Table of Contents

Developing Plugins **1**
- APIs available to Extensions 1
  - Standard Regular Expressions 1
- Creating New Plugins 2
  - Anatomy of a Plugin 2
  - Creating the Perl Module 3
  - Writing the Documentation Topic 3
  - Packaging for Distribution 3
  - Publishing for Public Use 4
- Hints on Writing Fast Plugins 4
- Security 4
- Recommended Storage of Plugin Specific Data 5
  - Plugin Internal Data 5
  - Web Accessible Data 5
- Integrating with configure 6
  - Structure of a Config.spec file 6
- Maintaining Plugins 7
  - Discussions and Feedback on Plugins 7
  - Maintaining Compatibility with Earlier Foswiki Versions 7
  - Handling deprecated functions 8
  - TWiki® Plugins 8

Developing Plugins

The usual way Foswiki is extended is by writing a *Plugin*. Plugins extend Foswiki by providing functions that 'listen' to events in the Foswiki core, and handling these events. These functions are called "Plugin Handlers" and they are described in depth in EmptyPlugin (`lib/Foswiki/Plugins/EmptyPlugin.pm`).

APIs available to Extensions

To be robust, extensions must avoid using any unpublished functionality from the Foswiki core. The following perl packages give access to features for extension authors. These APIs are not just for Plugins, they can be used in any type of extension. Click on the name of the package to see the full documentation.

- `Foswiki::Func` - bridge to core functions. This is the package you will use most.
- `Foswiki::Meta` - topic meta-data
- `Foswiki::OopsException` - special exception for invoking the 'oops' script
- `Foswiki::AccessControlException` - access control exception
- `Foswiki::Attrs` - parser and storage object for macro parameters
- `Foswiki::Time` - time parsing and formatting
- `Foswiki::Sandbox` - safe server-side program execution, used for calling external programs.
- Iterators - these are classes that implement the `Foswiki::Iterator` specification
    - `Foswiki::ListIterator` - utility class for iterator objects that iterate over list contents
    - `Foswiki::LineIterator` - utility class for iterator objects that iterate over lines in a block of text
    - `Foswiki::AggregateIterator` - utility class for iterator objects that aggregate other iterators into a single iteration

In addition the following global variables may be referred to:

- `$Foswiki::Plugins::VERSION` - plugin handler API version number
- `$Foswiki::Plugins::SESSION` - reference to `Foswiki` singleton object
- `$Foswiki::cfg` - reference to configuration hash
- `$Foswiki::regex` - see Standard Regular Expressions, below

Foswiki:Development.GettingStarted is the starting point for more comprehensive documentation on developing for Foswiki.

Note the APIs are available to all extensions, but rely on a `Foswiki` singleton object having been created before the APIs can be used. This will only be a problem if you are writing an extension that doesn't use the standard initialisation sequence.

Standard Regular Expressions

A number of standard regular expressions are available for use in extensions, in the `$Foswiki::regex` hash. these regular expressions are precompiled in an I18N-compatible manner. The following are guaranteed to be present. Others may exist, but their use is unsupported and they may be removed in future Foswiki versions.

In the table below, the expression marked type 'String' are intended for use within character classes (i.e. for use within square brackets inside a regular expression), for example:

```
my $isCapitalizedWord =
  ( $s =~ /[$Foswiki::regex{upperAlpha}][$Foswiki::regex{mixedAlpha}]+/ );
```

Those expressions marked type 'RE' are precompiled regular expressions that can be used outside square brackets. For example:

```
my $isWebName = ( $s =~ m/$Foswiki::regex{webNameRegex}/ );
```

Name	Matches	Type
upperAlpha	Upper case characters	String
upperAlphaNum	Upper case characters and digits	String
lowerAlpha	Lower case characters	String
lowerAlphaNum	Lower case characters and digits	String
numeric	Digits	String
mixedAlpha	Alphabetic characters	String
mixedAlphaNum	Alphanumeric characters	String
wikiWordRegex	WikiWords	RE
webNameRegex	User web names	RE
topicNameRegex	Topic names	RE
anchorRegex	#AnchorNames	RE
abbrevRegex	Abbreviations/Acronyms e.g. GOV, IRS	RE
emailAddrRegex	email@address.com	RE
tagNameRegex	Standard macro names e.g. %THIS_BIT% (THIS_BIT only)	RE

Creating New Plugins

With a reasonable knowledge of the Perl scripting language, you can create new plugins or modify and extend existing ones.

Anatomy of a Plugin

A (very) basic Foswiki plugin consists of two files:

- a Perl module, e.g. `lib/Foswiki/Plugins/MyFirstPlugin.pm`
- a documentation topic, e.g. `MyFirstPlugin.txt`

The Perl module can invoke other, non-Foswiki, elements, like other Perl modules (including other plugins), graphics, external applications, or just about anything else that Perl can call.

