

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Diplom Thesis Informatics

Mit der Trello-API rummuckeln

Sebastian Engel

24th July 2012

Referees

Name Erstgutachter
(Bioinformatik)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Name Zweitgutachter
(Biologie/Medizin)
Medizinische Fakultät
Universität Tübingen

Engel, Sebastian:

Titel der Arbeit

Diplom Thesis Bioinformatics

Eberhard Karls Universität Tübingen

Thesis period: 13th march to 11th September 2012

Abstract

Trello is a collaboration webservice to manage projects and assign their todo items to co-workers. There are many collaboration tools today, but most of them are very basic. Trello is very extensive and it is optimal for small businesses. But although it works fine like it's supposed to it has its limits. Trello as its state now is a closed system. Nothing gets in or out unless you use Trello itself. But sometimes it would be handy if you were able to get content from Trello out into other applications. For example a CMS which should contain completed theses which you are already managing in Trello.

So this thesis addresses small scripts which let Trello interact with other web-services and applications. For this purpose I wrote a wrapper of the Trello API in Ruby to accomplish this task in the most dynamic way possible.

Acknowledgements

Write here your acknowledgements.

Contents

List of Figures	v
List of Tables	vii
Nomenclature	ix
1 Introduction	1
2 Principles	3
2.1 Trello	3
2.1.1 How Trello works	3
2.1.2 Why Trello	4
2.1.3 Trello API	4
2.2 JSON	6
2.3 Ruby	7
2.3.1 RubyGems	8
3 Applications	11
3.1 Trello API wrapper	11
3.2 Command Line Interface	11
3.3 Export to HTML	13
3.3.1 Markdown	14
3.3.2 Twitter Bootstrap Framework	17
3.3.3 HTML5	18

3.3.4	Templating with ERB	18
3.4	One way sync to Google Calendar	22
3.5	Export to iCalendar	22
3.6	One way sync to Joomla	22
3.6.1	For every card an article	22
3.6.2	All cards in one article	22
3.7	One way sync to WordPress	22
3.8	Backup	22
3.8.1	Export	22
3.8.2	Import	22
3.8.3	Member import	23
3.8.4	Close all boards	23
4	Conclusion	25
5	Outlook	27
5.1	Trello Alfred Extension	27
5.2	Native applications	28
	Bibliography	29
	Index	31

List of Figures

2.1	A Trello board.	3
2.2	A opened card in Trello.	9
3.1	Connections between Trello, the API wrapper and the actual features. [rub][htm][joo12][goo12]	12
3.2	The browser view of the HTML converted from the Markdown in listing 3.5	16
3.3	Overview about kramdowns converting options. [kra12]	17
5.1	Alfred Extension for Trello: This command would add a card with the name <i>Visit the Reichstag</i> to the board called <i>Berlin</i> . . .	28

List of Tables

Nomenclature

API	Application Programming Interface
CLI	Command Line Interface
CMS	Content Management System
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
REST	Representational State Transfer
URI	Uniform resource identifier
XHTML	Extensible Hyper Text Markup Language
XML	Extensible Markup Language

Chapter 1

Introduction

Blablabla.....

Die Arbeit gliedert sich dazu wie folgt: Die Grundlagen von BlaBlaBla werden in Kapitel 1 erarbeitet. ... Eine Diskussion und ein kurzer Ausblick im Kapitel ?? beschließen diese Arbeit.

Bevor wir uns der Auswertung bzw. Bewertung der gewonnenen Primärdaten zuwenden, wollen wir zunächst einige grundlegende Begriffe der deskriptiven Statistik wiederholen.

Chapter 2

Principles

2.1 Trello

2.1.1 How Trello works

Trello is a webservice by the New York City based web corporation Fog Creek Software¹. It is a collaboration tool to manage projects, launched in 2011².



Figure 2.1: A Trello board.

There is the concept of so called *boards* which contains several configurable lists. Figure 2.1 shows a board with the three standard lists *To Do*, *Doing* and *Done*. In these lists the user can create todo items. These todo items are called *cards*. The cards can contain several additional data. Each card has a title and maybe a description, some assigned members, a due date, some

¹Official Fog Creek Software website: <http://www.fogcreek.com>

²The original launch post in the Trello blog: <http://blog.trello.com/launch/>

labels, votes, checklists, comments and attachments. The creator of the board is the owner in the first place and the owner can add other Trello users to his boards and cards. So everyone who's working on a project can see what's going on at the moment. Users who are assigned to a board can even create new todo items by themselves. If somebody works at more than one company with many projects each there is the concept of *organizations*. This is useful in order to ensure a clear separation.

