Table of Contents

TWiki Plugins......1
- Overview......1
- Installing Plugins......1
  - On-Site Pretesting......1
  - Some Notes on Plugin Performance......2
- Managing Installed Plugins......2
  - Enabling Plugins......2
  - Plugin Evaluation Order......2
  - Plugin-Specific Settings......2
  - Listing Active Plugins......3
    - %ACTIVATEDPLUGINS%......3
    - %PLUGINDESCRIPTIONS%......3
    - %FAILEDPLUGINS%......4
- The TWiki Plugin API......6
  - Available Core Functions......6
  - Predefined Hooks......6
  - Hints on Writing Fast Plugins......6
  - Version Detection......6
  - Security......7
- Creating Plugins......7
  - Anatomy of a Plugin......7
  - Creating the Perl Module......7
  - Writing the Documentation Topic......8
  - Packaging for Distribution......9
  - Measuring and Improving the Plugin Performance......9
  - Publishing for Public Use......9
- Recommended Storage of Plugin Specific Data......10
  - Plugin Internal Data......10
  - Web Accessible Data......10
- Integrating with configure......10
  - Structure of a Config.spec file......11
- Maintaining Plugins......12
  - Discussions and Feedback on Plugins......12
  - Maintaining Compatibility with Earlier TWiki Versions......12
  - Handling deprecated functions......13

TWiki Plugins

Add functionality to TWiki with readily available plugins; create plugins based on APIs

Overview

You can add plugins to extend TWiki functionality, without altering the core code. A plug-in approach lets you:

- add virtually unlimited features while keeping the main TWiki code compact and efficient;
- heavily customize an installation and still do clean updates to new versions of TWiki;
- rapidly develop new TWiki functions in Perl using the plugin API.

Everything to do with TWiki plugins - demos, new releases, downloads, development, general discussion - is available at TWiki.org, in the **TWiki:Plugins** web.

TWiki plugins are developed and contributed by interested members of the community. Plugins are provided on an 'as is' basis; they are not a part of TWiki, but are independently developed and maintained.

Relevant links on TWiki.org:

- TWiki:TWiki.TWikiPluginsSupplement - ***tip:*** supplemental documentation on TWiki plugins
- TWiki:Plugins.PluginPackage - list of all contributed plugin packages
- TWiki:Plugins.PluginDevelopment - discussion and feedback on contributed plugins
- TWiki:Plugins.PluginBrainstorming - open forum for new plugin ideas
- TWiki:Plugins.PluginPackageHowTo - template to create a new plugin package

See other types of extensions: TWikiAddOns, TWikiContribs, TWikiSkins

Installing Plugins

Each TWiki plugin comes with its own documentation: step-by-step installation instructions, a detailed description of any special requirements, version details, and a working example for testing. Many plugins have an install script that automates these steps for you.

Special Requirements: Some plugins need certain Perl modules to be preinstalled on the host system. Plugins may also use other resources, like graphics, other modules, applications, and templates. You should be able to find detailed instructions in the plugin's documentation.

Each plugin has a standard release topic, located in the TWiki:Plugins web at TWiki.org. There's usually a number of other related topics, such as a developers page, and an appraisal page.

On-Site Pretesting

The recommended approach to testing new plugins before making them public is to create a second local TWiki installation, and test the plugin there. You can allow selected users access to the test area. Once you are satisfied that it won't compromise your main installation, you can install it there as well.

InstalledPlugins shows which plugins are: 1) installed, 2) loading properly, and 3) what TWiki:Codev.PluginHandlers they invoke. Any failures are shown in the Errors section. The

`%FAILEDPLUGINS%` variable can be used to debug failures. You may also want to check your webserver error log and the various TWiki log files.

Some Notes on Plugin Performance

The performance of the system depends to some extent on the number of plugins installed and on the plugin implementation. Some plugins impose no measurable performance decrease, some do. For example, a Plugin might use many Perl libraries that need to be initialized with each page view (unless you run mod_perl). You can only really tell the performance impact by installing the plugin and by measuring the performance with and without the new plugin. Use the TWiki:Plugins.PluginBenchmarkAddOn , or test manually with the Apache `ab` utility. Example on Unix:
`time wget -qO /dev/null /twiki/bin/view/TWiki/AbcPlugin`

If you need to install an "expensive" plugin, but you only need its functionality only in a subset of your data, you can disable it elsewhere by defining the %DISABLEDPLUGINS% TWiki variable.