The plugin API handles the details of connecting your Perl module with the Foswiki core.

The module provides a lot of support for plugins development, including a plugin creator, automatic publishing support, and automatic installation script writer. If you plan on writing more than one plugin, you probably need it.

Creating the Perl Module

Copy file `lib/Foswiki/Plugins/EmptyPlugin.pm` to `<name>Plugin.pm`. The EmptyPlugin does nothing, but it contains all the information you need to create you own custom plugin.

Writing the Documentation Topic

The plugin documentation topic contains usage instructions and version details. (The doc topic is also included *in* the distribution package.) To create a documentation topic:

1. **Copy** the plugin topic template from EmptyPlugin?
2. **Customize** your plugin topic.
    - Important: In case you plan to publish your plugin on Foswiki.org, use Interwiki names for author names and links to Foswiki.org topics, such as Foswiki:Main/WikiGuest. This is important because links should work properly in a plugin topic installed on any Foswiki, not just on Foswiki.org.
3. **Save** your topic, for use in packaging and publishing your plugin.

OUTLINE: Doc Topic Contents
Check the plugins web on Foswiki.org for the latest plugin doc topic template. Here's a quick overview of what's covered:
Syntax Rules: <*Describe any special text formatting that will be rendered.*>"
Example: <*Include an example of the plugin in action. Possibly include a static HTML version of the example to compare if the installation was a success!*>"
Plugin Settings: <*Description and settings for custom plugin settings, and those required by Foswiki.*>"
Plugins Preferences <*If user settings are needed, link to preference settings and explain the role of the plugin name prefix*
Plugin Installation Instructions: <*Step-by-step set-up guide, user help, whatever it takes to install and run, goes here.*>"
Plugin Info: <*Version, credits, history, requirements - entered in a form, displayed as a table. Both are automatically generated when you create or edit a page in the web.*>

Packaging for Distribution

The is a powerful build environment that is used by the Foswiki project to build Foswiki itself, as well as many of the plugins. You don't **have** to use it, but it is highly recommended!

If you don't want to (or can't) use the BuildContrib, then a minimum plugin release consists of a Perl module with a WikiName that ends in `Plugin`, ex: `MyFirstPlugin.pm`, and a documentation page with the same name(`MyFirstPlugin.txt`).

1. Distribute the plugin files in a directory structure that mirrors Foswiki. If your plugin uses additional files, include them all:
    - `lib/Foswiki/Plugins/MyFirstPlugin.pm`
    - `data/Foswiki/MyFirstPlugin.txt`
    - `pub/Foswiki/MyFirstPlugin/uparrow.gif` [a required graphic]
2. Create a zip archive with the plugin name (`MyFirstPlugin.zip`) and add the entire directory structure from Step 1. The archive should look like this:

- `lib/Foswiki/Plugins/MyFirstPlugin.pm`
- `data/Foswiki/MyFirstPlugin.txt`
- `pub/Foswiki/MyFirstPlugin/uparrow.gif`

Publishing for Public Use

You can release your tested, packaged plugin to the Foswiki community through the web. All plugins submitted to Foswiki.org are available for public download and further development.

Publish your plugin by following these steps:

1. **Post** the plugin documentation topic to the Foswiki:Extensions web
2. **Attach** the distribution zip file(s) to the topic, eg: `MyFirstPlugin.zip`
3. Add a user support hub by visiting Foswiki:Support.CreateNewSupportHub
4. Optionally, check in the sources to the Foswiki subversion repository (see Foswiki:Development.HowToStartExtensionDevelopmentInSubversion)

Once you have done the above steps once, you can use the BuildContrib to upload updates to your plugin.

Thank you very much for sharing your plugin with the Foswiki community

Hints on Writing Fast Plugins

- Delay initialization as late as possible. For example, if your plugin is a simple syntax processor, you might delay loading extra Perl modules until you actually see the syntax in the text.
  - For example, use an `eval` block like this:
    ```
    eval { require IPC::Run }
    return "<font color=\"red\">SamplePlugin: Can't load required
    modules ($@)</font>" if $@;
    ```
- Keep the main plugin package as small as possible; create other packages that are loaded if and only if they are used. For example, create sub-packages of BathPlugin in `lib/Foswiki/Plugins/BathPlugin/`.
- Avoid using preferences in the plugin topic; set `$NO_PREFS_IN_TOPIC` if you possibly can, as that will stop Foswiki from reading the plugin topic for every page. Use Config.spec instead.
- Use registered tag handlers

Security

- Badly written plugins can open security holes in Foswiki. This is especially true if care isn't taken to prevent execution of arbitrary commands on the server.
- Don't allow sensitive configuration data to be edited by users. Use the `%Foswiki::cfg` hash for configuration options. Don't ask installers to edit topics in the System web.
  - Integrating with `configure` describes the steps
  - Foswiki:Extensions.MailInContrib has an example
  - Foswiki:Extensions.BuildContrib can help you with this

