

# 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**

Luis Javier González Caballero

December 12, 2019



# Acknowledgements

This book would not have been possible without the help of people from the [tiddlywiki google group](#).  
Special thanks to:

- Ton Gerner for his help with css classes.
- Riz for his help with templates.
- Mohammad Rahmani for his wonderful wikis and plugins.
- Chris Hunt for his [Tiddlywiki coding notes](#).



# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                          | <b>11</b> |
| 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    | Tiddlers                                     | 14        |
| 1.5.3    | The Story River                              | 15        |
| 1.5.4    | Tags   | 15        |
| 1.5.5    | Fields                                       | 16        |
| 1.5.6    | Text format                                  | 16        |
| 1.5.7    | Transclusion                                 | 16        |
| 1.5.8    | Templates                                    | 16        |
| 1.5.9    | Filters                                      | 16        |
| 1.5.10   | Macros and widgets                           | 16        |
| 1.5.11   | Mechanism                                    | 17        |
| 1.5.12   | Lists  | 17        |
| <b>2</b> | <b>Using tiddlywiki</b>                      | <b>19</b> |
| 2.1      | The philosophy of Tiddlywiki                 | 19        |
| 2.2      | Planning your wiki                           | 19        |
| 2.3      | Organizing microcontent with links           | 19        |
| 2.4      | Organizing tiddlers with tags                | 19        |
| 2.5      | Personalize behavior with macros and widgets | 19        |
| 2.6      | Importing content                            | 19        |
| <b>3</b> | <b>Loading tiddlywiki in the browser</b>     | <b>21</b> |
| 3.1      | Key points                                   | 21        |
| 3.2      | Introduction                                 | 22        |
| 3.3      | Architecture                                 | 22        |
| 3.4      | Tiddlers as Basic Coding Elements            | 23        |
| <b>4</b> | <b>Components of the tiddlywiki screen</b>   | <b>25</b> |
| 4.1      | Key points                                   | 25        |
| 4.2      | The TW whole screen                          | 26        |
| 4.2.1    | The story river                              | 26        |
| 4.2.2    | The right sidebars                           | 27        |
| 4.3      | Drawing its interface                        | 27        |
| <b>5</b> | <b>Customize the TW screen</b>               | <b>29</b> |
| 5.1      | Key points                                   | 29        |
| 5.2      | Introduction                                 | 30        |
| 5.3      | PageTemplate                                 | 30        |
| 5.4      | ViewTemplate                                 | 31        |
| 5.4.1    | Example                                      | 31        |
| 5.5      | EditTemplate                                 | 32        |
| 5.6      | Formating with CSS                           | 32        |
| 5.6.1    | Example                                      | 33        |
| 5.7      | Create a loading message                     | 34        |

|          |  |           |
|----------|--|-----------|
| 5.8      | Create a new button at the toolbars      | 34        |
| 5.8.1    | How to do it                             | 34        |
| 5.9      | Add a new global keyboard shortcut       | 35        |
| 5.9.1    | How to do it                             | 35        |
| 5.10     | Creating a left menu                     | 36        |
| 5.10.1   | Configure the story river position       | 36        |
| 5.10.2   | Create the tiddler menu                  | 37        |
| 5.10.3   | Create the styles for the left menu      | 37        |
| 5.10.4   | Create the entries for the left menu     | 37        |
| 5.10.5   | The next step                            | 37        |
| <b>6</b> | <b>Inside a tiddler</b>                  | <b>39</b> |
| 6.1      | Key points                               | 39        |
| 6.2      | Types of tiddlers                        | 40        |
| 6.2.1    | The tiddler type field                   | 40        |
| 6.3      | Normal tiddler                           | 42        |
| 6.4      | Tag tiddler                              | 42        |
| 6.5      | Alert tiddler                            | 43        |
| 6.6      | Dictionary tiddler                       | 43        |
| 6.7      | JSON tiddlers                            | 44        |
| 6.8      | CSS tiddlers                             | 44        |
| 6.9      | Template tiddlers                        | 44        |
| 6.10     | Other system tiddlers                    | 44        |
| 6.10.1   | <code>\$/config/EmptyStoryMessage</code> | 44        |
| 6.11     | Adding macros and widgets                | 45        |
| <b>7</b> | <b>Filters</b>                           | <b>47</b> |
| 7.1      | Key points                               | 47        |
| 7.2      | What is a filter?                        | 48        |
| 7.3      | The filter expression                    | 48        |
| 7.4      | Using filters                            | 49        |
| 7.5      | The filter basic step                    | 50        |
| 7.6      | Regular expressions                      | 51        |
| 7.6.1    | Writing regular expressions              | 51        |
| 7.6.2    | Using regular expressions                | 53        |
| <b>8</b> | <b>Macros</b>                            | <b>55</b> |
| 8.1      | Introduction                             | 55        |
| 8.2      | Variables and parameters                 | 55        |
| 8.2.1    | Defining variables                       | 55        |
| 8.2.2    | Parameters                               | 55        |
| 8.2.3    | Using variables and parameters           | 56        |
| 8.3      | Basic macros included                    | 56        |
| 8.4      | Writing your own macros                  | 56        |
| 8.5      | Using macros                             | 56        |
| <b>9</b> | <b>Widgets</b>                           | <b>57</b> |
| 9.1      | Introduction                             | 57        |
| 9.2      | The checkbox widget                      | 57        |
| 9.3      | The \$list widget                        | 58        |
| 9.4      | \$edit-text widget                       | 59        |
| 9.5      | \$button widget                          | 60        |
| 9.6      | Combining widgets                        | 61        |
| 9.6.1    | Collecting information                   | 61        |
| 9.6.2    | Change a tag                             | 62        |
| 9.7      | Writing your own widgets                 | 63        |
| 9.7.1    | Data structures                          | 64        |
| 9.7.1.1  | The parse tree                           | 64        |
| 9.7.1.2  | The widget tree                          | 64        |



|           |  |           |
|-----------|--|-----------|
| 9.7.1.3   | The dom node tree . . . . .                  | 64        |
| 9.7.1.4   | The dom . . . . .                            | 65        |
| 9.7.2     | Code notes . . . . .                         | 65        |
| 9.7.3     | Code of the \$select widget . . . . .        | 67        |
| <b>10</b> | <b>The scripting business</b>                | <b>69</b> |
| 10.1      | List scripting . . . . .                     | 69        |
| 10.2      | Interface scripting . . . . .                | 69        |
| 10.3      | Data tiddlers scripting . . . . .            | 69        |
| 10.4      | Template scripting . . . . .                 | 69        |
| 10.5      | Tiddlers scripting . . . . .                 | 69        |
| <b>11</b> | <b>Recipe book</b>                           | <b>71</b> |
| 11.1      | Personal todo-list . . . . .                 | 71        |
| 11.2      | Writting stories . . . . .                   | 71        |
| 11.3      | Image gallery . . . . .                      | 71        |
| 11.4      | Lesson planner . . . . .                     | 71        |
| 11.5      | Simple game . . . . .                        | 71        |
| <b>12</b> | <b>Languages</b>                             | <b>73</b> |
| 12.1      | Tiddlywiki languages . . . . .               | 73        |
| 12.2      | Writing international wikis . . . . .        | 73        |
| <b>13</b> | <b>Plugins</b>                               | <b>75</b> |
| 13.1      | Extending tiddlywiki functionality . . . . . | 75        |
| 13.2      | Creating plugins . . . . .                   | 75        |
| 13.3      | Where we can find plugins . . . . .          | 75        |
| <b>14</b> | <b>The people. The project</b>               | <b>77</b> |
| <b>15</b> | <b>Glossary</b>                              | <b>79</b> |
| <b>16</b> | <b>Resources</b>                             | <b>81</b> |



Mastering Tiddlywiki is a hard a long trip. Why? you can ask yourserf. I answer you. Tiddlywiki is a confluence of many technologies:

1. The first step is understanding Wikitext. It is a markup language like Markdonw: it specify code characters for the format.
2. You need some understanding of HTML. At the bottom, Tiddlywiki is a HTML file with Javascript code.
3. Tiddlywiki uses CSS too. CSS is a technology that goes hand in hand with HTML.
4. If you want to program you own widgets you need Javascript.
5. Although you can use tw as a single file, you can have a self-hosted wiki installing node.js and tiddlywiki from its repository.

But f you want to be a Tiddlywiki master, the first of all is to understand the underlying of Tiddliwiki. This is where this book is going to help you.

Before reading this publication download an empty copy of Tiddlywiki and try the different features it allows.