Define `DISABLEDPLUGINS` to be a comma-separated list of names of plugins to disable. Define it in Main.TWikiPreferences to disable those plugins everywhere, in the WebPreferences topic to disable them in an individual web, or in a topic to disable them in that topic. For example,

```
   * Set DISABLEDPLUGINS = SpreadSheetPlugin, EditTablePlugin
```

Managing Installed Plugins

Some plugins require additional settings or offer extra options that you have to select. Also, you may want to make a plugin available only in certain webs, or temporarily disable it. And may want to list all available plugins in certain topics. You can handle all of these management tasks with simple procedures:

Enabling Plugins

Plugins can be enabled and disabled with the configure script. An installed plugin needs to be enabled before it can be used.

Plugin Evaluation Order

By default, TWiki executes plugins in alphabetical order on plugin name. It is possible to change the order, for example to evaluate database variables before the spreadsheet CALCs. This can be done with `{PluginsOrder}` in the plugins section of configure.

Plugin-Specific Settings

Plugins can be configured with 1. preferences settings and/or 2. with configure settings. Older plugins use plugin preferences settings defined in the plugin topic, which is no longer recommended.

1. Use preferences settings:

Adinistrators can set plugin-specific settings in the local site preferences at Main.TWikiPreferences and users can overload them at the web level and page level. This approach is recommended if users should be able to overload settings. For security this is ***not recommended for system settings***, such as a path to an executable. By convention, preferences setting names start with the plugin name in all caps, and an underscore. For example, to set the cache refresh period of the TWiki:Plugins.VarCachePlugin , add this bullet in

Main.TWikiPreferences

- `Set VARCACHEPLUGIN_REFRESH = 24`

Preferences settings that have been defined in Main.TWikiPreferences can be retrieved anywhere in TWiki with `%<pluginname>_<setting>%`, such as `%VARCACHEPLUGIN_REFRESH%`.

To learn how this is done, use the TWiki:Plugins.VarCachePlugin documentation and Perl plugin code as a reference.

2. Use configure settings:

The administrator can set plugin settings in the configure interface. Recommended if only site administrators should be able to change settings. Chose this option to set sensitive or dangerous system settings, such as passwords or path to executables. To define plugin-specific configure settings,

- Create a Config.spec file in `lib/TWiki/Plugins/YourPlugin/` with variables, such as
  `$TWiki::cfg{Plugins}{RecentVisitorPlugin}{ShowIP} = 0;`
- In the plugin, use those those variables, such as
  `$showIP = $TWiki::cfg{Plugins}{RecentVisitorPlugin}{ShowIP} || 0;`

To learn how this is done, use the TWiki:Plugins.RecentVisitorPlugin documentation and Perl plugin code as a reference.

In either case, define a SHORTDESCRIPTION setting in two places:

- As a setting in the plugin documentation, which is needed for the extension reports on twiki.org. Example:
    - `Set SHORTDESCRIPTION = Show recent visitors to a TWiki site`
- As a global Perl package variable in the plugin package, which is needed by TWiki to show info on installed plugins. Example:
  `our $SHORTDESCRIPTION = 'Show recent visitors to a TWiki site';`

For better performance, make sure you define this in the plugin package:
`our $NO_PREFS_IN_TOPIC = 1;`

Listing Active Plugins

Plugin status variables let you list all active plugins wherever needed.

This site is running TWiki version **TWiki-6.0.0, Mon, 14 Oct 2013, build 26523**, plugin API version **6.00**

`%ACTIVATEDPLUGINS%`

On this TWiki site, the enabled plugins are: SpreadSheetPlugin, BackupRestorePlugin, ColorPickerPlugin, CommentPlugin, DatePickerPlugin, EditTablePlugin, ExplicitNumberingPlugin, GoogleAnalyticsPlugin, HeadlinesPlugin, InterwikiPlugin, JQueryPlugin, PreferencesPlugin, RenderListPlugin, SetGetPlugin, SlideShowPlugin, SmiliesPlugin, TablePlugin, TagMePlugin, TwistyPlugin.