2.1.2 Why Trello

Trello is not just one of hundreds of thousands of todo applications. It is streamlined for the purposes of small businesses. So for the needs in the university with small groups of people working on the same things it is perfect. Trello has proofed its value several months already. The Trello website is written in HTML 5 with the use of AJAX where it makes sense. Trello provides an iOS [tre12a] and Android [tre12c] app. Both are constantly evolving. So the system is state-of-the-art. In addition the company behind Trello is not just a start-up with three employees. That's important, too. A product of a small business, which is based just on the enthusiasm of the founders often doesn't last long. Fog Creek Software is over ten years old and has several products.

The first wish was to see the due dates of the cards anybody is assigned to in Google Calendar. But thinking about that there were many other use cases for small scripts which could run as cron jobs on a server to serve several regular tasks. These scripts are described in more detail in Chapter 3.

2.1.3 Trello API

Trello has an API which is still in beta at the moment I'm writing this. But it is already very extensive. [tre12d]

Authentication

Though the scripts which are used here need access to private boards in Trello there has to be any kind of authentication. For user applications with a frontend the Trello API provides OAuth2. But because of the concept of OAuth2 the user is required to enter his Trello username and password. [oau12] My scripts are supposed to run on servers as cron jobs. There is no user who could manually enter data. For this kind of applications Trello provides a key/token-system. Every user has a private key. With this key the user can generate a token. This token will be sent along every request to the Trello API. The token tells Trello which scope the request can see. While

generating a token one can specify the scope of the token and when it will expire. The possible expirations of a token are between one day and never. In our case we will use *never*. To generate a token one has to visit a special URL: `https://trello.com/1/authorize?key=SUBSTITUTEWITHYOURPRIVATEKEY&name=My+Application&expiration=never&response_type=token&scope=read,write` In this example the token would never expire and could read and write everything the user can access with the API. Other valid values instead of *never* for expiration would be *1day*, *30days*. *30days* is the default value. [tre12b]

REST

The Trello API is a *RESTful* web API. That means that the API is conform to the REST design model. REST is a common style of software architecture for distributed systems. It's built on four of the HTTP request methods: GET, POST, PUT and DELETE. An implementation of a REST web service follows four basic design principles:

- Use HTTP methods explicitly.
- Be stateless.
- Expose directory structure-like URIs.
- Transfer XML, JSON, or both.

[res12]

Following these conventions an URL of a RESTful web service looks like this:

```
https://api.trello.com/1/boards/4eea4ffc91e31d1746000046/  
checklists?key=PRIVATEKEY&token=TOKEN
```

This is a GET request to get all checklists of the board with the id `4eea4ffc91e31d1746000046`. If this URL is visited in a browser (with correct *key* and *token*) the browser will show plain JSON. In order Ruby can work with it, it must be able to capture this data somehow.

To fulfil the requirements of REST in Ruby there are several gems. Here the `RestClient` gem is used. In comparison to the `open-uri` library, which is included in the Ruby standard library, it's much more tidied up when it comes to POST requests.

```
1 response = RestClient.post(  
2   'https://api.trello.com/1/boards',  
3   :name => board['name'],  
4   :desc => board['desc'],
```

```
5 :token => $token
6 )
```

Listing 2.1: POST request using RestClient.

```
1 uri = URI('https://api.trello.com/1/boards')
2 req = Net::HTTP::Post.new(uri.path)
3
4 req.set_form_data(
5   'name' => board['name'],
6   'desc' => board['desc'],
7   'key' => $key,
8   'token' => $token
9 )
10
11 Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.
12   scheme == 'https') do |http|
13   response = http.request(req)
14 end
```

Listing 2.2: POST request with open-uri.

Listing 2.1 and listing 2.2 show the very same API call. But 2.1 is realised with RestClient and listing 2.2 with open-uri. Not even is the open-uri code much longer, but open-uri doesn't detect the correct scheme from the given URI. If the call should be performed in HTTPS this has to be set explicitly. That implies that for the handling of RESTful web services RestClient is the better choice.

