Overview

This page aims to provide basic information on how to approach development projects within Tiki and point to the appropriate resources. It is not meant to contain every possible detail.

Before you begin, consider the timeline of your project. Of course, it would be great to have it live tomorrow but be realistic. Too often, projects come up with very tight schedules that force around less than ideal solutions and extend longer than initially planned.

Tiki has a release schedule that allows planning your development targets ahead. With a release every 8 months, you are often much better off working from **trunk** and work for the next release than to build on top of the current **stable** version. If your needs extend beyond the easy extension points in Tiki, patch maintenance will become a burden that will cause delays and rework in the project. When working for the next release, you can contribute your patches and new extension points right away and evolve with the code base.

Code structure and primary components

Tiki's code evolved over the years and continues to evolve. From the start, the codebase was built to be flat and easy to understand. For those familiar with the MVC model, there are some references, but you won't find front controllers or actions within classes. In Tiki, the root PHP script is the controller. The model is loosely controlled by a set of libraries to interact with the database and the view is handled by Smarty templates.

The typical lifecycle of a request looks like this:

1. Input configuration to set-up filters
2. Inclusion of *tiki-setup.php*, which performs a handful of tasks from filtering the input, loading up the configuration to dealing with multiple environment issues to allow compatibility on different platforms
3. Feature checks
4. Handling of the user request
5. Setting up the page
6. Rendering via Smarty

The code is massive and it contains hundreds of features and configuration options. All of those are stored within the *$prefs* global variable, which is also available within Smarty. Many parts of the code will verify options to alter the behavior of the code. You may need to create new preferences yourself as you work in the code.

Handling of the user requests is done simply by inspecting the PHP request variables and calling methods in libraries. Tiki also includes multiple third party libraries. As much as possible, avoid calling them directly from the script and wrap the behavior within Tiki-specific libraries.

Setting up the page implies loading some information through the libraries located in the **lib/** directory and assigning values to the global *$smarty* object. The rendering will be made using templates that can be adapted per style as needed.

Tiki relies on JQuery for cross-browser compatibility of JavaScript and provides helpers to include JavaScript within your pages.

Content organization

Tiki is many things, but it primarily is a content management system. Before jumping straight into the development of an entirely new feature, you should consider re-using existing storage mechanism and building around those to complete your use case. Independent features with a limited scope allow to quickly build new functionality with targeted user experience, but they tend to create silos of unmaintained code. On the other hand, if you use existing containers, they are much more likely to evolve

by themselves and provide long-term benefits.

You should aim to build the features you need in a generic way so others can benefit from them and bundle **your application** as a profile. Think of profiles as recipes to configure Tiki and bind various features together to serve a use-case.

Begin by determining how to shape the content, then see what gaps need to be filled.

The primary content containers in Tiki are the following:

- Wiki pages, allowing for free-form content and custom reporting using plugins. The wiki pages come with complete revision history, multilingual support and dozens of other features.
- Trackers, allowing for user databases with structured content. They provide the basic support for CRUD without binding to fixed database tables.
- File galleries, allowing for file sharing, revision history and WebDAV access. They will also be used to store attachments to wiki pages and trackers.

Tiki also includes other content features like forums, spreadsheets, polls, calendars and blogs, but they are less likely to be involved in creating an application based on Tiki.

The required user interactions with the content determine which storage container is the best to use. When a lot of collaboration is required, wiki pages are a strong contender. The free-form nature allows for creativity. Trackers are useful when the structure of the information is important when specific fields need to be searched or sorted for example. File galleries have the advantage of being very easy to understand, but the web-based experience of manipulating external files is not as good. There is lower connectivity to the other features.

Global features within Tiki can bridge between the key characteristics of the content features. For example:

- Comments can be added to any content, allowing for more collaboration on tracker items or files in galleries.
- Attachments can be added to tracker items or wiki pages to re-use an existing document as a reference.
- Categories allow regrouping documents to provide some additional reporting capabilities to wiki pages even though they do not allow for structured content. They also allow the creation of document workflows through transitions between states.
- Tags can be added to all content to allow users to build their own organization of the content and create custom lists of items they keep track of.

All content features also come with benefits like content indexation for searching permission management.

As a general rule, it's easier to start small and adapt as needs change.

Permission management

Of course, Tiki will manage the user authentication and their respective permissions. Authentication can be bridged to user directories like LDAP and single sign-on solutions like OpenID, CAS and others.

From the moment the user is authenticated, he will be part of groups that will grant certain rights to content within Tiki. The permissions can be granted:

- Globally, unless other rules are specified
- Per-category, applying to all member objects
- Per object, overriding all of the above

Tiki contains a total of over 200 permissions that can be applied at any of those 3 levels. They typically connect to certain actions, like viewing a wiki page, editing a page, deleting a tracker item, assigning a category and many others.

As you develop for Tiki, you will also certainly need to check for a permission or even create a new one.

Extension points

Even though Tiki is monolithic by design to encourage contribution and avoid the dependency hell during upgrade cycles, there are several extension points built withing the system that allows for customization and easy extension.

The two primary extension points within Tiki are plugins and modules. Plugins allow to embed something within content and modules allow to attach something in the interface, currently in the left or right column, or anywhere through some template hacking (starting in Tiki7: top & bottom as well). They both can be used for various reasons.

Plugins can be used to:

- Act as a decorator to create a visual effect on content
- Embed information from another feature
- Provide some listing in a dashboard
- Perform actions on saving through the post-save hook

A plugin is composed of two functions stored within a PHP file. One of the functions is a descriptor used to determine some aspects of the execution and build the user interface to insert or edit the plugin, the other is the actual function used to execute the plugin. See Writing syntax plugins for more information.

Modules can be used to:

- Provide navigation to content, either through custom listings or menu objects
- Provide contextual information related to the current object, as used for the translation status or transition user interface

A plugin is minimally defined by a template. Ideally, it also provides a PHP file containing a descriptor function and an execution function to set-up the module. See Writing modules for more information.

Various other features may have extension points, like search indexing, print features and others. However, they change a lot over time and the code often is the best resource.

See also

- Preferences
- Database Schema Upgrade
- Database Access