

THE BOOK OF TIDDLYWIKI

ADVANCED CUSTOMIZATION

LUIS J. GONZÁLEZ CABALLERO

Your messy thoughts. Organized.



tiddlywiki.com

- your personal wiki
- laptop, mobile, tablet
- own your data 100%
- a single HTML file
- offline or in the cloud
- open source and free

Advanced Customization

Coordinator: Luis Javier González Caballero

Written with the help of the people of the [tiddlywiki google group](#)

February 27, 2020

Collaborators	Collaboration
Dr. Rizwan Ishak	Chapter 6
Mohammad Rahmani	Chapter 11
Eskha	Chapter 12

Contents

1	Introduction	11
1.1	Key points	11
1.2	What is tiddlywiki	12
1.3	Starting with tiddlywiki	13
1.4	Reasons to use tiddlywiki	13
1.5	Elements of TW	14
1.5.1	The screen	14
1.5.2	The Story River	14
1.5.3	Tiddlers	14
1.5.4	Wikitext	15
1.5.5	Tags	16
1.5.6	Fields	16
1.5.7	Text format	16
1.5.8	Transclusion	16
1.5.9	Templates	16
1.5.10	Filters	16
1.5.11	Macros and widgets	17
1.5.12	Mechanisms	17
1.5.13	Lists	17
2	Using tiddlywiki	19
2.1	The philosophy of Tiddlywiki	20
2.2	Planning your wiki	20
2.3	Organizing microcontent with links	21
2.4	Personalize Tiddlywiki	21
2.5	Importing content	22
3	Loading tiddlywiki in the browser	23
3.1	Key points	23
3.2	Introduction	24
3.3	Architecture	25
3.4	Starting Tiddlywiki	25
3.5	Tiddlers as Basic Coding Elements	26
3.6	The Tiddlers Store	26
4	Mechanisms	29
4.1	Key points	29
4.2	Alert Mechanism	30
4.3	Draft Mechanism	31
4.4	Drag and Drop Mechanism	31
4.5	History Mechanism	32
4.6	Info Mechanism	32
4.7	Plugin Mechanism	33
4.8	Popup Mechanism	33
4.9	Saving Mechanism	33
4.10	State Mechanism	34
4.11	Translation Mechanism	35
4.12	Upgrade Mechanism	35
4.13	Boot Mechanism	35
4.14	Startup Mechanism	36
4.15	Encryption Mechanism	36

4.16	Event Mechanism	36
4.17	Parsing Mechanism	37
5	Components of the tiddlywiki screen	39
5.1	Key points	39
5.2	The TW whole screen	40
5.2.1	The story river	40
5.2.2	The right sidebars	41
5.3	Drawing its interface	41
6	Page and tiddler layout customization	43
6.1	Modifying Page Layout with System Tags	43
6.2	Modifying Page Layout with CSS	43
6.3	Modifying Tiddler Layout with System Tags	44
6.4	Modifying Tiddler Layout with CSS	44
7	Customize the TW screen	45
7.1	Key points	45
7.2	Introduction	46
7.3	PageTemplate	46
7.4	ViewTemplate	47
7.4.1	Example	47
7.5	EditTemplate	48
7.6	Formating with CSS	48
7.6.1	Example	49
7.7	Create a loading message	50
7.8	Create a new button at the toolbars	50
7.8.1	How to do it	50
7.9	Add a new global keyboard shortcut	51
7.9.1	How to do it	51
7.10	Creating a left menu	52
7.10.1	Configure the story river position	52
7.10.2	Create the tiddler menu	53
7.10.3	Create the styles for the left menu	53
7.10.4	Create the entries for the left menu	53
7.10.5	The next step	53
8	Inside a tiddler	55
8.1	Key points	55
8.2	Types of tiddlers	56
8.2.1	User tiddlers	56
8.2.2	Internal tiddlers	56
8.2.3	The tiddler type field	56
8.3	Normal tiddler	58
8.4	Plugin tiddler	58
8.5	Tag tiddler	59
8.6	Alert tiddler	59
8.7	Dictionary tiddler	60
8.8	JSON tiddlers	60
8.9	CSS tiddlers	60
8.10	Template tiddlers	60
8.11	Other system tiddlers	61
8.11.1	\$/:/config/EmptyStoryMessage	61
8.12	Adding macros and widgets	61
9	Filters	63
9.1	Key points	63
9.2	What is a filter?	64
9.3	The filter expression	64

9.4	Using filters	65
9.5	The filter basic step	66
9.6	Regular expressions	67
9.6.1	Writing regular expressions	67
9.6.2	Using regular expressions	69
10	Macros	71
10.1	Key points	71
10.2	Introduction	72
10.3	Variables and parameters	72
10.3.1	Defining variables	72
10.3.2	Parameters	72
10.3.3	Using variables and parameters	72
10.3.4	Javascript macros	72
10.4	Using macros	73
11	Widgets	75
11.1	Key points	75
11.2	Introduction	76
11.3	First approach. The manual way	76
11.4	The checkbox widget	76
11.5	The \$list widget	77
11.5.1	Paragraphs in Tiddlywiki	78
11.6	\$edit-text widget	79
11.7	\$button widget	79
11.8	Combining widgets	80
11.8.1	Collecting information	80
11.8.2	Change a tag	81
11.9	Writing your own widgets	82
11.9.1	Data structures	83
11.9.1.1	The parse tree	83
11.9.1.2	The widget tree	83
11.9.1.3	The dom node tree	83
11.9.1.4	The dom	84
11.9.2	Code notes	84
11.9.3	Code of the \$select widget	86
12	The Tiddlywiki internals	89
12.1	The Tiddlywiki architecture	89
12.2	The \$tw object	90
12.3	More information inside Tiddlywiki	94
13	Plugins	95
13.1	Introduction	95
13.1.1	Purpose	95
13.1.2	Task manager plugin	95
13.1.3	Content of the plugin	95
13.2	Level 1: Primary	95
13.2.1	Button to Create New Task	95
13.2.2	Create a List of Unfinished Tasks	96
13.2.2.1	List tasks	96
13.2.3	Create a List of Finished Tasks	97
13.2.4	Put all Sections Together	98
13.2.5	Bundle as a Plugin	98
13.2.6	The Tinka process	99
13.2.7	Distributing Rome Plugin	99
13.2.8	Packaging Rome as a Plugin	101
13.2.9	Using Rome Plugin	102

14 Tiddlywiki internationalization	105
14.1 Key points	105
14.2 Languages internals	106
14.3 The new tiddler button	106
14.4 Writing international wikis and plugins	107
14.4.1 Tiddlywiki support to internationalization	107
14.4.2 Translating messages	107
14.4.3 Translating messages in you plugin	107
15 Examples	109
15.1 A simple to-do list	109
15.1.1 Introduction	109
15.1.2 The Todolist plugin	109
15.1.3 The Todolist tiddlers	110
15.1.3.1 Icons	110
15.1.3.2 Information tiddlers	110
15.1.3.3 The styles	110
15.1.3.4 Templates	111
15.1.3.5 The macros	111
15.1.4 The Todolist main macros	111
15.2 more projects	114
16 The people of Tiddlywiki	115
17 Glossary	117
18 Resources	119



The Long Trip

Tiddlywiki is the mix of many technologies: from HTML and CSS to the javascript code. This box show all of this technologies:

1. **Start city:** when you first discover tiddlywiki you donwload an empty wiki and you start adding tiddlers.
2. **Markup Language city:** The first you discover is the Wikitext format.
3. **Fields Village:** You travel through Tags plain and realize the existence of Fields. Here are the Transclusion hills.
4. **The HTML Land:** If you go deeper, you realize that HTML is at the bottom of everything.
5. **The CSS Plain:** Soon you want to change the look of your tiddler. You must deal with CSS at the Styles zone. And with the templates hills.
6. **The Filters Magic Academy:** Suddenly you must learn a mysterious and exoteric language: Filters. You do so in this hidden academy... if you find it. If you do not learn this, you never master Tiddlywiki.
7. **The javascript Mountains:** Now you have already reached a place where few people have been. You will write here your macros and your first javascript code: the widgets.
8. **The plugins abyss:** Now its time to reach the plugins abyss and start sharing your wikis. This is an important step.
9. **Node.js Hills:** This is not a place for anyone. Here you find the npm shore and you will find the github way.
10. **The Developers Highlands:** The last step. You have already been in all places and now you reach the Highlands. Here you can collaborate with the developers and with the great master of all masters. This is a very sparsely populated place.

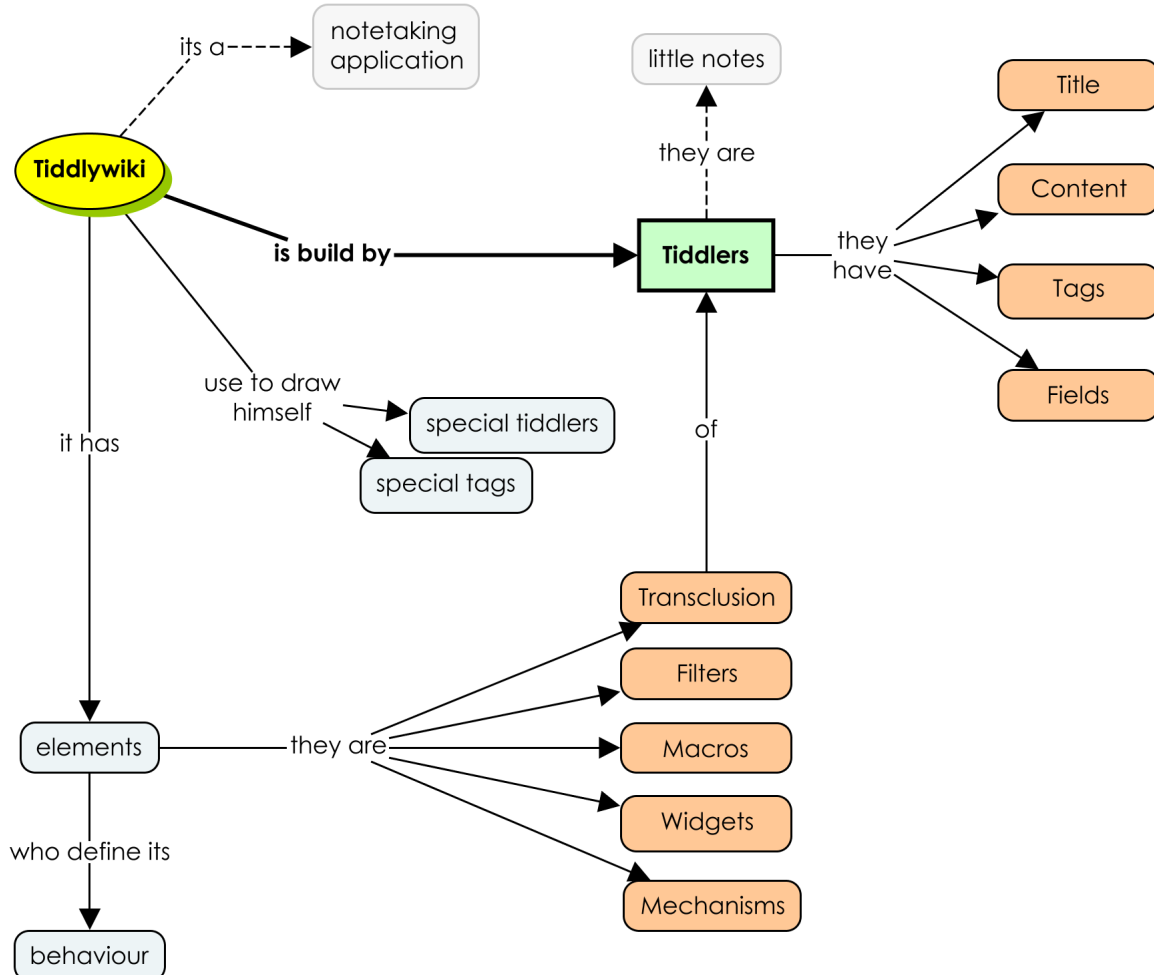
1 Introduction

1.1 Key points

- Tiddlywiki is more than a note-taking application.
- It is an advanced way of organizing your information.
- All notes you will add to it are called tiddlers.
- You can download an empty wiki from its web site.
- It is portable and multi platform.
- To add format to your text inside the tiddlers you use format characters: `//text//` for italics, `__text__` for underline, `[[Tiddler]]` for links etc. This is called Wikitext.
- It use tags to organize, classify etc the tiddlers.
- It use fields to add extra information to the tiddlers. For example the created date field, the modified date field etc.
- You can include the information of a tiddler inside another tiddler without writing twice. This is called Transclusion.
- You can use templates to personalize the way that tiddlywiki shows the tiddlers.
- Tiddlywiki uses Filters to get a group of tiddlers. Filters are written in a little language inside tiddlywiki. For example, to get all tiddlers tagged with the“Person” tag you write `{{ [tag[Person]] }}`.
- You can add code to your wiki in Macros and Widgets. This is used to personalize the wiki.

1.2 What is tiddlywiki

Figure 1.1: What is Tiddlywiki



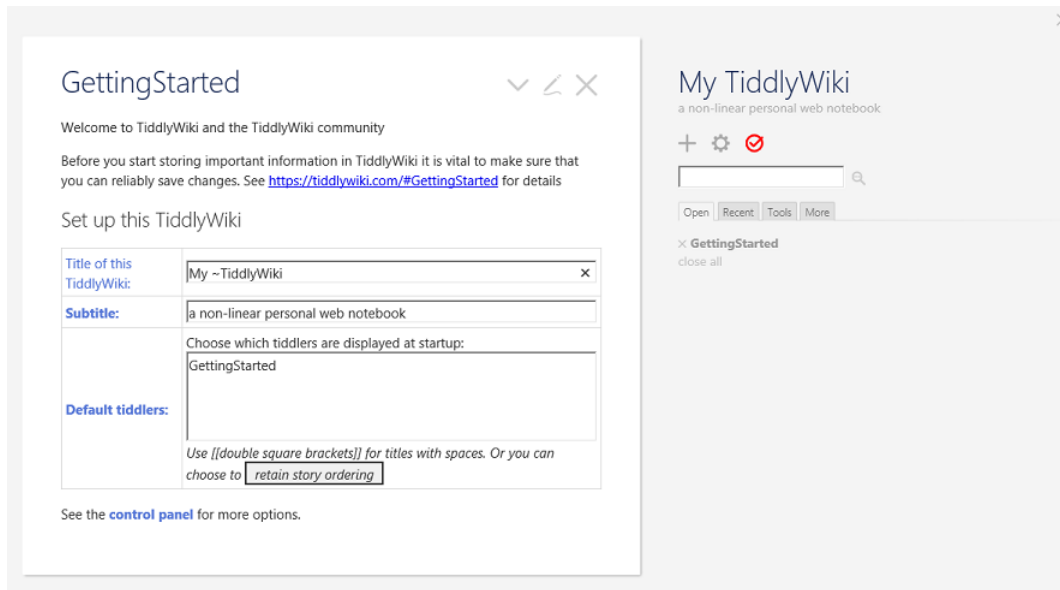
TiddlyWiki is a personal and a non-linear notebook for organizing and sharing complex information. It is an open-source single page application wiki in the form of a single HTML file that includes all javascript code, the CSS format and the content. It is designed to be easy to customize and re-shape depending on application. It facilitates re-use of content by dividing it into small pieces called Tiddlers. It is not an application but a large html page that runs in almost all web browsers so it is very portable: you can use it in a USB stick, in a phone or tablet or as a web page in some internet servers. Tiddlywiki is made of tiddlers: little text areas or notes with a title and a content. You can add all the tiddlers you want and each of them will contain certain information.

It was created by the British software developer Jeremy Ruston in 2004. Tiddlywiki is free and open source software and is distributed under the terms of the BSD license.

Tiddlywiki introduces the concept of microcontent: the smallest structured and addressable piece of information, the smallest semantically meaningful units. This small piece of information is called tiddler. The purpose of this tiddlers is recording and organizing information is so that it can be used as many times as necessary.

You can find the basic information of Tiddlywiki in the web site: <https://tiddlywiki.com/>. I recommend you take a look.

Figure 1.2: The first run of Tiddlywiki



1.3 Starting with tiddlywiki

The first step will be download an empty wiki page to our computer:

1. Go to <https://tiddlywiki.com/>.
2. Look for the tiddler“GettingStarted”
3. Click on the red button,“Download Empty”
4. Save the empty tiddlywiki in your computer.

Once you have the file you can open it with the browser (best with Firefox or Chrome). You will see the page of the Figure 1.2. Assign a title and a subtitle to your wiki. You will notice that the tick icon above the subtitle change to red.

This shows us an important thing: as an html web page, the TW file can't save by itself. You musts save them by hand if you don't want to loose the data. Click in this red icon and save the file in the same location (maybe you have to configure your browser). The next time you open the empty wiki in the browser the title and subtitle will have changed. Other operations you can do is to create an initial tiddler and change the“Default tiddlers” text area to the title of this first tiddler. To create tiddlers you will click on the "plus" icon under the subtitle.

The First Rule of using Tiddlywiki: backup your wiki file regularly retaining some backward copies.

I insist: while you can edit any TiddlyWiki (even tiddlywiki.com), the changes will only exist on the editor's computer, so if you close the browser without saving your wiki you will lose your work,

1.4 Reasons to use tiddlywiki

You have powerful reasons to use tiddlywiki:

- You can use them without installing any software.
- It is multiplatform: use your wiki in your tablet, mac, linux, windows or phone systems.
- It is portable: you can put in a USB stick, copy to your computer or upload to many internet servers.

- Its nonlinear approach allows you to use the information in new and helpful ways.
- You can organize your information and knowledge your way.
- You can change the appearance and behavior of your wiki and adapt them to your preferences.
- Tiddlywiki promotes information sharing.

1.5 Elements of TW

1.5.1 The screen

When you open a Tiddlywiki file you can see a left size with all open tiddlers and a right side with the title, buttons, and menus. In chapter 3 we will see the whole screen (look at the Figure 5.1).

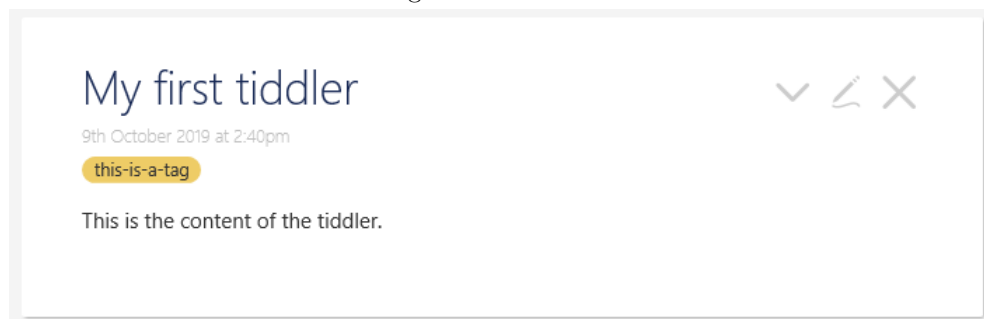
1.5.2 The Story River

The left side of the Tiddlywiki is called the Story River and shows all open tiddlers.

A typical wiki contains hundreds or thousands of tiddlers, some of them open in the left side and others closed stored in the file. You can search the closed tiddlers with the search bar or with the menus.

1.5.3 Tiddlers

Figure 1.3: A tiddler



All notes you can add to Tiddlywiki are written in Tiddlers. It consist of a Title on the top, the tags and its content. A Tiddler is the basic element of Tiddliwiki. All things inside TW are made of tiddlers. The GettinStarted page you can see is a tiddler. The right menus too.

The most important thing in a tiddler is its title: this will be unique. In Figure 1.3 you can see the first tiddler:

- Buttons to edit and close the tiddler.
- Title:“My first tiddler”
- The field: created date.
- Tags: only one:“this-is-a-tag”. You can add as many tags as you want.
- Content:“This is the content of the tiddler (the text field)”.

When you click the“pen” icon at the top you can see a tiddler in edit mode:

Figure 1.4: Creating a tiddler

- Buttons to delete, discard and save the tiddler
- Title at the top: “New Tiddler”
- Tag zone
- The editor Toolbar
- The content of the tiddler
- The type of the tiddler
- The fields zone. There are predefined fields but you can add your own fields.

If you close the tiddler it disappears from the screen but if you save the tiddler you can search it with the search bar, under the save button or choosing in the right menu More / All or look in the recent tab.

1.5.4 Wikitext

It is an easy way of formatting your text. In figure of page 61 you have all formatting options.

Tiddlywiki defines a set of rules for each formatting option. Each rule is defined in a wikirule tiddler. You can see all this formatting rules adding this code in a new tiddler:

```
{{{ [ all [ shadows ] module-type [ wikirule ] ] }}}
```

You will see the `$/core/modules/parsers/wikiparser/rules/dash.js` tiddler for the dash and `$/core/modules/parsers/wikiparser/rules/list.js` for wikilist.

You can selectively deactivate this rules writing at the top of the tiddler:

- `\rules_except_dash_list`: This code deactivate the dash and list wikitext rules.
- `\rules_only_list`: This code applies only the list wikirule.

1.5.5 Tags

Tagging is a way of organizing tiddlers into categories. For example, if you had tiddlers representing various individuals, you could tag them as friend, family, colleague etc to indicate these people's relationships to you. By tagging your tiddlers, you can view, navigate and organize your information.

Tiddlywiki has a Tag manager. Look in SideBar > Tools > Tag manager or open the tiddler `$/Tag-Manager`. With this tiddler you can change the color and add an icon for the tag .

1.5.6 Fields

A tiddler has fields. There are system fields like the created and modified date (hidden) but you can add your own ones. Look at <https://tiddlywiki.com/#TiddlerFields>

1.5.7 Text format

The formatted text inside a tiddler is called "Wikitext". The best way of learning Wikitext is playing with the toolbar above the tiddler in edition mode. For example, if you want some words to be italicized and click in the italic icon of the tool bar while editing a tiddler, Tiddlywiki will add two slashes around the words: `//some words//`. WikiText is a concise, expressive way of typing a wide range of text formatting, hypertext and interactive features. It allows you to focus on writing without a complex user interface getting in the way. It is designed to be familiar for users of Markdown, but with more of a focus on linking and the interactive features.

Another example: you can make a link of a tiddler inside other tiddler writting `[Tiddler Title Linked]`.

1.5.8 Transclusion

Transclusion is the process of referencing one tiddler "A" from another tiddler "B" such that the content of "A" appears to be a part of "B". It avoid Avoid having duplicate information.

- To show the information of a tiddler inside other, write `{{Tiddler Title}}`.
- To show the content of a field write `{{Tiddler Tittle !! field name}}`.
- to show the content of a field in the same tiddler write `{{!! field name}}`

1.5.9 Templates

Is a role a tiddler can have. Is like a shape for other tiddlers. It tells other tiddlers the way they have to display: how to show the title, the tags, its content and the other tiddlers. When you download an empty tiddler the initial template for all tiddlers is:

- `$/core/ui/ViewTemplate` if the tiddler is in view mode (you are not editing it)
- `$/core/ui/EditTemplate` if you are editing the tiddler.

They (View and Edit templates) are tiddlers and Special tags `$/tags/ViewTemplate` and `$/tags/EditTemplate` can be placed on any tiddler to make it appear in view or edit modes. You can search them in the advanced search and look for its content. And you can add your own templates. Various other templates are and can be used in tiddlywiki. The View and Edit templates above, are however the most important to designing your tiddlywiki.

1.5.10 Filters

You can think of TiddlyWiki as a database in which the records are tiddlers. A database typically provides a way of discovering which records match a given pattern, and in TiddlyWiki this is done with filters.

A filter is a concise notation for selecting a particular set of tiddlers. For example, to show the titles of all tiddlers tagged with the "learn" tag we can write `{{[tag[learn]] }}`. The `[tag[learn]]` item is the filter and the brackets the way to add the links.

1.5.11 Macros and widgets

Tiddlywiki is highly customizable. It use macros and widgets to personalize its appearance (Tiddlywiki use templates for this too).

A macro is a named snippet of text. When you use it, Tiddlywiki shows its content.

A widget is a piece of code to perform some actions.

1.5.12 Mechanisms

All elements of tiddlywiki fit together through a number of mechanisms. For example, the History Mechanism keeps track of a list of tiddlers comprising the navigation history. The StartupMechanism runs the installed startup modules at the end of the boot process.

This are the Mechanisms::

- AlertMechanism
 - What: It manage the alerts.
 - How:
 - * When creating an alert appears in the upper left

1.5.13 Lists

When you look some information inside Tiddlywiki it gives you a list of tiddlers, so the management of lists is an important question in Tiddlywiki.

The more simplest is a tiddler list. This is a list with a few number of tiddlers. For example:

[[First Tiddler]] SecondTiddler [[Third tiddler]] Finaltiddler

is a list with 4 tiddlers. You can see tiddlers inside [[]] and other tiddlers not. If your tiddler contains spaces you have to surround with [[]].

You can construct a list with filters. For example if you find this code in a tiddler:

<\$list filter="[tag[\$:/tags/PageTemplate]]">

this means a list with all tiddlers tagged with the \$:/tags/PageTemplate tag.

The most important thing here is the order of the tiddlers inside the list. In the first example the order is clear but in the second is not. Tiddlywiki will order the list alphabetically. But, imagine you need other order in this \$list inside your tiddler. In this case you can include this order in the “list” field of the tiddler. Tiddlywiki will respect that order.

2 Using tiddlywiki

- Is not a good choice start your tiddlywiki writing tiddlers directly.
- If you do so, maybe you fill your tiddlywiki with unnecessary information and forget the most important.
- It is a good habit to do a little planning.
- Set the goal of your wiki.
- Establishes good main tags and other transversal tags. You can color them.
- Write tiddlers dividing your information in very small pieces.
- Weaves all the story transcluding this very small tiddlers in a big one, writing only a small piece of code.
- Set links to individual tiddlers. It is very very easy.
- Set list of links with widgets and filters. You can list a subset of all of your tiddlers based in tags or fields.
- Create index tiddlers tagging them with special tags.
- Change the appearance with css tiddlers.
- Change, eliminate or add elements to the Tiddlywiki screen changing some special tiddlers and special tags.

2.1 The philosophy of Tiddlywiki

The purpose of recording and organizing information is so that it can be used again. The value of recorded information is directly proportional to the ease with which it can be re-used.

The philosophy of tiddlers is that we maximize the possibilities for re-use by slicing information up into the smallest semantically meaningful units with rich modeling of relationships between them. Then we use aggregation and composition to weave the fragments together to present narrative stories.

TiddlyWiki aspires to provide an algebra for tiddlers, a concise way of expressing and exploring the relationships between items of information.

TiddlyWiki let you to keep all your things in a non-linear notebook. Divide the complex notes in simple contents and store each of them in a single tiddler. We called each of this simple contents “microcontent”.

Tiddlers are the fundamental units of information in TiddlyWiki. Tiddlers work best when they are **as small as possible** so that they can be reused by weaving them together in different ways.

TiddlyWiki gives you different ways to structuring this microcontent:

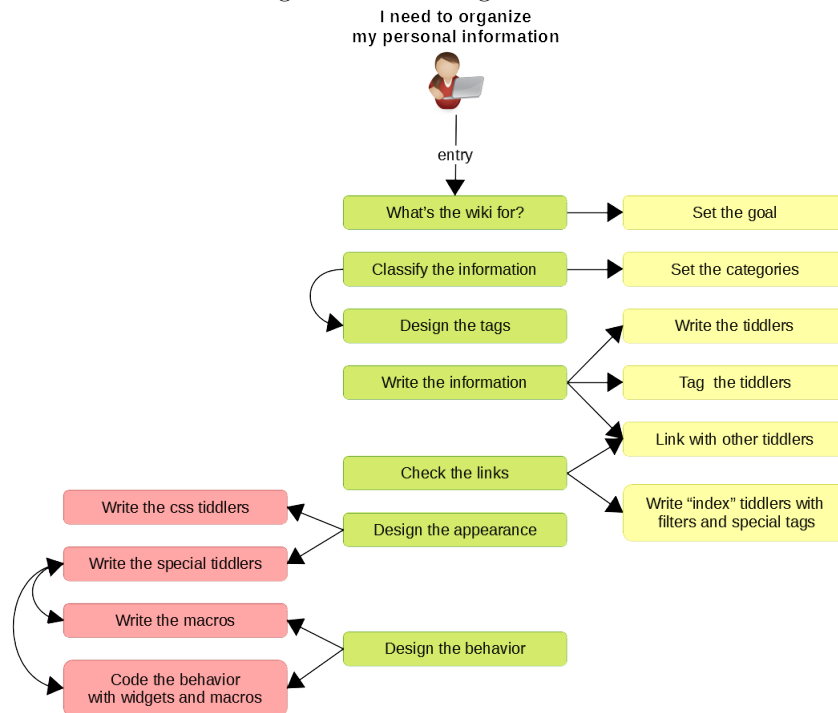
- Format the tiddler content with Wikitext.
- Use tags to group tiddlers.
- Use fields to store simple tiddler data.
- Include the content of a tiddler inside other tiddler.
- Link your tiddlers with Tiddler links.
- Add external links.
- Search information in your wiki with filters. This produce a list of tiddler titles.
- Customize this list of this filters with widgets and macros.
- Add simple data inside DataTiddlers.