`%PLUGINDESCRIPTIONS%`

- SpreadSheetPlugin (2013-10-10, $Rev: 26482 (2013-10-14) $): Add spreadsheet calculation like `"$SUM( $ABOVE() )"` to TWiki tables or anywhere in topic text

- BackupRestorePlugin (2013-02-16, $Rev: 25448 (2013-10-14) $): Administrator utility to backup, restore and upgrade a TWiki site
- ColorPickerPlugin (2013-02-15, $Rev: 25074 (2013-10-14) $): Color picker, packaged for use in TWiki forms and TWiki applications
- CommentPlugin (2013-02-10, $Rev: 24977 (2013-10-14) $): Quickly post comments to a page without an edit/preview/save cycle
- DatePickerPlugin (2013-09-04, $Rev: 26272 (2013-10-14) $): Pop-up calendar with date picker, for use in TWiki forms, HTML forms and TWiki plugins
- EditTablePlugin (2013-01-13, $Rev: 25108 (2013-10-14) $): Edit TWiki tables using edit fields, date pickers and drop down boxes
- ExplicitNumberingPlugin (1.6, $Rev: 19806 (2010-11-09) $): Use the **`##.`**, **`##..`** etc. notation to insert outline numbering sequences (1, 1.1, 2, 2.1) in topic's text. Also support numbered headings.
- GoogleAnalyticsPlugin (2011-05-14, $Rev: 21272 (2011-05-14) $): Adds Google Analytics javascript code to specified pages
- HeadlinesPlugin (2013-02-16, $Rev: 25104 (2013-10-14) $): Show headline news in TWiki pages based on RSS and ATOM news feeds from external sites
- InterwikiPlugin (2013-02-12, $Rev: 25126 (2013-10-14) $): Text **`ExternalSite:Page`** links to a page on an external site based on aliases defined in a rules topic
- JQueryPlugin (2013-09-28, $Rev: 26439 (2013-10-14) $): jQuery JavaScript library for TWiki
- PreferencesPlugin (2013-09-08, $Rev: 26286 (2013-10-14) $): Allows editing of preferences using fields predefined in a form
- RenderListPlugin (2013-01-28, $Rev: 24820 (2013-10-14) $): Render bullet lists in a variety of formats
- SetGetPlugin (2013-01-28, $Rev: 24822 (2013-10-14) $): Set and get variables in topics, optionally persistently across topic views
- SlideShowPlugin (2013-04-07, $Rev: 25715 (2013-10-14) $): Create web based presentations based on topics with headings.
- SmiliesPlugin (2013-01-13, $Rev: 24784 (2013-10-14) $): Render smilies as icons, like `:-)` for 🙂 or `:eek:` for 😮
- TablePlugin (2013-09-25, $Rev: 26425 (2013-10-14) $): Control attributes of tables and sorting of table columns
- TagMePlugin (2013-10-23, $Rev: 26549 (2013-10-25) $): Tag wiki content collectively or authoritatively to find content by keywords
- TwistyPlugin (2013-03-22, $Rev: 25508 (2013-10-14) $): Twisty section JavaScript library to open/close content dynamically

`%FAILEDPLUGINS%`

Plugin	Errors
SpreadSheetPlugin	none
BackupRestorePlugin	none
ColorPickerPlugin	none
CommentPlugin	none
DatePickerPlugin	none
EditTablePlugin	none
ExplicitNumberingPlugin	none
GoogleAnalyticsPlugin	none
HeadlinesPlugin	none
InterwikiPlugin	none
JQueryPlugin	none
PreferencesPlugin	none

RenderListPlugin	none
SetGetPlugin	none
SlideShowPlugin	none
SmiliesPlugin	none
TablePlugin	none
TagMePlugin	none
TwistyPlugin	none