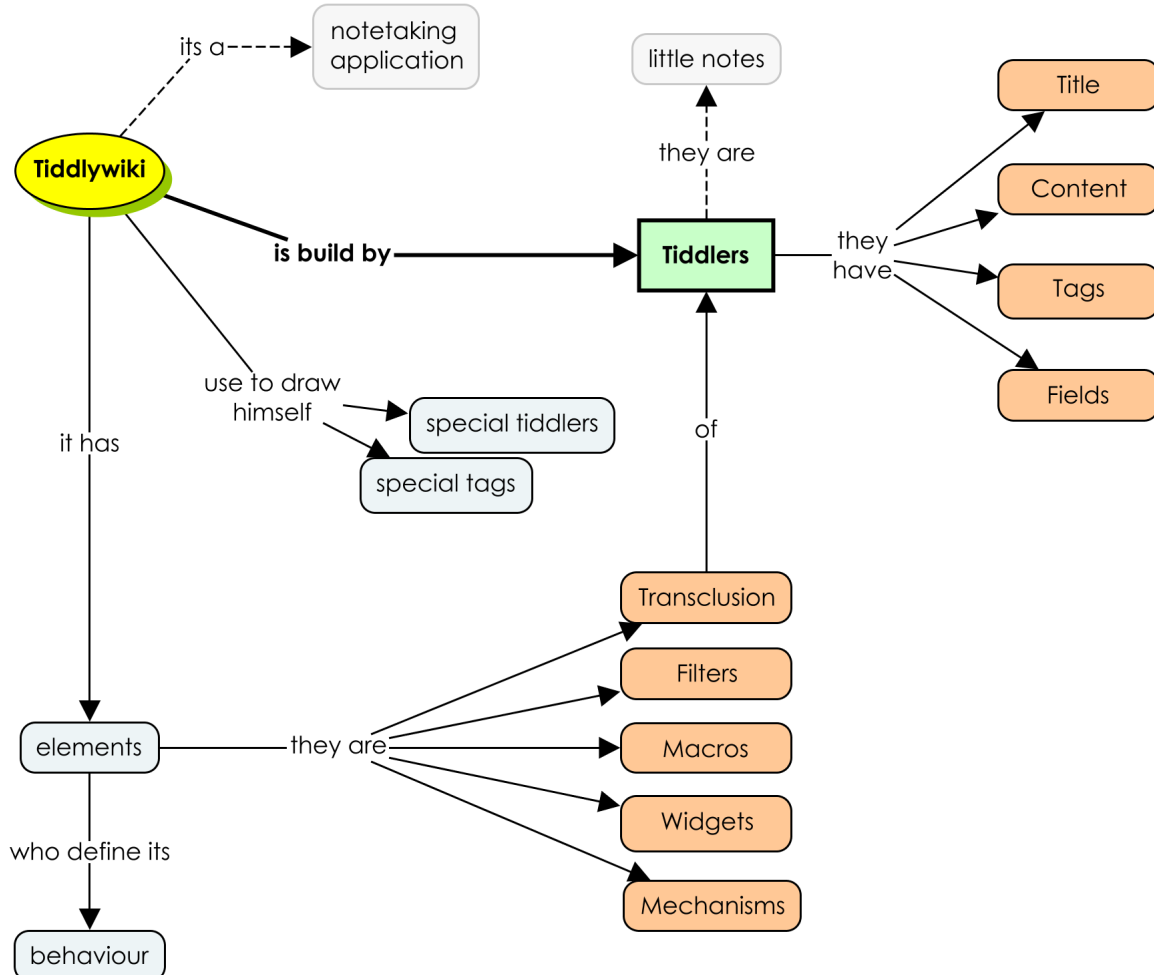
# 1 Introduction

## 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 other 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



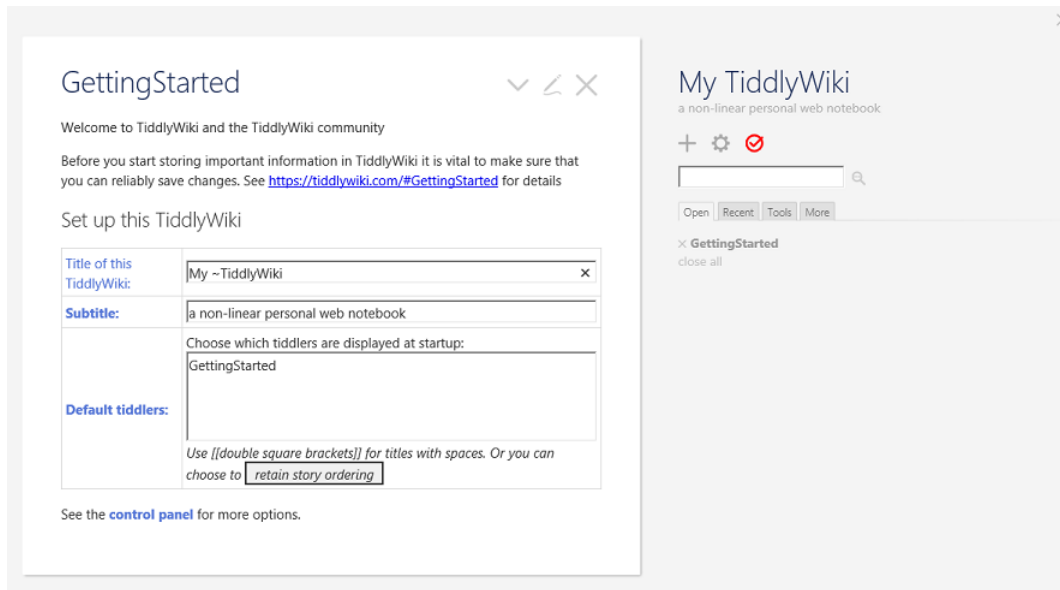
TiddlyWiki is a personal and a non-linear notebook for organizing and sharing complex information. It is an open-source single page application wiki in the form of a single HTML file that includes all javascript code, the CSS format and the content. It is designed to be easy to customize and re-shape 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 must save them by hand if you don't want to lose 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.

## 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 promote information sharing.

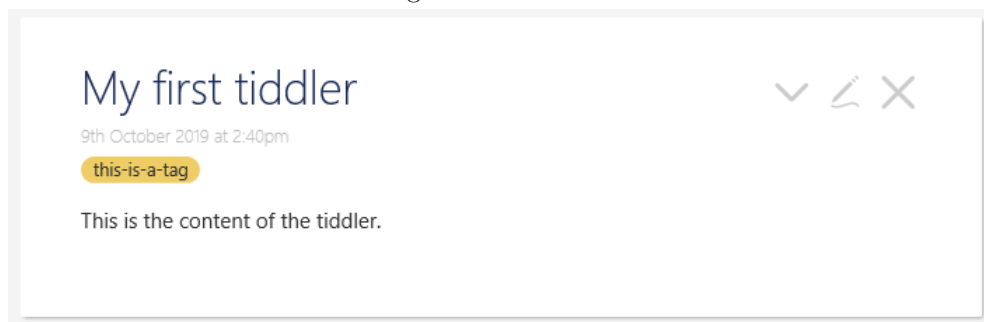
## 1.5 Elements of TW

### 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 4.1).

### 1.5.2 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”.

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



Figure 1.4: Creating a tiddler

Draft of 'New Tiddler'

New Tiddler

tag name add

**B** / **S** **U** **X<sup>2</sup>** **X<sub>2</sub>** **AB** **[[\*]]** **{{\*}}** **QUOTE** **QUOTE** **LIST** **LIST** **H1** **H2** **H3** **LINK**

Type the text for this tiddler

Type: content type

Add a new field: field name field value add

- Buttons to delete, discard and save the tiddler
- Title at the top: “New Tiddler”
- Tag zone
- The format 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. You can search it with the search bar, under the save button or choosing in the right menu More / All.

### 1.5.3 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.

### 1.5.4 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. Open the tiddler `$/TagManager`. With this tiddler you can change the color and add an icon for the tag .

## 1.5.5 Fields

A tiddler has field. There are system field like the created and modified date but you can add your own ones.

## 1.5.6 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.

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

## 1.5.7 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}}`

## 1.5.8 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. You can search them in the advanced search and look for its content. And you can add your own templates.

## 1.5.9 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.

## 1.5.10 Macros and widgets

Tiddlywiki is highly customizable. It use macros and widgets to personalize its appearance.

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.

### 1.5.11 Mechanism

All elements of tiddlywiki fit together through mechanism. For example, the HistoryMechanism 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.

### 1.5.12 Lists

When you look some information inside Tiddlywiki it gives you a list of tiddlers, so the manage of list 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