2.2 JSON

All the responses to Trello API calls use JSON. It is a subset of the JavaScript programming language. Despite its relation to JavaScript it's language independent. JSON is a data-interchange format like XML. But JSON is built on two structures. One is a list of key/value pairs. In most programming languages this is realised as a hash, struct, object or associative array. The other structure is an ordered list of values. This is realised as array, list, vector or sequence in popular programming languages. In JSON itself these structures are called *object* and *array*. Objects start and end with curly brackets. Each key is followed by a colon and the key/value pairs are separated by commas. Arrays start and end with square brackets. The values are separated by commas. Both can be arbitrarily nested. At every point one of my scripts saves content at any other place than Trello it's in the JSON format, too. That's

because it guarantees easy compatibility with Trello. JSON can be saved in files, too. A JSON file has the suffix `.json`. [jso12]

```
1 {  
2   "id": "4eea4ffc91e31d1746000046",  
3   "name": "Example Board",  
4   "desc": "This board is used in the API examples",  
5   "lists": [{  
6     "id": "4eea4ffc91e31d174600004a",  
7     "name": "To Do Soon"  
8   }, {  
9     "id": "4eea4ffc91e31d174600004b",  
10    "name": "Doing"  
11  }, {  
12    "id": "4eea4ffc91e31d174600004c",  
13    "name": "Done"  
14  }]  
15 }
```

Listing 2.3: JSON example.

In listing 3.6 a JSON example is shown. This is the response of

```
https://api.trello.com/1/boards/4eea4ffc91e31d1746000046?  
lists=open&list_fields=name,desc&key=PRIVATEKEY&token=TOKEN
```

The JSON in listing 3.6 starts with a curly bracket. That means the uppermost structure is an object. Here are a few key/value pairs like "id": "4eea4ffc91e31d1746000046". The key "list" has an array as value. So their value is in squared brackets. Each element of the array is an object again.

2.3 Ruby

Ruby is a modern general-purpose object-oriented programming language. Its big difference to most other languages is that it focuses on humans rather than computers.

Yukihiro Matsumoto, the designer of Ruby, said once:

Ruby is simple in appearance, but is very complex inside, just like our human body.[rub00]

That means, that Ruby is very easy readable and is intuitive for humans even so it can perform complex tasks. This is achieved with English keywords instead of brackets and curly brackets. The result for the programmer of

this consistent philosophy is a very easy to read language which is also very plain. Because of the English words instead of abstract characters Ruby is easy understandable. Even non-programmers understand mostly whats going on. So programmers produce way less errors while writing the code. A wrongly spelled word is more intuitive recognisable than a missing bracket or semicolon. [rub12a] More about Ruby can be found at <http://www.ruby-lang.org>.

2.3.1 RubyGems

Ruby has a good amount of methods and classes every Ruby installation provides. But there are hundreds of extensions for special use cases – to communicate with RESTful Web APIs for example – made by third party developers. In Ruby such extensions are called *gems*. To manage and publish these third party libraries there is the standard *RubyGems*. It provides a standard format for third party libraries for Ruby, a tool to manage the installation of gems and a server for distributing the gems. [rub12c] Some Ruby distributions are delivered with several gems. Gems can be added to an existing Ruby installation at any time.

To install an additional gem on a Unix operating system the following command can be used:

```
gem install gemname
```

Where **gemname** is the name of the respective gem. If the installation performed without errors, the gem is ready to use. [rub12d]

To use an installed gem in a Ruby script the following code at the top of the script before the code starts is necessary:

```
1 require 'gemname'
```

Listing 2.4: Using the gem *gemname*

Again, *gemname* stands for the name of the respective gem. If any gems are used in these script which are not part of the Ruby standard library, they are listed at the beginning of the related description. Further information at <http://doc.rubygems.org> and <http://rubyforge.org/projects/rubygems/>.

Ob-La-Di, Ob-La-DaAAAA in list **Doing**

1 vote 2 2 Aug 4 at 4:00 PM (past due)

2 2/3

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

TROLLOLO!