Handler	**Plugins**
afterRenameHandler	TagMePlugin
afterSaveHandler	TagMePlugin
beforeCommonTagsHandler	EditTablePlugin PreferencesPlugin TwistyPlugin
beforeSaveHandler	CommentPlugin
commonTagsHandler	SpreadSheetPlugin BackupRestorePlugin CommentPlugin EditTablePlugin ExplicitNumberingPlugin HeadlinesPlugin JQueryPlugin SlideShowPlugin SmiliesPlugin
initPlugin	SpreadSheetPlugin BackupRestorePlugin ColorPickerPlugin CommentPlugin DatePickerPlugin EditTablePlugin ExplicitNumberingPlugin GoogleAnalyticsPlugin HeadlinesPlugin InterwikiPlugin JQueryPlugin PreferencesPlugin RenderListPlugin SetGetPlugin SlideShowPlugin SmiliesPlugin TablePlugin TagMePlugin TwistyPlugin
postRenderingHandler	EditTablePlugin GoogleAnalyticsPlugin PreferencesPlugin
preRenderingHandler	InterwikiPlugin SmiliesPlugin TablePlugin
startRenderingHandler	RenderListPlugin ***This handler is deprecated*** - please check for updated versions of the plugins that use it!

19 plugins

%FAILEDPLUGINS%

The TWiki Plugin API

The Application Programming Interface (API) for TWiki plugins provides the specifications for hooking into the core TWiki code from your external Perl plugin module.

Available Core Functions

The TWikiFuncDotPm module (`lib/TWiki/Func.pm`) describes **all** the interfaces available to plugins. Plugins should **only** use the interfaces described in this module.

⚠ ***Note:*** If you use other core functions not described in `Func.pm`, you run the risk of creating security holes. Also, your plugin will likely break and require updating when you upgrade to a new version of TWiki.

Predefined Hooks

In addition to TWiki core functions, plugins can use **predefined hooks**, or **callbacks**, as described in the `lib/TWiki/Plugins/EmptyPlugin.pm` module.

- All but the initPlugin are commented out. To enable a callback, remove the leading `#` from all lines of the callback.

TWiki:Codev.StepByStepRenderingOrder helps you decide which rendering handler to use.

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 $@;
      ```
- Keep the main plugin package as small as possible; create other packages that are loaded if and only if they are used. For example, create sub-packages of BathPlugin in `lib/TWiki/Plugins/BathPlugin/`.
- Avoid using preferences in the plugin topic; Define `$NO_PREFS_IN_TOPIC` in your plugin package as that will stop TWiki from reading the plugin topic for every page. Use Config.spec or preferences settings instead. (See details).
- Use registered tag handlers.
- Measure the performance to see the difference.

Version Detection

To eliminate the incompatibility problems that are bound to arise from active open plugin development, a plugin versioning system is provided for automatic compatibility checking.

- All plugin packages require a `$VERSION` variable. This should be an integer, or a subversion version id.

- The `initPlugin` handler should check all dependencies and return 1 if the initialization is OK or 0 if something went wrong.
    - The plugin initialization code does not register a plugin that returns 0 (or that has no `initPlugin` handler).

- `$TWiki::Plugins::VERSION` in the `TWiki::Plugins` module contains the TWiki plugin API version, currently **6.00**.
    - You can also use the `%PLUGINVERSION{}%` variable to query the plugin API version or the version of installed plugins.

Security

- Badly written plugins can open huge security holes in TWiki. 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. it is better to add sensitive configuration options to the `%TWiki::cfg` hash than adding it as preferences in the plugin topic.
    - Integrating with `configure` describes the steps
    - TWiki:Plugins.MailInContrib has an example
    - TWiki:Plugins.BuildContrib can help you with this
- Always use the TWiki::Sandbox to execute commands.
- 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.

Creating Plugins

With a reasonable knowledge of the Perl scripting language, you can create new plugins or modify and extend existing ones. Basic plug-in architecture uses an Application Programming Interface (API), a set of software instructions that allow external code to interact with the main program. The TWiki Plugin API provides the programming interface for TWiki.

Anatomy of a Plugin

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

- a Perl module, e.g. `MyFirstPlugin.pm`
- a documentation topic, e.g. `MyFirstPlugin.txt`

The Perl module can be a block of code that talks to with TWiki alone, or it can include other elements, like other Perl modules (including other plugins), graphics, TWiki templates, external applications (ex: a Java applet), or just about anything else it can call. In particular, files that should be web-accessible (graphics, Java applets ...) are best placed as attachments of the `MyFirstPlugin` topic. Other needed Perl code is best placed in a `lib/TWiki/Plugins/MyFirstPlugin/` directory.