### 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

### 2.3 Organizing microcontent with links

### 2.4 Organizing tiddlers with tags

### 2.5 Personalize behavior with macros and widgets

### 2.6 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.

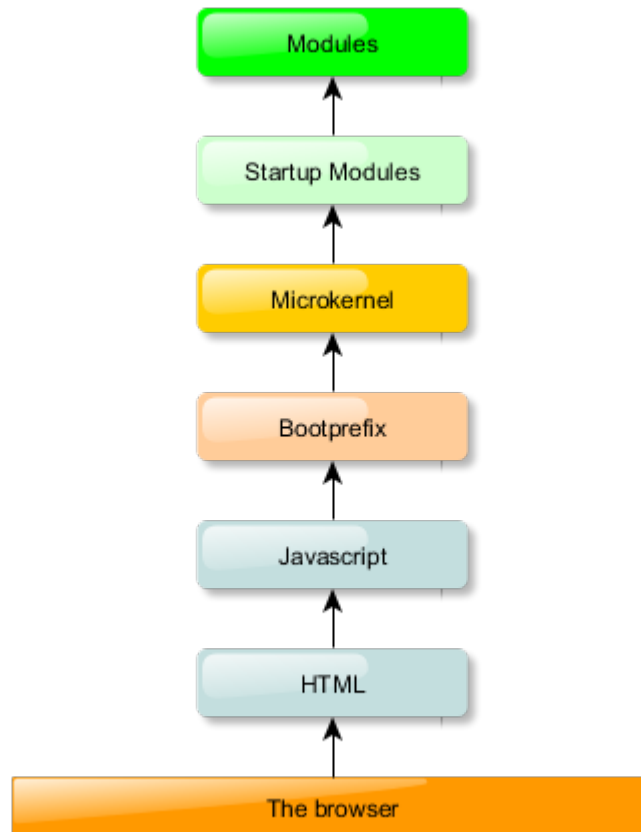
## 3 Loading tiddlywiki in the browser

### 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...

## 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</div>
<div id="storeArea">The tiddlers of the wiki</div>

<div id="bootKernelPrefix" type="application/javascript">
<script>
  var bootprefix = (function($tw){
    "use strict";
    $tw = $tw || Object.create(null);
    ...
  })($tw);
</script>
</div>
...
</body>
</html>
```

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.

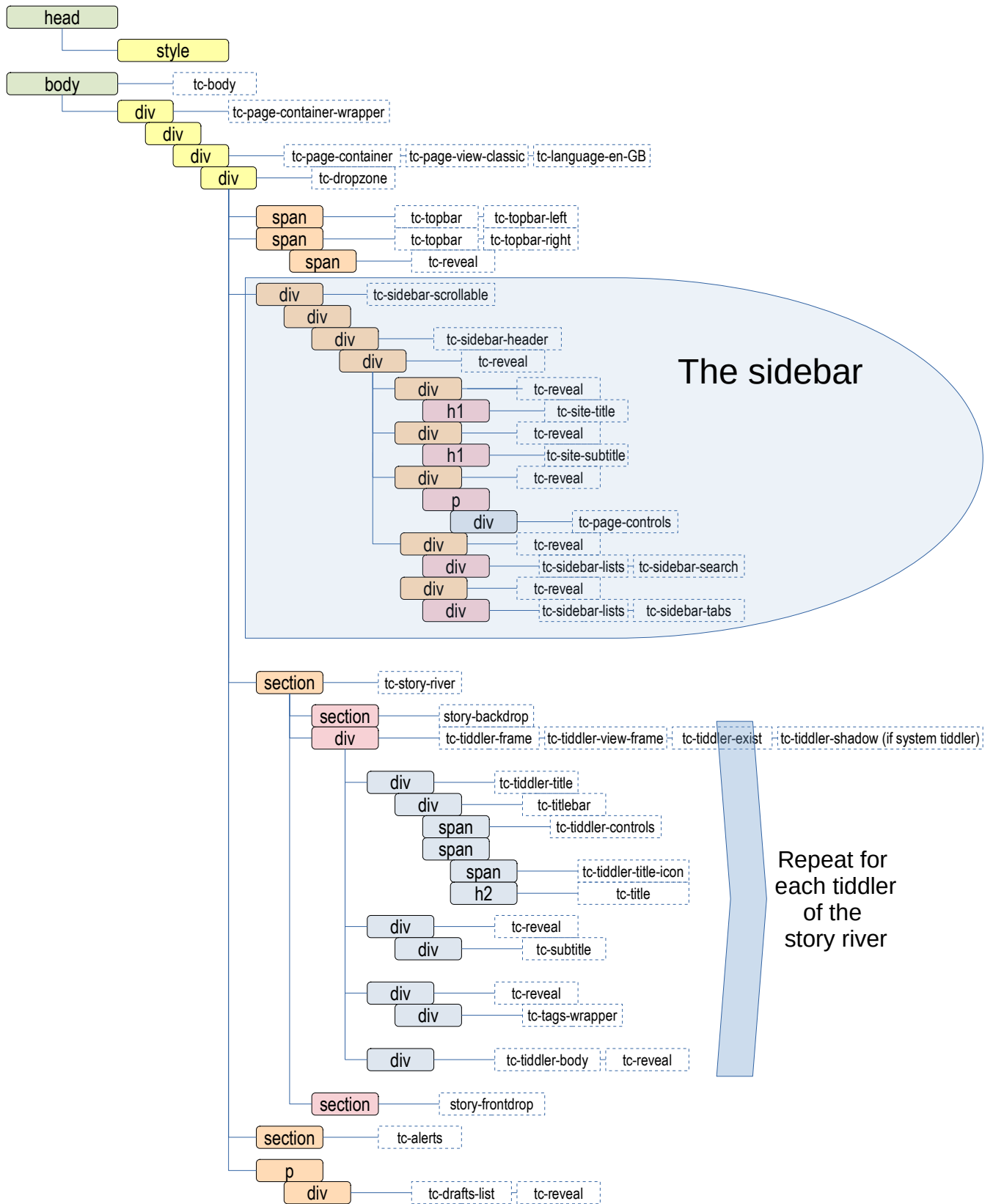


1. First of all the operating system loads the browser.
2. The browser loads the HTML page.
3. After some css styles and the store area we can see the javascript code. The browser runs this code.
4. It loads the bootprefix. The bootprefix is responsible for preparing the kernel to boot on different engines 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 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 6.2.1](#) you can see all roles of a tiddler.



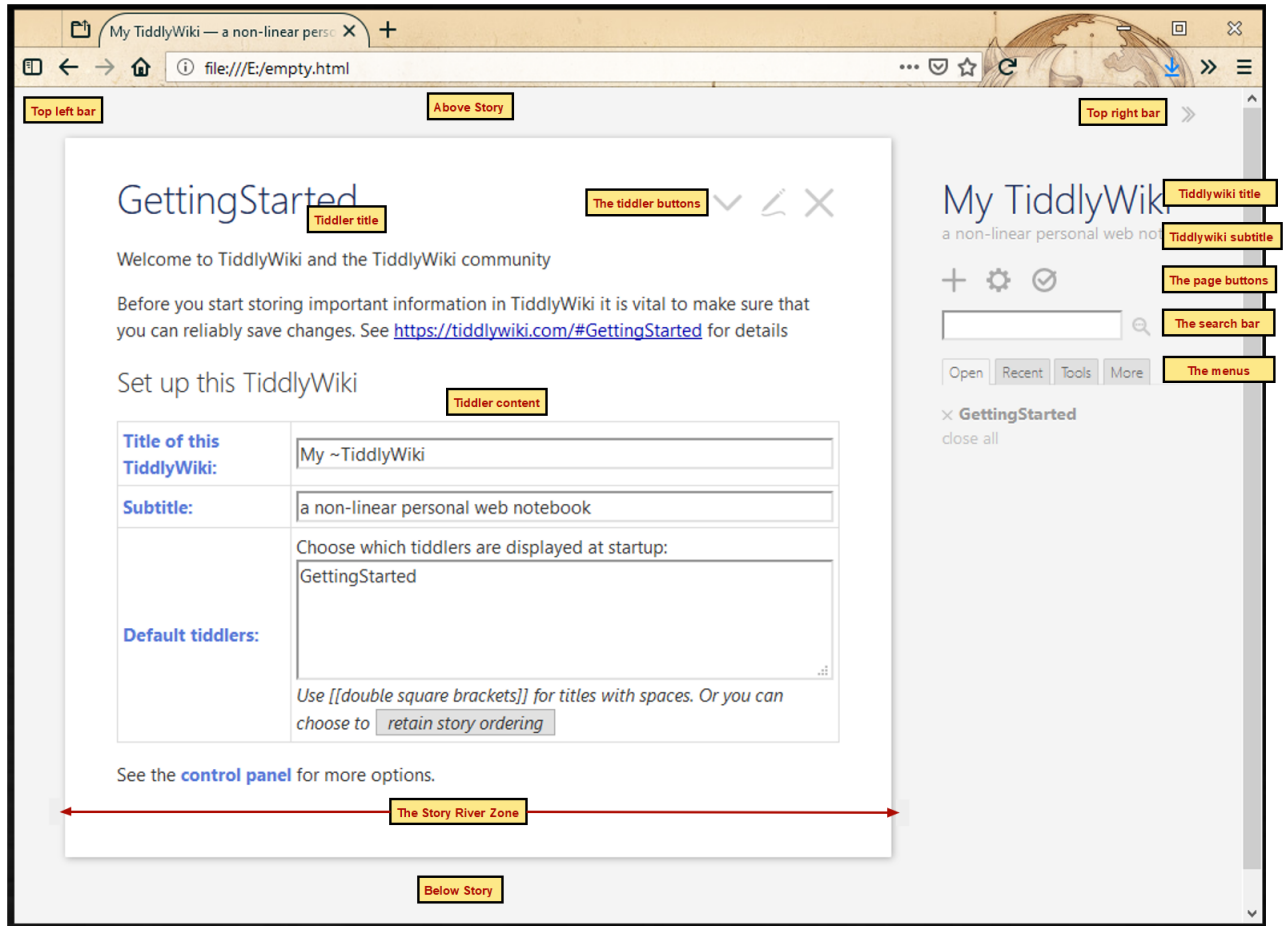
## 4 Components of the tiddlywiki screen

### 4.1 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.

## 4.2 The TW whole screen

Figure 4.1: The Tiddlywiki page



In Figure 4.1 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.

### 4.2.1 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 open each under the other but you can configure it so that only the active tiddler appears.

## 4.2.2 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 4.2](#)

## 4.3 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)

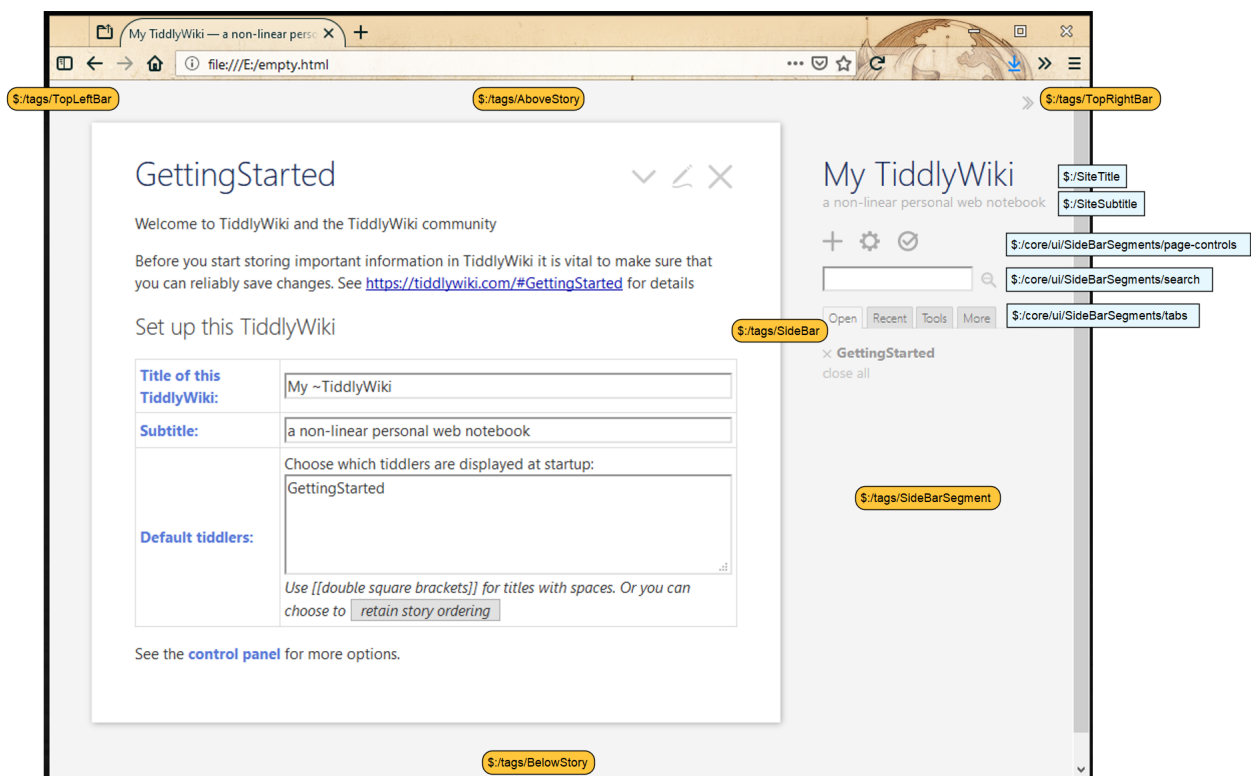
All elements are tagged with `$/tags/PageTemplate`.

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 4.2](#).

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 4.2: The tags of Tiddlywiki places



## 5 Customize the TW screen

### 5.1 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

## 5.2 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

## 5.3 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.



## 5.4 ViewTemplate

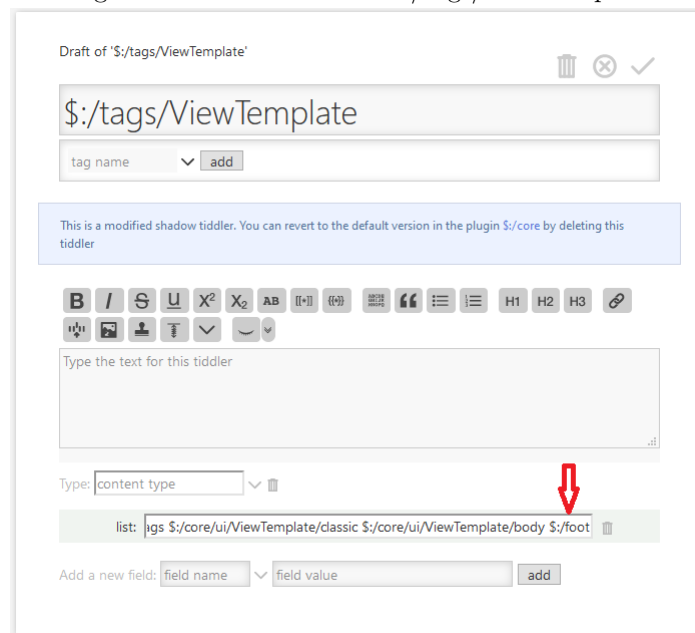
This is the template Tiddlywiki uses to display a single tiddler when you are reading it. Its title is `$/core/ui/ViewTemplate`. As in the PageTemplate, you can edit it and look at its code:

```
<$list filter="[all[shadows+tiddlers]tag[$:/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`.

### 5.4.1 Example

Figure 5.1: 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 5.1).
- Save the tiddler.

You will see a foot in all tiddlers with the created date. In Figure 5.2 you will see the new tiddler appearance.

## 5.5 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.

## 5.6 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

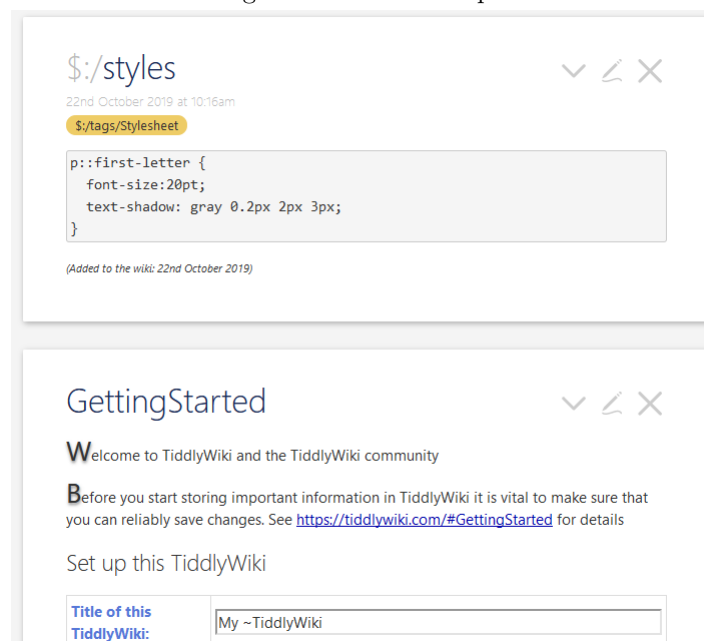
If you read this tiddler you will find all css clases for this theme. At the end of this chapter, on page 38 we will see a map with many css standard classes. The best way of using this map is search the classes in it in the main css Tiddlywiki style sheet, `$/themes/tiddlywiki/vanilla/base`. In this 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...     |

### 5.6.1 Example

Figure 5.2: 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 5.2 you see the style tiddler and the new appearance of the paragraphs.

## 5.7 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.

## 5.8 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.

### 5.8.1 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.

## 5.9 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

### 5.9.1 How to do it

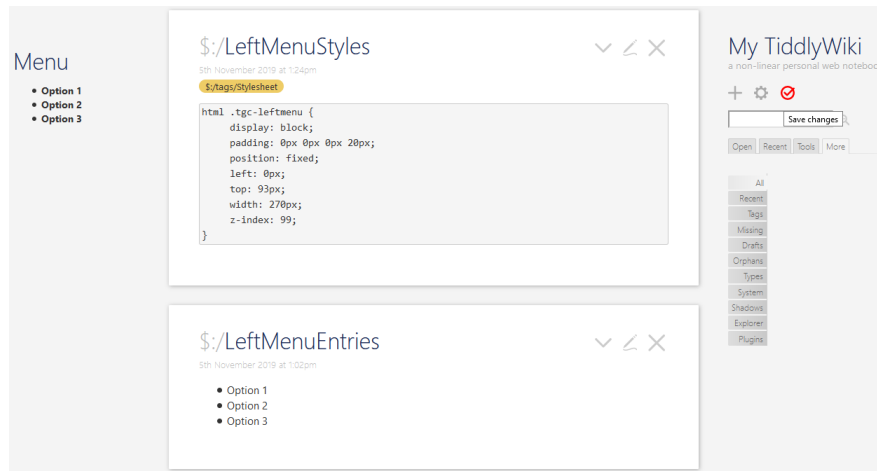
- Create a new tiddler. This new tiddler is only for information:
  - Title: `$/config/ShortcutInfo/control-panel`
  - Text: `ctrl-alt-P`.
- Go to the control panel.
  - Look for the new entry, control-panel.
  - Expand and click in the pen icon.
  - In the text area type the keys CTRL+ALT+P
- Create other tiddler:
  - Title: `$/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 5.3: A left menu



## 5.10 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

### 5.10.1 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.

### 5.10.2 Create the tiddler menu

Create a new tiddler. For example, `$/LeftMenu` and tag it with `$/tags/PageTemplate`

Add this code

```
<div class="tgc-leftmenu tc-table-of-contents">
@@.tc-site-title
Menu
@@
<$scrollable class='tgc-scrollable-menu'>
{{$/LeftMenuEntries}}
</$scrollable>
</div>
```