2.2 Planning your wiki

When we create a wiki, the temptation is to start creating tiddlers and write our notes directly on them. However, a small analysis should be done. In the Figure 2.1 you can see a little approach to this analysis:

- **Set the goal of your wiki.** It’s important if you don’t want to fill it with unnecessary information.
- **Classify the information.** Setting the hierarchy of your information will help you later to find it more easily. At this point you can set the main tags.
- **Design the tags.** In addition to the main tags it may be interesting to add other "cross" tags (for example tags for place, people, status... etc)
- **Write the information.** Now that you know the information you want to add, the tags and the categories it is time of writing the tiddlers, tagging them appropriately. Write the tiddlers and tag them.
- **Check the links.** Check your tiddlers adding links where appropriate. At this point maybe you have to add some “index” tiddlers writing good filters.
- **Design the appearance.** If you want to customize the appearance of your tiddlywiki you have to write the css tiddlers and the special Tiddlywiki tiddlers.
- **Design the behavior.** Write the macros and organize the actions writing code in the tiddlers and special tiddlers.

Figure 2.1: Planning the wiki



2.3 Organizing microcontent with links

TiddlyWiki is designed to be non-linear, structuring content with stories, tags, hyperlinks, and other features. You can organize and retrieve your notes in ways that conform to your personal thought patterns, rather than feel chained to one preset organizational structure.

The first thing you have to do is divide your information in very small pieces and write them in tiddlers. These tiddlers can then be woven together to create longer units: stories, articles, lists, image galleries, and much more. TiddlyWiki's features are specially designed to help you relate and connect tiddlers together in multiple ways, facilitating your future retrieval of your notes and even helping you see unexpected relationships among your tiddlers and the information they contain.

What features can you use?

- **Links:** Tiddlywiki allows you to easily add links to other tiddlers. You just have to write `[[Title of the tiddler]]` and you already have your link. Add as many links as you want.
- **Filters:** The filters help you choosing a subset of tiddlers based on your tags or fields and showing it as a list of tiddlers. You can add links to this list too.
- **Tags:** Tags allows you in many ways:
 - Tags divide your knowledge in categories.
 - Tags allow you adding custom appearance depending of this categories.
 - They help you in creating indexes.

2.4 Personalize Tiddlywiki

TW allows a deep customization through the different elements that it puts at our disposal:

- **Startup tiddlers:** Yo can choose which tiddlers opens at startup time in the Control panel tiddler.
- **CSS Styles:** You can change the font, colors, borders of the screen writing css tiddlers.

- **Elements of the Tiddlywiki screen:** Changing some special tiddlers and special tags you can change this elements. You can even delete some and add others.
- **Widgets:** You can add more complex actions to your tiddlywiki writing code in tiddlers with macros and widgets.

2.5 Importing content

Open your favorite text editor and write this lines:

```
title :$/MyTags
tags : one two [[number three]] four five six seven eight nine ten
```

Save this file as Mytags.tid.

Now drag this file into your tiddlywiki file. You have a new tiddler, “\$/MyTags” tagged with all that tags. If you add a new tiddler all this tags are in the tags drop down field.

You can add in this way all fields you want: title, tags, text (the tiddler content) and your personal fields.

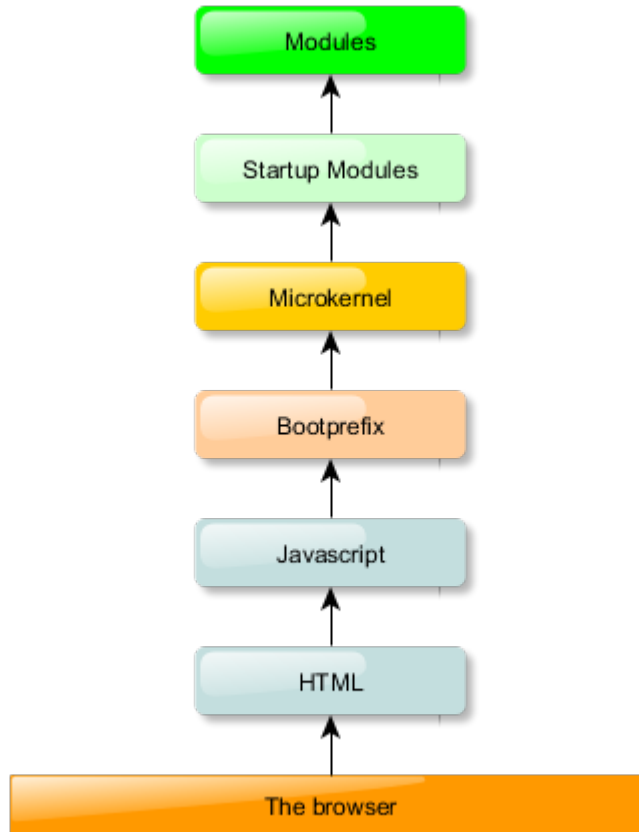
3 Loading tiddlywiki in the browser

3.1 Key points

- When the browser loads the Tiddlywiki file it runs its javascript code.
- The Bootprefix of the code loads the Microkernel.
- The Microkernel is the the only part of Tiddlywiki that is not managed by tiddlers.
- The rest of the application is managed by modules stored in tiddlers as javascript code.
- A tiddler can contain many kinds of data: text, images, javascript code (modules), JSON data...
- A tiddler can have many roles: a plugin, data, formatting code...
- All the tiddlers that make up the core of Tiddlywiki are stored in the "\$:/core" tiddler.

3.2 Introduction

Figure 3.1: Architecture of Tiddlywiki



If you edit the empty.html wiki file you will see something like this:

```
<html>
<head> ... </head>
<body>
<div id="styleArea">css styles</div>
<div id="storeArea">The tiddlers of the wiki</div>

<div id="libraryModules" style="display:none">
<script>
...
</script>
</div>

<div id="bootKernelPrefix" ... type: application/javascript>
<script>
/*\
  title: $:/boot/bootprefix.js
  type: application/javascript
  ...
</script>
</div>

<div id="bootKernel" style="display:none">
<script>
/*\
  title: $:/boot/boot.js
  type: application/javascript
  ...
</script>
</div>
```

```
</body>
</head>
```

So, at the bottom Tiddlywiki is a html page with a lot of javascript code inside it. When the browser loads this file, it runs the javascript code and the content of the page change. At the end of this chapter there is a diagram of the html code after the browser has loaded this file.

3.3 Architecture

What happens when loading a tiddlywiki file? In Figure 3.1 we see its architecture.

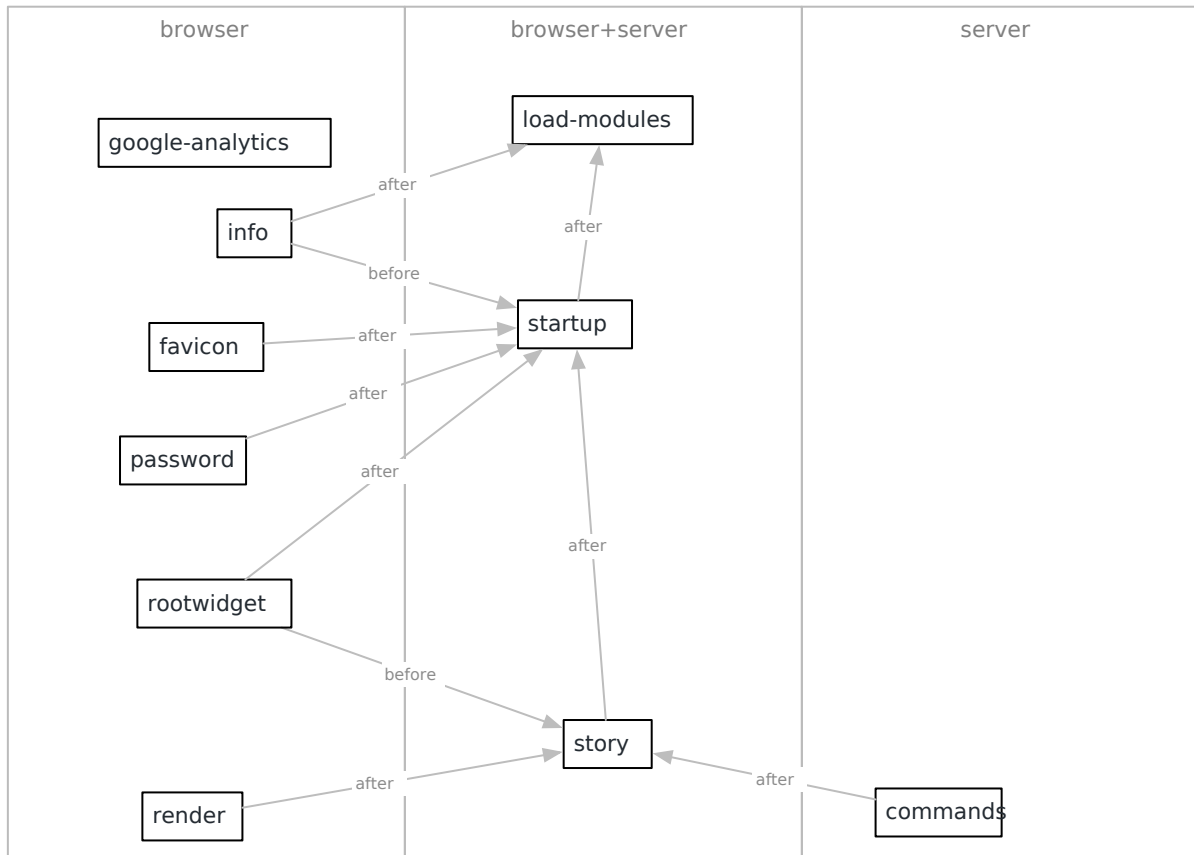
1. First of all the operating system loads the browser.
2. The browser loads the HTML page.
3. After some css styles and the store area we can see the javascript code. The browser runs this code.
4. It loads the bootprefix. The bootprefix is responsible for preparing the kernel to boot on different engines e.g. browsers and node.js.
5. The microkernel is the first thing to run, when the application is started and it puts some initial objects and functions into the application tree, which are needed to load and manage tiddlers. After the microkernel built this initial application tree, the remaining parts of the application can be loaded as module tiddlers.
6. The microkernel load the startup modules (startup tiddlers).
7. At the top we have all modules (tiddlers).
8. Loading startup tiddlers

3.4 Starting Tiddlywiki

In the Figure 3.2 we can see the startup process. After the browser loads the empty.html file it runs the bootkernel and boot javascript code. Then, Tiddlywiki takes control and code for the rest of the process is located in tiddlers. In this table we can see these start-tiddlers and its functions:

name	tiddler	function
load-modules	<code>\$/core/modules/startup/load-modules.js</code>	Load core modules
info	<code>\$/core/modules/startup/info.js</code>	Collect the info tiddlers
startup	<code>\$/core/modules/startup.js</code>	Startup logic and kick off some services
favicon	<code>\$/core/modules/startup/favicon.js</code>	The favicon tiddler
password	<code>\$/core/modules/startup/password.js</code>	Starts password mechanism
rootwidget	<code>\$/core/modules/startup/rootwidget.js</code>	Setup the root widget and the core root widget handlers
story	<code>\$/core/modules/startup/story.js</code>	Prepare the story list
render	<code>\$/core/modules/startup/render.js</code>	Title, stylesheet and page rendering
windows	<code>\$/core/modules/startup/windows.js</code>	Setup root widget handlers for the messages concerned with opening external browser windows
browser-messaging	<code>\$/core/modules/browser-messaging.js</code>	Browser message handling
commands	<code>\$/core/modules/startup/commands.js</code>	Command processing
css-escape	<code>\$/core/modules/startup/CSSescape.js</code>	Polyfill for <code>CSS.escape()</code> .

Figure 3.2: Startup process



3.5 Tiddlers as Basic Coding Elements

Only a small part of the Tiddlywiki is not managed by tiddlers: the Microkernel. After the microkernel built this initial application tree, the remaining parts of the application can be loaded as module tiddlers.

A tiddler is the smallest unit of the TiddlyWiki system. It can contain any data like plain text, WikiText markup, JavaScript code (module tiddler), JSON structures (JSON structures might even contain additional tiddlers). Plug-ins are implemented this way to pack multiple tiddlers in a single plug-in tiddler), images in SVG format or even binary images encoded with base64. Internally Tiddlers are immutable objects containing a bunch of key:value pairs called fields. The only required field of a tiddler is the title field. In section 8.2.3 you can see all roles of a tiddler.

3.6 The Tiddlers Store

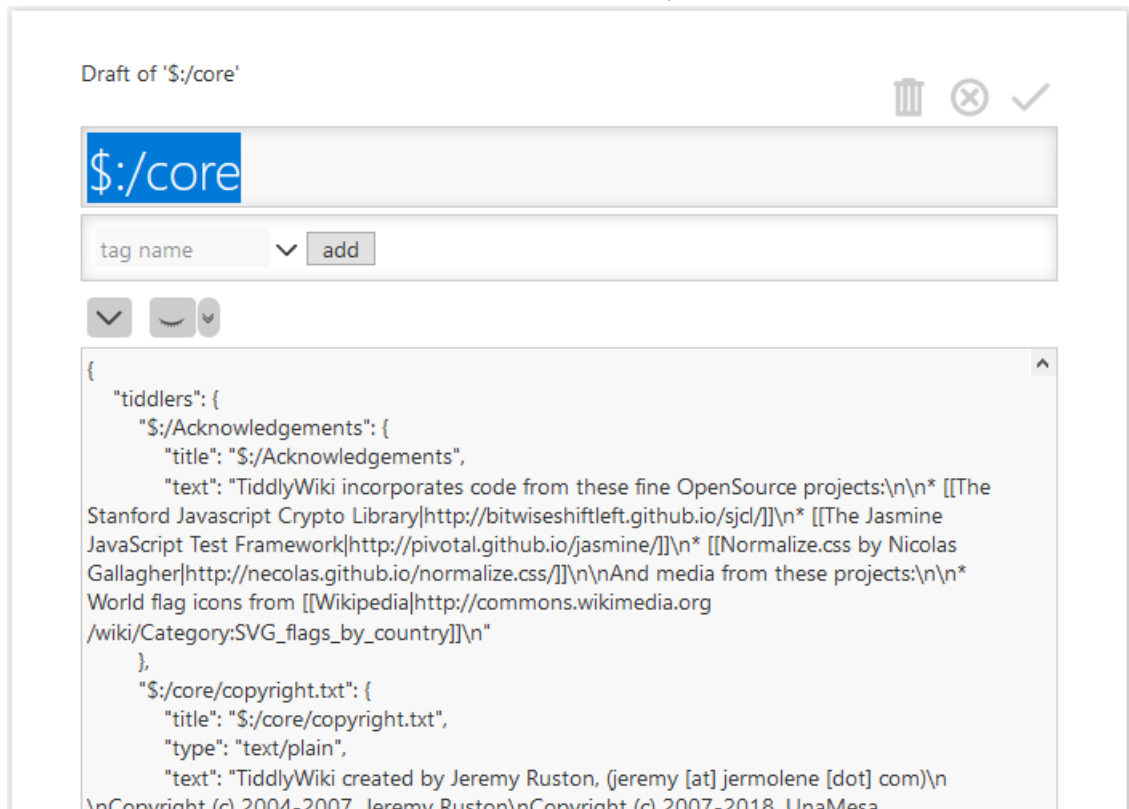
All Tiddlywiki is built as a plugin, included the core tiddlers. If you build your own tiddlywiki with node.js and the core files of the npm repository, all the tiddlers are stored in the \$:/core tiddler and become shadow tiddlers like any other plugin. If you remember, before loading tiddlywiki in the browser all this core tiddlers are located in the <div id="store Area"> section of the html file.

If you edit a shadow tiddler, for example "\$:/Acknowledgements" it appears a message: "You are about to edit a ShadowTiddler. Any changes will override the default system making future upgrades non-trivial. Are you sure you want to edit "\$:/Acknowledgements"?"

After the edition you will have two "\$:/Acknowledgements" tiddlers: your new one and the other located in "\$:/core" JSON shadow tiddler. And if you delete your new "\$:/Acknowledgements" Tiddlywiki loads the shadow one.

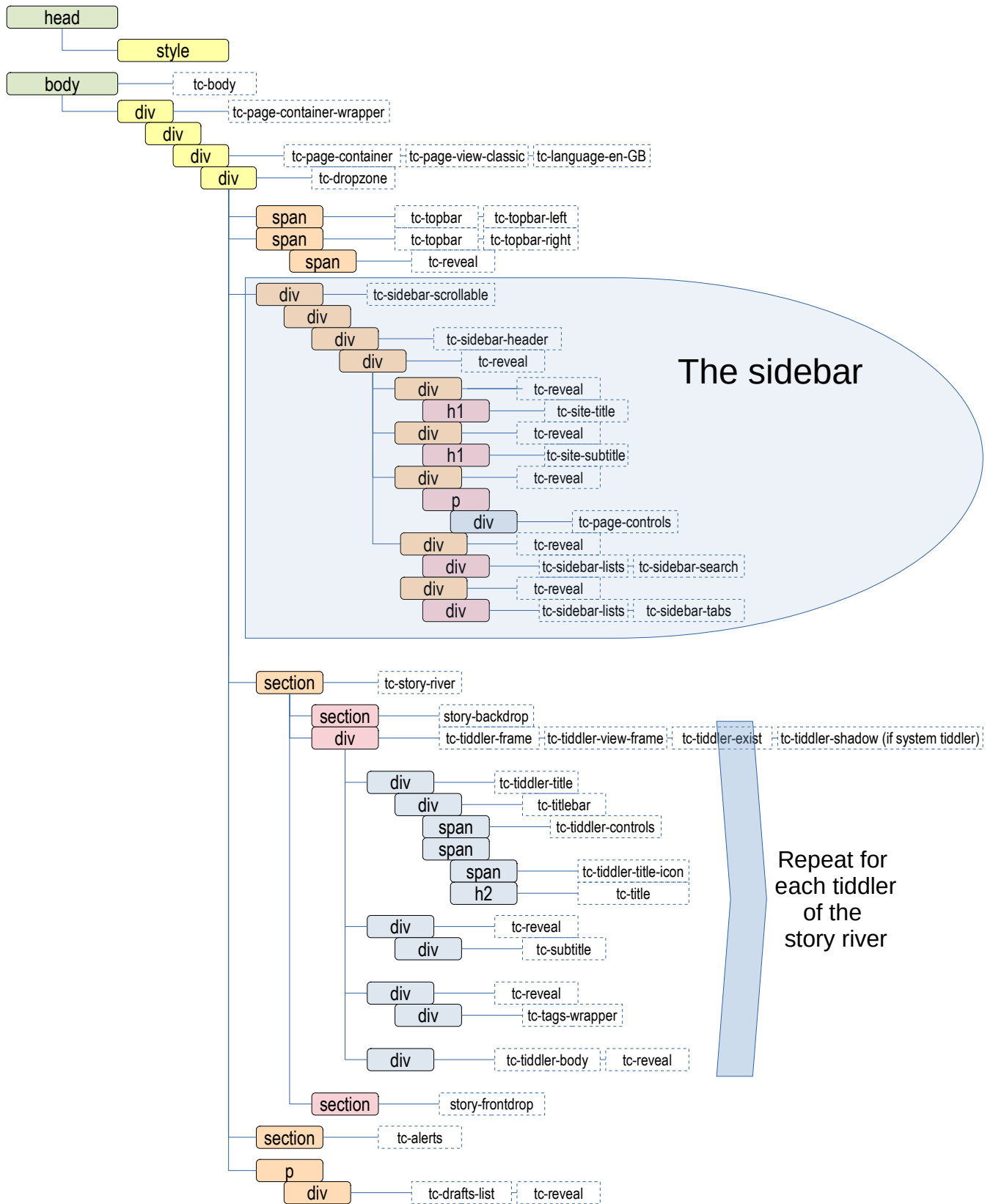
Now a little tricky question: you can edit the \$:/core tiddler. It is a JSON tiddler composed with core tiddlers. Look at figure 3.3. It starts with { "tiddlers": { "\$:/Acknowledgements":

Figure 3.3: Editing \$:/core



So “\$:/Acknowledgements” is the first Shadow tiddler of this core plugin. And you can see that the next tiddler is “\$:/core/copyright.txt”. If you delete the “\$:/Acknowledgements” content from the \$:/core tiddler and save the wiki it disappears completely unless you have edited it and have a copy as a normal tiddler.

Be careful!. There is no other copy of this “\$:/core” tiddler like the others shadows tiddlers, so you can’t delete it to recover the original one. If you delete it, your Tiddlywiki will never run again.

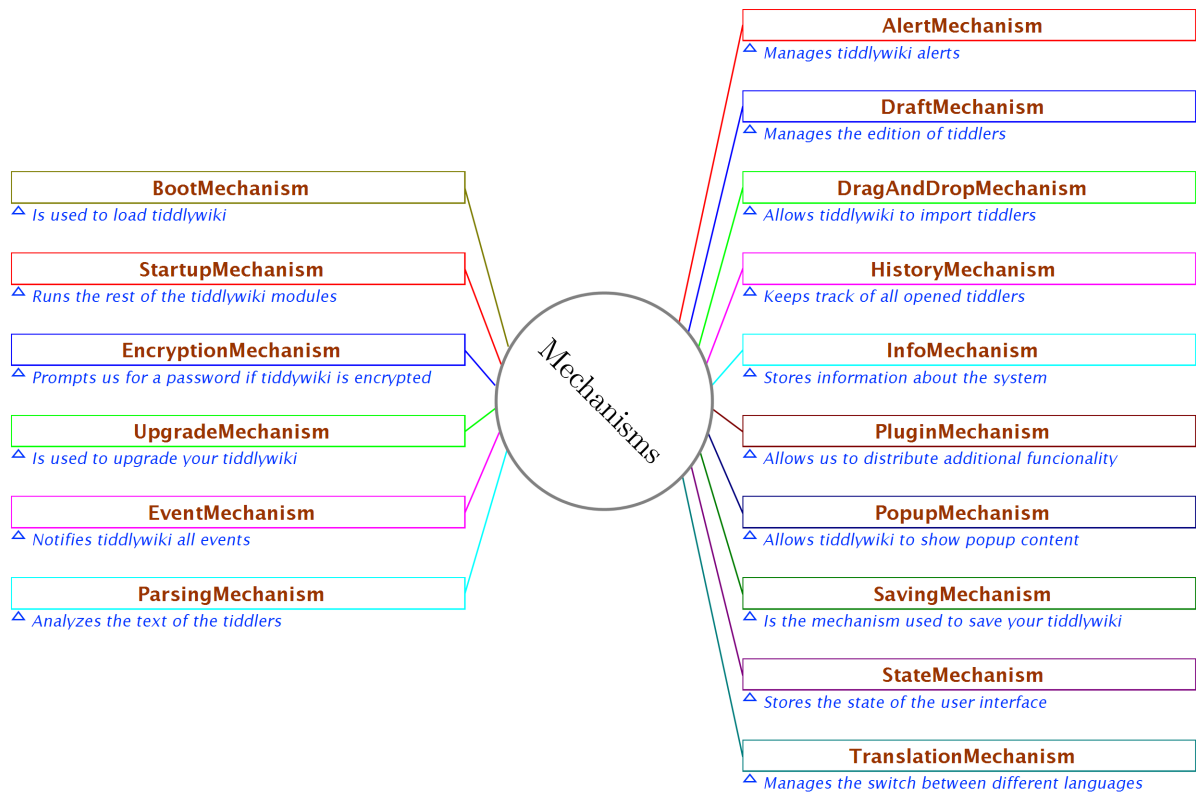


4 Mechanisms

4.1 Key points

- The AlertMechanism manage the alerts
- The DraftMechanism manage the edition of tiddlers.
- The DragAndDropMechanism allows us to import and manipulate tiddlers.
- The HistoryMechanism keeps track to all opened tiddlers. Each time a user open a tiddler it is added to the end of \$:/HistoryList JSON tiddler.
- The InfoMechanism stores information about the system.
- The PluginMechanism allows to distribute and manage additional functionality.
- The PopupMechanism allows us to show popup content.
- The SavingMechanism is the mechanism used to save your tiddlywiki..
- The StateMechanism is a way of storing the state of the elements of the user interface.
- The TranslationMechanism manages and switches between different languages.
- The UpgradeMechanism is used to upgrade your wiki.
- The BootMechanism runs the heart of Tiddlywiki and it is used to load the rest of the modules.
- The StartupMechanism runs all javascript modules whose module-type is set to “startup”.
- The EncryptionMechanism prompts for a password if the tiddlywiki is encrypted.
- The EventMechanism notify Tiddlywiki all events.
- The ParsingMechanism analyze the text of the tiddlers against a set of rules. The parser’s module-type is “parser” and the rule’s “wikirule”.

Figure 4.1: Mechanisms



4.2 Alert Mechanism

What

The AlertMechanism manage the Tiddlywiki alerts.

How

- When creating an alert it appears in the upper left.
- Tiddlywiki display them with the “\$/core/ui/AlertTemplate” template.
- The “\$/core/ui/PageTemplate/alerts” manage how Tiddlywiki display them.
- The alert can be deleted with the trash icon
- You can change its position with the .tc-alerts style. For example, if you want the alerts at the bottom create a style tiddler with this code: `.tc-alerts{position:fixed;bottom:0;right:0;max-width:500px;z-index:20000;}`
- The alerts has four fields:
 - title: By default, alert titles have the prefix \$:/temp/alerts/
 - text: The text of the message

- modified: Date of the alert
- component: Component name associated with the alert
- tags: Must include “\$/tags/Alert”

4.3 Draft Mechanism

What

It manages the tiddlers that are being edited.

How

- When a tiddler is in “edit mode” (you are creating a new tiddler or editing an existing one) a field "draft.of" is added.
- They also have a "draft.title" field, with the title with which the tiddler will be saved.
- They are drawn with the edit template.
- The “Navigator” widget incorporates handlers for the following events:
 - tm-new-tiddler
 - tm-edit-tiddler
 - tm-cancel-tiddler
 - tm-save-tiddler
- This “Navigator” widget manipulates the current store, the story list and history lists in response to this messages.
- The draft tiddlers are excluded from search operations
- The Page template includes a new section at the bottom, “\$/core/ui/PageTemplate/drafts”. It shows in red all tiddlers saved with your wiki in draft mode.

4.4 Drag and Drop Mechanism

What

TiddlyWiki uses drag and drop to power two separate features:

- Importing Tiddlers into TiddlyWiki
- Manipulating tiddlers within a TiddlyWiki
 - Entries in the "Open" tab of the sidebar can be reordered by drag and drop; new tiddlers can be opened by dragging their titles into the list
 - Entries within a tag pill dropdown can be reordered by drag and drop; new tiddlers can be assigned the tag by dragging their titles into the list
 - Entries in the control panel "Appearance"/"Toolbars" tab can be reordered by drag and drop. (Less usefully, new entries can be added to the toolbars by dragging their titles into the list)

How

Tiddlywiki supports drag and drop adding a zone in the html file (look at page 28 for this “dropzone”). It uses the standard HTML 5 drag and drop APIs. However, this is an area that is rightly notorious for cross-browser compatibility problems. Therefore, some features that you might expect to work won’t necessarily work in all browsers.