- Make sure that all user input is checked and validated. Be especially careful to filter characters that might be used in perl string interpolation.
- Avoid `eval`, and if you must use it make sure you sanitise parameters
- Always use the Foswiki::sandbox to execute commands. Never use backtick or qx//.
- Use `Foswiki::Func::checkAccessPermission` to check the access rights of the current user.
- Always audit the plugins you install, and make sure you are happy with the level of security provided. While every effort is made to monitor plugin authors activities, at the end of the day they are uncontrolled user contributions.

Recommended Storage of Plugin Specific Data

Plugins sometimes need to store data. This can be plugin internal data such as cache data, or data generated for browser consumption such as images. Plugins should store data using Foswiki::Func functions that support saving and loading of topics and attachments.

Plugin Internal Data

You can create a plugin "work area" using the `Foswiki::Func::getWorkArea()` function, which gives you a persistent directory where you can store data files. By default they will not be web accessible. The directory is guaranteed to exist, and to be writable by the webserver user. For convenience, `Foswiki::Func::storeFile()` and `Foswiki::Func::readFile()` are provided to persistently store and retrieve simple data in this area.

Web Accessible Data

The internal data area is not normally made web-accessible for security reasons. If yoou want to store web accessible data, for example generated images, then you should use Foswiki's attachment mechanisms.

Topic-specific data such as generated images can be stored in the topic's attachment area, which is web accessible. Use the `Foswiki::Func::saveAttachment()` function to store the data.

Recommendation for file name:

- Prefix the filename with an underscore (the leading underscore avoids a name clash with files attached to the same topic)
- Identify where the attachment originated from, typically by including the plugin name in the file name
- Use only alphanumeric characters, underscores, dashes and periods to avoid platform dependency issues and URL issues
- Example: `_GaugePlugin_img123.gif`

Such auto-generated attachments han be hidden from users by setting the 'h' attribute in the attachment attributes.

Web specific data should be stored in the attachment area of a topic in the web that you specify for the purpose, e.g. Web.BathPlugPictures. Use the `Foswiki::Func::saveAttachment()` function to store the data in this topic.

Integrating with `configure`

Some extensions have setup requirements that are best integrated into `configure` rather than trying to use preference settings. These extensions use `Config.spec` files to publish their configuration requirements.

`Config.spec` files are read during configuration. Once a `Config.spec` has defined a configuration item, it is available for edit through the standard `configure` interface. `Config.spec` files are stored in the 'plugin directory' e.g. `lib/Foswiki/Plugins/BathPlugin/Config.spec`.

Structure of a `Config.spec` file

The `Config.spec` file for a plugin starts with the plugin announcing what it is:

```
# ---+ BathPlugin
# This plugin senses the level of water in your bath, and ensures the plug
# is not removed while the water is still warm.
```

This is followed by one or more configuration items. Each configuration item has a *type*, a *description* and a *default*. For example:

```
# **SELECT Plastic,Rubber,Metal**
# Select the plug type
$Foswiki::cfg{BathPlugin}{PlugType} = 'Plastic';

# **NUMBER**
# Enter the chain length in cm
$Foswiki::cfg{BathPlugin}{ChainLength} = '30';

# **BOOLEAN EXPERT**
# Turn this option off to disable the water temperature alarm
$Foswiki::cfg{BathPlugin}{TempSensorEnabled} = '1';
```

The type (e.g. `**SELECT**`) tells `configure` to how to prompt for the value. It also tells configure how to do some basic checking on the value you actually enter. All the comments between the type and the configuration item are taken as part of the description. The configuration item itself defines the default value for the configuration item. The above spec defines the configuration items
`$Foswiki::cfg{BathPlugin}{PlugType}`,
`$Foswiki::cfg{BathPlugin}{ChainLength}`, and
`$Foswiki::cfg{BathPlugin}{TempSensorEnabled}` for use in your plugin. For example,

```
if( $Foswiki::cfg{BathPlugin}{TempSensorEnabled} && $curTemperature > 50 ) {
    die "The bathwater is too hot for comfort";
}
```

The `Config.spec` file is read by configure, and `configure` then writes `LocalSite.cfg` with the values chosen by the local site admin.