### 5.10.3 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 20px;
  position: fixed;
  left: 0px;
  top: 93px;
  width: 270px;
  z-index: 99;
}
```

### 5.10.4 Create the entries for the left menu

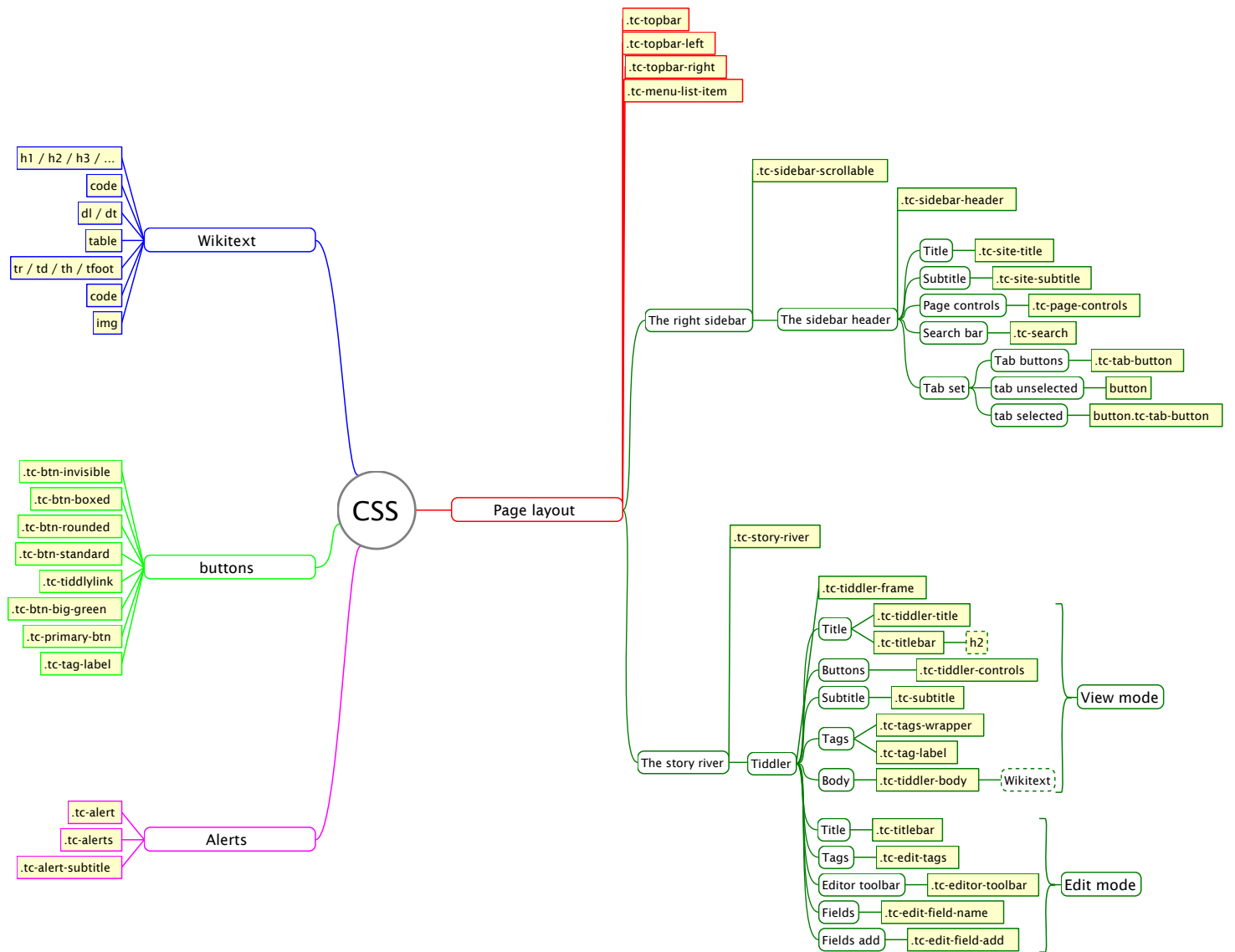
If you read the code of the tiddler left menu you can see the line: `{{$/LeftMenuEntries}}`. This mean that all things you put in this tiddler appears on your left menu. You can add to this tiddler:

- A toc in the usual way.
- A tabbed tiddler
- Some tag bubbles

The options are endless.

### 5.10.5 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.





## 6 Inside a tiddler

### 6.1 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.

## 6.2 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?

**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.

**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.

**Macro tiddler:** a tiddler that stores macros.

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

**Other system tiddlers:** There are special tiddlers that have utility to configure tiddlywiki. They are usually tiddlers with a specific name.

### 6.2.1 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                       | Tag                               |
|-----------------------------|----------------------------------|-----------------------------------|
| Normal tiddler              |                                  |                                   |
| Tag tiddler                 |                                  |                                   |
| Alert tiddler               |                                  | \$/tags/Alert                     |
| 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 &amp; 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>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:

- Add a tiddler
- In the Type field write: "image/png"
- Add other field called "\_canonical\_uri" with this content: "files/beach.png" (Figure 6.1)
- Save the tiddler

Figure 6.1: 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](#)

files/Beach.png

Type: image/png ▼ 🗑️

\_canonical\_uri: files/Beach.png 🗑️

Add a new field: field name ▼ field value add

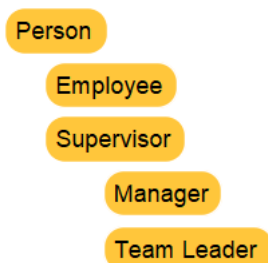
## 6.3 Normal tiddler

You add a normal tiddler with the plus button on the left side of Tiddlywiki. 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.

## 6.4 Tag tiddler

Figure 6.2: Tag hierarchy



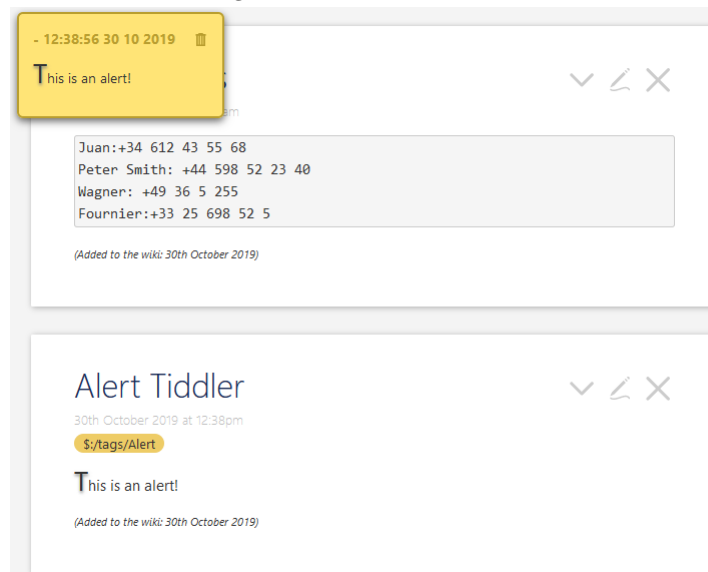
A tag tiddler is a normal tiddler titled with the name of a tag. The main advantage to do this is to construct a hierarchy of tags. For example, imagine you want to create this hierarchy:

- 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.

## 6.5 Alert tiddler

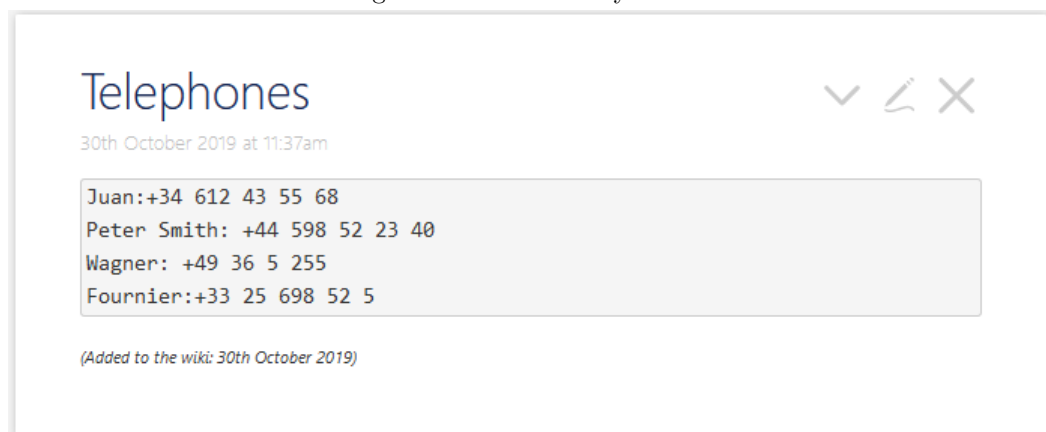
Figure 6.3: An alert tiddler



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

## 6.6 Dictionary tiddler

Figure 6.4: 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 6.4 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
```