The plugin API handles the details of connecting your Perl module with main TWiki code. When you're familiar with the Plugin API, you're ready to develop plugins.

The TWiki:Plugins.BuildContrib 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/TWiki/Plugins/EmptyPlugin.pm` to `<name>Plugin.pm`. The `EmptyPlugin.pm` module contains mostly empty functions, so it does nothing, but it's ready to be used. Customize it. Refer to the Plugin API specs for more information.

If your plugin uses its own modules and objects, you must include the name of the plugin in the package name. For example, write `Package MyFirstPlugin::Attrs;` instead of just `Package Attrs;`. Then call it using:

```
use TWiki::Plugins::MyFirstPlugin::Attrs;
$var = MyFirstPlugin::Attrs->new();
```

Writing the Documentation Topic

The plugin documentation topic contains usage instructions and version details. It serves the plugin files as FileAttachments for downloading. (The doc topic is also included *in* the distribution package.) To create a documentation topic:

1. **Copy** the plugin topic template from TWiki.org. To copy the text, go to TWiki:Plugins/PluginPackage and:
   - enter the plugin name in the "How to Create a Plugin" section
   - click Create
   - select all in the Edit box & copy
   - Cancel the edit
   - go back to your site to the TWiki web
   - In the JumpBox enter your plugin name, for example `MyFirstPlugin`, press enter and create the new topic
   - paste & save new plugin topic on your site
2. **Customize** your plugin topic.
   - Important: In case you plan to publish your plugin on TWiki.org, use Interwiki names for author names and links to TWiki.org topics, such as TWiki:Main/TWikiGuest . This is important because links should work properly in a plugin topic installed on any TWiki, not just on TWiki.org.
3. **Document** the performance data you gathered while measuring the performance
4. **Save** your topic, for use in packaging and publishing your plugin.

   OUTLINE: Doc Topic Contents
   Check the plugins web on TWiki.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 %VARIABLES%, and those required by TWiki.*>"

   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 TWiki:Plugins web.*>"

Packaging for Distribution

The TWiki:Plugins.BuildContrib is a powerful build environment that is used by the TWiki project to build TWiki 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 (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 TWiki. If your plugin uses additional files, include them all:
   - `lib/TWiki/Plugins/MyFirstPlugin.pm`
   - `data/TWiki/MyFirstPlugin.txt`
   - `pub/TWiki/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/TWiki/Plugins/MyFirstPlugin.pm`
   - `data/TWiki/MyFirstPlugin.txt`
   - `pub/TWiki/MyFirstPlugin/uparrow.gif`

Measuring and Improving the Plugin Performance

A high quality plugin performs well. You can use the TWiki:Plugins.PluginBenchmarkAddOn to measure your TWiki:Plugins.PluginBenchmarks . The data is needed as part of the Documentation Topic.

See also Hints on Writing Fast Plugins.

Publishing for Public Use

You can release your tested, packaged plugin to the TWiki community through the TWiki:Plugins web. All plugins submitted to TWiki.org are available for download and further development in TWiki:Plugins/PluginPackage .

Publish your plugin by following these steps:

1. **Post** the plugin documentation topic in the TWiki:Plugins/PluginPackage :
   - enter the plugin name in the "How to Create a Plugin" section, for example `MyFirstPlugin`
   - paste in the topic text from Writing the Documentation Topic and save
2. **Attach** the distribution zip file to the topic, ex: `MyFirstPlugin.zip`
3. **Link** from the doc page to a new, blank page named after the plugin, and ending in `Dev`, ex: `MyFirstPluginDev`. This is the discussion page for future development. (User support for plugins is handled in TWiki:Support .)
4. **Put** the plugin into the SVN repository, see TWiki:Plugins/ReadmeFirst (optional)

NEW 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 TWiki community 🙂

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 TWikiFuncDotPm functions that support saving and loading of topics and attachments.

Plugin Internal Data

You can create a plugin "work area" using the `TWiki::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, `TWiki::Func::storeFile()` and `TWiki::Func::readFile()` are provided to persistently store and retrieve simple data in this area.