A range of types are available for use in `Config.spec` files:

`BOOLEAN`	A true/false value, represented as a checkbox
`COMMAND` *length*	A shell command
`LANGUAGE`	A language (selected from `{LocalesDir}`
`NUMBER`	A number
`OCTAL`	An octal number

`PASSWORD` *length*	A password (input is hidden)
`PATH` *length*	A file path
`PERL`	A simplified perl data structure, consisting of arrays, hashes and scalar values
`REGEX` *length*	A perl regular expression
`SELECT` *choices*	Pick one of a range of choices
`SELECTCLASS` *package-specifier*	Select a perl package (class) e.g. `SELECTCLASS Foswiki::Plugins::BathPlugin::*Plug` lets the user select between all packages with names ending in `Plug`, `Foswiki::Plugins::BathPlugin::RubberPlug`, `Foswiki::Plugins::BathPlugin::BrassPlug` etc.
`STRING` *length*	A string
`URL` *length*	A url
`URLPATH` *length*	A relative URL path

All types can be followed by a comma-separated list of *attributes*.

`EXPERT`	means this an expert option
`M`	means the setting is mandatory (may not be empty)
`H`	means the option is not visible in `configure`

See `lib/Foswiki.spec` for many more examples.

`Config.spec` files are also used for other (non-plugin) extensions. in this case they are stored under the `Contrib` directory instead of the `Plugins` directory.

Maintaining Plugins

Discussions and Feedback on Plugins

Usually published plugins have a support hub in the Support web on Foswiki.org.Support hubs have links to where to discuss feature enhancements and give feedback to the developer and user communities.

Maintaining Compatibility with Earlier Foswiki Versions

The plugin interface (Foswiki::Func functions and plugin handlers) evolve over time. Foswiki introduces new API functions to address the needs of plugin authors. Plugins using unofficial Foswiki internal functions may no longer work on a Foswiki upgrade.

Organizations typically do not upgrade to the latest Foswiki for many months. However, many administrators still would like to install the latest versions of a plugin on their older Foswiki installation. This need is fulfilled if plugins are maintained in a compatible manner.

Tip: Plugins can be written to be compatible with older and newer Foswiki releases. This can be done also for plugins using unofficial Foswiki internal functions of an earlier release that no longer work on the latest Foswiki codebase. Here is an example; the has more details.

```
if( $Foswiki::Plugins::VERSION >= 1.1 ) {
    @webs = Foswiki::Func::getListOfWebs( 'user,public' );
} else {
```

```
        @webs = Foswiki::Func::getPublicWebList( );
    }
```

Handling deprecated functions

From time-to-time, the Foswiki developers will add new functions to the interface (either to Foswiki::Func, or new handlers). Sometimes these improvements mean that old functions have to be deprecated to keep the code manageable. When this happens, the deprecated functions will be supported in the interface for at least one more Foswiki release, and probably longer, though this cannot be guaranteed.

When a plugin defines deprecated handlers, a warning will be shown in the list generated by %FAILEDPLUGINS%. Admins who see these warnings should check Foswiki.org and if necessary, contact the plugin author, for an updated version of the plugin.

Updated plugins may still need to define deprecated handlers for compatibility with old Foswiki versions. In this case, the plugin package that defines old handlers can suppress the warnings in %FAILEDPLUGINS%.

This is done by defining a map from the handler name to the `Foswiki::Plugins` version *in which the handler was first deprecated*. For example, if we need to define the `endRenderingHandler` for compatibility with `Foswiki::Plugins` versions before 1.1, we would add this to the plugin:

```
package Foswiki::Plugins::SinkPlugin;
use vars qw( %FoswikiCompatibility );
$FoswikiCompatibility{endRenderingHandler} = 1.1;
```

If the currently-running Foswiki version is 1.1 *or later*, then the *handler will not be called* and *the warning will not be issued*. Foswiki with versions of `Foswiki::Plugins` before 1.1 will still call the handler as required.

TWiki® Plugins

Most plugins written for TWiki can also be run in Foswiki, by installing the TWikiCompatibilityPlugin. See Foswiki:Extensions.TWikiCompatibilityPlugin for more information.

Edit | Attach | Print version | History: %REVISIONS% | Backlinks | Raw View | More topic actions

Topic revision: r1 - 10 Sep 2009 - 14:41:57 - ProjectContributor

- . System
- Log In

- **Toolbox**
- Users
- Groups
- Index
- Search
- Changes
- Notifications
- RSS Feed

- Statistics
- Preferences

- **User Reference**
- BeginnersStartHere
- TextFormattingRules
- Macros
- FormattedSearch
- QuerySearch
- DocumentGraphics
- SkinBrowser
- InstalledPlugins

- **Admin Maintenance**
- Reference Manual
- AdminToolsCategory
- InterWikis
- ManagingWebs
- SiteTools
- DefaultPreferences
- WebPreferences

- **Categories**
- Admin Documentation
- Admin Tools
- Developer Doc
- User Documentation
- User Tools

- **Webs**
- Public
- System

Copyright © by the contributing authors. All material on this site is the property of the contributing authors.
Ideas, requests, problems regarding Wiki? Send feedback