## 6.7 JSON tiddlers

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

## 6.8 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.

## 6.9 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.

## 6.10 Other system tiddlers

### 6.10.1 `$/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:

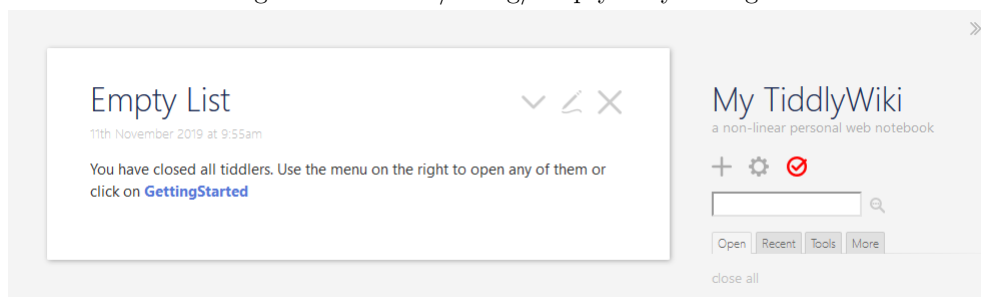
1. Create the tiddler you want to show when all tiddlers are closed. For example, “Empty List”. Add your message in the content, for example:

```
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:

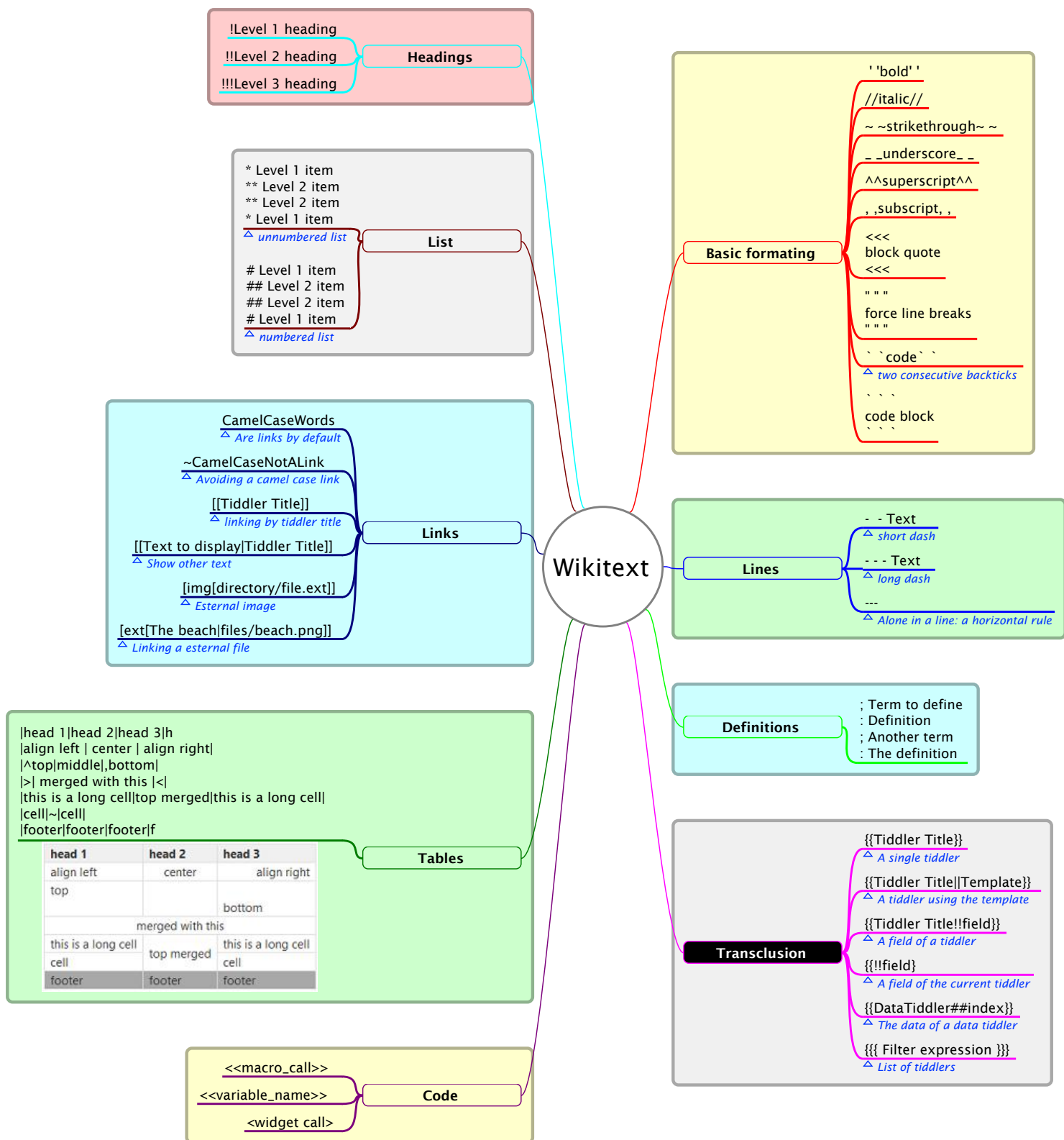
Figure 6.5: The `$/config/EmptyStoryMessage`



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.

## **6.11 Adding macros and widgets**

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





# 7 Filters

## 7.1 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.

## 7.2 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 1.5.12.

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>>).

## 7.3 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. Maybe the managers will be listed twice).
- SimpleFilter1 □ = SimpleFilter2: Union of the two list without duplication. (OR combination)
  - 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.

## 7.4 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 7.1: 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 7.1. You know each line is the title of a tiddler, so in your wiki you have 7 tiddlers tagged with “employee”.

Figure 7.2: 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 7.2 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 `<li>` 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).

## 7.5 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 7.3: 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 | \$ = a string of characters                    |
| output    | those input titles that start with \$          |
| ! output  | those input tiddlers that do not start with \$ |

In looking for matches for \$, 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.

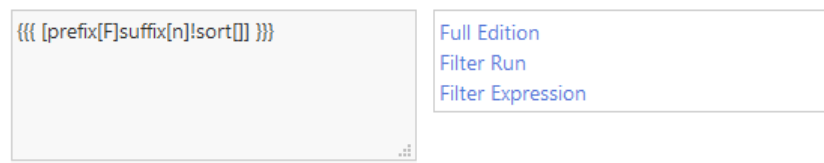
- Parameter: The parameter of the operator. In this operator is a sequence of characters so you can write `[prefix[H]]` to show all tiddlers with the tittle starting with “H”.
- Output: What list produce the operator.
- !Output: What list produce the operator if you put, for example, `[!prefix[H]]`.

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 7.4.

Figure 7.4: 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.

## 7.6 Regular expressions

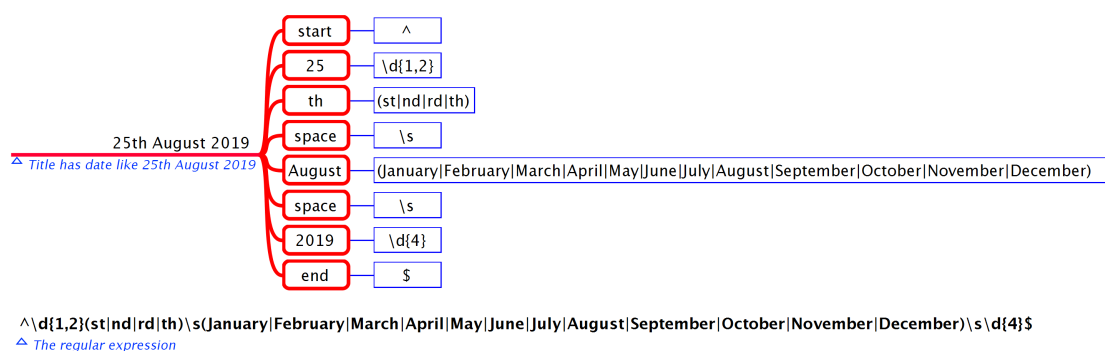
### 7.6.1 Writing regular expressions

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 7.5: 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 7.5 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}$
```

## 7.6.2 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 7.5 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  $\wedge\{d{2}\}.*\{d{2}\}$ .



## 8 Macros

### 8.1 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.*

### 8.2 Variables and parameters

#### 8.2.1 Defining variables

There are two ways of defining variables. The first with the `<$set>` sidet and the second with the `<$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>`

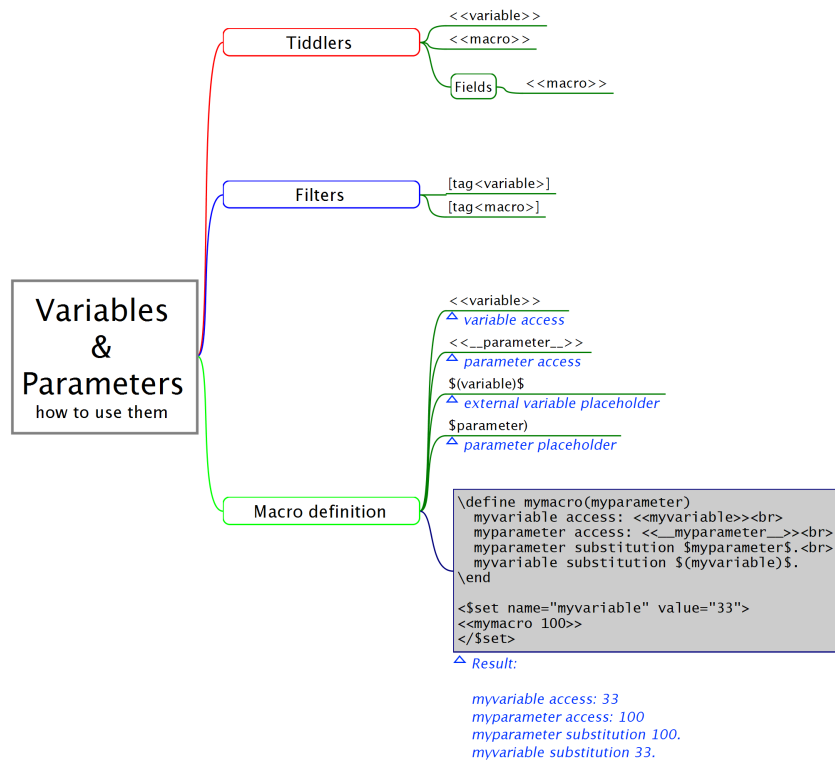
```
<$vars greeting="Hi" me={{!!title}} sentence=<<helloworld>> >
  <<greeting>>! I am <<me>> and I say: <<sentence>>
</$vars>
```

#### 8.2.2 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.

### 8.2.3 Using variables and parameters

Figure 8.1: Using variables



We can use variables in parameters in many places: in a tiddler, inside the definition of a macro, in fields, in filters, in links... The way of using them depends of the place and is very very tricky.

## 8.3 Basic macros included

## 8.4 Writing your own macros

## 8.5 Using macros

# 9 Widgets

## 9.1 Introduction

Widgets are trickiest part of 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. The easy an worst way is create a tiddler for each task and other tiddler with all this tasks:

```
*[[ Study_javascript ]]  
*[[ Download_github ]]  
*[[ Create_the_repository ]]  
*[[ Download_the_code ]]
```

This has a terrible problem: you have no control of the tasks you finish. Of course you can edit the tasks tiddlers and add a tag, “done” for the tiddlers you finish but this is a big effort.