The general sequence of a drag and drop operation is as follows:

1. The user clicks down and drags the pointer on a draggable element such as the DraggableWidget, ButtonWidget or LinkWidget
2. The draggable element moves with the mouse pointer until the click is released
3. Moving the pointer over droppable elements such as the DroppableWidget displays a highlight indicating that the item can be dropped
4. The configured actions are performed if the drag ends on a droppable element

4.5 History Mechanism

What

The History mechanism keeps track of a list of tiddlers comprising the navigation history. Each time you click on a link to a tiddler, the title of the target tiddler is added to the top of the stack.

So, tiddlywiki remembers all the tiddlers you have opened in the current session. The history grows each time you open a tiddler. The initial tiddlers opened at the load of the html files are not added to the history.

How

Tiddlywiki stores all open tiddlers in the system tiddler \$:/HistoryList. This history list is stored in JSON to allow additional details about the coordinates of the DOM node that initiated the navigation. The history list also maintains the field current-tiddler that contains the name of the tiddler at the top of the stack.

4.6 Info Mechanism

What

It stores information about the system

How

System tiddlers in the namespace \$:/info/ are used to expose information about the system (including the current browser) so that WikiText applications can adapt themselves to available features. These are the tiddlers:

- \$:/info/browser
- \$:/info/browser/language
- \$:/info/browser/screen/width
- \$:/info/browser/screen/height
- \$:/info/node
- \$:/info/url/full

- \$:/info/url/host
- \$:/info/url/hostname
- \$:/info/url/origin
- \$:/info/url/pathname
- \$:/info/url/port
- \$:/info/url/protocol
- \$:/info/url/search

4.7 Plugin Mechanism

What

Plugins are bundles of tiddlers that are distributed and managed as a single unit by being packed into a single JSON tiddler. Users can install them with drag and drop, or using the plugin library.

The tiddlers within registered plugins behave as ShadowTiddlers: they can be freely overwritten by creating a tiddler with the same title, but deleting that tiddler restores the underlying tiddler value from the plugin.

How

You can pack all your tiddlers inside a new one manually or with the [Tinka plugin](#).

Plugins conventionally have a title of the form \$:/plugins/publisher/name. Plugins that are part of the core TiddlyWiki distribution have titles of the form \$:/plugins/tiddlywiki/name.

When running TiddlyWiki under Node.js, plugins can also be stored as individual tiddler files in Plugin-Folders.

If you want to know what plugins include your wiki go to the control panel, plugin section.

4.8 Popup Mechanism

What

The popup mechanism allows blocks of content to be selectively displayed and positioned with respect to an anchor.

How

Tiddlywiki handles popups by:

- The Reveal widget
- The Button widget
- State tiddlers

This means that you have to program the popus with widgets and status tiddlers.

4.9 Saving Mechanism

The saving mechanism is the mechanism by which TiddlyWiki generates a new HTML file and stores it.

What

After modifying your tiddlywiki you have to save it. This can be done in many ways:

- Manually: clicking in the save-wiki button
- Node.js: if you run Tiddlywiki under this server
- [Tiddlyserver](#): if you run Tiddlywiki under this static server
- [TiddlyDesktop](#): If you run Tiddlywiki with TiddlyDesktop installed.
- Android: You have to install AndTidWiki.
- etc

Tiddlywiki has a tiddler type: saver to handle this saving methods:

Saver	Description
<code>\$/core/modules/savers/andtidwiki.js</code>	Handles saving changes via the AndTidWiki Android app
<code>\$/core/modules/savers/beaker.js</code>	Saves files using the Beaker browser'
<code>\$/core/modules/savers/download.js</code>	Handles saving changes via HTML5's download APIs
<code>\$/core/modules/savers/manualdownload.js</code>	Handles saving changes via HTML5's download APIs
<code>\$/core/modules/savers/fsosaver.js</code>	Handles saving changes via MS FileSystemObject ActiveXObject
<code>\$/core/modules/savers/github.js</code>	Saves wiki by pushing a commit to the GitHub v3 REST API
<code>\$/core/modules/savers/gitlab.js</code>	Saves wiki by pushing a commit to the GitLab REST API
<code>\$/core/modules/savers/put.js</code>	Saves wiki by performing a PUT request to the server
<code>\$/core/modules/savers/tiddlyfox.js</code>	Handles saving changes via the TiddlyFox file extension
<code>\$/core/modules/savers/tiddlyie.js</code>	Handles saving changes via Internet Explorer BHO extension (TiddlyIE)
<code>\$/core/modules/savers/twedit.js</code>	Handles saving changes via the TWEdit iOS app
<code>\$/core/modules/savers/upload.js</code>	Handles saving changes via upload to a server.
<code>\$/core/modules/savers/nodewebkit.js</code>	Handles saving changes in the NW.js environment.

How

Tiddlywiki does not save all tiddlers. It uses the filter defined in the tiddler `$/config/SaverFilter` who stores the tiddlers that are not going to be saved. It contains the filter:

```
[ all [] ] _ $:/HistoryList _ $:/StoryList _ $:/Import _ $:/isEncrypted _ $:/
UploadName _ [ prefix [ $:/state / ] ] _ [ prefix [ $:/temp / ] ]
```

The template used to save the wiki is `$/core/templates/tiddlywiki5.html` but you can send a parameter to the save button to specify which template you want to use.

4.10 State Mechanism

What

The state mechanism in TiddlyWiki is at the heart of how complex user interfaces can be built from WikiText. It is a way of storing the state of the user interface.

How

This state is stored in a state tiddler: a tiddler that is used to store the state of the user interface. We have two cases:

1. This state is shared by all tiddlers: we use a single tiddler.
2. The state is different for each tiddler:
 - a) We can store the state in a field of each tiddler
 - b) We can use a dynamically generated unique title for each tiddler we need. We can use the `qualify` macro for this option.

4.11 Translation Mechanism

What

It manages and switches between language plugins that provide translations of the TiddlyWiki user interface.

How

- The title of the current language plugin is read from the tiddler `$/language`.
- Translation plugins are bundles of tiddlers that each contain an independent translatable string. The strings are transcluded as needed.
- Translatable strings are generally in the namespace `$/language/`, for example:
 - `$/language/EditTemplate/Shadow/OverriddenWarning`
 - `$/language/Docs/ModuleTypes/isfilteroperator`
 - `$/language/EditTemplate/Fields/Add/Value/Placeholder`

4.12 Upgrade Mechanism

What

It is used to upgrade your tiddlywiki to the last version.

How

Open <https://tiddlywiki.com/upgrade.html>. Drag your tiddlywiki file to the browser screen that appears. You can download the `upgrade.html` file to use offline.

4.13 Boot Mechanism

What

At its heart, TiddlyWiki5 is a relatively small boot kernel that runs at first. The rest of the functionality is added via modules located in tiddlers. At the end of the boot process the `StartupMechanism` schedules the execution of startup modules to bring up the rest of TiddlyWiki.

How

The tiddler `$/boot/boot.js` is a barebones TiddlyWiki kernel that is just sufficient to load the core plugin modules and trigger a startup module to load up the rest of the application. This boot kernel includes:

- Several short shared utility functions
- A handful of methods implementing the module mechanism
- The `$tw.Tiddler` class (and field definition plugins)
- The `$tw.Wiki` class (and tiddler deserialization methods)
- Code for the browser to load tiddlers from the HTML DOM
- Code for the server to load tiddlers from the file system

4.14 Startup Mechanism

What

The startup mechanism runs the installed startup modules at the end of the boot process.

How

The boot process will run all tiddlers with their module-type set to startup. Startup modules are executed in sequence according to their declared dependencies: the javascript code includes the `exports.after` and `export.before` sentences. Look at Figure 3.2.

4.15 Encryption Mechanism

What

Tiddlywiki can be encrypted using the set password tool at the right sidebar. Opening an encrypted TiddlyWiki in the browser prompts for a password before decrypting and displaying the content.

How

The EncryptionMechanism is implemented with the following elements:

- A PasswordVault within the BootMechanism that holds the current encryption password
- The ability of the BootMechanism to read a block of encrypted tiddlers from the TiddlyWiki file, to prompt the user for a password, and to decrypt the tiddlers
- Handlers for the messages `WidgetMessage: tm-set-password` and `WidgetMessage: tm-clear-password` that handle the user interface for password changes
- The `EncryptWidget` within the main file template that encrypts a filtered list of tiddlers with the currently held password
- The `$/isEncrypted` tiddler that contains "yes" or "no" according to whether there is a password in the password vault

4.16 Event Mechanism

What

Most of the mechanisms need a way to get notified, when anything in the wiki store changes.

How

The core -plug-in adds an event system to the bare wiki store.

The functions providing the event system are added via the wikimethod module type.

4.17 Parsing Mechanism

What

The parsing mechanism analyses the text of a tiddler against a set of parsing rules, producing a tree representing the structure of the text. The RenderingMechanism is used to transform parse trees into render trees of widget nodes.

How

TiddlyWiki5 includes ParserModules for several types of tiddler:

- WikiText
- Raw HTML
- Plain text
- Images (bitmap, SVG and PDF)

You can search this parser tiddlers with the code: `{{{ [all[shadows]module-type[parser]] }}}}`. The Wiki-Text parser is the most complex. It is formed by a set of rules contained in separated tiddlers with the module-type: wikirule (you can search for these tiddlers with the code `{{{ [all[shadows]module-type[wikirule]] }}}}`).

The output of parsing a tiddler is an object containing a tree of parse nodes corresponding to the original text as a JSON string.

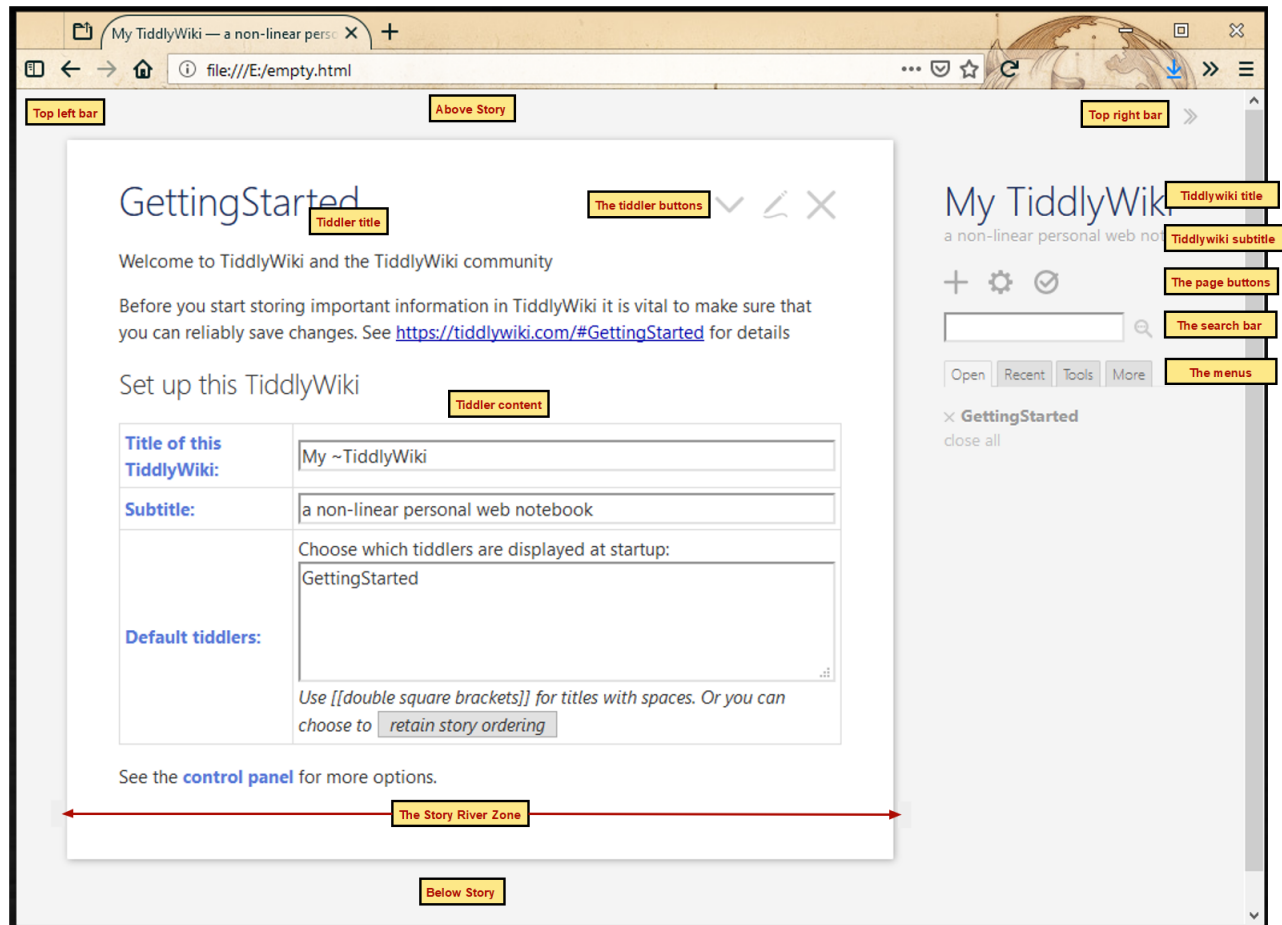
5 Components of the tiddlywiki screen

5.1 Key points

- Tiddlywiki page is divided in two main sides. The left is the Story River. The right, the sidebars
- In the left side appears the tiddlers you open.
- The right side shows the title, subtitle, buttons and the menus.
- All inside Tiddlywiki is made of tiddlers. A tiddler can have many roles: template, macro, widget, css formatting code, etc. depending of its tag or content-type field.
- Tiddlywiki uses special tiddlers and special tags to configure its appearance.

5.2 The TW whole screen

Figure 5.1: The Tiddlywiki page



In Figure 5.1 we can see the Tiddlywiki elements. The most important things are the Story River on the left and the right elements: title, menus etc. The tiddlers that we open will appear in the navigation story on the left. With the right menus we can find all the tiddlers of the file.

These are the elements that we can see in the Tiddlywiki page. But in order to personalize our wiki we must go deeper. You have to know that all these elements are tiddlers: the title and subtitle of your Tiddlywiki are tiddlers. The right menus too. And the search text box is a tiddler too. Even the buttons are tiddlers. The top right and left bars, the Above story, the Below story, all are tiddlers, **system** and **shadow** tiddlers.

Inside Tiddlywiki, all is a tiddler. And you can search and open them. For example if you look for the Subtitle in the search bar in advanced mode (click in the Advanced search icon on the right of the search bar) you will find a tiddler called `$/SiteSubtitle`. It's a shadow tiddler. If you change its content, the Tiddlywiki subtitle will change. And if you look for the save button you will find the tiddler called `$/core/ui/Buttons/save` with the code of the button.

5.2.1 The story river

This place of the Tiddlywiki page is where all tiddlers you will open appear. You can configure the way Tiddlywiki will open them in the configuration tiddler (click the gearwheel icon on the right). At first all tiddlers opens each under the other but you can configure it so that only the active tiddler appears.

5.2.2 The right sidebars

The right place is where you can see the wiki title, subtitle, the page buttons, the search bar.... and under them the sidebars (the menus).

You can see 4 tabs: Open for the open tiddlers, Recent for the tiddlers you have opened, Tools to customize and configure your wiki and More where you can find some utilities. Of course they are four tiddlers: `$/core/ui/SideBar/Open`, `$/core/ui/SideBar/Recent`, `$/core/ui/SideBar/Tools` and `$/core/ui/SideBar/More`. All of them has the tag: `$/tags/SideBar`. So if you want to include your own tab create a tiddler and add to them this tag.

This is an important question about Tiddlywiki: The tiddlers that make up the screen are chosen with its tags. For example, above we have seen that if you put the tag `$/tags/SideBar` to a tiddler it appears on the left side. There are many tags for all places of the screen. You can see then in [Figure 5.2](#)

5.3 Drawing its interface

We know all are tiddlers but how Tiddlywiki draws its interface? This is where templates come in action.

To tell tiddlywiki where to place all elements we have a template: `$/core/ui/PageTemplate`. Try this: Edit this tiddler and add at the bottom of all code a new line with the code: `<hr><hr><hr>`. Then click the save tiddler button and close it. You can see tree lines below the story river.

Looking this template we can see that it draws this elements:

1. `$/core/ui/PageTemplate/topleftbar`.- The top left bar
2. `$/core/ui/PageTemplate/toprightbar`.- The top right bar
3. `$/core/ui/PageTemplate/sidebar`.- The right side: title, subtitle, menus...
4. `$/core/ui/PageTemplate/story`.- All open tiddlers: the Story River
5. `$/core/ui/PageTemplate/alerts`.- Special tiddlers called alerts (tiddlers with the `$/tags/Alert` tag)

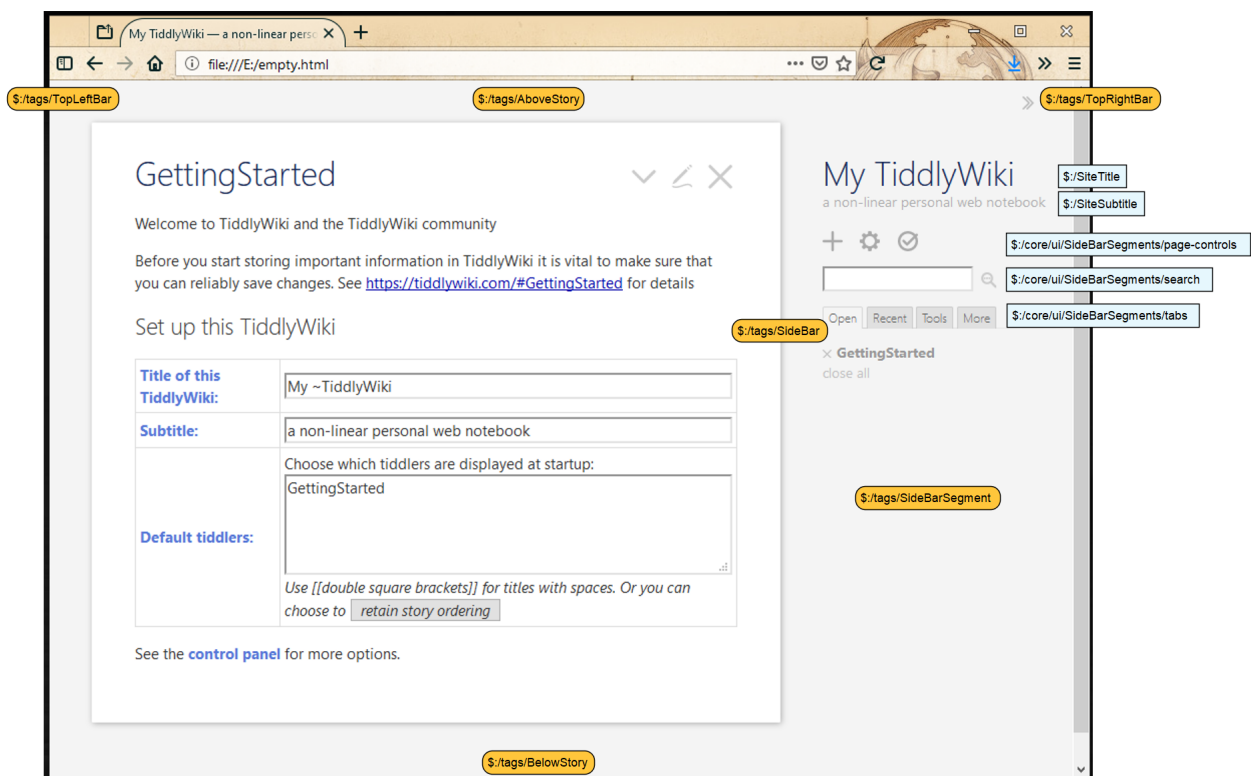
All elements are tagged with `$/tags/PageTemplate`.

The code of all this elements allows us to add additional elements very easily. We only have to create a new tiddler and tagged it with the correct tag. For example, if we want to show some information above the story river we create a tiddler with this information and add the tag `$/tags/AboveStory`. The most important tags are in [Figure 5.2](#).

What's this? The light blue little squares shows the tiddlers containing the title, subtitle, buttons, search bars and tabs. And the orange bubbles show the tags. So, if you want to include:

- A top left bar: create a tiddler with the buttons and tagged it with `$/tags/TopLeftBar`.
- A top right bar: There is a tiddler with this tag: `$/tags/TopRightBar`: the tiddler `$/core/ui/TopBar/menu`. It is used to hide the right zone.
- Content above the story river: Add this content in a tiddler tagged `$/tags/AboveStory`.
- Content below the story river Add this content in a tiddler tagged `$/tags/BelowStory`.
- A new tab in the right menus: Add this content in a tiddler tagged `$/tags/SideBar`.
- New content below the right menu: Add this content in a tiddler tagged `$/tags/SideBarSegment`.

Figure 5.2: The tags of Tiddlywiki places



6 Page and tiddler layout customization

Written by Dr. Rizwan Ishak

6.1 Modifying Page Layout with System Tags

TiddlyWiki's layout is determined by tiddlers carrying special SystemTags while template of TiddlyWiki is written in HTML. Together these provides two major methods to add, remove and customize the user interface of TiddlyWiki.

The page layout of TiddlyWiki is determined by tiddlers carrying the tag "\$:/tags/PageTemplate". The tiddlers in Tiddlywiki carrying the tag \$:/tags/PageTemplate are:

- \$:/core/ui/PageTemplate/topleftbar
- \$:/core/ui/PageTemplate/toprightbar
- \$:/core/ui/PageTemplate/sidebar
- \$:/core/ui/PageTemplate/story
- \$:/core/ui/PageTemplate/alerts
- \$:/core/ui/PageTemplate/pluginreloadwarning

Each of these tiddlers correspond to an element displayed in the page. If the user removes the tag "\$:/tags/PageTemplate" from any of these tiddlers, the corresponding element will stop being displayed in the layout.

Adding the tag \$:/tags/PageTemplate to a new tiddler will make it a part of the layout of TiddlyWiki. Unless otherwise specified, the newly added elements will be displayed at bottom of the page. This behaviour can be modified by using CSS or by changing the order of tiddlers in the list field of tiddler \$:/tags/PageTemplate.

There are several other system tags that will let the user to add or remove elements from TiddlyWiki interface. For a complete list refer [SystemTags](#). For Example: Adding tag "\$:/tags/SideBar" will add the tiddler as a sidebar tab. Adding the tag "\$:/tags/AboveStory" will display the tiddler permanently above the story river.

6.2 Modifying Page Layout with CSS

Each element in the page has one or more CSS classes styling it. The CSS classes associated with the major page template elements are:

- Top left bar: .tc-topbar .tc-topbar-left
- Top right bar: .tc-topbar .tc-topbar-right
- Sidebar: .tc-sidebar-scrollable
- Story river: .tc-story-river Alerts: .tc-alerts

If user wants to hide a particular element from being displayed, they can create a stylesheet tiddler and add the style display:none; to the corresponding class.

CSS offers far more styling options than changing the visibility of an element. A complete description of CSS and its application to the each element is out of scope of this reference material. User is directed to familiarize themselves with various CSS properties from third party sources.

All major browsers provide the options to inspect a displayed element on html page and view the CSS properties attached to that element. You may find the way to do it on your browser's website or forums.

`display:none` property merely hides the display of the html element. It will not stop the element from loading to the DOM structure. Widespread use of the same will be a inefficient usage of resources.

6.3 Modifying Tiddler Layout with System Tags

The layout of each tiddler is determined by tiddlers carrying the tag “`$/tags/ViewTemplate`”. The tiddlers Tiddlywiki carrying the tag “`$/tags/ViewTemplate`” are:

- `$/core/ui/ViewTemplate/title`
- `$/core/ui/ViewTemplate/unfold`
- `$/core/ui/ViewTemplate/subtitle`
- `$/core/ui/ViewTemplate/tags`
- `$/core/ui/ViewTemplate/classic`
- `$/editions/tw5.com/operator-template`
- `$/core/ui/ViewTemplate/body`
- `$/core/ui/ViewTemplate/import`
- `$/core/ui/ViewTemplate/plugin`

Each of these tiddlers correspond to an element displayed in all the tiddlers or a particular subset of tiddlers. If the user removes the tag “`$/tags/ViewTemplate`” from any of these tiddlers, the corresponding element will stop being displayed in the layout.

Conversely, adding the tag “`$/tags/ViewTemplate`” to a new tiddler will make it a part of the layout of tiddlers. Unless otherwise specified, the newly added elements will be displayed at bottom of the tiddler. This behavior can be modified by using CSS or by changing the order of tiddlers in the list field of tiddler “`$/tags/ViewTemplate`”

6.4 Modifying Tiddler Layout with CSS

Each element in the tiddler has one or more CSS classes styling it. The CSS classes associated with the major tiddler elements are:

- Tiddler frame: `.tc-tiddler-frame.tc-tiddler-view-frame`
- Title: `.tc-title`
- View Toolbar: `.tc-tiddler-controls`
- Subtitle: `.tc-subtitle`
- Tag-pill: `.tc-tag-label`
- Body: `.tc-tiddler-body`

If user wants to hide a particular element from being displayed, they can create a stylesheet tiddler and add the style `display:none`; to the corresponding class.

CSS offers far more styling options than changing the visibility of an element.

You can apply CSS styles to a subset of tiddlers having a common tag. Refer: How to apply custom styles by tag

7 Customize the TW screen

7.1 Key points

- The PageTemplate is used to display the Tiddlywiki screen.
- It shows all tiddlers tagged with `$/tags/PageTemplate`. Its order is in the list field of the tiddler with the same title, `$/tags/PageTemplate`.
- The ViewTemplate is used to display a tiddler in view mode.
- It shows all tiddlers tagged with `$/tags/ViewTemplate`. Its order is in the list field of the tiddler with the same title, `$/tags/ViewTemplate`.
- The EditTemplate is used to display a tiddler when you are editing or creating.
- It shows all tiddlers tagged with `$/tags/EditTemplate`. Its order is in the list field of the tiddler with the same title, `$/tags/EditTemplate`.
- The text/css tiddlers tagged with `$/tags/Stylesheet` are used to format the text of all Tiddlywiki elements.
- You can add more buttons adding the tiddler button with its code.
- You can create new keyboard shortcuts
- Creating a left menu is not complicated

7.2 Introduction

One major feature of TiddlyWiki that many new users are unaware of is the degree to which TiddlyWiki can be customized, just by adding or removing SystemTags in key shadow tiddlers or in your own custom tiddlers.

- You can add and remove default features in tiddlers in either viewing or editing mode (let's say you find the tiddler subtitle distracting, or you want to add yourself a reminder that you will see when you edit tiddlers)
- You can also add and remove default features from the general page layout (maybe you want to add a clock to the sidebar, or replace one of the page control buttons with your own)
- You can also rearrange the order in which these features are displayed (perhaps you would like tags above tiddler titles, or the subtitle of your TiddlyWiki below the page control buttons)

Once you know what you are doing, all of these things are actually pretty easy to do.

There are three main templates in tiddlywiki:

- PageTemplate
- ViewTemplate
- EditTemplate

7.3 PageTemplate

The `$/core/ui/PageTemplate` tiddler is the template that draws the whole screen of tiddlywiki. It has the tag. All tiddlers **inside** this tiddler one are tagged with the `$/tags/PageTemplate` tag.

What means“inside” this tiddler? If you look at this tiddler you can see this code:

```
<$list filter="[all [shadows+tiddlers]tag[$:/tags/PageTemplate]!has[draft.of]]" variable="listItem"><$transclude tiddler=<<listItem>>/></$list>
```

This is a list who transcludes (shows) all tiddlers tagged whit the `$/tags/PageTemplate` tag. If you tag a new tiddler with that tag it will be included in the main wiki page. You can find the sort of all the elements tagged in the list field of the tiddler `$/tags/PageTemplate` (the same title as the tag). You can show your tiddler at the top of the screen if you include it as the first element of the list field in that tiddler.