[Edit the card description.](#)

Bla

67%

- ☐ la
- ☒ le
- ☒ lt

[Add item](#)

Attachments [Attach](#)

RB [kreditrechner.rb](#)
a minute ago - 332 B
[Delete](#)

Foto_4.JPG
Jul 19 at 2:02am - 1.72 MB
[Make Cover](#) - [Delete](#)

Activity

Juuro attached [kreditrechner.rb](#) to this card
a minute ago

Juuro
This card is splendid! Wonderful!
a minute ago - [Delete](#)

Labels

[Edit Labels...](#)

Members

[Assign...](#)

Actions

- [Add checklist...](#)
- [Due date...](#)
- [Attach File...](#)
- [Move...](#)
- [Subscribe](#) ☒
- [Archive](#)

[More Actions...](#)

Figure 2.2: A opened card in Trello.

Chapter 3

Applications

3.1 Trello API wrapper

These scripts fulfill very different tasks, but they have also much in common. For example almost every script loads single cards. At least potentially. So I wrote a set of functions and classes which represent Trello for Ruby. This is kind of a translation of Trello to Ruby and vice versa. Additionally now the scripts can use the functions and in consequence they can stay very lightweight and clean. Almost everything that's possible with the Trello API is possible with this API wrapper, too. But it covers not all features, because the API is still in beta phase, so it changes quite quickly.

The API wrapper has also functions to pre-process data for Ruby. From a developers point of view, Trello is all about cards. Cards are the only things in Trello with real data, not just meta data. So if the task is so get a board from the API it means to get the cards of the board. There is an API call to get all cards which are in a specific board. But with this call the developer doesn't get all information about the cards. So the API wrapper has to execute the API call for a single card to cumulate all information about all cards of the board. This is the function of the API wrapper to keep the actual script clean. So the developer can work with the data and hasn't to worry about determining them.

3.2 Command Line Interface

Almost every script needs some informations. An information which every scripts need is key and token of the user which account should be used for the access to Trello. The scripts have to know which cards, lists and boards they have to look at. So these information has to be passed to the scripts, too. At first we set this information at the top of the script. But it emerged that it's

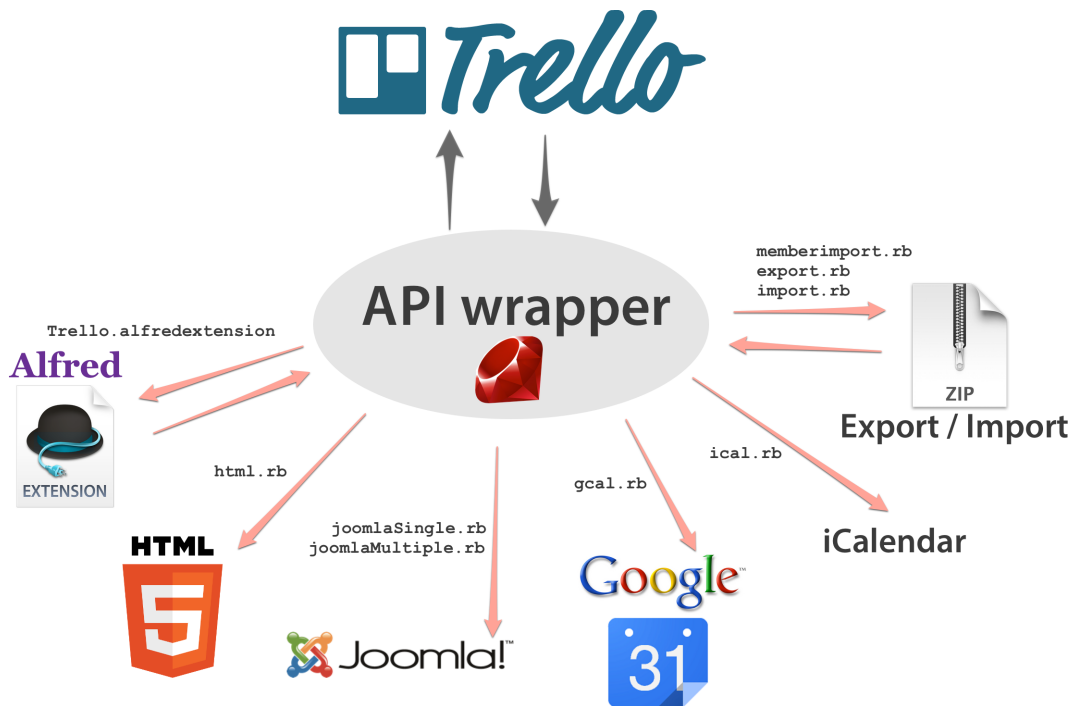


Figure 3.1: Connections between Trello, the API wrapper and the actual features. [rub][htm][joo12][goo12]