## 9.2 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) |

For our little project we use the tag mode. This is the tasks tiddler code:

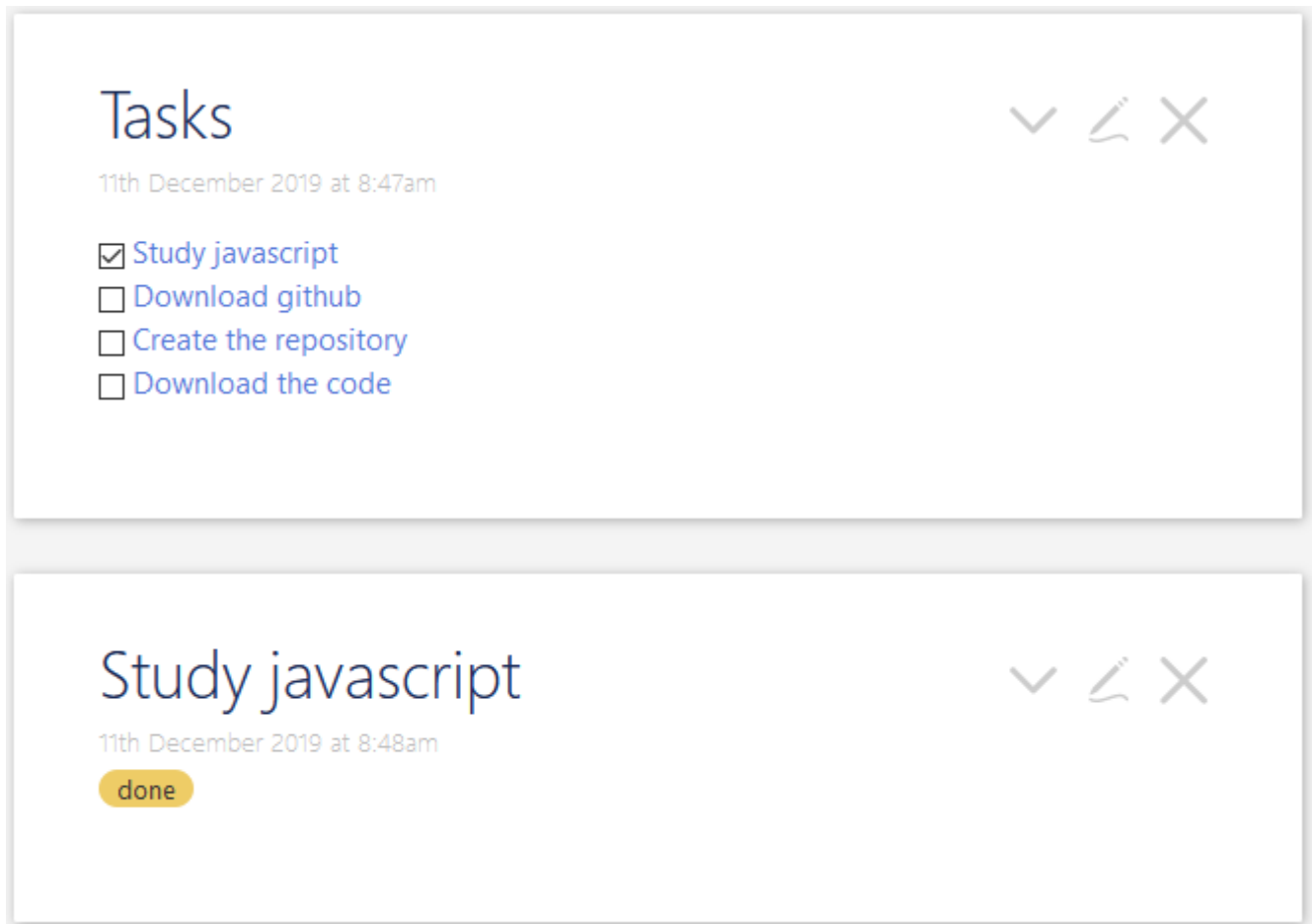
```
<$checkbox tiddler="Study_javascript" tag="done">[[ Study_javascript ]]</  
$checkbox><br/>
```

```

<$checkbox_tiddler="Download_github" _tag="done">_ [[ Download_github]] </
  $checkbox><br/>
<$checkbox_tiddler="Create_the_repository" _tag="done">_ [[ Create_the_
  repository]] </ $checkbox><br/>
<$checkbox_tiddler="Download_the_code" _tag="done">_ [[ Download_the_code]] </
  $checkbox><br/>

```

Figure 9.1: Adding a checkbox



Adding new tasks is still a complex process. We need an easy way of listing the tasks. We solve this with other tiddler: \$list.

## 9.3 The \$list widget

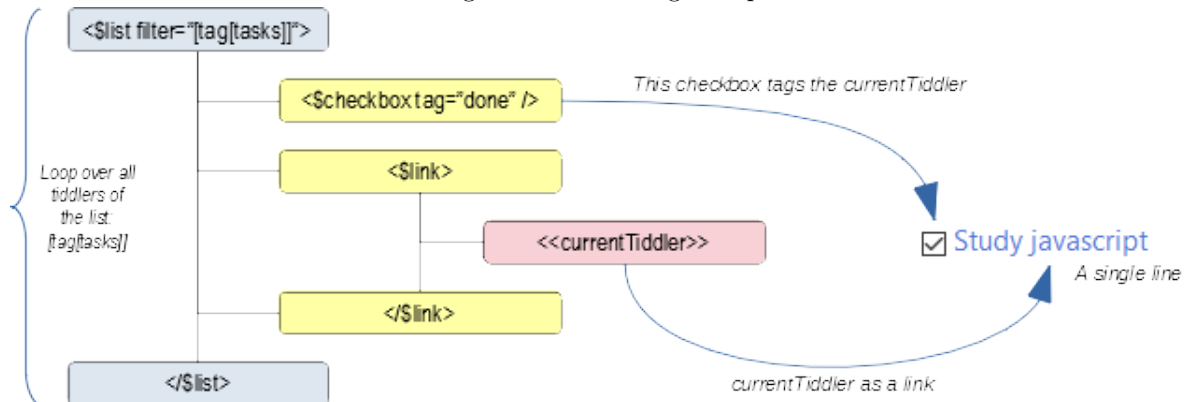
We have to simplify things. We have to tell Tiddlywiki which tiddlers are tasks. We do this adding the “task” tag for task tiddlers. We saw the \$list widgets in the “Filters” chapter. Is an easy way to list custom tiddlers. For our purpose we want to list all task tiddlers:

```

<$list_filter="[tag [task]]">
<$checkbox_tag="done"/><$link><currentTiddler></ $link><br/>
</ $list>

```

Figure 9.2: The widget loop



If you tag all tasks with the tag “task” the output will be the same of Figure 9.1. 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 9.2.

In this example our widget loops over the tasks tiddlers writing the lines: `<$checkbox tag="done"/>` `<$link><<currentTiddler>></$link><br/>`. There are 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 needs a blank line before them. for example, if you add to a tiddler:

```
This is my list :
*item1
*item2
*item3
```

You will see this output:

```
This is my list : *item1*item2*item3
```

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

```
This is my list :

*item1
*item2
*item3
```

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.

## 9.4 \$edit-text widget

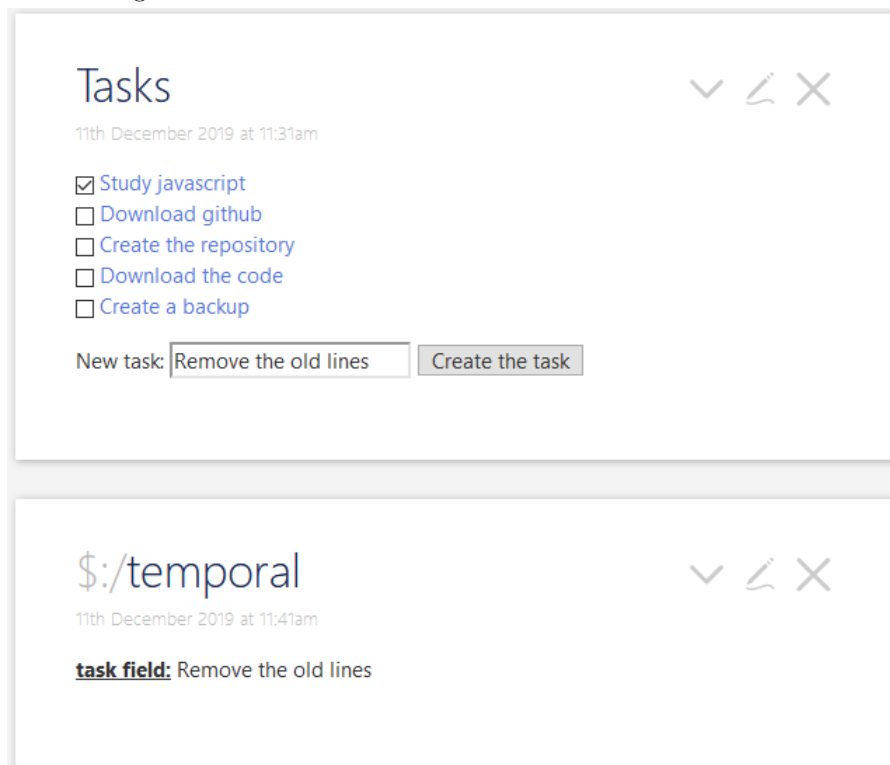
Maybe we can simplify the adding of new task. First we analyze the `$edit-text` widget. It provide us a simple way of edit temporal data. We use it for writing the text of the new task:

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

All things you write to this text little rectangle applies immediately to the field “task” of the tiddler “\$:/temporal”. In Figure 9.3 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.