What about the appearance of all this elements? Each element in the page has one or more CSS classes styling it. The CSS classes associated with the major page template elements are:

- Top left bar: `.tc-topbar .tc-topbar-left`
- Top right bar: `.tc-topbar .tc-topbar-right`
- Sidebar: `.tc-sidebar-scrollable`
- Story river: `.tc-story-river` Alerts: `.tc-alerts`

If user wants to hide a particular element from being displayed, they can create a stylesheet tiddler and add the style `display:none;` to the corresponding class.

CSS offers far more styling options than changing the visibility of an element. A complete description of CSS and its application to the each element is out of scope of this reference material. User is directed to familiarize themselves with various CSS properties from third party sources. All major browsers provide the options to inspect a displayed element on html page and view the CSS properties attached to that element. You may find the way to do it on your browser's website or forums. `display:none` property merely hides the display of the html element. It will not stop the element from loading to the DOM structure. Widespread use of the same will be a inefficient usage of resources.

7.4 ViewTemplate

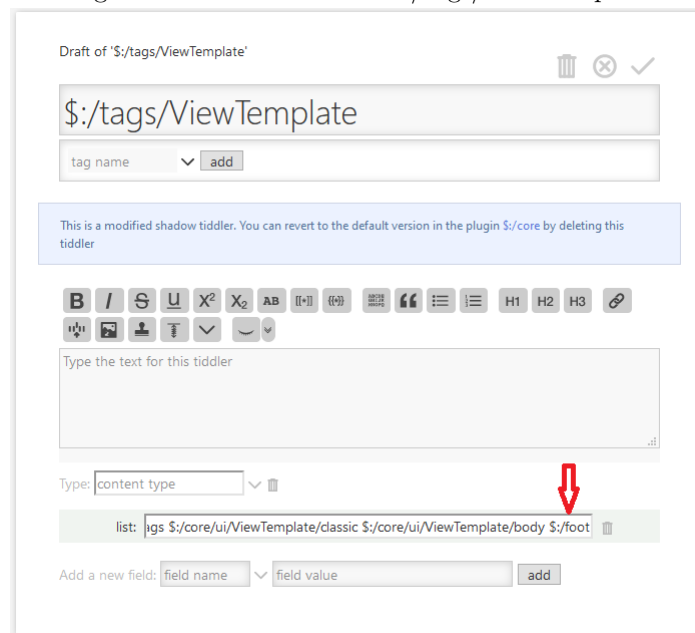
This is the template Tiddlywiki uses to display a single tiddler when you are reading it. Its title is `$/core/ui/ViewTemplate`. As in the PageTemplate, you can edit it and look at its code:

```
<$list filter="[all [shadows+tiddlers] tag[$:/tags/ViewTemplate]!has[draft.of  
  ]]" variable="listItem">  
  
<$transclude tiddler=<<listItem>>/>  
  
</$list>
```

It shows a list with all tiddlers tagged with the `$/tags/ViewTemplate` tag sort with the list field of the tiddler `$/tags/ViewTemplate`.

7.4.1 Example

Figure 7.1: The list field of `$/tags/ViewTemplate`



We want to illustrate the power of this templates. In this section we will add a foot at all tiddlers with the date in which the tiddler was added to the wiki.

The steps are:

- Add a new tiddler with the title `$/foot`
- Add this code to the tiddler:

```
<small>  
  //(Added to the wiki: <$view field="created" format="date" template="  
    DDth_MMM_YYYY"/>)//  
</small>
```

- Add the tag `$/tags/ViewTemplate` to the template
- Open the tiddler `$/tags/ViewTemplate`

- Add our tiddler, `$/foot` at the end of the list field of that tiddler, `$/tags/ViewTemplate` (look at Figure 7.1).
- Save the tiddler.

You will see a foot in all tiddlers with the created date. In Figure 7.2 you will see the new tiddler appearance.

7.5 EditTemplate

The `$/core/ui/EditTemplate` tiddler is the tiddler Tiddlywiki uses to display a tiddler when you are editing or creating it. And inside this tiddler you find:

```
<$list filter="[ all [ shadows+tiddlers ] tag [ $:/tags/EditTemplate ] !has [ draft . of ] ]" variable="listItem">
<$set name="tv-config-toolbar-class" filter="[<tv-config-toolbar-class>] [<listItem>encodeuricomponent [] addprefix [ tc-btn -]]">

<$transclude tiddler=<<listItem>>/>

</$set> </$list>
```

So it shows all tiddlers tagged with `$/tags/EditTemplate` sorting with the list field of the tiddler with the same name as this tag.

7.6 Formating with CSS

You know HTML uses CSS to format the text. With CSS you can add colors, change font size, add borders and many other things.

If you open the control panel and show the Appearance tab you will see two themes: Snow White and Vanilla. And in Pallete tab you can choose many color combinations. Tiddlywiki uses CSS to change this settings. For example, the Vanilla base configuration is in the tiddler `$/themes/tiddlywiki/vanilla/base`. You can see that it is tagged with `$/tags/Stylesheet`.

Inside we find the css configuration for many html tags:

p: Paragraphs

h1: Title 1

h2: Title 2

hr: Lines

table, td, tr: Tables

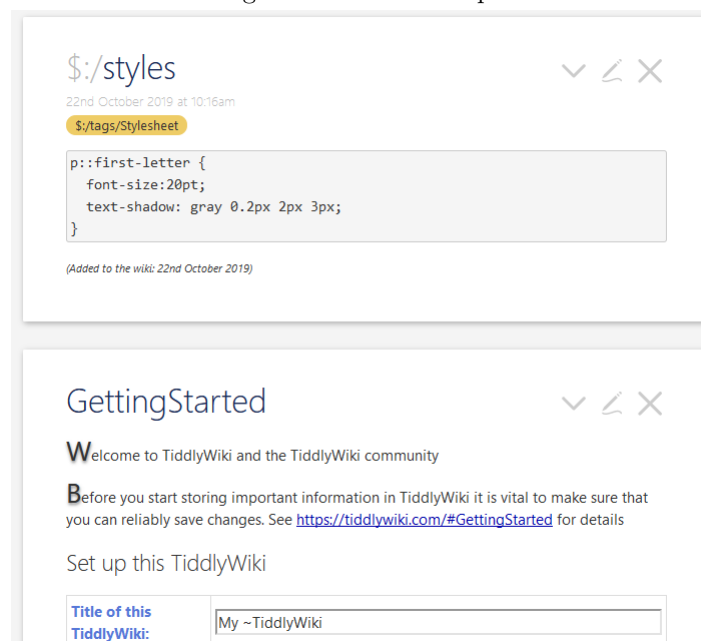
If you read this tiddler you will find all css clases for this theme. At the end of this chapter, on page 54 we will see a map with many css standard classes. The best way of using this map is search the classes in it in the main css Tiddlywiki style sheet, `$/themes/tiddlywiki/vanilla/base`. In this way you can see the values of the classe you are looking.

Other way of play with this classes is with the firefox inspector. With it you can dinamicaly change its values. In the next table we can see some properties of css classes.

Property	Example	Description
display	display: none	Hide the element
border	border: 2px dotted red	Draw a border
border-radius	border-radius: 20px	Rounded corners
color	color: red	Text color
background-color	background-color: lightgray	Background color
font-style	font-style: italic	italic, normal
font-weight	font-weight: bold	
font-size	font-size: 15pt	Font size
font-family	font-family: "Times New Roman", Times, serif;	
column-count	column-count: 3;	Number of columns
text-align	text-align: right	left, right, center, justify
transform	transform: rotate(20deg);	rotate, skewY, scaleY...

7.6.1 Example

Figure 7.2: CSS Example



We can add a custom stylesheet tiddler for our own wikis. Imagine you want the first word of all paragraphs bigger. These are the steps:

- Create a tiddler, `$:/styles`
- Add this content:

```
p::first-letter {
  font-size:20pt;
  text-shadow: gray 0.2px 2px 3px;
}
```

- Add the tag `$/tags/Stylesheet` to our tiddler.
- If you want, you can add the Type, under the content: `text/css` (only for readability)

In Figure 7.2 you see the style tiddler and the new appearance of the paragraphs.

7.7 Create a loading message

If the Tiddlywiki is big the loading in the browser may be delayed. This is a way to show a little message while it is loading:

- Create a new tiddler. Its title is not important.
- Add the loading message in its content. For example, “The wiki is loading. Please, wait...”
- Add the tag `$/tags/RawMarkupWikified/TopBody`
- Add the type `text/vnd.tiddlywiki`
- Save.

7.8 Create a new button at the toolbars

You know there are three main toolbars:

- Page toolbar: buttons on the right side of TW.
- View toolbar: buttons at the top of all tiddler in view mode
- Edit toolbar: buttons at the top of the tiddler you are editing

You can see all buttons in this toolbars in the control panel, appearance, toolbars. In this section we will add a new button. The new button will appears in the toolbars and you can show or hide it.

7.8.1 How to do it

- First of all you have to search for an icon for the new button. You can search at [Flaticon](#) or [Feather icons](#) and import the file to our wiki. Change its size to 22px x 22px
- Create the tiddler button:

– Content: the code for the button.

- * For example, to create a new kind of tiddler. The tooltip appears if you leave the cursor over the button:

```
<$button tooltip="Create a new kind of tiddler" aria-label="
  Create new kind of tiddler" class=<<tv-config-toolbar-class>>>
  {{icon tiddler}}
<$action-sendmessage $message="tm-new-tiddler"
  title="New kind of tiddler"
  text=<<content>>
  tags="kind-1"
  color=#ffff80 />
</$button>
```

– Caption: `{{icon tiddler}}` Text-button. To show the icon and the button text.

- * Description: short description for the button.

– Tags:

- * `$/tags/PageControls` for the Page toolbar
- * `$/tags/ViewToolbar` for the View toolbar

- * `$/tags/EditToolbar` for the Edit toolbar
- If you want to change the position of the button in the control panel add the tiddler to the list fields of the three tag tiddlers in the position you want:
 - `$/tags/PageControls`
 - `$/tags/ViewToolbar`
 - `$/tags/EditToolbar`
- Save and reload the wiki
- Open the control panel, appearance, toolbars. The new button will appear in the three toolbars. You can show or hide it.

7.9 Add a new global keyboard shortcut

In the control panel, keyboard shortcuts appears the shortcuts of your tiddler. In this section we learn how to create a new shortcut. Imagine you want to create a new shortcut, CTRL+ALT+P to open the tiddler Control panel

7.9.1 How to do it

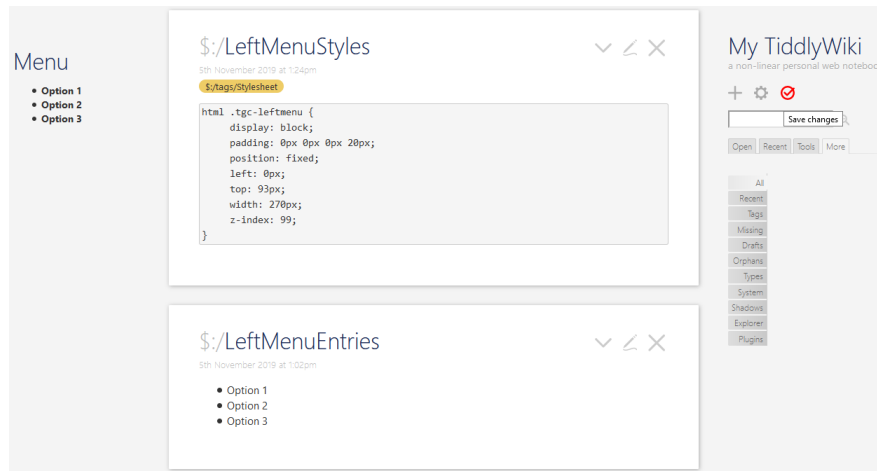
- Create a new tiddler. This new tiddler is only for information:
 - Title: `$/config/ShortcutInfo/control-panel`
 - Text: `ctrl-alt-P`.
- Go to the control panel.
 - Look for the new entry, control-panel.
 - Expand and click in the pen icon.
 - In the text area type the keys CTRL+ALT+P
- Create other tiddler:
 - Title: `$/control-panel` (for example. It can be other title)
 - Text:


```
<$navigator story="$:/StoryList" history="$:/HistoryList"> <$action-navigate $to="$:/ControlPanel"/> </$navigator>
```
 - Tag: `$/tags/KeyboardShortcut`
 - Add field key: `((control-panel))`. This is very important. It associate this code with the first tiddler, the shortcut.
- Save and reload the wiki.

Notes:

- We have chosen the control-panel suffix.
- The tiddler title `$/config/ShortcutInfo/control-panel` is formed by "`$/config/ShortcutInfo/`" + suffix
- We have chosen the tiddler title `$/control-panel`.
- The link between the code and the keys is made with the key field. It is formed by `((+ suffix +))`
- The action-navigate has to be enclosed in the \$navigator widget.

Figure 7.3: A left menu



7.10 Creating a left menu

In this section we will create a simple left menu. The steps are:

- Configure the story river position.
- Create the tiddler menu.
- Create the styles for the left menu.
- Create the entries for the left menu

7.10.1 Configure the story river position

The manual way

Go to the control panel, appearance, Theme Tweaks. Look for the Story left position and put a value of 230.

The styles

Tiddlywiki have some css classes to automatize this. For example you can add a stylesheet tiddler tagged with `$/tags/Stylesheet` and add the code:

```
html .tc-story-river {  
  left: 230px;  
  width: 770px;  
}
```

But in this way the right menu hides under the story river. You can add :

```
.tc-sidebar-scrollable {  
  left: 1000px;  
}
```

to improve the appearance.

7.10.2 Create the tiddler menu

Create a new tiddler and tag it with `$/tags/PageTemplate`

Add this code (insert an empty line before `<$transclude>`):

```
<div class="tgc-leftmenu tc-table-of-contents">

<$list filter='[tag[$:/tags/LeftMenu]]'>

  <$transclude/>
</$list>

</div>
```

7.10.3 Create the styles for the left menu

Tiddlywiki has not styles for showing this menu. You have to add them. Add a stylesheet tiddler tagged with `$/tags/Stylesheet` and add the code:

```
html .tgc-leftmenu {
  display: block;
  padding: 0px 0px 0px 0px;
  position: fixed; left: 5px;
  top: 93px;
  width: 270px;
  z-index: 99;
}
```

7.10.4 Create the entries for the left menu

If you read the code of the tiddler left menu you can see the line: `{{$/tags/LeftMenu}}`. This mean that all things you put in this tiddler appears on your left menu.

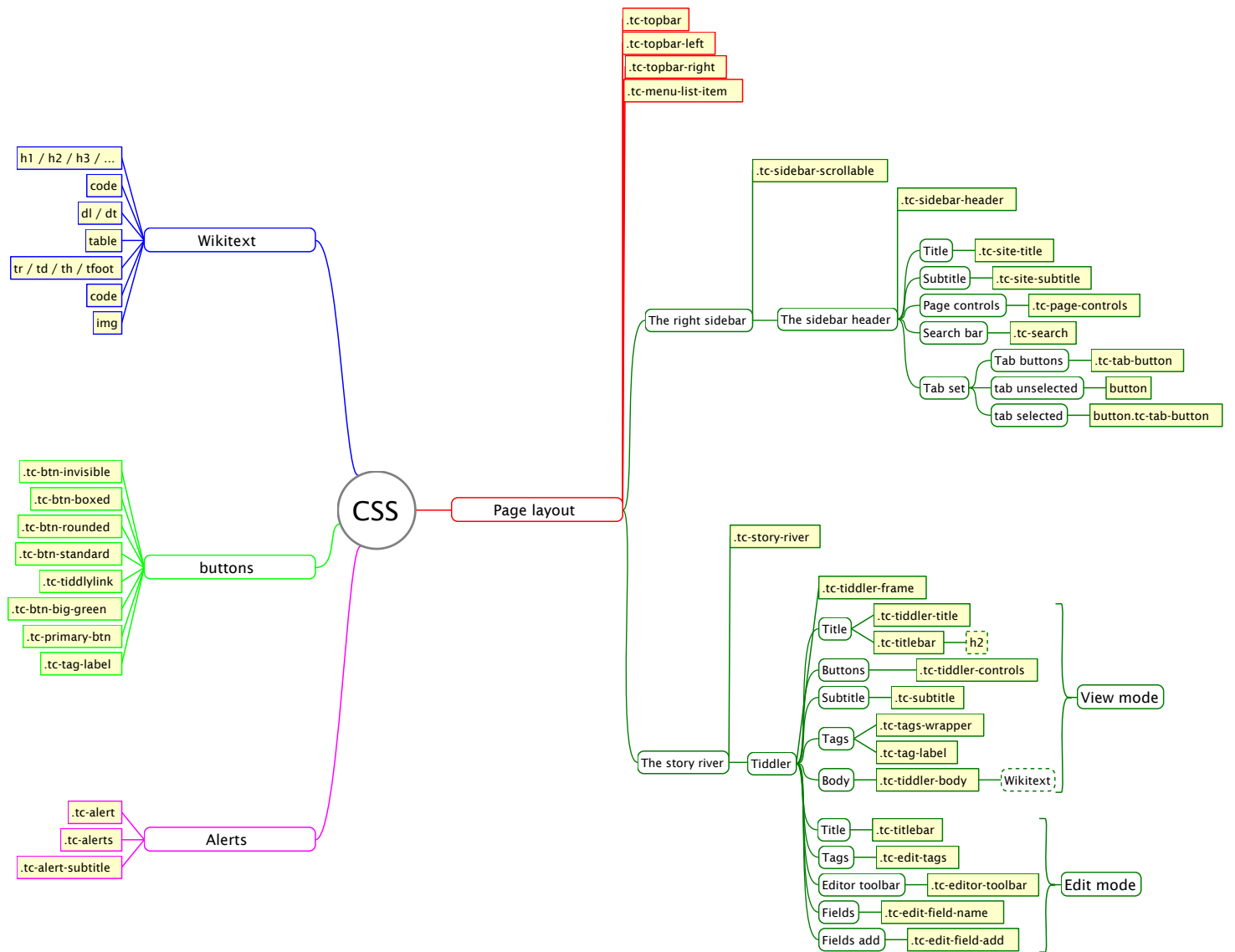
To simplify things, lets make a tiddler to put this tag automatically. Create a new tiddler, Buttons and add this code:

```
<$list filter="[all[shadows] prefix[$:/core/ui/Buttons]]">
  <$checkbox tag="$:/tags/LeftMenu" > <<currentTiddler>>
  </$checkbox><br/>
</$list>
```

You will see all Tiddlywiki core buttons. If you mark the checkbox of a button it will add to your left bar. Remember to save your tiddlywiki.

7.10.5 The next step

This is the easy way. You can add a lot of trimmings. For example, you can add a button to show or hide this menu with the `$reveal` widget like the right menu.



8 Inside a tiddler

8.1 Key points

- A tiddler can perform many roles in Tiddlywiki
- There are many kinds of tiddlers depending of the Content Type field an its tags.
- For the external “image” tiddler types, you have to add the “__canonical_uri” field with the link of the external file.
- Normal tiddler: tiddler with your notes.
- Tag tiddler: a tiddler with the same title as a tag.
- Alert tiddler: a tiddler that appears on top of all other tiddlers.
- Dictionary and JSON tiddlers: tiddlers for storing short information.
- CSS tiddlers: tiddlers with css classes for formatting your tiddlers.
- Template tiddlers: tiddlers used as a template for other tiddlers.
- Macro tiddlers: tiddlers with your code
- Javascript tiddlers: Macros and Widgets.

8.2 Types of tiddlers

Tiddlers are the heart of Tiddlywiki. Once the Microkernel loads the tiddler functions (addTiddler(tdlr), deleteTiddler(tdlr), etc) the rest of the code is stored in tiddlers. This means that a tiddler can hold several roles, not only for our notes.

All tiddlers have a field. It is below the text in edit mode: Type. It is used to tell Tiddlywiki the type of the tiddler. But Tiddlywiki not only look this field to determine the role of the wiki. TW also look the tags assigned to the tiddler

How many of roles does a tiddler have?

8.2.1 User tiddlers

Normal tiddler: It stores your notes and thinkings.

Tag tiddler: Is a tiddler with the same title as a tag.

Alert tiddler: It shows an alert that remains open on the screen.

Image tiddler: If you drag a picture into a Tiddlywiki it stores the image in a tiddler.

Dictionary tiddler: Is a tiddler for storing data.

JSON tiddler: Another way of storing data.

Macro tiddler: a tiddler that stores macros.

8.2.2 Internal tiddlers

CSS tiddler: It stores the css rules for text formatting.

Template tiddler: It stores the template for other tiddler or even the whole Tiddlywiki screen.

Javascript tiddler: A tiddler with javascript code. It stores the tiddlywiki code and your widgets.

Shadow tiddler: Tiddlers that are loaded from within Plugins.

System tiddler: Tiddlers that starts with the special string \$:/

Plugin tiddler: It is a JSON tiddler whose data are tiddlers. The core of Tiddlywiki is stored in one of this tiddlers.

Parser tiddler: Is responsible to transform block of text to a parse-tree. It defines the rules to transform wikitext to html.

Wiki rule: It defines all elements used by the parser tiddlers. Tiddlywiki has rules who transform wikitext to html. For example, the dash rule transform the “- - -” wikitext to <hr>.

Saver tiddler: It has the code for saving the tiddlywiki file in all possible environments.

8.2.3 The tiddler type field

If you add a tiddler you can see under the content a field: Type. In this field Tiddlywiki stores the content type of the tiddler if it is not text. This table shows the kind of content that supports TW. The additional content rows contains content not documented in the TW site. The next table show all roles a tiddler can have:

Role	Type field	Module type	Tag
Normal tiddler			
Tag tiddler			
Alert tiddler			\$:/tags/Alert

Role	Type field	Module type	Tag
JSON tiddler	application/json		
Dictionary tiddler	application/x-tiddler-dictionary		
<i>External content</i>			
GIF image tiddler	image/gif		
JPG image tiddler	image/jpeg		
PNG image tiddler	image/png		
ICO image tiddler	image/x-icon		
SVG image tiddler	image/svg+xml		
PDF tiddler	application/pdf		
CSS tiddler			\$/tags/Stylesheet
<i>Code & shortcuts</i>			
Keyboard shortcut			\$/tags/KeyboardShortcut
Macro tiddler			\$/tags/Macro
Widget	application/javascript		\$/tags/Macro
javascript code tiddler	application/javascript		
<i>Template tiddlers</i>			
Page template			\$/tags/PageTemplate
View template			\$/tags/ViewTemplate
Edit template			\$/tags/EditTemplate
<i>Buttons</i>			
Page toolbar			\$/tags/PageControls
View toolbar			\$/tags/ViewToolbar
Edit toolbar			\$/tags/EditToolbar
<i>Additional content</i>			
Audio mp3	audio/mp3		
Video mp4	video/mp4		
<i>Parsers & Savers</i>			
Parser	application/javascript	parser	
Rule	application/javascript	wikirule	
Saver	application/javascript	saver	
<i>Other</i>			
Splash string			\$/tags/RawMarkupWikified/TopBody

Example. Adding a jpg external image: Imagine you have a directory called files in the same location as your wiki and inside this directory a png image, beach.png. If you want to add a tiddler with this image these are the steps:

Figure 8.1: Adding external files

Draft of 'Beach'

Beach

tag name add

This tiddler shows content stored outside of the main *TiddlyWiki* file. You can edit the tags and fields but cannot directly edit the content itself

[files/Beach.png](#)

Type:

Add a new field: add

- Add a tiddler
- In the Type field write: "image/png"
- Add other field called "_canonical_uri" with this content: "files/beach.png" (Figure 8.1)
- Save the tiddler

8.3 Normal tiddler

You add a normal tiddler with the plus button on the left side of Tiddlylwiki. Add the title and the content.

This tiddler contains wikitext. Wikitext is normal text with formatting characters. At the end of the chapter you can see a mindmap with all options.

8.4 Plugin tiddler

It is a JSON tiddler whose content is the tiddlers that make up the plugin. It has some fields to identify it:

author: The author of the plugin

core-version: Minimum version of TW for the plugin to work.

dependents: Other plugin dependencies.

description: A little description of the plugin

list: If the plugin includes a readme or license information we have to add them here. For example, list:
readme license

plugin-priority: If the plugin needs to load another one first.

plugin-type: It contains usually, "plugin".

version: The plugin version.

A plugin has two information tiddlers:

- readme: stored in \$:/<plugin-name>/readme. It's basic information about the plugin.
- license: stored in \$:/<plugin-name>/license. The license under which the plugin is published.

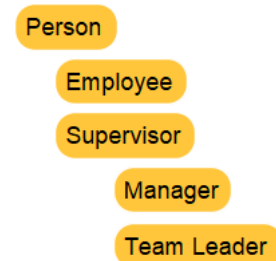
8.5 Tag tiddler

A tag tiddler is a normal tiddler titled with the name of a tag. The main advantage to do this is to construct a hierarchy of tags. For example, imagine you want to create this hierarchy:

- Create the “Person” tiddler without tags.
- Create the “Employee” tiddler and tag it with the “Person” tag.
- Create the “Supervisor” tiddler and tag it with the “Person” tag.
- Create the “Manager” tiddler with the “Supervisor” tag.
- Create the “Team Leader” tiddler with the “Supervisor” tag.

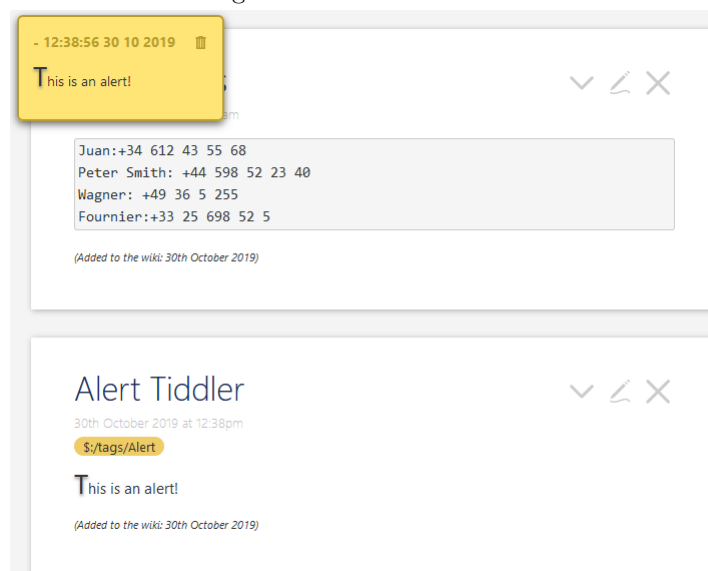
This is useful, for example to add an index to the sidebar.

Figure 8.2: Tag hierarchy



8.6 Alert tiddler

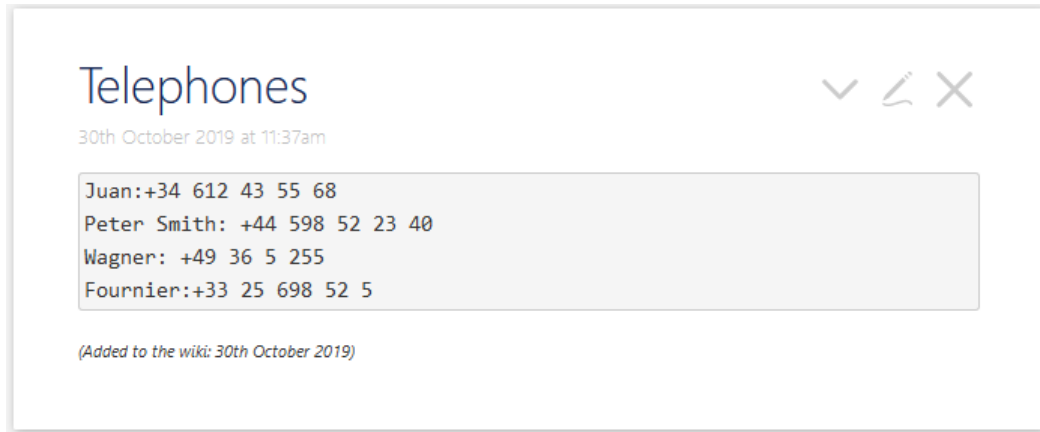
Figure 8.3: An alert tiddler



Tiddlywiki can show alerts. An alert is a tiddler with the `$:/tags/Alert` tag.