very unpractical to hard code this in each script. So it would be impossible to use the same Ruby file with several Trello accounts. For every Trello account the user has to generate a dedicated file. The solution for this problem is a command-line interface (CLI). With a CLI the user can pass information to the script in a predefined format, so the script knows exactly what to do. For every other call the user can specify different information for one and the same script.

The Ruby class `OptionParser`[rub12b] provides easy customisable command-line option analysis. The developer is able to specify its own options for each script. For this purpose a dedicated class is used. In order to let the actual script *know* about the CLI arguments the developer has to require the respective CLI class with the command-line option definitions.

```
1 ruby html.rb -c 4ffd78a2c063afeb066408b8
```

Listing 3.1: Example usage of a script with CLI.

An example usage of a script with CLI would look like Listing 3.1. The `-c` is an command-line option. If there is a string behind the option, like in this case, the string is a so called *argument*. But there are command-line options which stand for its own. Those are called *flags*. Flags are only for polar decisions.

```
1 # Trello list(s)
```



```

2 opts.on("-l", "--lists x,y,z", Array, "Ids of one or more
   Trello lists.") do |lists|
3   options.lists = lists
4 end

```

Listing 3.2: Definition of a command-line option

Listing 3.2 shows the definition of the option `-l` for passing one or more IDs of lists to a script. In Line 2 the word `Array` casts the list argument to an `Array` object.

`OptionParser` provides an automated help option. If the user types

```
ruby script.rb -h
```

he gets the explanation the developer wrote in the CLI class for this script with all possible options. This list is automatically generated by the definitions of the command-line options like in Listing 3.2. It works with `-help` and `--help` instead of `-h`, too.

```

1 Usage: ical.rb [options]
2 Select the input cards with -c, -l, -b or -a
3
4 Specific options:
5 -a, --[no-]all          Set this if all due dates of all
   cards of all boards this user can see shall be used.
6 -l, --lists x,y,z       Ids of one or more Trello lists.
7 -b, --boards x,y,z      Ids of one or more Trello boards.
8 -c, --cards x,y,z       Ids of one or more Trello cards.
9 -k MANDATORY, --key     Your Trello key.
10 -t MANDATORY, --token   The Trello token.

```

Listing 3.3: Output of the `-h` option.

Listing 3.3 shows the Output of `ruby ical.rb -h`. These are the basic CLI commands used by every script. For some scripts there are additional commands. They are explained in their respective sections.

3.3 Export to HTML

Used gems:

- `erb`
- `json`
- `open-uri`

- pp
- kramdown

The `html.rb` script exports the data of one or more cards to an HTML file. The resulting HTML file lists the cards one below another. The order is determined by the order in the command-line argument. If the command-line looks like listing 3.4 the script will process the list with the id `4ffd78ff7f0c71780cc5aa1c` at first. That means in the HTML file are all cards in this list and below these the single card with the id `4ffd78a2c063afeb066408b8`. In addition to the command-line options described in section 3.2 the option `--title` is used. Here the user has to specify a title for the web page. The title will be displayed at the top of the page in `<h1>` HTML tags.

```
1 ruby html.rb -l 4ffd78ff7f0c71780cc5aa1c  
  -c 4ffd78a2c063afeb066408b8 --title 'Madness'
```

Listing 3.4: Example for a `html.rb` call.

Each card is displayed with all their information. This includes title, description, members, due date, labels, votes, checklists, comments and attachments. Trello itself distinguishes between photos and other attachments. Normal attachments are linked under the description. Photos are embedded in the HTML code as thumbnails. Trello detects JPEG, GIF and PNG files as pictures and displays them as thumbnails. In addition to these formats the resulting HTML file displays TIFF, PSD, BMP and JPEG2000 as thumbnails, too. All modern Browsers support these formats.