## 9.5 \$button widget

Figure 9.3: 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" $field="task" text=""/>
CreateTask
</$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><br/>
</$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>

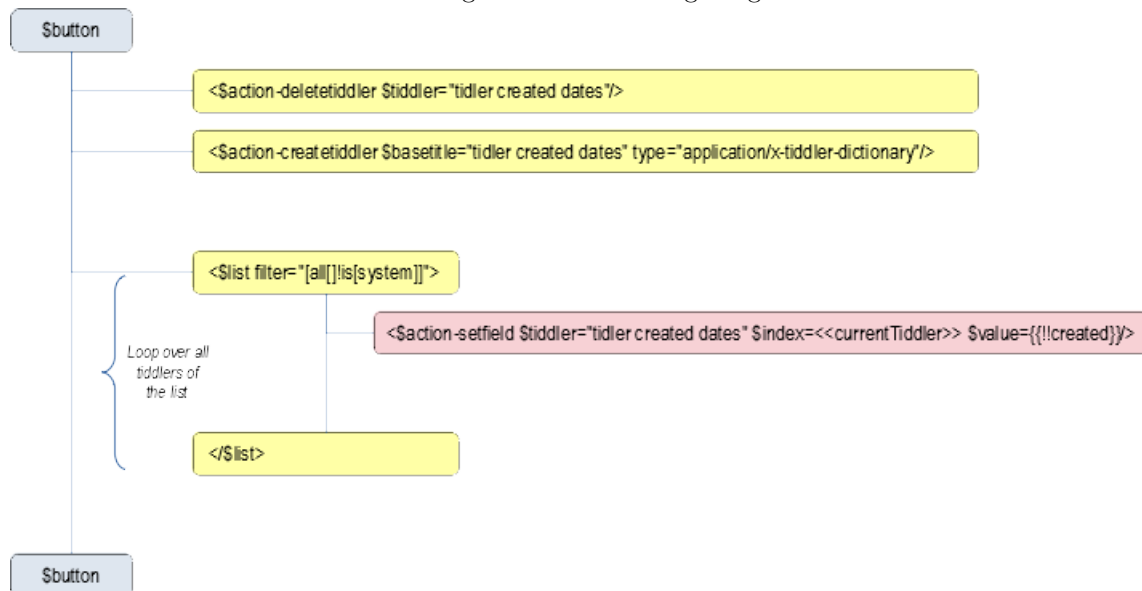
```

The output is in Figure 9.3.

## 9.6 Combining widgets

### 9.6.1 Collecting information

Figure 9.4: 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:

```

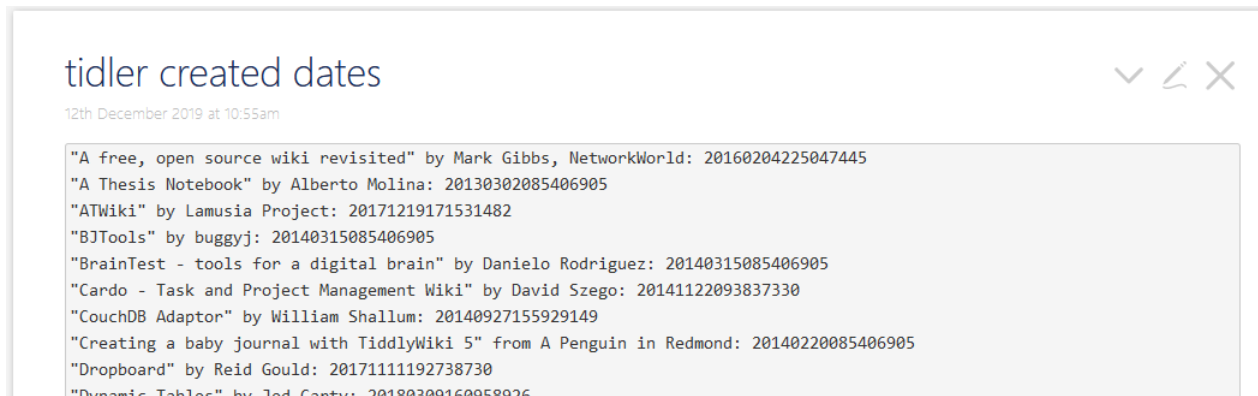
<$button>
<$action-deletetiddler $tiddler="tidler created dates"/>
<$action-createtiddler $basetitle="tidler created dates" type="application/
  x-tiddler-dictionary"/>

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

```

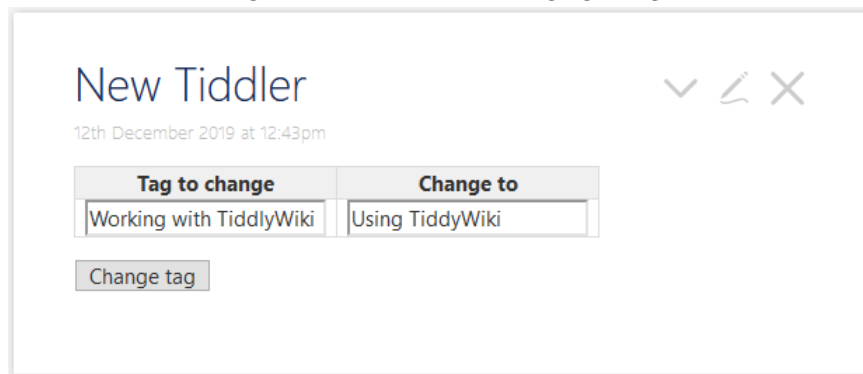
This will loop over all elements of the list inserting the title and the created field in the tiddler “tidler created dates”. In Figure 9.4 its a little diagram of the code. Now, if you open this tiddler you will see the dates. The first four numbers are the year, the next two the month and the next two the day. And after that there is the hour, minute... (look at Figure 9.5)

Figure 9.5: Tiddler created dates



## 9.6.2 Change a tag

Figure 9.6: 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>
```

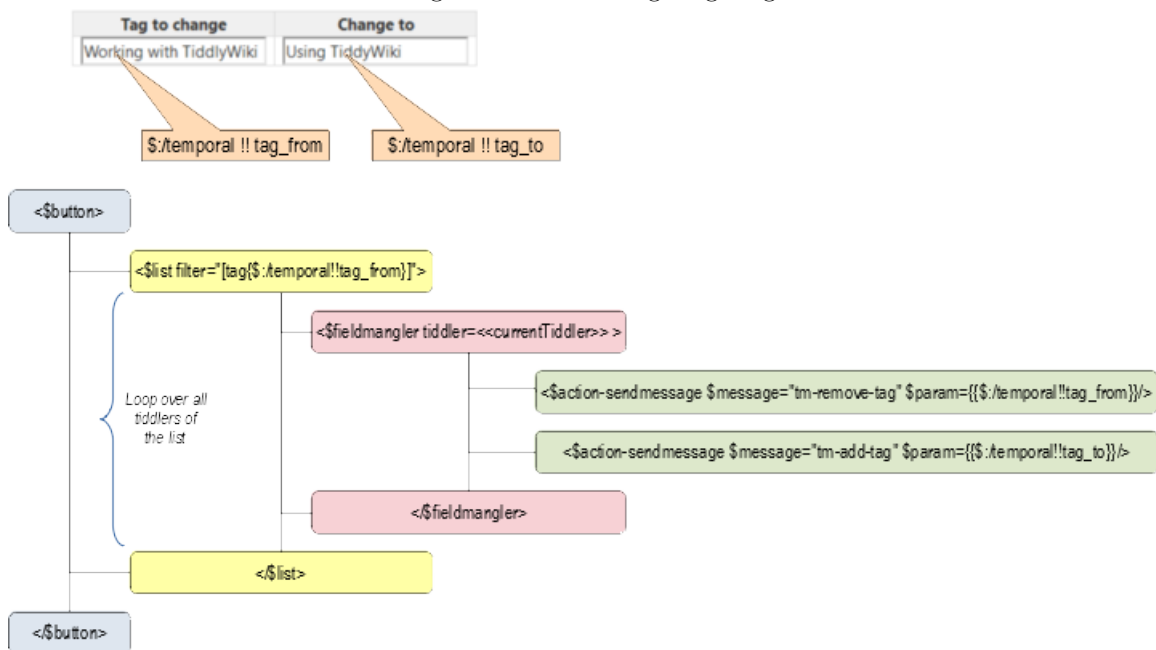


Change\_tag  
</\$button>

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 9.7: The change tag diagram



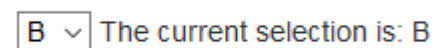
## 9.7 Writing your own widgets

The purpose of a widget is, usually to render; it is a tool by which "authors" can create rich and dynamic wiki content quickly. Widgets are one of the mechanisms by which TiddlyWiki5 becomes a "generative" writing tool as opposed to mere static. While there are out-of-the-box widgets which provide the rich authoring capabilities which most people need, there is also an infrastructure for 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 9.8 is the output for this code:

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

Figure 9.8: The select widget output



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

## 9.7.1 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 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.

### 9.7.1.1 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>15</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.

### 9.7.1.2 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 wiktext 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.

### 9.7.1.3 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.

#### 9.7.1.4 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.

#### 9.7.2 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.

### 9.7.3 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;
})();

```

## **10 The scripting business**

Pieces of code.

### **10.1 List scripting**

Lists, nested lists, actions and buttons inside lists...

### **10.2 Interface scripting**

How to use buttons, edittext, radiobuttons, checkboxes...

### **10.3 Data tiddlers scripting**

How to use data tiddlers.

### **10.4 Template scripting**

Revealing sections of tiddlywiki, state tiddlers...

### **10.5 Tiddlers scripting**

Creating, removing and editing tiddlers with code, navigating with code...





# **11 Recipe book**

In this section we will show master code pieces to various situations. Personal wikis with many uses. Maybe the sections will be:

## **11.1 Personal todo-list**

## **11.2 Writting stories**

## **11.3 Image gallery**

## **11.4 Lesson planner**

## **11.5 Simple game**



## **12 Languages**

### **12.1 Tiddlywiki languages**

### **12.2 Writing international wikis**



## **13 Plugins**

### **13.1 Extending tiddlywiki functionality**

### **13.2 Creating plugins**

### **13.3 Where we can find plugins**



## 14 The people. The project

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.





## 15 Glossary

**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



## 16 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