8.7 Dictionary tiddler

Figure 8.4: A dictionary tiddler



A dictionary tiddler is a way of storing data. Create a tiddler in the usual way and tag it with the application/x-tiddler-dictionary tag. Now you can enter your data using this notation: data:content. In Figure 8.4 you can see a tiddler to store telephone numbers. Each term is called “index”.

If you can use this data you can add a tiddler and transclude the index you want. For example, if you add the text:

```
Peter Smith: {{Telephones##Peter Smith}}
```

the output will be:

```
Peter Smith: +44 598 52 23 40
```

8.8 JSON tiddlers

They are data tiddlers too but in JSON format. If you want to look at a JSON datatiddler open the \$:/HistoryList tiddler.

8.9 CSS tiddlers

You can change the way you display tiddlywiki elements by adding css code in CSS tiddlers. They are tiddlers with the \$:/tags/Stylesheet tag.

8.10 Template tiddlers

They are tiddlers to customize the way Tiddlywiki shows some tiddlers. They are text and “instructions” that tell Tiddlywiki how to shows all tiddler elements.

8.11 Other system tiddlers

8.11.1 \$:/config/EmptyStoryMessage

Here you can add a little message that appears if you close all tiddlers. When you close all tiddlers this message appears as plain text without Wikitext or any format options.

If you want a more complicated message you can transclude a custom tiddler with the `$:/core/ui/ViewTemplate` template. These are the steps:

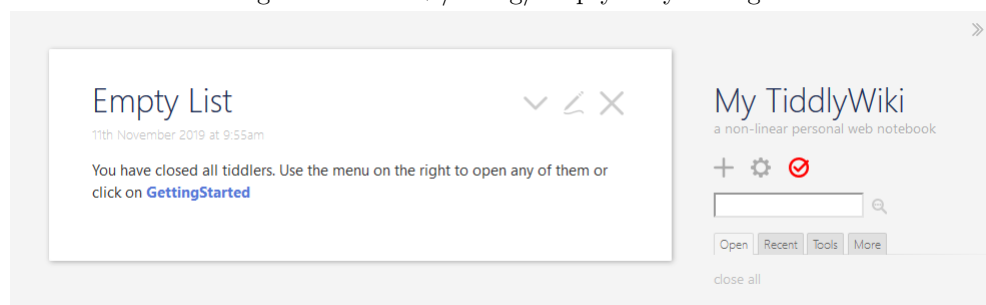
1. Create the tiddler you want to show when all tiddlers are closed. For example, “Empty List”. Add your message in the content, for example:

You have closed all tiddlers. Use the menu on the right to open any of them or click on `[[GettingStarted]]`

2. Create the tiddler `$:/config/EmptyStoryMessage`
3. Add the code `{{Empty List|$:/core/ui/ViewTemplate}}` to its content.
4. Save all.

Now, if you close all tiddler this is the screen:

Figure 8.5: The `$:/config/EmptyStoryMessage`



You can see in this figure there are no open tiddlers (look at the right open tab menu) but it shows the Empty List tiddler.

8.12 Adding macros and widgets

Here you can find basic information. Advanced information in later chapters

Wikitext

Headings

!Level 1 heading
!!Level 2 heading
!!!Level 3 heading

List

* Level 1 item
** Level 2 item
** Level 2 item
* Level 1 item
△ *unnumbered list*

Level 1 item
Level 2 item
Level 2 item
Level 1 item
△ *numbered list*

Basic formatting

'**bold**'
//*italic*//
~ ~~~strikethrough~~~ ~
_ underscore _
^^^{superscript}^^
,subscript, ,
<<< block quote >>>
" " " force line breaks " " "
`code`
△ *two consecutive backticks*
` ` ` code block ` ` `

Links

CamelCaseWords
△ *Are links by default*
~CamelCaseNotALink
△ *Avoiding a camel case link*
[[Tiddler Title]]
△ *linking by tiddler title*
[[Text to display|Tiddler Title]]
△ *Show other text*
[img[directory/file.ext]]
△ *External image*
[ext[The beach|files/beach.png]]
△ *Linking a external file*

Lines

- - Text
△ *short dash*
- - - Text
△ *long dash*

△ *Alone in a line: a horizontal rule*

Definitions

; Term to define
: Definition
; Another term
: The definition

Tables

|head 1|head 2|head 3|h
align left	center	align right
^top	middle	,bottom
>	merged with this	<
this is a long cell	top merged	this is a long cell
cell	~	cell
footer	footer	footer

head 1	head 2	head 3
align left	center	align right
top		bottom
merged with this		
this is a long cell	top merged	this is a long cell
cell		cell
footer	footer	footer

Transclusion

{{Tiddler Title}}
△ *A single tiddler*
{{Tiddler Title|Template}}
△ *A tiddler using the template*
{{Tiddler Title!!field}}
△ *A field of a tiddler*
{{{field}}}
△ *A field of the current tiddler*
{{DataTiddler##index}}
△ *The data of a data tiddler*
{{{ Filter expression }}}
△ *List of tiddlers*

Code

<<macro_call>>
<<variable_name>>
<widget call>

9 Filters

9.1 Key points

- Filters are a way to find information in your wiki. They produce a list of tiddlers.
- Each filter expression is divided in many simple filters.
- Each simple filter is called a filter step.
- Each simple step is enclosed in [].
- Each simple filter represents a simple search criteria.
- The simplest way of using filters is enclosing them: {{{ the filter }}}
- You can customize the list produced by the filter using widgets.
- Write {{{ [search1] [search2] }}} to find tiddlers that are in search1 OR in search2.
- Write {{{ [search1] +[search2] }}} to find tiddlers that are in search1 AND in search2.
- Write {{{ [search1] -[search2] }}} to find tiddlers that are in search1 AND NOT in search2.
- Write {{{ [!search] }}} to find all tiddlers that are not in the search.
- The main part of a simple filter is the filter operator.
- There are many filter operators.
- It is impossible to show all operators here, so it is important to understand the filter operator reference in the [Tiddlywiki site](#).
- You can concatenate operators in a simple filter expression. For example, use {{{ [prefix[F]suffix[n]!sort[]] }}} to show the tiddlers that start with F and end with n sorted in descendant order.

9.2 What is a filter?

Filters are the heart of Tiddlywiki. More than a half or all customizations you make will include filters. This means that understanding filters is crucial.

To understand filters we need to realize some points:

- Tiddlywiki is a database of tiddlers. You saw in the code of Section 3.2 that there is a section of the html file called Store Area dedicated of storing the tiddlers.
- The main task of knowledge management is to find what information matches a certain pattern. In Tiddlywiki this means finding what tiddlers match a given pattern.
- You can do this task with Filters.
- When you search your information with a filter, Tiddlywiki gives you a list of tiddlers called “the output”. You find the List concept in Section 1.5.13.

So a filter is a concise notation for selecting a particular set of tiddlers. To work with filters you must perform two steps:

1. **Prepare the filter expression.** This includes the filter operators (tag, prefix, contains, field...) and parameters.
2. **Use the filter.** This includes using widgets (\$list, \$link, \$transclude...) and variables (<<currentTiddler>>).

9.3 The filter expression

A filter expression is many simple filters written one after another:

SimpleFilter1 □ SimpleFilter2 □ SimpleFilter3 ...

You must separate the simple filters with a white space. We call each simple expression “a filter step”.

This simple expressions can be preceded with the prefixes =, +, -, ~ or nothing:

- SimpleFilter1 □ SimpleFilter2: Union of the two list (OR combination)
 - Example: [tag[manager]] produces a list with all tiddlers tagged with the “manager” tag.
 - Example: [tag[manager]] □ [tag[employee]] produces a list with all managers and all employees (it list a tiddler if it has the “manager” tag OR the “employee” tag. The managers are listed only once even if they are duplicates).
- SimpleFilter1 □ = SimpleFilter2: Union of the two list with duplication
 - Example: [tag[manager]] □ = [tag[employee]] produces the list of all managers and all employees. Each person is listed once.
- SimpleFilter1 □ + SimpleFilter2: The tiddlers must match both filters. It is the intersections of the two list (AND combination)
 - Example: [tag[manager]] □ + [tag[has_car]] produce a list with all managers with car (it list a tiddler if it has the “manager” tag AND the “has car” tag).
- SimpleFilter1 □ - SimpleFilter2: The tiddlers must match the first filter expression and must not match the second. It is the difference of the two list (AND NOT combination).
 - Example: [tag[employee]] □ - [tag[manager]] produce a list of all employees that are not managers (it list a tiddler if it has the employee tag AND NOT has the manager tag).
- SimpleFilter1 □ ~ SimpleFilter2: It produces the list of the first filter. If it is empty, it produces the list of the second filter (ELSE combination)
 - Example: [tag[manager]] □ ~ [tag[employee]]. It produces a list with all managers. If there is no managers it list all employees.

9.4 Using filters

Imagine you have a wiki with the employees of your company. Some of them are managers and a few have a car given by the company. Your tags are: “employee” for all employees, “manager” for the managers and “has car” for the employees with assigned car.

Figure 9.1: The simple output



All employees

14th November 2019 at 12:46pm


Oliver Anderson
Harry Baker
James Collings
Thomas Dixon
Emily Foster
Emma Grant
Sophie James

Using filters can be tricky. The simplest use is enclosing the filter expression between three curly brackets:

- Add a new tiddler: “All employees”
- In the content of this tiddler add this text: `{{[tag[employee]]}}`
- Save the it.

You will see the output of the Figure 9.1. You know each line is the title of a tiddler, so in your wiki you have 7 tiddlers tagged with “employee”.

Figure 9.2: The desire output



Employees State

14th November 2019 at 12:16pm

- Harry Baker — Atlanta
- Thomas Dixon — Mississippi
- Emily Foster — Nevada
- Sophie James — Texas

But if you want a more complicated output you have to use widgets. Imagine you want to show the list of the Figure 9.2 with the tiddlers you have in your wiki. This is a list with the employees who have an assigned car and his state. You have to write this code:

```
<$list filter="[tag[employee]]+[tag[has car]]">
  <li><$link><currentTiddler></$link>—{!!state}</li>
</$list>
```

In this code we can see:

- We use the \$list widget.

- With each tiddler of the filter output, we show:
 - A list `` element.
 - The link (`$link` widget) of current tiddler (`<<currentTiddler>>` variable).
 - A long dash (the `- - -` element).
 - The content of the “state” field of the current tiddler (`{{!!state}}` element).

9.5 The filter basic step

Each Simple filter is called “a basic step”. You may have notice that all SimpleFilter expression is surrounded between `[]`. So each Filter Basic step is:

```
[ ! operator:suffix parameter ! operator:suffix parameter ... ]
```

You can repeat this sequence as many times as you need. Each occurrence of “!operator:suffix parameter” use the list produced by its predecessors. This list is called “the input”.

All elements are optional:

- The `!` means not. For example, `[!tag[has-car]]`: tiddlers without the tag “has-car”.
- Operator: the most important part of the filter. In the above examples the operator is “tag”.
- Suffix: some operators has special flags. For example, the tag operator has a flag, `strict` that modifies its behaviour in some cases so the output with `{{{ [tag[]] }}` is different from `{{{ [tag:strict[]] }}`.
- Parameter: In the above examples the parameter is the desire tag, “employee”, “manager” or “has-car”. Usually the parameter is surrounded with `[]` too. If there is no parameter: write `[]` as above.

The main trick of filters is understand the filter operators. There are a lot of them so write about all is out of the scope of this publication. If you want to know more about them you have to go to the [Filter Operators](#) section of Tiddlywiki.

Let’s look an example:

Figure 9.3: The prefix operator

prefix Operator

3rd February 2015 at 8:27pm

Filter Operators Negatable Operators String Operators

purpose	filter the input titles by how they start
input	a selection of titles
parameter	S = a string of characters
output	those input titles that start with S
! output	those input tiddlers that do not start with S

In looking for matches for S, capital and lowercase letters are treated as different.

[Examples](#)

How to read this tiddler:

- Purpose: What you can use the operator for.
- Input: The first “!operator:suffix parameter” occurrence has all tiddlers as the input. For the second occurrence, the input is the list produced by the first and so on. Each “!operator:suffix parameter” filters the list produced by the predecessors.

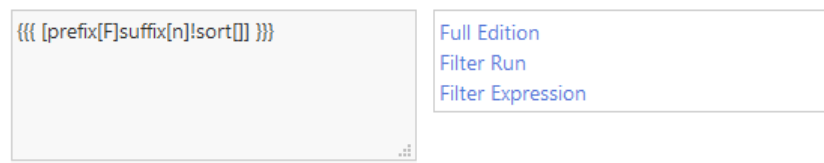
- Parameter: The parameter of the operator. In this operator is a sequence of characters so you can write `[prefix[H]]` to show all tiddlers with the tittle starting with “H”.
- Output: What list produce the operator.
- !Output: What list produce the operator if you put, for example, `[!prefix[H]]`.

Example: `{{[prefix[F] suffix [n]!sort []] }}` This **simple filter expression** has three repetitions of “!operator:suffix parameter” and it means: “Show all tiddlers that begin with F and end with n in descending order”:

- `{{{` : Show the result filter expression as a list of tiddlers.
- `[` : Start of the Simple Filter Expression.
- `prefix[F]`: produce the list of all tiddlers whose title begins with “F”.
- `suffix[n]`: Filter this list and produces a list with the tiddlers whose title ends with “n”
- `!sort[]` Sort descending the list of the tiddlers whose title starts with “F” and ends with “n”.
- `]` : Ends the Filter Simple Expression.
- `}}}` : Finish the code.

If you add a tiddler with this code to the tiddlywiki site you can see the Figure 9.4.

Figure 9.4: A filter example



Remember that this is a single step of a filter expression and that you can concatenate as many single filter expression as you want. For example, you can show the tiddlers whose title starts with “F” and ends with “n” tagged with “Filter Syntax” with this code:

```
{{{ [ prefix [F] suffix [n] ] + [tag [ Filter Syntax ] ] }}
```

We have here two simple filter expressions, the second one with the “+” prefix forming a more complex filter expression.

9.6 Regular expressions

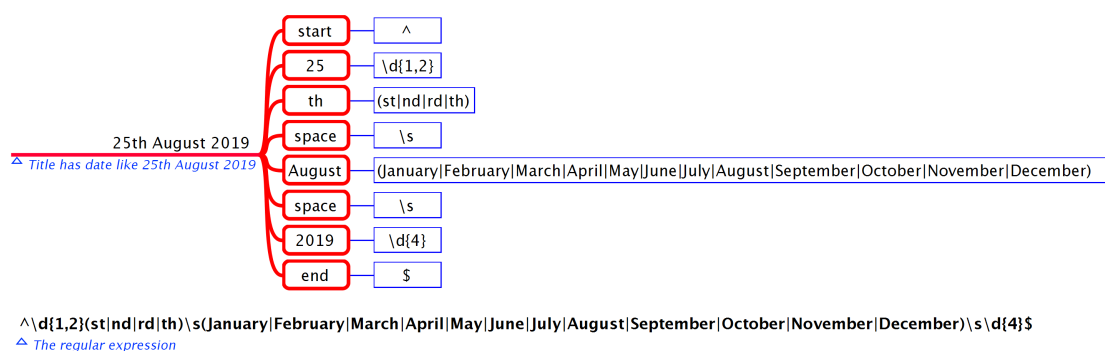
9.6.1 Writing regular expressions

Regular expressions are concise strings of characters that denote patterns of text to search for. Tiddlywiki uses the Javascript regular expressions. The most important is the representation of this patterns. This table shows the most important patterns:

character	meaning
.	Any character
*	zero or more repetitions of the character or previous group
+	one or more repetitions of the character or previous group
^	Start of line

character	meaning
\$	End of line
[]	Group or characters
()	Sub expressions
(a b c)	a or b or c
?	Coincides with the previous element zero or one times
??	Coincides with the previous element one or no times
{n}	Coincides with the previous element n times
{n,m}	with the previous element at least n times and at most m times
[a-z]	Any character between a and z
\w	Any character of a word: a-zA-Z
\W	Any character other than the previous one
\s	Blank space
\S	Any character that is not a blank space
\d	A dígit
\D	Anything other than a digit
\t	a tab
\r	Carriage return
\n	New line
\1 \2 \3 ...	The group number 1 or number 2 or the n
\	The following character is treated as literal (e.g. \ * indicates the asterisk)

Figure 9.5: Building the regular expression



The most complicated thing is to build the regular expression. I recommend dividing the string you want to look for in its main parts. For example, imagine you want to search all tiddlers that has a date in his title like 25th August 2019. In Figure 9.5 you can see all parts of this string:

- \wedge : The start of the line
- $\d{1,2}$: One or two digits

- (st|nd|rd|th): One of this: st or nd or rd or th.
- \s: a space
- (January|February|March|April|May|June|July|August|September|October|November|December): The month
- \s: A space
- \d{4}: Four digits
- \$: The end of the line

So the regular expression is

```
^\d{1,2}(st|nd|rd|th)\s(January|February|March|April|May|June|July|August|September|October|November|December)\s\d{4}$
```

9.6.2 Using regular expressions

Using regular expressions is not complicated. Tiddlywiki has a filter operator: `regexp` to define this kind of searches. Create a new tiddler and add this in the body:

Tiddlers titled with two letters: `{{[regexp[^\w{2}$]]}}`.

Tiddlers ending with “wiki” in its title: `{{[regexp[wiki$]]}}`.

Tiddlers created in August 2014: `{{[regexp:created[~201408]]}}`.

If you want more complicated expressions is better to define a variable with the pattern. Tiddlywiki has problems with groups. For example, if you want to search all tiddlers ending with the words “wiki” or “tiddler” and write: `{{[regexp[(wiki|tiddlers)$]]}}` Tiddlywiki shows an error. You have to write:

```
<$set _name="pattern" _value="(wiki|tiddlers)$">
```

```
{{[regexp<pattern>]]}}
</$set>
```

So, if you want to show all tiddlers who has a date in its title, as in Figure 9.5 you have to write:

```
<$set _name="pattern" _value="^\d{1,2}(st|nd|rd|th)\s(January|February|March|April|May|June|July|August|September|October|November|December)\s\d{4}$">
```

```
{{[regexp<pattern>]]}}
</$set>
```

One tricky question in regular expressions are capturing groups. Each subexpression (a pattern surrounded by parentheses) is called a capturing group. We can refer to the first capture group by `\1`. The second by `\2` and so respectively.

Imagine you want to show all tiddlers that starts and ends with “Wiki”:

```
<$set _name="pattern" _value="^(Wiki).*\1$">
{{[regexp<pattern>]]}}</$set>
```

The (Wiki) is the first capturing group of the expression. If you want to refer later to this pattern you write `\1`.

Be careful with this. Maybe you want to search a tiddler that start and end with any two numbers, so you write this: `^\d{2}.*\1$`:

- `^`: Start of line
- `(\d{2})`: two digits, first capturing group.
- `.*`: Any character repeated many times

- \1: Two digits again.
- \$: End of line

You will find only the tiddlers who start and end WITH THE SAME TWO DIGITS. If you want to search not the same numbers you have to write $\wedge\{d{2}\}.*\{d{2}\}$.

10 Macros

10.1 Key points

- Tiddlywiki includes some core macros.
- Macros simplify the writing of tiddlers.
- Try not to add complex code to your normal tiddlers.
- Add this code inside other macro tiddler.
- This simplify the maintenance of your wiki.
- You define a macro with `\define name-of-the-macro(param1:"default value1" param2:"default value2")`
- You use this macro in your tiddlers writing `<<name-of-the-macro "param1 value" "param2 value">>`
- You can't pass variables as parameters of the macro using the `<<name-of-the-macro>>` notation.
- To pass variables you must write: `<$macrocall $name:"the-name-of-the-macro" current:<<currentTiddler>> />` notation.
- There are other kind of macros: the javascript macros written in javascript language.
- These macro are complex to write.

10.2 Introduction

A macro is a named snippet of text. WikiText can use the name as a shorthand way of transcluding the snippet. In the next code we define a macro for a greeting:

```
\define sayhi(name:"Bugs Bunny" address:"Rabbit Hole Hill")
Hi, I'm $name$ and I live in $address$.
\end
```

Later we can use it in this way:

```
<<sayhi "Donald Duck" Disneyland>>
```

And the result is: *Hi, I'm Donald Duck and I live in Disneyland.*

10.3 Variables and parameters

10.3.1 Defining variables

There are two ways of defining variables. The first with the `<$set>` sidget and the second with the `<$vars>` widget. With the first we define a variable with each set. With the second we can define multiple variables at once.

- ```
<$set name="myvariable" value="the_value">
<<myvariable>>
</$set>

<$vars greeting="Hi" me={{!!title}} sentence=<<helloworld>> >
<<greeting>>! I am <<me>> and I say: <<sentence>>
</$vars>
```

### 10.3.2 Parameters

In the sayhi macro of the introduction we can see two parameters: name and address. A parameter is like a variable but in the scope of a macro. They have values when using the macro.

### 10.3.3 Using variables and parameters

We can use variables in parameters in many places: in a tiddler, inside the definition of a macro, in fieds, in filters, in links... The way of using them depends of the place and is very very tricky. In [Figure 10.1](#) you can see the way how do we have to refer to variables and parameters.

### 10.3.4 Javascript macros

There is other kind of macro. We can define a macro in a javascript tiddler. These are Javascript Macros. Tiddlywiki comes with several of this macros. For example imagine that you have to write a javascript macro to insert a little index with the titles of your tiddler that is, the `!Title1` to `!!!Title3` wikitext.

Insert a new tiddler an tag it with `“$/tags/Macro”`. Assign it the type: `application/javascript` and write this code:

```
(function () {

"use_□strict ";
```

```

exports.name="index";
exports.params=[{ name: "head" }];

exports.run=function(head){
 var currentTiddlerName=this.getVariable("currentTiddler");
 var currentTiddler=this.wiki.getTiddler(currentTiddlerName);
 var value=currentTiddler.getFieldString("text");

 var result=" "+head+"\n<small>\n\n"
 result=result+"@@.tc-tiddler-frame\n@@width:300px;\n@@background-color: Beige;\n";
 var partial;
 var pattern=/^(!+)(\w.+?)$/gm;

 while((partial=pattern.exec(value))!=null){
 result=result+Array(partial[1].length+1).join("*")+
 partial[2]+"\n";
 result=result+"@@</small>"
 }
 return result;
}

})();

```

This macro reads all text of the current tiddler looking for the pattern:

```
pattern = /^(!+)(\w.+?)$
```

These are the titles of the tiddler. The macro returns a wikitext string with these titles.

Writing javascript macros is more difficult than writing the first. You have to know the \$tw tree object to access all the tiddlywiki methods included. An introduction of this question is in the Chapter 12.

## 10.4 Using macros

You can see all the core macros included with Tiddlywiki in <https://tiddlywiki.com/#Core%20Macros>. To use them you have to write:

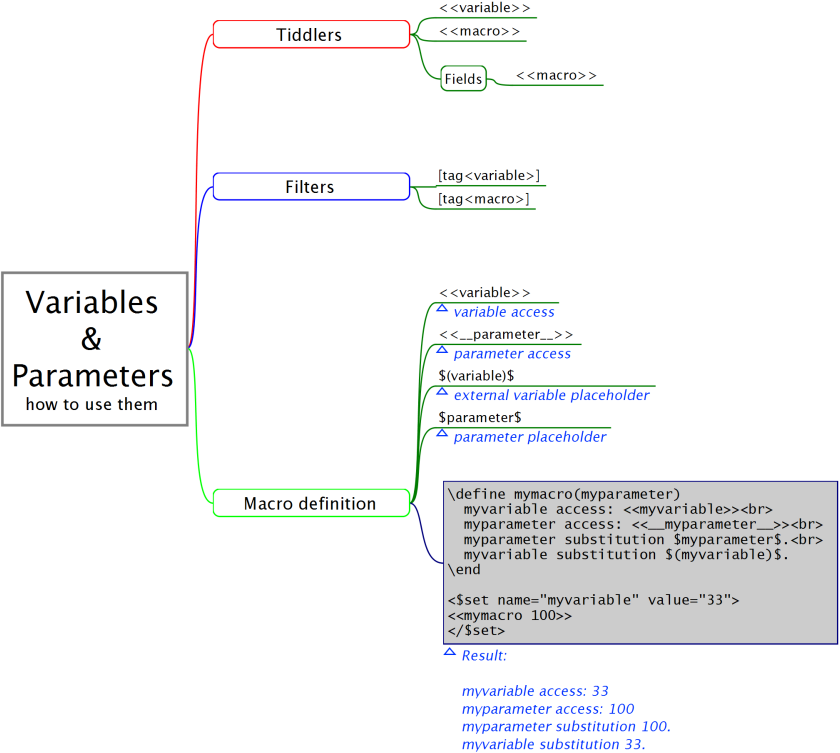
```
<<my-personal-macro param1:"value1" param2:"value2">>
```

There is a tricky question. All the parameters must be string no variables are allowed, so we can't write: ~~<<my-personal-macro current=<<currentTiddler>>>>~~. This is a very important restriction because we need to pass variables to our macros. It is such an important issue that Tiddlywiki allows us another way to call macros, the \$macrocall widget:

```
<$macrocall $name="my-personal-macro" current=<<currentTiddler>> param1="value1" param2="value2"/>
```

Here I can show a little question about macro design. Try not to add complex code to the tiddlers you use as information tiddlers. This add unnecessary complexity to your tiddlywiki and makes it difficult to maintain. Is better to add additional tiddlers tagged with "\$:/tags/Macro" and write your code named with an easy name to remember. After all the code you can add some information about the macros included in the tiddler.

Figure 10.1: Using variables



# 11 Widgets

## 11.1 Key points

- You can add dynamic behaviors to your wiki using widgets.
- Instead manually writing all tiddlers of a category, you can use the \$list tiddler. It build a dynamic list of them.
- If you add a new tiddler to this category the list will change immediately.
- You can use the checkbox widget in tag mode to add a tag to a tiddler.
- The \$edit-text widget use a tiddler or a field of a tiddler to store its content.
- You use the \$button widget to launch actions as delete all tiddlers of a list (\$list widget).
- You can combine widgets.
- With javascript you can program your own widgets.

## 11.2 Introduction

Widgets are trickiest part of Tiddlywiki. With widgets you can have a more dynamic Tiddlywiki. Each widget has a special purpose: add a checkbox, create a button, display a field etc. There are core widgets and other widgets you can add as plugins.

We learn widgets solving a problem: Create a custom to-do wiki.

## 11.3 First approach. The manual way

In this to-do wiki each task is a tiddler with the “task” tag. You need a tiddler for each task and other tiddler to control the tasks. This are the steps:

- Create the “Compose ballad” tiddler tagged with “task”.
- Create the “Make the beds” tiddler tagged with “task”.
- Create the “Kill the Dragon” tiddler tagged with “task”.
- Create the “Get the Ring” tiddler tagged with “task”.
- Create the “Go to Mordor” tiddler tagged with “task”.
- Create the “Task control” tiddler with this code:

```
* [[Compose ballad]]
* [[Make the beds]]
* [[Kill the Dragon]]
* [[Get the Ring]]
* [[Go to Mordor]]
```

This approach has a lot of work. If you want to add a new task you have, not only to create the tiddler tasks but change the “Task control” tiddler to add this new task. Tiddlywiki has a best way of adding them.

## 11.4 The checkbox widget

Before continuing we will review the "checkbox" widget.

If you read the tiddler for this widget you find that you can save the status of the widget in:

- A tag (tag mode): the easiest
- A tiddler: useful for shared status.
- A field (field mode): of the same tiddler or other tiddler.
- An index of a data tiddler (index mode).

