SpidercoreDev

NOTE: this project is now actively underway at http://www.tikipro.org

My first objective is simple: I would like to make Tiki much more open to other people's code.

I want to put phpBB, Gallery, SquirrelMail,or any of the multitude of other open source web applications into Tiki. Tiki is a great piece of software, one which I settled on after reviewing all open source CMS systems. The adoption of ADODB was a major undertaking, and one that truly will move Tiki into much more demanding environments.

My second objective is simpler: I want to modify as **little** existing Tiki code as possible.

There is a lot of great code in Tiki and I have no interest in rewriting any of it. However, it needs to be better organized so external code can coexist peacefully within Tiki code, and coexist with other alien code trees within Tiki. My planning is a bit more in depth, however, I am strictly adhering to mose's #1 rule: **USE WHAT EXISTS**.

got code?

There is a separate module in source forge for this code called spidercore. If you would like a copy of this code, simply check it out of CVS with:

cvs :pserver:user@cvs.sourceforge.net:/cvsroot/tikiwiki checkout spidercore

Phase 1.9 - Tiki Packages (aka Directory Restructuring)

My design is simple: Put all files related to each feature into their own directory

!!!First attempt complete! see http://cvs.sourceforge.net/viewcvs.py/tikiwiki/spidercore/ for an initial reorganization, or an in progress demo. Most everything has been put in what seems to me a much more logical, modular grouping. Nothing was thrown away, nothing was added. I do not want to write any functionality because I know the code that's there will work once I fix paths and includes.

Change as little code as possible (and absolutely NO database tables) - simply reorganize the codebase into properly encapsulated directories. Advancing Dennis Heltzel's concepts and name, each directory is known a TikiPackage.

The Golden Rule of Tiki Packages: Tiki should place the **fewest possible requirements** possible for a TikiPackage

A Tiki Package is simply a directory in the Tiki root directory. The only requirement is a single file called "setup_inc.php" that will get included by TikiSetup. Optional files will include a "local_inc{{.$tikidoman}.php" for settings particular to a user's install. A TikiPackage will be all of the integrated features in lib/* or it could be an external app, like phpBB, adopted to use the Tiki framework.

The solution is low tech and dirt simple. I do not see **insisting** on a large API infrastructure when some Packages might only need to execute a few lines of code. Other Packages might need a complex architecture. So let developer's decide for themselves how simple or complex to make their own Package. This should also allow ports to stay as virgin as possible, and stay much more up to date with their own code base.

Tiki's current "lib" directory works very well and is the model for where I would like to start. My answer is to simply stuff all code for given features (wiki, blog, article, forum) into the existing lib directories. I would like to move the lib/* directories up out of the lib directory so that the Tiki root is one or two php files and bunch of directories, such as:

```
/path/to/tiki/
|- db/
|- core/
|- blog/
|- wiki/
|- phpBB/
|- gallery/
|- newFeature/
|---> setup_inc.php
|---> local_inc.php
|---> local_inc.www.multisite.com.php
|---> /templates
|---> /modules
|---> /admin
```

There is a common KernelPackage that is simply a chopping and moving of the existing tikilib & tiki-setup files. See Phase 2.0 for details...

Phase 2.0 - KernelPackage - TikiCoreTwo & class encapsulation

I want to simply break up tikilib & tiki-setup into a few classes. When a new Package, such as phpbb gets ported to TikiTwo, all it needs is a database connection, and a framework within which to render it's pages. a phpBB port would simply invoke the same setup files that all integrated features now call.

There have been many other proposals with a lot of great ideas. However, this proposal is intended to be much more evolutionary rather than revolutionary. A primary goal is to preserve as much as the existing Tiki code and infrastructure as possible instead of a large scale rewrite.

This core will:

- Maintain tiki's "keep it simple" philosophy and require only a modest amount of new code. A goal is to reuse nearly all existing code. In short, initial work will be a cut-n-paste job to get functions out of the bloated lib files and into a properly encapsulated classes framework (e.g. so 'create_page' does not get parsed when reading a forum post, etc.)
- Focused on scalability for both **feature depth** and **speed**. Some sites want a lot of features, and others want less features a more speed because of traffic demands. The core should be scalable, much like the

Linux kernel, where new functionality can be added or removed where desired. Unused functionality should not even be parsed.
- Core functionality will be much more Object Oriented and have greater use of inheritance (extend). The reduces code and makes extending Tiki infastructure for your own specialized purposes much easier.

The migration will be done in a few key areas first (wiki pages will be the first to move). This will prototype the design and allow several development cycles to happen to ensure a proper design. When complete, classes like TikiLib or WikiLib will be deprecated and only kept for legacy features.

Some detail:

A new class structure is defined from which all Tiki classes inherit. The existing lib/* directory structure will be maintiained, and all existing libs can stay in place as migration takes place. The basic class inheritance looks like:

- class TikiFeature - Cool feature such as Wiki or Blog or Article, etc...
- --> class TikiContent - Virtual Base class upon which all "content" oriented classes derive (wiki, article, etc.)
- ----> class TikiBase - a few utility functions common to every Tiki class
- –> class TikiDB - little changed from current core

An instantiated class should typically represent a single object, such as a wiki page or article. Aggregate data functions are currently in the same class. Perhaps a derived TikiFeatureGroup class might be designed, but this is uncertain for now.

A few rules of thumb:

1. There should be _NO_ presentation code (no smarty assignments, no html generation, etc) in any class derived from TikiBase. Two notable exceptions might be extreme error conditions or convenient $rs->GetMenu() calls
2. All derived class should support a core set of methods. In real OO land, there would be several pure virtual functions named: **load**, **store**, **verify**, **expunge** (delete has a naming conflict with native php function). These functions should always follow these names so derived classes can properly call into base methods

Example: imaginary TikiFeature derived class, see TikiPage for real example

```
class TikiFeature extends TikiBase
{
// member var that hold corresponding data
var mRow;

function TikiFeature( $iFeatureID = NULL ) {
// be sure to call base constructor!!!
TikiBase::TikiBase();
$this->mFeatureID = $iFeatureID;
}

function load() {
```

```
if( $this->mFeatureID ) {
$this->mRow = {... select data from db where feature_id=$this->mFeatureID ...}
}
return( count( $this->mErrors ) == 0 );
}

// This verifies data integrity. NOTE: pass by reference is crucial, because
// some modifications might be necessary (length truncation, etc.);
function verify( &$inParams ) {
// clean up variables
foreach( array_keys($inParams) as $key ) {
$inParams[$key] = trim( $inParams[$key] );
}

if( empty( $inParams['required_column'] ) ) {
$this->mErrors['required_column'] = SOME_ERROR_MESSAGE;
} elseif( strlen( $inParams['required_column'] ) > FEATURE_LENGTH ) {
$inParams['required_column'] = substring( $inParams['required_column'],
0, FEATURE_LENGTH );
}

{... continue testing hash values various conditions ...}

return( count( $this->mErrors ) == 0 );
}

// this method stores the data.
function store( &$inParams ) {
// ALWAYS call our verify first to ensure data is safe to go into db
if( $this->verify( $inParams ) ) {
$this->mDB->StartTrans();
if( data exists ) {
{... update existing rows ...}
} else {
{... insert new row ...}
}
$this->mDB->CompleteTrans();
}
return( count( $this->mErrors ) == 0 );
}

funciton expunge() {
{... delete appropriate rows for this feature in all necessary tables ...}
return( count( $this->mErrors ) == 0 );
}
}
```