 'FROM `tiki_callouts` ';
$sql .= 'WHERE `extension` = "' . $extension . '" ';
$sql .= 'AND `location` = "' . $location . '" ';
$sql .= 'ORDER BY `sequence`;';

$actions = $dbTiki->getCol($sql, 'action');

if(DB::isError($actions))
{
trigger_error('Callout::_getActions(): error in query (' . $sql . ')',
E_USER_ERROR);
$actions = array();
}

return $actions;
}

function _doActions($actions)
{
if$actions
return;

/* If we have an array, we invoke each one in sequence.
*/
if(is_array($actions))
{
/* NB: We are calling _doActions recursively here. If someone were
* to create and action array which referred to itself, then this
* loop would not terminate. Given the small unlikelihood of this
* occuring accidently, no attempt is made to detect it. --dayton
*/
foreach($actions as $a)
Callouts::_doActions($a);
return;
}

/* We have a single action, which should be a string.
*/
if(!is_string($actions))
{
trigger_error("Callout::_doActions: action is not a string, \"$actions\"",
E_USER_ERROR);
return;
```

```
}

/* Now, do we have a php command, a file to include, or the name of a
* function to invoke. We'll guess as best we can.
*/
if'^.*\.php$', $actions
missing page for plugin INCLUDE

else if';', $actions
{
eval($actions);
}
else if(is_callable($actions))
{
/* NB: is_callable() does not detect language constructs such as
* 'echo' or 'include' as functions. To envoke these, make them a
* statement.
*/
$actions();
}
else
trigger_error("Callout::_doActions: Unknown action, \"$actions\"", E_USER_ERROR);
}
}
```