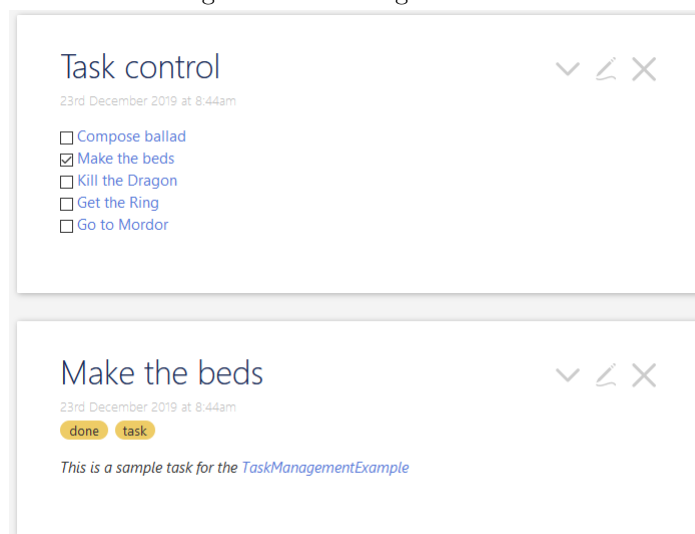
These are the attributes for the checkbox widget:

| Attribute | Description                                                                                                                                   |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| tiddler   | Title of the tiddler to manipulate (defaults to the current tiddler)                                                                          |
| tag       | The name of the tag to which the checkbox is bound                                                                                            |
| invertTag | When set to yes, flips the tag binding logic so that the absence of the tag causes the checkbox to be checked                                 |
| field     | The name of the field to which the checkbox is bound                                                                                          |
| index     | New in: 5.1.14 The index of the tiddler, a DataTiddler, to which the checkbox is bound be sure to set the tiddler correctly                   |
| checked   | The value of the field corresponding to the checkbox being checked                                                                            |
| unchecked | The value of the field corresponding to the checkbox being unchecked                                                                          |
| default   | The default value to use if the field is not defined                                                                                          |
| class     | The class that will be assigned to the label element                                                                                          |
| actions   | New in: 5.1.14 A string containing ActionWidgets to be triggered when the status of the checkbox changes (whether it is checked or unchecked) |

Let's see how we can use this widget.

## 11.5 The \$list widget

Figure 11.1: Adding a checkbox



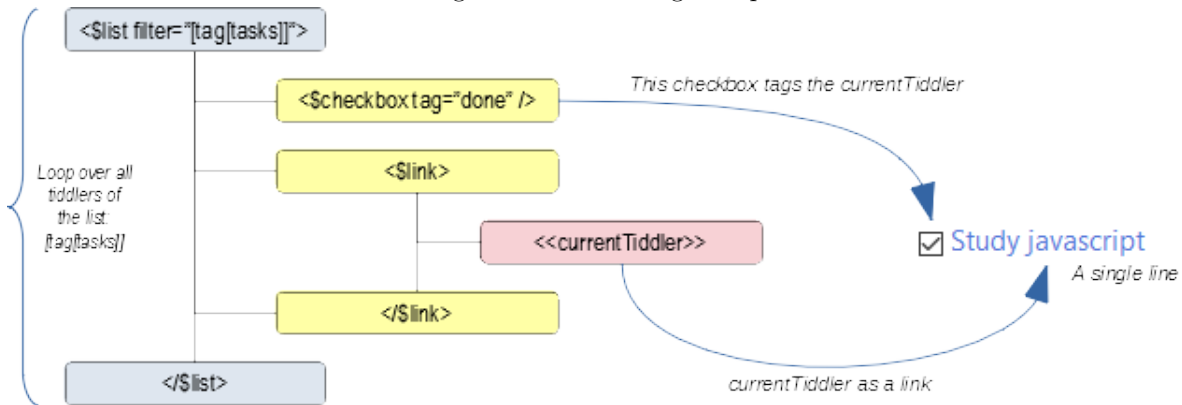
Adding the task manually as in Section 11.3 is a hard work. We have to simplify things. We will use the \$list widget to list the task. We use for this the “task” tag. We saw the \$list widgets in the “Filters” chapter. Is an easy way to list custom tiddlers. For our purpose we want to list all task tiddlers:

```
<$list filter="[tag [task]]">
<$checkbox tag="done"/><$link <<currentTiddler>>/></$link>

</$list>
```

If you tag all tasks with the tag “task” the output will be the same of Figure 11.1. The difference is that creating new tasks is easy: create a new tiddler and tag it with the “task” tag. The list will update immediately. You can see this widget as a loop: it loops over all tiddlers of the filter list executing the code between <\$list..> and </\$list>. Look at Figure 11.2.

Figure 11.2: The widget loop



In this example our widget loops over the tasks tiddlers writing the lines:

```
<$checkbox tag="done"/> <$link><<currentTiddler>></$link>

```

We use the checkbox widget in “tag mode”. If you check the “Make the beds” checkbox Tiddlywiki will add the tag “done” to this tiddler permanently (of course you have to save your wiki if you don’t want to lose all your work.).

### 11.5.1 Paragraphs in Tiddlywiki

There is an important question here: the widget inserts lines one after another without any blank lines in the middle. This can cause problems with wikitext: you know that there are wikitext formats that need a blank line before them. For example, if you add to a tiddler:

```
This is my list :
* item 1
* item 2
* item 3
```

You will see this output:

```
This is my list : * item 1 * item 2 * item 3
```

If you want the items as a list you have to add a blank line:

```
This is my list :

* item 1
* item 2
* item 3
```

In many cases in the list widget you have to add this extra line after the <\$list> first line. For example, if you want to show our output as a wikitext list with bullets you have to write:

```
<$list filter="[tag[task]]">

* <$link><<currentTiddler>></$link>
</$list>
```

If you don’t add the empty line after the \* element the output will be wrong.



## 11.6 \$edit-text widget

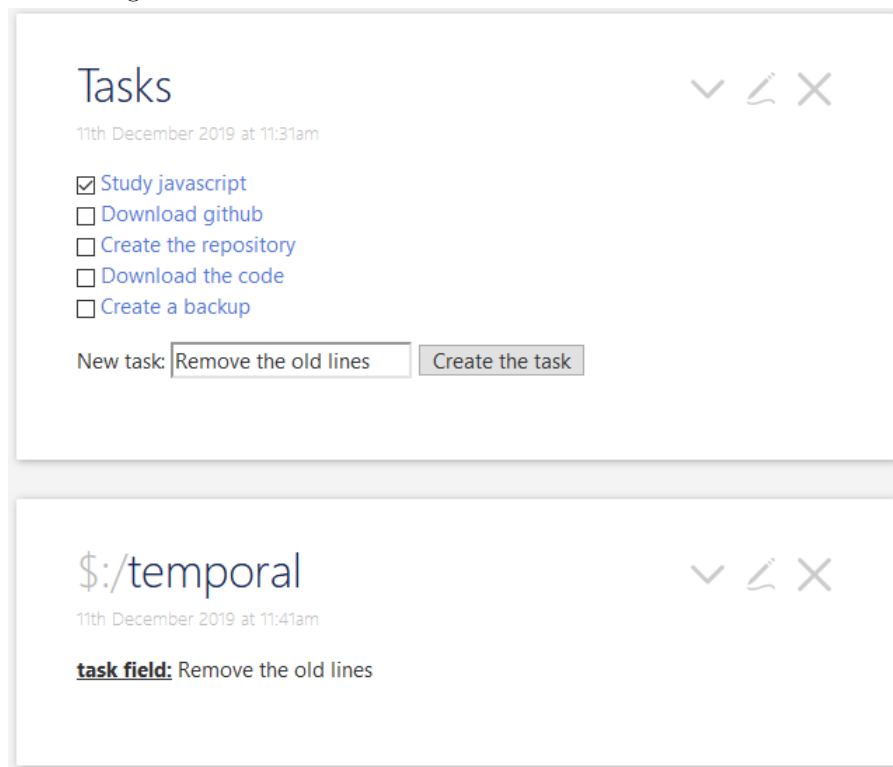
Maybe we can simplify the adding of new task. First we analyze the \$edit-text widget. It provide us a simple way of edit temporal data. We use it for writing the text of the new task:

```
New task: <$edit-text tiddler="$:/temporal" field="task"/>
```

All things you write to this text little rectangle applies immediately to the field “task” of the tiddler “\$:/temporal”. In Figure 11.3 you see the field “task” of this “\$:/temporal” tiddler. We use this value later to add our new task. The big problem: you cannot use this widget to edit fields of the same tiddler because of the lost control of it.

## 11.7 \$button widget

Figure 11.3: The tasks list with a text area to add new tasks



A button execute some code. You can use a button to delete a tiddler, to add it, to create an alert (an special tiddler) to add some tag to a tiddler or a list of tiddler (looping with the \$list widget inside the \$button widget) etc.

To create a new task we write:

```
<$button>
<$action-createtiddler $basetitle={{$:/temporal!!task}} tags="task"/>
<$action-setfield $tiddler="$:/temporal" $field="task" text=""/>
Create task
</$button>
```

When we click the button Tiddlywiki creates a new tiddler titled with the text area and tagged with “task”. Then delete the text area.

The all code of our Tasks tiddler is this:

```

<$list filter="[tag[task]]">
<$checkbox tag="done"/> <$link><<currentTiddler>></$link>

</$list>

New task: <$edit-text tiddler="$/temporal" field="task"/>
<$button>
<$action-createtiddler $basetitle={{$/temporal!!task}} tags="task"/>
<$action-setfield $tiddler="$/temporal" $field="task" text=""/>
Create task
</$button>

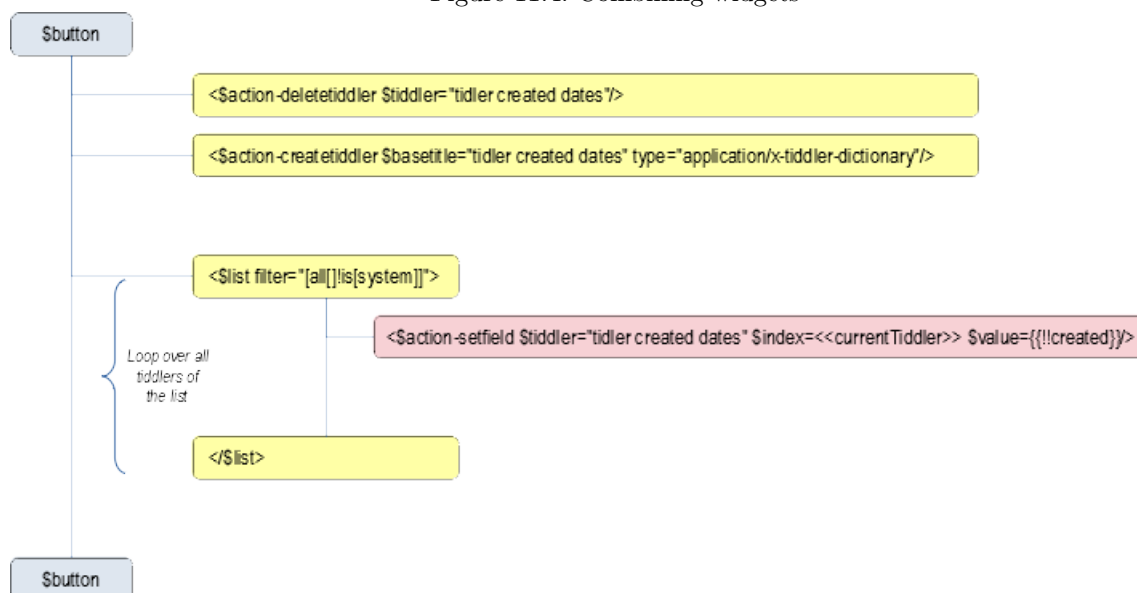
```

The output is in Figure 11.3.

## 11.8 Combining widgets

### 11.8.1 Collecting information

Figure 11.4: Combining widgets



Imagine you want to create a data tiddler with all tiddlers of your wiki and the date they were created. The data tiddler will “tidler created dates”. This is the code:

```

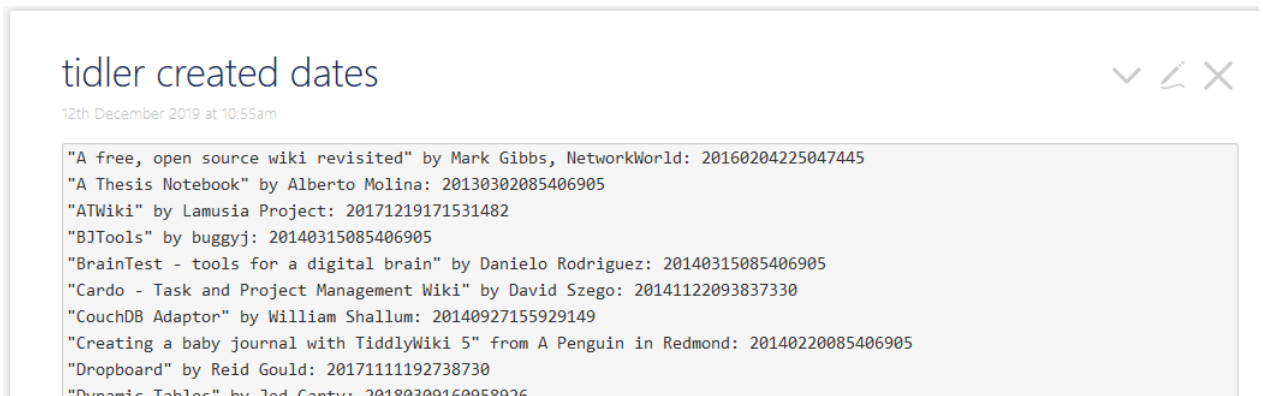
<$button>
<$action-deletetiddler $tiddler="tidler created dates"/>
<$action-createtiddler $basetitle="tidler created dates" type="application/
x-tiddler-dictionary"/>

<$list filter="[all[]!is[system]]">
<$action-setfield $tiddler="tidler created dates" $index=<<currentTiddler>>
 $value={{!!created}}/>
</$list>
Collect Dates
</$button>

```

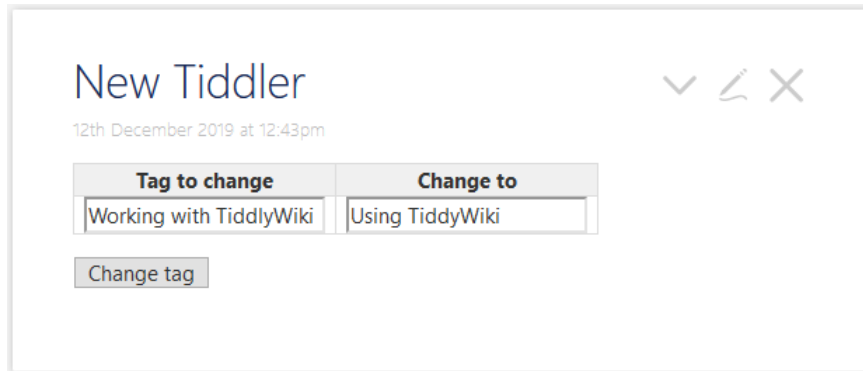
This will loop over all elements of the list inserting the title and the created field in the tiddler “tidler created dates”. In Figure 11.4 its a little diagram of the code. Now, if you open this tiddler you will see the dates. The first four numbers are the year, the next two the month and the next two the day. And after that there is the hour, minute... (look at Figure 11.5)

Figure 11.5: Tiddler created dates



## 11.8.2 Change a tag

Figure 11.6: Tiddler for changing a tag



You want to create a tiddler to modify a tag from its original value to other value. Let's look what widgets we need. If you investigate inside the Tiddlywiki site you will find this information:

- There are two messages: tm-remove-tag and tm-add-tag.
- The messages are sent inside a \$button with the action-sendmessage widget.
- This two message needs the \$fieldmangler to set to which tiddler the change applies.
- You need two \$edit-text widget. One for the tag to change and the other for the "change to" tag.

This is the code:

```
|_Tag_to_change_|_Change_to_|h
|<$edit-text_tiddler="$:/temporal"_field="tag_from"/>|<$edit-text_tiddler="
$:/temporal"_field="tag_to"/>|

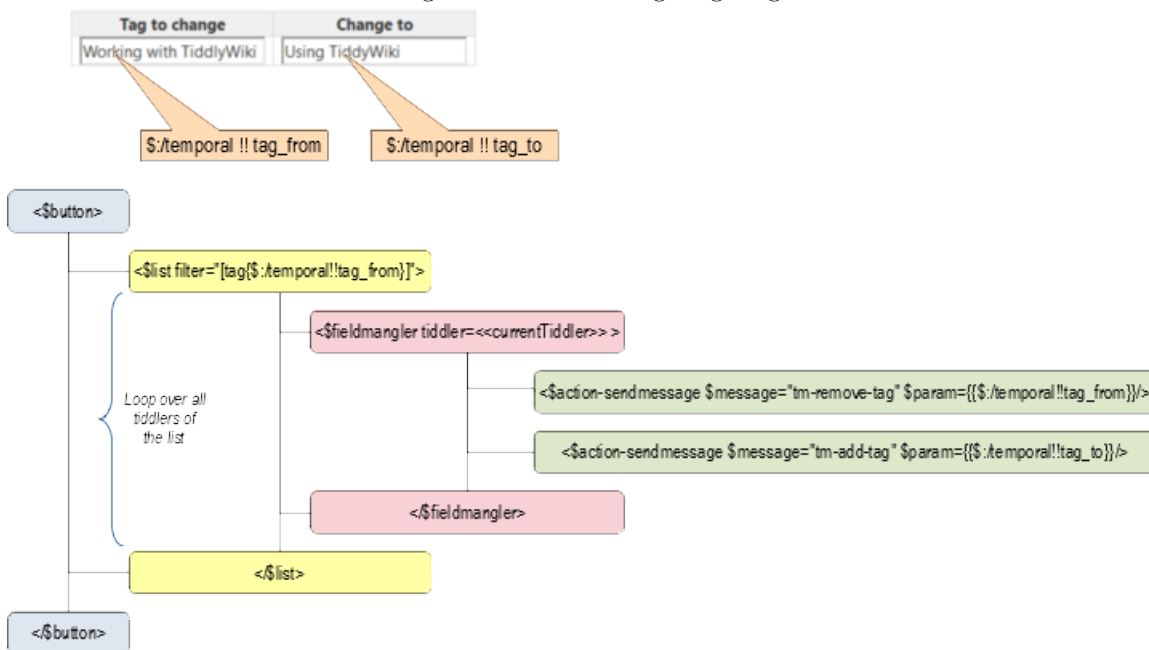
<$button>
<$list_filter="[tag{$:/temporal!!tag_from}]">
<$fieldmangler_tiddler=<<currentTiddler>>>
<$action-sendmessage_$message="tm-remove-tag"_$param={{$:/temporal!!
tag_from}}/>
<$action-sendmessage_$message="tm-add-tag"_$param={{$:/temporal!!tag_to}}/>
</$fieldmangler>
</$list>
```

Change\_tag  
</\$button>

Notes:

- The three above lines are for drawing the top table with the two edit text zones.
- The action widgets are triggered by the \$button widget.
- We loop over all tiddlers tagged with tag\_from: it is the [tag{\$/temporal!!tag\_from}] filter
  - We insert the messages inside a \$fieldmangler widget to set the tiddler to modify.
  - For each tiddler of the list we send the two messages with the \$action-sendmessage tiddler.

Figure 11.7: The change tag diagram



## 11.9 Writing your own widgets

Figure 11.8: The select widget output

**B** The current selection is: B

writing widgets to either extend or specialize TiddlyWiki5. The purpose of this document is to give new TiddlyWiki5 developers some understanding of how widgets work and how to create their own.

By way of example we'll look at a custom widget developed for a recent project. That widget provides a "select" mechanism, allowing users to choose from a drop-down list of items; the item selected is passed to the "child" macros and widgets of the widget. Thus, the Figure 11.8 is the output for this code:

```
<$select list="A_B_C" name="selection"> The current selection is: <<
 selection>></ $select>
```

This widget is not a core widget. You can drag it from the Tiddlywiki where this tiddler is to yours.

### 11.9.1 Data structures

To understand widgets we need some understanding of four core data structures:

1. The Parse Tree
2. The Widget Tree
3. The Dom Node Tree
4. The Dom itself.

The choreography of the work needed on these four data structures may not be intuitive, so I'll give my understanding at the level needed to get started.

This question is out of the goal of this book. We show how to add javascript code to your tiddler.

#### 11.9.1.1 The parse tree

The Parse Tree is generated by TiddlyWiki5 core parsers as an overall interpretation of all the tiddlers which need to be presented in the browser. One of the first things which happens when a tiddler is to be presented is that it is parsed<sup>17</sup> by the specific parser for the indicated tiddler "type". Parsing a tiddler results in a parse tree for that tiddler which is injected into the overall Parse Tree. When an instance of our widget is initialized, the Parse Tree location of "this" instance of our widget is passed to our widget via "parseTreeNode". Our widget needs to maintain that location via `this.parseTreeNode` so that we can reference it later, and repeatedly, to do things like reading widget attributes when we render our widget.

#### 11.9.1.2 The widget tree

The Widget Tree is generated during rendering and is instantiated by a combination of our `initialize()` and `render()` functions; it establishes hierarchic relationships between the widgets rendered in the wiki. In a sense, the Widget Tree overlays the Parse Tree, providing a widget-only view with links back to the Parse Tree. We need to know about it because the Widget Tree does at least two very important things for us:

1. It manages the refresh process, so that changes to tiddlers or attributes ripple through the branches of the tree efficiently. In the case of our `<$select...>` widget example we use this mechanism to refresh the widget itself if any of the widget attributes are changed, otherwise we pass the refresh request down through the hierarchy.
2. It allows us to create "widget variables" which are visible anywhere in the branches from the widget which "sets" such variables. In the case of the `<$select....>` widget example we use this mechanism to "pass" the selection result to child widgets and macros, so that the selection can inform the wikitext contained by the `<$select....>` widget.

Two properties of the widget function codify the Widget Tree: `this.parentWidget` and `this.children`. Generally the core Widget function manages these for us.

#### 11.9.1.3 The dom node tree

The Dom Node Tree is generated by our widget ( `domNodes` in `Widget()` ); it establishes hierarchic relationships between the Widget Tree and the any Dom nodes which we want the widget mechanism to manage. Typically a widget needs to add a single, root Dom node which we create in our render function, but in principle a widget can include several Dom nodes without a common root. We care because the Dom tree is used to manage deletion of the Dom nodes which we create within a widget. To make that mechanism work, the root Dom node created by our widget needs to be bolted into the Dom node tree. We also need to make certain that the default `Widget.prototype.removeChildDomNodes` deletes the Dom nodes which our widget creates, or, if it doesn't then write a replacement function which does.

#### 11.9.1.4 The dom

The Dom is the Dom. We need to know about it, understand it, "program" it, because it is the mechanism by which we do the job of rendering. Widgets "do" their rendering via a render function, which is where we perform standard Dom manipulations to render the html which we desire from our widget.

#### 11.9.2 Code notes

The javascript for the <\$select....> widget is available at the end of this chapter.

If you look through the code you will see that it is largely a copy-paste of almost any one of the standard widgets. There are seven functions in all; the first five functions represent a pattern common to most widgets; the last two functions are helper functions specific to our widget:

Core Widget Functions:

```
SelectWidget.prototype.render
SelectWidget.prototype.execute
SelectWidget.prototype.refresh
SelectWidget.prototype.removeChildDomNodes
SelectWidget.prototype.create
```

Helper Functions:

```
SelectWidget.prototype.getOptionList
SelectWidget.prototype.handleChangeEvent
```

#### Render

Given that the purpose of a widget is to render, the heart of a widget is the render function. Typically, we don't need to do any customization of the render function, but we need to understand what it does. The reason that we don't typically need to do much in render is that most of the work is typically done by two major functions invoked by render, namely execute and create, and we do most of our customization in those functions.

The main steps of "render" are as follows:

1. Maintain a link to the parent DOM node, i.e. the DOM element immediately above any DOM elements which we create with this widget.
2. Fetch any attributes which were included in the "call" to this widget.
3. Perform calculations needed ahead of construction of the DOM elements for this widget, including making any "child" widgets required by this widget.
4. Create the actual DOM elements for this widget. Insert the DOM elements for this widget into the parent DOM node structure.
5. Use renderChildren to render each of the child widgets. These child widgets are rendered as DOM child elements of the DOM node we created earlier with create. These child widgets were either created by the earlier execute function or could be widgets wrapped by this widget in the wikiText.

```
SelectWidget.prototype.render = function(parent, nextSibling) {
this.parentDomNode = parent;
this.computeAttributes();
this.execute();
var domNode = this.create(parent, nextSibling);
this.domNodes.push(domNode);
parent.insertBefore(domNode, nextSibling);
this.renderChildren(domNode, null);
};
```

## Execute

The execute function is invoked by the render function, and it typically does two things for us:

1. Fetches attributes which were included in the widget's invocation, whether wikiText or otherwise.
2. Makes child widgets

Child widgets can come from the originating wikitext, or be created programmatically as part of the the execute function.

```
SelectWidget.prototype.execute = function() {
 // get attributes
 this.filter = this.getAttribute("filter");
 this.list = this.getAttribute("list");
 this.tiddler =
 this.getAttribute("tiddler", this.getVariable("currentTiddler"));
 this.selectClass = this.getAttribute("class");
 this.setName = this.getAttribute("name", "currentTiddler");
 // make child widgets
 this.makeChildWidgets();
};
```

In this case we don't create any child widgets as part of this widget, but we do need to make sure that we make any child widgets which originate in the wikitext. For a good example of creating child widgets programmatically I recommend taking a look at the 'execute' function of the `<$edit....>` widget of the tiddler: `$/core/modules/widgets/edit.js` at the core of Tiddlywiki.

## Create

The create function is invoked by the render function. It's job is to create the actual DOM nodes forming the rendering of the widget. It is really that simple.

There are a couple of TiddlyWiki5 core functions used in the example which are worth noting for future use:

**\$tw.utils.domMaker** This utility function is provided in the core to help to create DOM nodes with a "class" attached. You have a choice of doing this in one line with domMaker:

```
var domNode = $tw.utils.domMaker("div",{class:this.selectClass});
```

Or you can do the same with two lines using plain old DOM manipulation, which would look like this:

```
var domNode = this.document.createElement("div");
domNode.className = this.selectClass;
```

**\$tw.utils.addEventListeners** This utility function registers event handler functions to DOM objects. In the `<$select....>` widget we add an event handler to the Dom `<select>` entity for "change" events occurring on the widget. Of course we need to provide the event handler function too, and in this case we provide `Select.prototype.handleChangeEvent(event)`.

```
$tw.utils.addEventListeners(select,[
 {name: "change", handlerObject: this, handlerMethod: "handleChangeEvent"}
]);
```

**Widget.prototype.setVariable and Widget.prototype.getVariable** `this.setVariable` and `this.getVariable` are the function invocations used to set and get widget variables accessible downstream by child widgets and macros. By "setting" a variable with `setVariable`, any child widget can access that variable directly via `getVariable`. The widget mechanism does the work of searching back up the widget tree for the nearest, corresponding, `setVariable`.

## Refresh

The refresh function is invoked either externally, typically by the widget's parent widget by invoking refreshChildren. For the <\$select....> widget we do a refresh if and only if the driving attributes have changed. In that case we invoke the standard widget refreshSelf function, which removes child DOM nodes of this widget and then re-renders the widget, this time according to the new attributes.

If none of the attributes has changed then all we do here is invoke refreshChildren in case any of the child widgets need or want to do a refresh based on changed tiddlers or changed attributes etc.

## RemoveChildDomNodes

The removeChildDomNodes function is invoked by refreshSelf to remove the DOM nodes which we added to the domNodes array which we built during render. Typically all of the DOM nodes created by a widget are children of a single root DOM node, so that removeChildDomNodes is a boiler-plate copy of the corresponding, default Widget function.