Web Accessible Data

Topic-specific data such as generated images can be stored in the topic's attachment area, which is web accessible. Use the `TWiki::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`

Web specific data can be stored in the plugin's attachment area, which is web accessible. Use the `TWiki::Func::saveAttachment()` function to store the data.

Recommendation for file names in plugin attachment area:

- Prefix the filename with an underscore
- Include the name of the web in the filename
- Use only alphanumeric characters, underscores, dashes and periods to avoid platform dependency issues and URL issues
- Example: `_Main_roundedge-ul.gif`

Integrating with `configure`

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

`Config.spec` files are read during TWiki 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/TWiki/Plugins/BathPlugin/Config.spec`.

Structure of a `Config.spec` file

The `Config.spec` file for an extension starts with the extension announcing what it is:

```
# ---+ Extensions
# ---++ 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
$TWiki::cfg{BathPlugin}{PlugType} = 'Plastic';

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

# **BOOLEAN EXPERT**
# Set this option to 0 to disable the water temperature alarm
$TWiki::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 `$TWiki::cfg{BathPlugin}{PlugType}`, `$TWiki::cfg{BathPlugin}{ChainLength}`, and `$TWiki::cfg{BathPlugin}{TempSensorEnabled}` for use in your plugin. For example,

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

The Config.spec file is read by `configure`, which 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 perl structure, consisting of arrays and hashes
REGEX *length*	A perl regular expression
SELECT *choices*	Pick one of a range of choices
SELECTCLASS *root*	Select a perl package (class)
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/TWiki.spec` for many more examples.

`Config.spec` files for non-plugin extensions are stored under the `Contrib` directory instead of the `Plugins` directory.

Note that from TWiki 5.0 onwards, CGI scripts (in the TWiki `bin` directory) provided by extensions must also have an entry in the `Config.spec` file. This entry looks like this (example taken from PublishContrib)

```
# **PERL H**
# Bin script registration - do not modify
$TWiki::cfg{SwitchBoard}{publish} = [ "TWiki::Contrib::Publish", "publish", { publishing => 1 } ]
```

`PERL` specifies a perl data structure, and `H` a hidden setting (it won't appear in `configure`). The first field of the data value specifies the class where the function that implements the script can be found. The second field specifies the name of the function, which must be the same as the name of the script. The third parameter is a hash of initial context settings for the script.

TWiki:TWiki/SpecifyingConfigurationItemsForExtensions has supplemental documentation on configure settings.

Maintaining Plugins

Discussions and Feedback on Plugins

Each published plugin has a plugin development topic on TWiki.org. Plugin development topics are named after your plugin and end in `Dev`, such as `MyFirstPluginDev`. The plugin development topic is a great resource to discuss feature enhancements and to get feedback from the TWiki community.

Maintaining Compatibility with Earlier TWiki Versions

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

Organizations typically do not upgrade to the latest TWiki for many months. However, many administrators still would like to install the latest versions of a plugin on their older TWiki 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 TWiki releases. This can be done also for plugins using unofficial TWiki internal functions of an earlier release that no longer work on the latest TWiki codebase. Here is an example; the TWiki:TWiki.TWikiPluginsSupplement#MaintainPlugins has more details.

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

Handling deprecated functions

From time-to-time, the TWiki developers will add new functions to the interface (either to TWikiFuncDotPm, 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 TWiki 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 TWiki.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 TWiki 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 `TWiki::Plugins` version *in which the handler was first deprecated*. For example, if we need to define the `endRenderingHandler` for compatibility with `TWiki::Plugins` versions before 1.1, we would add this to the plugin:

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

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

Related Topics: DeveloperDocumentationCategory, AdminDocumentationCategory, TWiki:TWiki.TWikiPluginsSupplement

-- ***Contributors:*** TWiki:Main.PeterThoeny , TWiki:Main.AndreaSterbini , TWiki:Main.MikeMannix , TWiki:Main.CrawfordCurrie , TWiki:Main.ArthurClemens , TWiki:Main.WillNorris

---

This topic: TWiki > TWikiPlugins
Topic revision: r36 - 2011-06-06 - TWikiContributor