### 11.9.3 Code of the \$select widget

```
/*\
title: $:/core/modules/widgets/selectWidget.js
type: application/javascript
module-type: widget
Implements the <$select widget - to render a <select> dom element containing an <option>
 dom element
for each item in an option list. The option list is generated from a filter expression,
 or a list
tiddler, or a plain text tiddler. The current selection is stored in a widget variable
 accessible
by the child widgets or via template insertion to the any enclosed text and/or child
 widgets.
...
<$select filter="...." list="...." tiddler="...." name="...."/>
...
*/
(function(){
/*jslint node: true, browser: true */
/*global $tw: false */
"use strict";
var Widget = require("$:/core/modules/widgets/widget.js").widget;
var SelectWidget = function(parseTreeNode,options) {
this.initialise(parseTreeNode,options);
};
SelectWidget.prototype = new Widget();
SelectWidget.prototype.render = function(parent,nextSibling) {
this.parentDomNode = parent;
this.computeAttributes();
this.execute();
var domNode = this.create(parent,nextSibling);
this.domNodes.push(domNode);
parent.insertBefore(domNode,nextSibling);
this.renderChildren(domNode,null);
};
SelectWidget.prototype.execute = function() {
// get attributes
this.filter = this.getAttribute("filter");
this.list = this.getAttribute("list");
this.tiddler = this.getAttribute("tiddler",this.getVariable("currentTiddler"));
this.selectClass = this.getAttribute("class");
this.setName = this.getAttribute("name","currentTiddler");
// make child widgets
this.makeChildWidgets();
};
/*
Selectively refreshes the widget if needed. Returns true if the widget or any of its
 children needed re-rendering
*/
SelectWidget.prototype.refresh = function(changedTiddlers) {
```



```

var changedAttributes = this.computeAttributes();
if(changedAttributes.filter || changedAttributes.list || changedAttributes.tiddler) {
this.refreshSelf();
return true;
} else {
return this.refreshChildren(changedTiddlers);
}
};
SelectWidget.prototype.removeChildDomNodes = function() {
$tw.utils.each(this.domNodes, function(domNode) {
domNode.parentNode.removeChild(domNode);
});
this.domNodes = [];
};
SelectWidget.prototype.create = function() {
// create a <div> container for the <select>
var domNode = $tw.utils.domMaker("div",{class:this.selectClass});
// create the <select> element
var select = this.document.createElement("select");
select.className = this.selectClass;
// get the list of select options
var optionList = this.getOptionList();
// fetch the current selection, defaulting to the first option in the option list
var selection = this.getVariable(this.setName);
if(!selection) this.setVariable(this.setName, optionList[0], this.parseTreeNode.params);
// create and add the <option> elements
for (var i=0; i < optionList.length; i++) {
var option = this.document.createElement("option");
if(selection && selection === optionList[i]) {
option.setAttribute("selected", "true");
}
option.appendChild(this.document.createTextNode(optionList[i]));
select.appendChild(option);
}
// add a selection handler
$tw.utils.addEventListeners(select, [
{name: "change", handlerObject: this, handlerMethod: "handleChangeEvent"}
]);
// insert the <select> into the enclosing domNode
domNode.appendChild(select);
return domNode;
};
SelectWidget.prototype.getOptionList = function() {
var optionList = [];
if(this.filter) {
// process the filter into an array of tiddler titles
var defaultFilter = "[!is[system]sort[title]]";
optionList = this.wiki.filterTiddlers(this.getAttribute("filter", defaultFilter), this.getVariable("currentTiddler"));
} else if(this.list) {
// parse the given list into an array
optionList = $tw.utils.parseStringArray(this.list);
} else {
// process either the given, or the current tiddler as a list tiddler
optionList = this.wiki.getTiddlerList(this.tiddler, []);
if(optionList.length === 0){
// process the tiddler text as a list
optionList = this.wiki.getTiddlerText(this.tiddler).split("\n");
}
}
return optionList ? optionList : [];
};
SelectWidget.prototype.handleChangeEvent = function(event) {
// set the widget variable to inform the children
this.setVariable(this.setName, event.target.value, this.parseTreeNode.params);
// refresh this widget, and thereby the child widgets AND the enclosed content of this
// widget
this.refreshSelf();
return true;
};
exports.select = SelectWidget;
})();

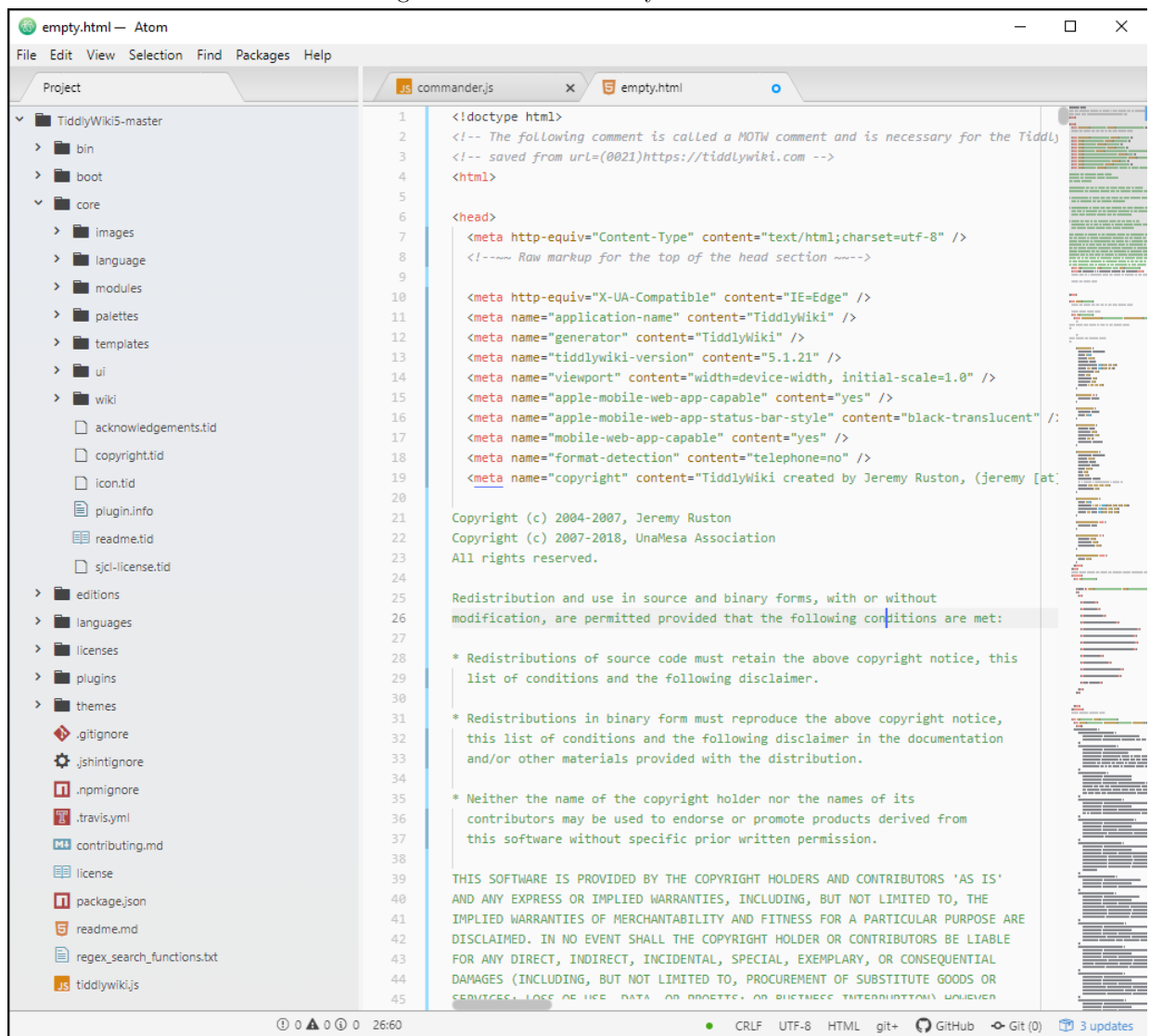
```



# 12 The Tiddlywiki internals

## 12.1 The Tiddlywiki architecture

Figure 12.1: The directory structure

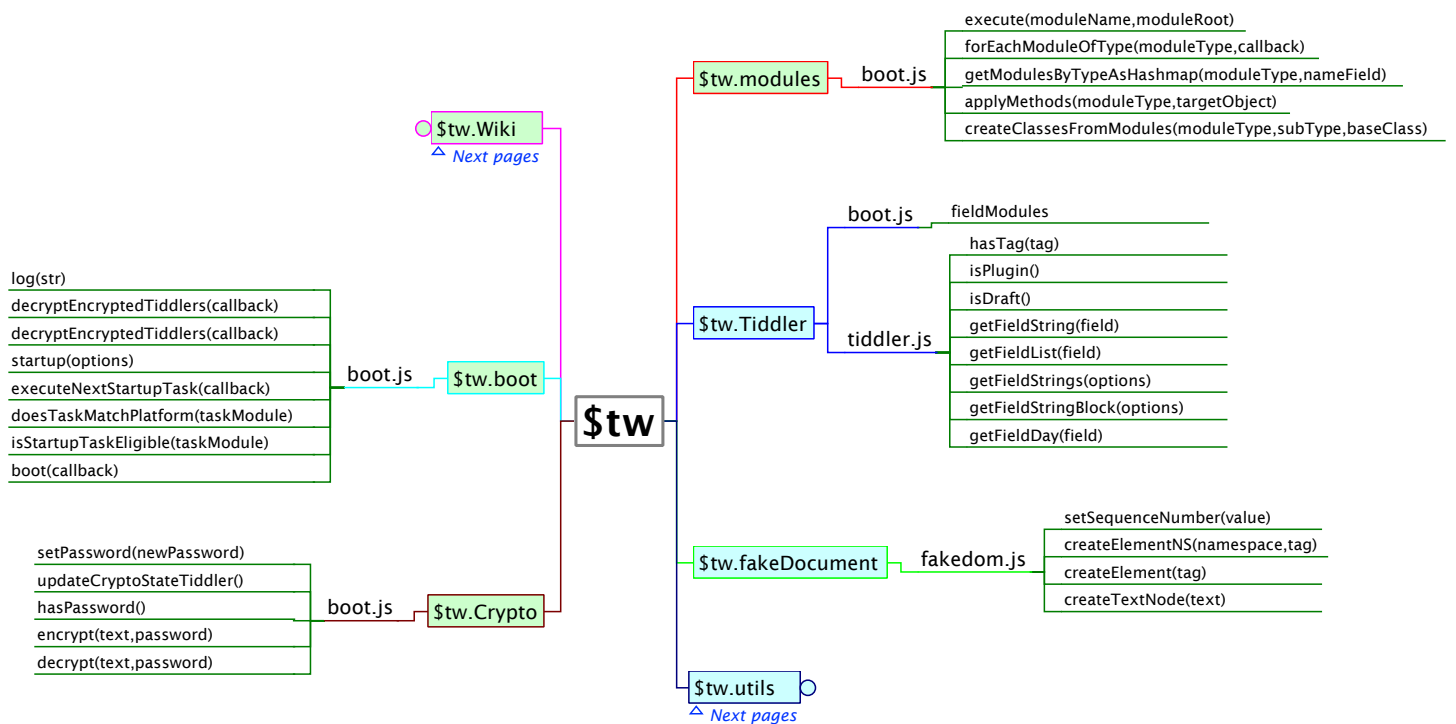


If you clone the [github Tiddlywiki repository](#) you will see all this files:

| Directory                             | Description                                                                                            |
|---------------------------------------|--------------------------------------------------------------------------------------------------------|
| bin                                   | Batch files                                                                                            |
| boot                                  | Javascript files with the Microkernel that loads Tiddlywiki                                            |
| core/images                           | The images included in Tiddlywiki                                                                      |
| core/languages                        | The languages. Initially, only en-GB                                                                   |
| core/modules                          | This files contains global data to be inserted into \$tw                                               |
| core/modules/commands                 | The commands listed in <a href="https://tiddlywiki.com/#Commands">https://tiddlywiki.com/#Commands</a> |
| core/modules/editor/engines           | Generic purpose Javascript libraries                                                                   |
| core/modules/editor/operations        | The text editor toolbar operations                                                                     |
| core/modules/filters                  | The code for all filters                                                                               |
| core/modules/indexsrs                 | Indexers for field and tags                                                                            |
| core/modules/infor                    | Publishes system information                                                                           |
| core/modules/macros                   | The code of some macros                                                                                |
| core/modules/parsers                  | Parsers for different content types                                                                    |
| core/modules/parsers/wikiparser/rules | Defines the wikirules                                                                                  |
| core/modules/savers                   | Savers handle different methods for saving files from the browser                                      |
| core/modules/server/routes            | Defines how individual URL patterns are gabdcked by the built-in HTTP server                           |
| core/modules/server/authenticators    | Defines how requests are authenticated by the built-in HTTP server.                                    |
| core/modules/startup                  | Startup functions                                                                                      |
| core/modules/storyviews               | Customize the animation and behaviour of list widgets                                                  |
| core/modules/upgraders                | Applies upgrade processing to tiddlers during an upgrade import                                        |
| core/modules/utlis                    | Add methods to \$tw.utlis                                                                              |
| core/modules/widgets                  | The javascript code for the widgets                                                                    |
| core/palettes                         | Colors included in the Control Panel, Appearance / Palette tab.                                        |
| core/templates                        | Some templates                                                                                         |
| core/ui                               | Some tiddlers                                                                                          |
| core/wiki                             | Some tiddlers                                                                                          |
| editions                              | Different editions of Tiddlywiki                                                                       |
| languages                             | All languages of Tiddlywiki                                                                            |
| licenses                              | Text of “TiddlyWiki5 Entity Contributor License Agreement”                                             |
| plugins                               | Several plugins                                                                                        |
| themes                                | Themes of Tiddlywiki (look in the Control Panel, Appearance / Theme tab)                               |

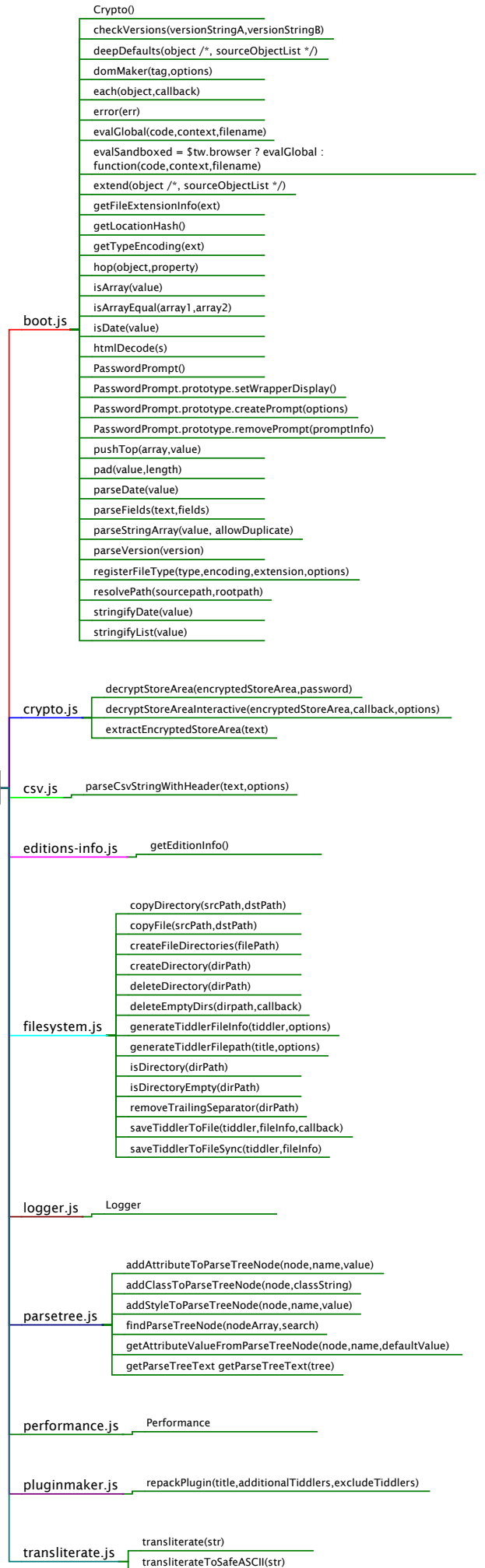
## 12.2 The \$tw object

In the next pages you will see the methods for all this \$tw tree





## \$tw.utils



compileFilter(filterString)  
filterTiddlers(filterString,widget,source)  
getFilterOperators()  
parseFilter(filterString)

filters.js

\$tw.wiki

wiki.js

addToHistory(title,fromPageRect,historyTitle)  
addToStory(title,fromTitle,storyTitle,options)  
addEventListener(type,listener)  
addIndexersToWiki()  
checkTiddlerText(title,targetText,options)  
clearCache(title)  
clearGlobalCache()  
clearTiddlerEventQueue()  
countTiddlers(excludeTag)  
deleteTextReference(textRef,currTiddlerTitle)  
dispatchEvent(type /\*, args \*/)   
doesPluginInfoRequireReload(pluginInfo)  
doesPluginRequireReload(title)  
enqueueTiddlerEvent(title,isDeleted)  
extractTiddlerDataItem(titleOrTiddler,index,defaultText)  
findDraft(targetTitle)  
forEachTiddler(/\* [options,]callback \*/)   
findListingsOfTiddler(targetTitle,fieldName)  
generateDraftTitle(title)  
generateNewTitle(baseTitle,options)  
getChangeCount(title)  
getCacheForTiddler(title,cacheName,initializer)  
getCreationFields()  
getGlobalCache(cacheName,initializer)  
getMissingTitles()  
getModificationFields()  
getOrphanTitles()  
getSizeOfTiddlerEventQueue()  
getSubTiddler(title,subTiddlerTitle)  
getTagMap()  
getTextReference(textRef,defaultText,currTiddlerTitle)  
getTiddlers(options)  
getTiddlerAsJson(title)  
getTiddlerBacklinks(targetTitle)  
getTiddlersAsJson(filter)  
getTiddlerData(titleOrTiddler,defaultData)  
getTiddlerDataCached(titleOrTiddler,defaultData)  
getTiddlerLinks(title)  
getTiddlerText(title,defaultText)  
getTiddlersWithTag(tag)  
importTiddler(tiddler)  
initParsers(moduleType)  
invokeUpgraders(titles,tiddlers)  
isBinaryTiddler(title)  
isDraftModified(title)  
isImageTiddler(title)  
isSystemTiddler(title)  
isTemporaryTiddler(title)  
makeTiddlerIterator(titles)  
makeTranscludeWidget(title,options)  
makeWidget(parser,options)  
parseText(type,text,options)  
parseTextReference(title,field,index,options)  
parseTiddler(title,options)  
readFiles(files,options)  
readFile(file,options)  
readFileContent(file,type,isBinary,deserializer,callback)  
removeEventListener(type,listener)  
renderText(outputType,textType,text,options)  
renderTiddler(outputType,title,options)  
search(text,options)  
setTextReference(textRef,value,currTiddlerTitle)  
setText(title,field,index,value,options)  
sortTiddlers(titles,sortField,isDescending,isCaseSensitive,isNumeric,isAlphaNumeric)  
sortByList(array,listTitle)  
setTiddlerData(title,data,fields)  
getTiddlerList(title,field,index)

## 12.3 More information inside Tiddlywiki

Look for all tiddlers tagged with “\$:/tags/ControlPanel/Advanced”:

| Tiddler                                | Description                                                                                  |
|----------------------------------------|----------------------------------------------------------------------------------------------|
| \$:/core/ui/ControlPanel/EditorTypes   | These tiddlers determine which editor is used to edit specific tiddler types.                |
| \$:/core/ui/ControlPanel/LoadedModules | Loaded tiddler modules linked to their source tiddlers                                       |
| \$:/core/ui/ControlPanel/Parsing       | Here you can globally disable/enable wiki parser rules.                                      |
| \$:/core/ui/ControlPanel/Stylesheets   | This is the rendered CSS of the current stylesheet tiddlers tagged with \$:/tags/Stylesheet. |
| \$:/core/ui/ControlPanel/TiddlerFields | This is the full set of TiddlerFields in use in this wiki.                                   |



# 13 Plugins

Written by Mohammad Rahmani

## 13.1 Introduction

Sistan is a tutorial for creating plugins in Tiddlywiki. It was written as a small chapter to be used in book developed by Luis Gonzalez.

To get the online latest version visit: <http://sistan.tiddlyspot.com>

### 13.1.1 Purpose

This tutorial shows how one can create a simple Tiddlywiki plugin. For demo purpose a task manager has been developed.

### 13.1.2 Task manager plugin

For pedagogical purpose the task manager plugin has been developed in three levels:

- Level 1: primary, very simple with minimum features and uses Tinka for making plugin in browser
- Level 2: intermediate, has more features and uses intermediate scripting skills and developed using Tinka
- Level 3: advanced, is professional one uses Node.js for creating the plugin.

### 13.1.3 Content of the plugin

This plugin contains four tiddlers:

- create-new-task-button: it defines the macro for creating and listing the tasks.
- list-tasks-finished.- List of finished tasks.
- list-tasks-unfinished.- List of unfinished tasks.
- Task Manager Level 1: Rome.- The task list.

## 13.2 Level 1: Primary

### 13.2.1 Button to Create New Task

#### A button

It is important to create new task in an easy and quick way. To do this a \$button widget has been used here.

The \$button will

- Create a new tiddler to store the task as its title
- Add the task tag to show tiddler is a task

## The script

To implement the create new task button the following code has been developed:

```
\define create-new-task ()
 <$action-sendmessage $message="tm-new-tiddler " title="new task " tags="
 task ">/>\end

<$button actions=<<create-new-task>>>New task</button>
```

## Description

This script has two parts: i. a \$button to create the user interface and lets user click it and create the new task and ii. a macro to perform the actions.

- The \$button create a button with New task label
- The \$button on click calls create-new-task macro to perform the actions.
- The macro create-new-task uses the \$action-sendmessage with tm-new-tiddler to create a new tiddler. This new tiddler stores the task as its title. It also set tags as task means the new tiddler will be tagged with task
- The new tiddler will be shown in screen under focus and user can simply enters the task title in the new tiddler title.

The complete code of create new task is given in create-new-task-button tiddler.

```
\define create-new-task ()
 <$action-sendmessage $message="tm-new-tiddler " title="new task " tags="
 task "text=<<now "Today is DDth, MMM YYYY">>/>
\end

<$button actions=<<create-new-task>>>New task</button>
```

## 13.2.2 Create a List of Unfinished Tasks

### 13.2.2.1 List tasks

The first step is to create a list of tasks. To do this a \$list widget with proper filter is used. Each task is stored in a tiddler tagged with task. When a task is done, it also gets the done tag.

```
<$list filter=" [tag [task] ! tag [done]] ">
</list>
```

## Description

- The \$list widget list tiddler based on the given filter
- The [tag[task]!tag[done]] selects all tiddlers tagged with task but NOT with done

## Display task title and checkbox

The task list needs to have

- A checkbox to be checked when
- The task is done the title of task to be shown

So the above code is completed as below

```
<$list filter=" [tag [task] ! tag [done]] ">
 <$checkbox tag=done/>
 <$link to=<<currentTiddler>>>{!! title }></$link>

</$list>
```

### Description

- The \$checkbox widget shows a square box that is ticked (checked) when activated. It is bound to tags and when checked, adds the done tag to current tiddler.
- The \$link widget creates a link to task tiddler, on click it navigates to desired tiddler.

### Empty message

It is semantic to show an empty message when the task list is empty. That means all tasks have been done.

The complete code has been given below

```
<$list filter=" [tag [task] ! tag [done]] " emptyMessage="You are all done!
 Nothing in task list ">
 <$checkbox tag=done/>
 <$link to=<<currentTiddler>>>{!! title }></$link>

</$list>
```

### Description

The emptyMessage is a \$list attribute and is shown when the list is empty. Here the message "You are all done! Nothing in task list" will be shown.

The complete code of displaying list of tasks is given in list-tasks-unfinished tiddler.

## 13.2.3 Create a List of Finished Tasks

### List of tasks done

The second step is to create a list of task have been done. To do this a \$list widget with proper filter is used. Each task done has the task and done tags simultaneously.

The blow code shows how finished tasks are displayed

```
<$list filter=" [tag [task] tag [done]] " emptyMessage="Nothing done yet! ">
 <$checkbox tag=done/>
 <~<$link to=<<currentTiddler>>>{!! title }></$link>~

</$list>
```

### Description

- The \$list widget lists tiddlers based on the given filter
- The [tag[task]tag[done]] selects all tiddlers tagged simultaneously with task AND done

The body of \$list widget has two parts:

- The \$checkbox widget shows a square box that is ticked (checked). It is bound to tags and when unchecked, removes the done tag from current tiddler. The \$link widget creates a link to task tiddler, on click it navigates to desired tiddler.
- The \$link has been wrapped in ~...~ to format the output as strikethrough text, which implies the task has been done.

### Empty message

It is semantic to show an empty message when the finished tasks list is empty. That means no task has been finished yet.

The `emptyMessage` is a `$list` attribute and is shown when the list is empty. Here the message "Nothing done yet!" will be shown.

The complete code of displaying list of tasks is given in `list-tasks-finished` tiddler.

## 13.2.4 Put all Sections Together

### A complete script

This section describes how to put all elements of task manager together to have a working application.

The task manager has three elements

1. A button to create a new task. This has been developed in `create-new-task-button`
2. A list to display unfinished tasks, which has been developed in `list-tasks-unfinished`
3. A list to display finished tasks, which has been developed in `list-tasks-finished`

### A User Interface

A user interface in brief a UI is used to display all elements of task manager in a simple order to let user easily interact with application.

Here we also use a name for our application. We called this Level 1: Task Manager, Rome. So, Rome is the name of our application.

To create the interface and put all elements together, a new tiddler is required as below

**title:** Task Manager Level 1: Rome

**text:** all sections of task manager in proper order as below

- the button to create new tasks
- the list of unfinished tasks
- the list of finished tasks

So, at the end the body of tiddler is like below

```
! My daily tasks
```

```
{{create-new-task-button}}
```

```
!! Unfinished tasks
```

```
> {{list-tasks-unfinished}}
```

```
!! Finished tasks
```

```
> {{list-tasks-finished}}
```

Navigate to Task Manager Level 1: Rome to see the working application.

## 13.2.5 Bundle as a Plugin

There are several different methods to bundle an application as a plugin. These methods are based on

1. Creating and packaging in browser
2. Creating and using the Node.js

Here the first method which is simpler is used. To learn how Tinka works see [Packaging with Tinka](#). A short description is also given in this [wiki Using Tinka to Create Plugins](#)

### 13.2.6 The Tinka process

is a Control Panel extension that helps with creating and modifying plugins in the browser. You can get the extension visiting: <https://tinkaplugin.github.io/#%24%3A%2Fplugins%2Fahahn%2Ftinka>.

The steps to create a new plugin are:

1. From control panel, click on Tinka Plugin Management
2. Next click on the Create new Plugin tab
3. Select plugin type e.g.
  - a) plugin, or
  - b) theme
4. Set the plugin path: This is the name of the tiddler where the finished plugin is going to be stored in. ~TiddlyWiki uses a special naming theme for these, so you will mostly find that the actual tiddlers that contain a plugin are named in this scheme:
  - a) \$:/plugins/myOrganisation/pluginName
5. Set the author: Name of the plugin author
6. Set the source: Website or URL of the plugin, also the place where updates are found
7. Set the dependents: List of plugins this plugin depends on (usually empty, but e.g. \$:/core)
8. Set the list: List of tiddlers contained in the plugin, that will serve as readme tiddlers, when inspecting a plugin via the control panel. (e.g. \$:/plugins/myOrganization/pluginName/readme)
9. Set the version: Version of the plugin with format of x.x.x e.g. 1.0.0
10. Set the core-version: Usually the minimal ~TiddlyWiki version your plugin requires in order for it to work (e.g. >=5.1.8)
11. Set the description: For plugins, this contains the plugin title that is shown in the control panel
12. Set the name: For themes, this contains the theme title that is shown in the control panel
13. Add the tiddlers to plugin in Step 2: Add Tiddlers
14. Package the plugin
15. Save and reload

### 13.2.7 Distributing Rome Plugin

The simplest way is to drag and drop the plugin tiddler from source wiki to target wiki. So for Rome plugin you can

#### First method


- Goto \$:/ControlPanel
- Open the Plugins tab and drag and drop the Rome a simple task manager to target wiki







#### Second method

- Create a link to plugin in a readme or home tiddler for Rome it is \$:/plugins/kookma/rome
- Drag and drop the plugin tiddler link to the target wiki

There are more advanced methods will be discussed in next parts of creating Task Manager Plugin.

Figure 13.1: Creating a plugin with tinka


\$/ControlPanel

Info Appearance Settings Saving Plugins Keyboard Shortcuts Tinka Plugin Management

Installed Create a new Plugin Archive

## Create New Plugin

*Usage: Enter the necessary metadata for your plugin and use the Filter selection below to pick the tiddlers that should be added to the plugin. After selecting the tiddlers, press 'Package Plugin'. Refer to the [Documentation](#) for further help.*

Step 1: Enter Metadata

**Plugin-Type:**  ▼

|                      |                                                                                                                   |               |                                                           |
|----------------------|-------------------------------------------------------------------------------------------------------------------|---------------|-----------------------------------------------------------|
| <b>Plugin Path:</b>  | <input type="text" value="\$:/plugins/"/> <input type="text" value="kookma"/> / <input type="text" value="rome"/> |               |                                                           |
| Author:              | <input type="text" value="Mohammad Rahmani"/>                                                                     | Source:       | <input type="text" value="http://sistan.tiddlyspot.com"/> |
| Dependents:          | <input type="text"/>                                                                                              | List:         | <input type="text" value="e.g. readme usage"/>            |
| <b>Plugin Title:</b> | <input type="text" value="Rome a simple task manager"/>                                                           |               |                                                           |
| Version:             | <input type="text" value="0.1.0"/>                                                                                | Core-Version: | <input type="text" value="5.1.21"/>                       |

**bold** = *required field*

Step 2: Add Tiddlers

## 13.2.8 Packaging Rome as a Plugin

### Create the plugin

In this tutorial Tinka plugin packager is used. After you make sure Tinka is installed in your Tiddlywiki and working fine, then follow the below steps to create the Rome plugin.

### Step One

Goto \$:/ControlPanel, open "Tinka Plugin Management" tab and then click on "Create a new plugin". Fill in the inputs as figure below. Note that the bold items are the required ones, without them Tinka will fail to create your plugin.

### Step Two

In this step, add all tiddlers required for creating plugin. In this Level 1: Task Manager example, all these tiddlers are tagged with Rome, so they are:

1. Create-new-task-button
2. list-tasks-finished
3. list-tasks-unfinished
4. Task Manager Level 1: Rome

To add these tiddlers to your Rome plugin, continue with step two. Use the Filter search with [tag[Rome]] to list all tiddlers included in Rome plugin. Then click the provided checkbox to add these tiddlers to plugin.

The Figure 13.2 shows how step two looks like.

Figure 13.2: Tinka step two

Default search | Filter search

Enter Filterstring to select tiddlers:  X

- ☒ Task Manager Level 1: Rome
- ☒ list-tasks-unfinished
- ☒ list-tasks-finished
- ☒ create-new-task-button

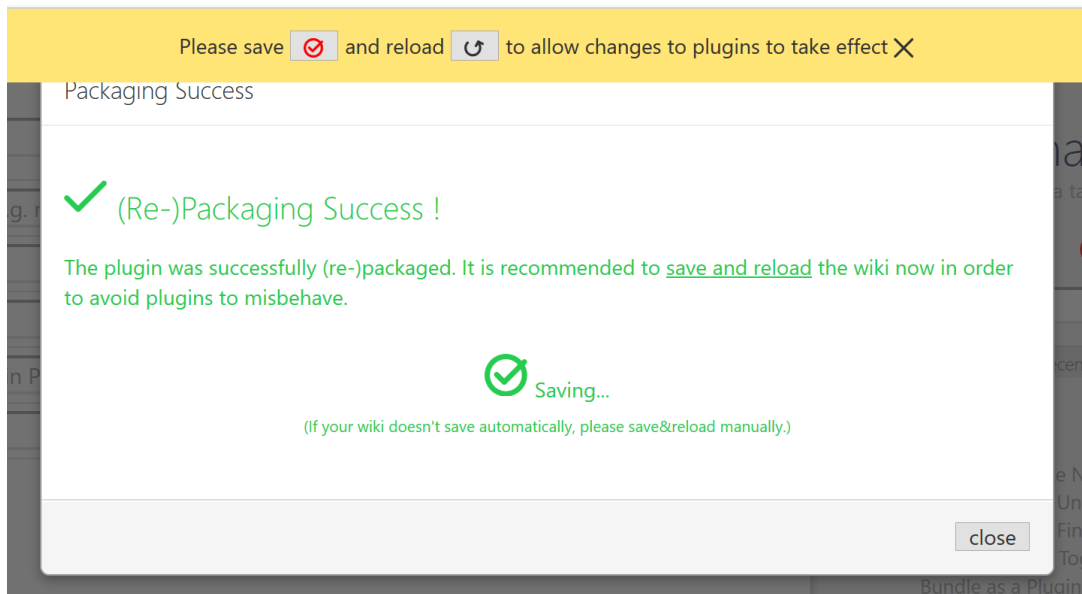
**Added Tiddlers**

- ☒ create-new-task-button
- ☒ list-tasks-finished
- ☒ list-tasks-unfinished
- ☒ Task Manager Level 1: Rome

### Final step

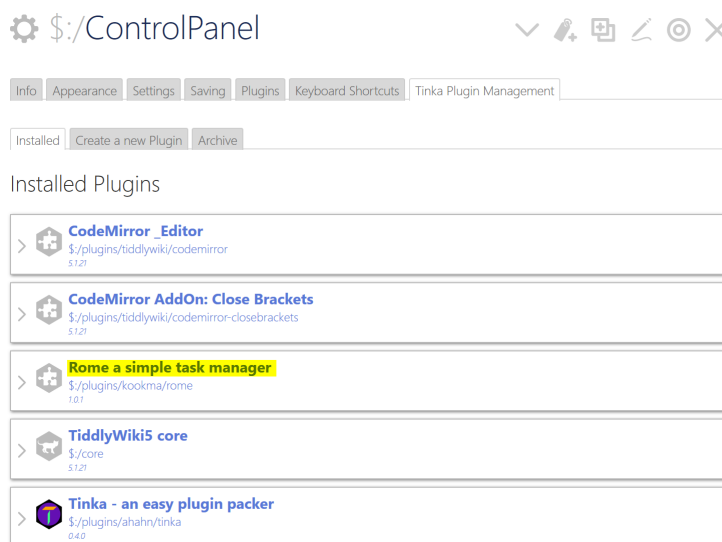
The next step is to click the Package plugin shown in Figure 13.2. After that Tinka will show a message as Figure 3. indicating the plugin has created successfully. Save the wiki and reload (See Figure 13.3).

Figure 13.3: Tinka final step



To make sure the plugin has been created, open the `$/ControlPanel`, goto Plugins tab and find the new plugin in the list of installed plugin. If everything goes well you should see the new Rome plugin in the list as below.

Figure 13.4: Tinka installed



### 13.2.9 Using Rome Plugin

After installing the Rome plugin in your Tiddlywiki, you can simply embed its interface whenever you like.

As an example here, a new tiddler called My First Plugin: Task manager was created with the below content:



```
{{Task_Manager_Level_1: Rome}}
```

Use task manager in sidebar

If you like to use the Rome task manager in the sidebar do as following

1. Open My First Plugin: Task manager
2. Tagg it with \$:/tags/SideBar
3. Add the caption field with My Tasks as value



# 14 Tiddlywiki internationalization

Written with the help of: Eshka from  
google groups

## 14.1 Key points

- Tiddlywiki current language can be set in the Basics sub-panel of info panel in the ControlPanel.
- New languages can be installed with the Plugin Manager of the ControlPanel
- Tiddlywiki manage internationalization by some shadow tiddlers.
- Instead of hard-code the text of the Tiddlywiki elements it transclude this language tiddlers.
- We can all contribute to the translation of Tiddlywiki.
- We can generate international plugins and tiddlywikis in the same way: create your own language tiddlers for the plugin. And transclude this tiddlers in your plugin in the same way that the new tiddler button.

## 14.2 Languages internals

Figure 14.1: The language shadow tiddlers

|                                                          |                                                           |
|----------------------------------------------------------|-----------------------------------------------------------|
| <code>{{{ [all[shadows]prefix[\$:/language/] ]}}}</code> | <code>\$:/language/Buttons/AdvancedSearch/Caption</code>  |
|                                                          | <code>\$:/language/Buttons/AdvancedSearch/Hint</code>     |
|                                                          | <code>\$:/language/Buttons/Cancel/Caption</code>          |
|                                                          | <code>\$:/language/Buttons/Cancel/Hint</code>             |
|                                                          | <code>\$:/language/Buttons/Clone/Caption</code>           |
|                                                          | <code>\$:/language/Buttons/Clone/Hint</code>              |
|                                                          | <code>\$:/language/Buttons/Close/Caption</code>           |
|                                                          | <code>\$:/language/Buttons/Close/Hint</code>              |
|                                                          | <code>\$:/language/Buttons/CloseAll/Caption</code>        |
|                                                          | <code>\$:/language/Buttons/CloseAll/Hint</code>           |
|                                                          | <code>\$:/language/Buttons/CloseOthers/Caption</code>     |
|                                                          | <code>\$:/language/Buttons/CloseOthers/Hint</code>        |
|                                                          | <code>\$:/language/Buttons/ControlPanel/Caption</code>    |
|                                                          | <code>\$:/language/Buttons/ControlPanel/Hint</code>       |
|                                                          | <code>\$:/language/Buttons/CopyToClipboard/Caption</code> |
|                                                          | <code>\$:/language/Buttons/CopyToClipboard/Hint</code>    |
|                                                          | <code>\$:/language/Buttons/Delete/Caption</code>          |
|                                                          | <code>\$:/language/Buttons/Delete/Hint</code>             |
|                                                          | <code>\$:/language/Buttons/Edit/Caption</code>            |
|                                                          | <code>\$:/language/Buttons/Edit/Hint</code>               |

The value of the current language is stored into the system tiddler: “\$/language”. Tiddlywiki uses ISO language code abbreviations. for instance, “en-GB” for British English.

Tiddlywiki implements many languages. And it provides a simple way of making new translations: it has a tiddlywiki that help us in this task, a tiddlywiki [edition for translators](#). Here we find instructions and it provides a way to perform them easily.

But, how manage Tiddlywiki this internationalization? Instead of use hard-coded names of all elements of Tiddlywiki, it uses the content of some shadow tiddlers. They are tiddlers whose title begins with “\$/language/”.

Let’s look an example: the new tiddler button.

## 14.3 The new tiddler button

We can learn about the language process by analyzing the code for the “new tiddler” button. This code is inside the tiddler “\$/core/ui/Buttons/new-tiddler”:

```
<$button actions={{{$:/core/ui/Actions/new-tiddler}} } tooltip={{$/language/
Buttons/NewTiddler/Hint}} aria-label={{$/language/Buttons/NewTiddler/
Caption}} class=<<tv-config-toolbar-class>>>

 <$list filter="[<tv-config-toolbar-icons>match[yes]]">
 {{{$/core/images/new-button}}}
 </$list>

 <$list filter="[<tv-config-toolbar-text>match[yes]]">

 <$text text={{$/language/Buttons/NewTiddler/Caption}}/>

 </$list>

</$button>
```

You can find here references to these language tiddlers:

- Tooltip: instead of an english string like “Create a new tiddler” it transclude the tiddler “\$/language/Buttons/NewTiddler/Caption”

- Text button: instead of “new tiddler”, it transclude the tiddler “\$/language/Buttons/NewTiddler/”

You can list all this language tiddlers with this transclusion: `{{[all [shadows] prefix[$:/language/] ]}}`. Look at Figure 14.1.

So if you want a version of Tiddlywiki in your language you just have to replace these language tiddlers. Tiddlywiki will show all this elements in the correct language.

## 14.4 Writing international wikis and plugins

### 14.4.1 Tiddlywiki support to internationalization

When you download an empty Tiddlywiki its language is English. Later you can download more language plugins. To view your wiki in the new language you have to go to ControlPanel > Information > Basic and change it in the language selector.

Doing so you change the “\$/language” tiddler, the place where Tiddlywiki stores the current language. At the beginning it contains the value “\$/languages/en-GB”. If you install the Spanish language and changes it in the Control panel, the new value will be “\$/languages/es-ES”

Tiddlywiki translates its elements using the translation mechanism. Tiddlywiki core doesn’t currently support translations for plugins, That doesn’t stop you making a plugin that has multiple translations, but it means that you have to roll your own, including building a macro to translate all the plugin messages.

### 14.4.2 Translating messages

Translating information is displayed in the plugin manager (ControlPanel). For each language plugin, the plugin manager displays some information in different tabs: readme, usage, content... This information can be displayed in the wiki current language.

For instance, to translate the “readme” part for “<author\_name>/<plugin\_name>” plugin, you need to create the translation in the following tidder: “\$/plugins/<author\_name>/<plugin\_name>/<lang\_code>/readme” where <lang\_code> is the ISO language name, for instance “fr-FR” for french from France.

Tip: you can check the language code of installed languages in the Plugins / languages panel of the ControlPanel.

Tiddlywiki will automatically display the translated information corresponding to selected “Current language” if this information exists.

### 14.4.3 Translating messages in you plugin

This section describes an option to translate messages in your plugin.

#### Step 1

Create the following tiddler and replace the text in bracket by your name, plugin name and for <an-pn> by a short prefix representing your <author\_name>-<plugin\_name>.

title: \$:/ plugins/<author\_name>/<plugin\_name>

tags: \$:/tags/Macro

text:

```

\define<an-pn>-base(){$:/plugins/<author_name>/<plugin_name>/lang

\define<an-pn>-lingo(txt)
 <$transclude_tiddler="$(<an-pn>-base)$/$(<languageTitle>$/$txt$"mode="
 inline">
 <$transclude_tiddler="$(<an-pn>-base)$/$txt$"mode="inline">
 Missing_tiddler_$(<an-pn>-base)$/txt
 </$transclude>
</$transclude>
\end

```

## Step 2

Store each text message in English in tiddlers named with a short message description according to this pattern:

```
$:/plugins/<author_name>/<plugin_name>/lang/<short message description>
```

## Step 3

Store translated texts in tiddlers named:

```
$:/plugins/<author_name>/<plugin_name>/lang/<lang_code>/<short message description>
```

## Step 4

When you need to display a text message, use the macro defined in step 1 like this:

```
<< <an-pn>-lingo "<short message description>" >>
```

# 15 Examples

## 15.1 A simple to-do list

### 15.1.1 Introduction

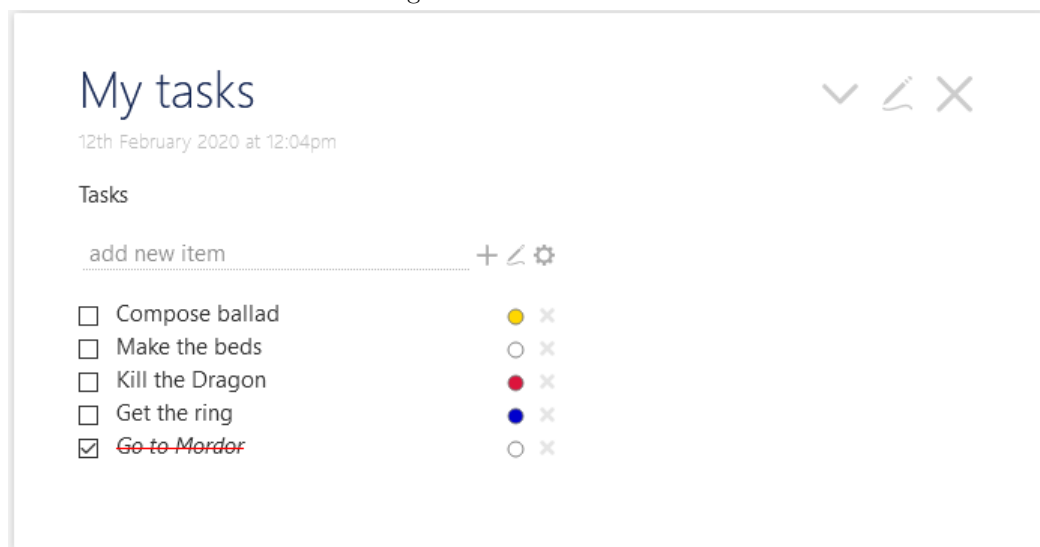
We need a more complex task manager. These are the features we need:

- It will support multiple lists.
- We need a way to prioritize the tasks.
- We need to archive complete tasks.

We will do this by analyzing the wonderful plugin Todolist written by Mohammad Rahmani. Todolist is a Pure tiny Tiddlywiki plugin for creating simple Todolist. The project code and demo are in [GitHub demo page](#).

### 15.1.2 The Todolist plugin

Figure 15.1: The task list



To create the default list you can write: `<<todolist-ui caption:"Tasks:" width:"" base:"">>` in the task tiddler. You can manage multiple list changing the “base” parameter.

Inside the “My tasks” tiddler you will find the caption of the list at the top, “Tasks” and below it there is:

1. An “add new item” space to write the tasks.
2. The buttons on the right:
  - a) A button for adding the task

- b) Another button for editing the tasks below it.
  - c) A last button to show the configuration options for this list.
- 3. The unfinished tasks:
  - a) The checkbox to mark the task as done.
  - b) The text of the task.
  - c) A circle with the priority color.
  - d) The delete button.
- 4. The finished tasks.

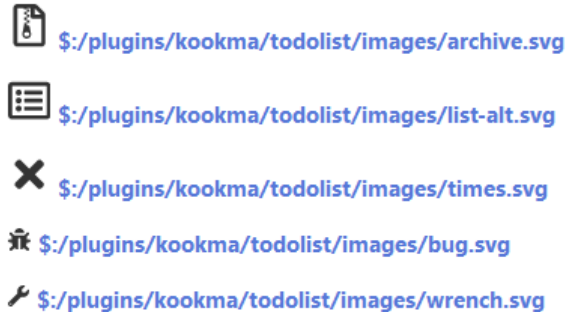
### 15.1.3 The Todolist tiddlers

You know that everything in Tw is inside the tiddlers, so we will analyze the tiddlers we need:

- Image tiddlers: We need some icons for the buttons.
- Information: In this tiddlers we will store some plugin information.
- Templates: Maybe we need some templates.
- Styles: To change its appearance we will create CSS tiddlers.
- Macros. The most important part of the plugin.

#### 15.1.3.1 Icons

Figure 15.2: The Todolist icons



#### 15.1.3.2 Information tiddlers

What information we need to show in a plugin? Here you find it:

- A tiddler for the history. Here you find all previous versions of the plugin: `$:/plugins/kookma/todolist/history`
- A tiddler for the license: `$:/plugins/kookma/todolist/license`
- A tiddler for the internal tiddlers of the plugin: `:$:/plugins/kookma/todolist/internals`
- The readme tiddler with its usage: `$:/plugins/kookma/todolist/readme`

#### 15.1.3.3 The styles

We find two tiddlers for the styles:

- `$:/plugins/kookma/todolist/styles/main.css` : The main css styles.
- `$:/plugins/kookma/todolist/styles/other.css` : Other css styles.



#### 15.1.3.4 Templates

We find only one template for the archived tasks: `$/plugins/kookma/todolist/template/archive-list`

#### 15.1.3.5 The macros

It is essential to facilitate the use of the plugin: a single macro call with little parameters: `<<todolist-ui caption:"Tasks:" width:"" base:"">>`

- `todolist-ui`: the name of the Todolist main macro contained in the tiddler `$/plugins/kookma/todolist/macros/ui` ! `! todolist-ui(caption:"A plain todo list", width:"100%" base:"base")` .
- `caption`: The tiddler will show this caption on the top of the list. You can add Wikitext format characters in this caption. For example. `caption:"__The ring tasks__"` will underscore it.
- `width`: the width of the task list: `width:"50%"`
- `base`: the name of the list. Each base has its own archived tiddlers and tasks. So when you add a task it will belong only to this list. If you do not add a base Todolist will use the default list.

The `<<todolist-ui>>` macro will call the other macros.

In the last page of this chapter there is a map with the tiddler macros and the macros it contains.

### 15.1.4 The Todolist main macros

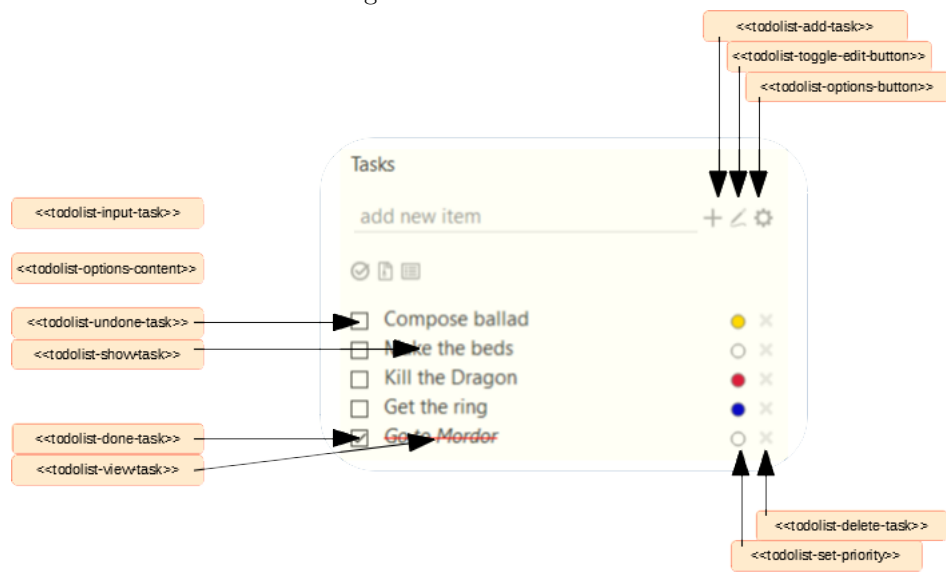
This is the main loop inside the `$/plugins/kookma/todolist/macros/ui` tiddler used to draw the list:

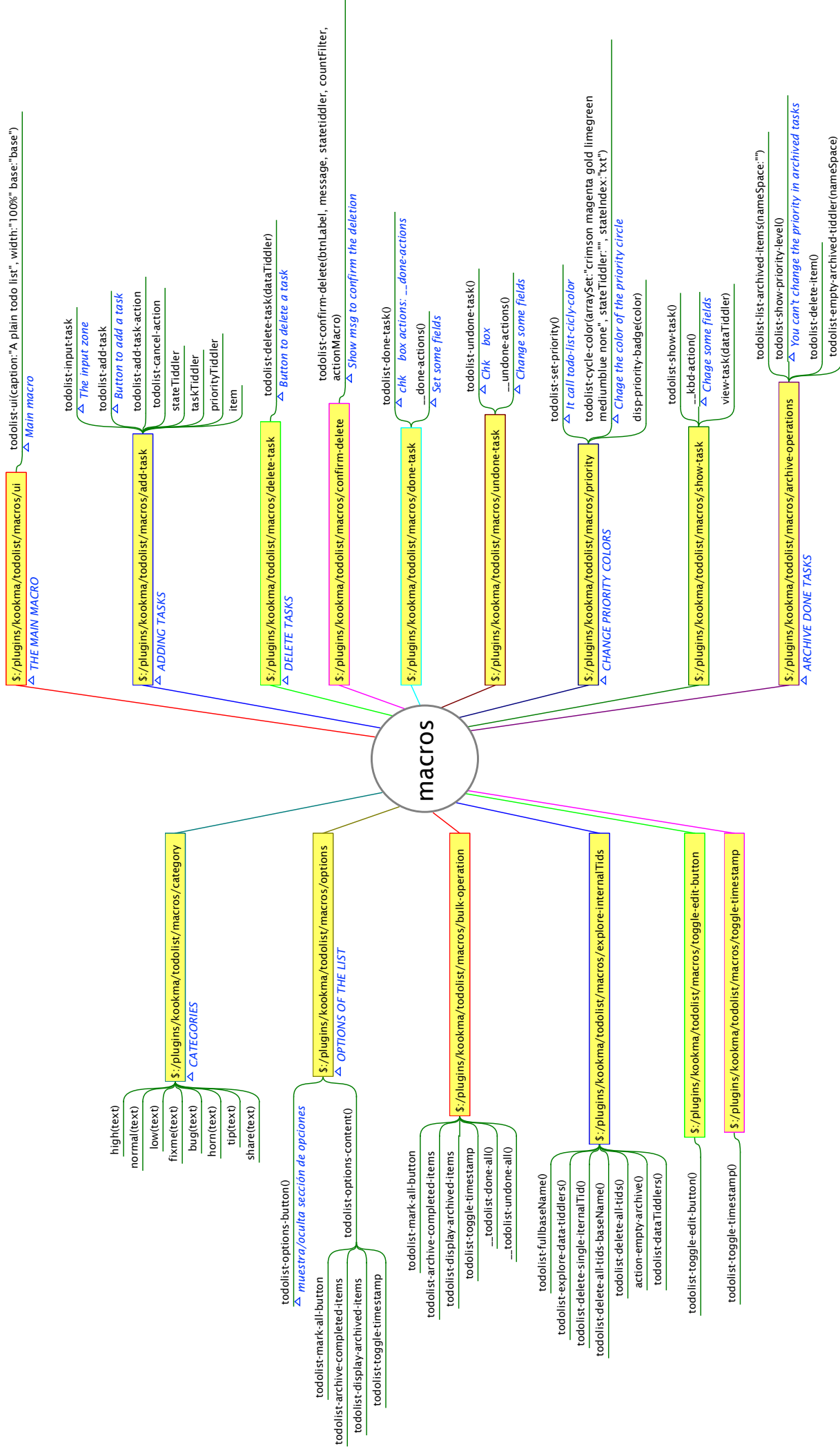
```
<$list filter="[<taskTiddler>indexes []! sort []]" variable="item">
 <div class="kk-todolist-row"> _____
 <div class="kk-todolist-done"><<todolist-done-task>></div> _
 <div class="kk-todolist-desc"><<todolist-show-task>></div>
 <div class="kk-todolist-priority"><<todolist-set-priority>></div>
 <div class="kk-todolist-delete">
 <$macrocall $name="todolist-delete-task" dataTiddler=<<taskTiddler>>/>
 </div>
 </div>
</$list>

<$list filter="[<doneTiddler>indexes []! sort []]" variable="item">
 <div class="kk-todolist-row">
 <div class="kk-todolist-done"><<todolist-undone-task>></div>
 <div class="kk-todolist-desc">
 <$transclude tiddler=<<doneTiddler>> index=<<item>>/>
 </div> _____
 <div class="kk-todolist-priority"><<todolist-set-priority>></div>
 <div class="kk-todolist-delete">
 <$macrocall $name="todolist-delete-task" dataTiddler=<<doneTiddler>>/>
 </div>
 </div>
</$list>
```

In the Figure 15.3 you can see which macro takes care of each element. We recommend the reader to analyze the rest of the operation of this great plugin. This is a task that will serve you later.

Figure 15.3: The macros





**15.2 more projects**

## 16 The people of Tiddlywiki

I want to add a chapter with all the people under this project. From Jeremy though the core developers to the wiki writers.

And the project, its philosophy and the future of tiddlywiki.



# 17 Glossary

**History:** it stores all tiddlers opened in the current session.

**Parser.** A Parser is provided by a module with module-type: parser and is responsible to transform block of text to a parse-tree. The core plug-in provides a recursive descent WikiText parser which loads it's individual rules from individual modules. Thus a developer can provide additional rules by using module-type: wikirule. Each rule can produce a list of parse-tree nodes. A simple example for a wikirule producing a <hr> from --- can be found in horizrule.js

**Pragma:** A pragma is a special component of WikiText that provides control over the way the remaining text is parsed.

**Story river:** Zone of the Tiddlywiki page for showing the tiddlers

**Tiddler:** Little notes that integrate Tiddlywiki

**Tags:** Words to classify the tiddlers





## 18 Resources

- Tiddlywiki for developers, Jeremy Ruston. [Tiddlywiki for developers](#).
- The [Playground](#), by Ton Gerner.
- RegExp in Tiddlywiki, by Mohammad Rahmani. <http://tw-regexp.tiddlyspot.com/>
- Tiddlywiki coding , by Chris Hunt. [Tiddlywiki coding notes](#). The link of the first version