This script generates static HTML. Of course the goal could also be reached with a dynamic solution with PHP or Ruby on Rails. But the upside is, that the user of the respective website hasn't to wait for the webserver. Dynamic websites are mostly fast in the meanwhile, but with static HTML files the developer is on the safe side. Especially if the data to be displayed doesn't change every minutes this approach pays off. The server hasn't to generate the whole data with every visit. It has just to send the static HTML files.

3.3.1 Markdown

Markdown is a small lightweight plain text formatting syntax, designed by John Gruber. It's designed for the use with blogs and CMS. In these use cases HTML is often too much. Markdown represents most of the features of HTML that are needed for writing. The designer of Markdown, had the goal that a text, written in Markdown, is still easy to read. John Gruber provides a software tool, written in the Perl programming language, that converts the Markdown formatted text to valid HTML. [mar04]

```

1 ### iCloud:
2
3 1.   Shared Photo Streams Now you can *share* just the **
      photos** you want, with just the people you choose.
4 2.   Reminder
5
6 -----
7
8 Here is an example of AppleScript:
9
10     tell application "Foo"
11         beep
12     end tell
13
14 ![Apple logo](http://upload.wikimedia.org/wikipedia/
      commons/f/fa/Apple_logo_black.svg "Apple logo")

```

Listing 3.5: Example for a text written in Markdown.

```

1 <h3>iCloud:</h3>
2
3 <ol>
4   <li>Shared Photo Streams Now you can <em>share</em>
      just the <strong>photos</strong> you want, with
      just the people you choose.</li>
5   <li>Reminder</li>
6 </ol>
7
8 <hr>
9
10 <p>Here is an example of AppleScript:</p>
11
12 <pre>
13   <code>tell application "Foo"
14     beep
15   end tell</code>
16 </pre>
17
18 <p></p>

```

Listing 3.6: Listing 3.5 converted to HTML.

Listing 3.5 shows a small example of Markdown. The `###` in line 1 is a header equal to `<h3>` in HTML. The first list item of the ordered list in line 3 contains italic and bold words. In line 6 is a horizontal line. After a normal line of text a code block starts in line 10. At the end in line 14 there is a picture with title and alt texts. After the conversion it looks like listing 3.6 in HTML. The appearance, of course, depends on the used CSS on the respective websites. The appearance in Trello is like in figure 3.2.

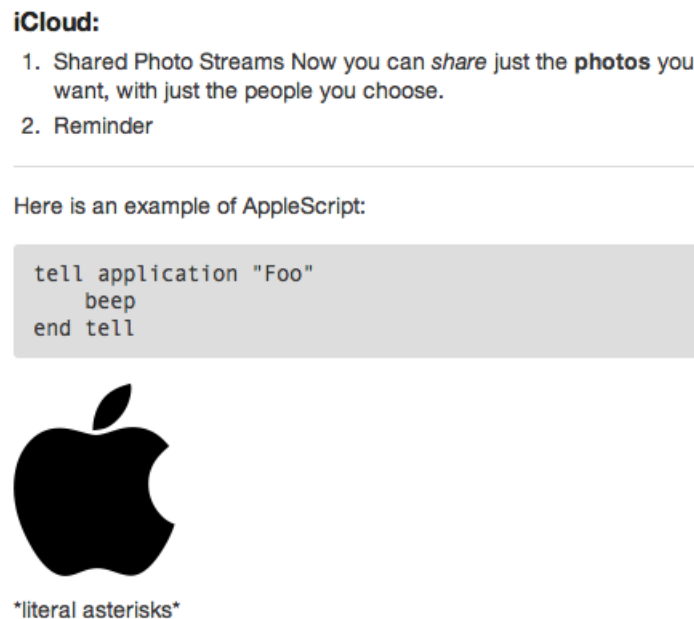


Figure 3.2: The browser view of the HTML converted from the Markdown in listing 3.5

Meanwhile Markdown became quite popular. Many blogging platforms support it, at least there are Markdown plug-ins for most platforms. Trello supports it in the description of cards. In the unlikely case that markdown reaches its limits inline HTML can be used. The only restriction is, that HTML block-level-elements have to be separated to the previous and following Markdwon blocks.

For converting Markdown to HTML the gem `kramdown` is used. Figure 3.3 describes the convert options of `kramdown`. It converts to LaTeX and a special `kramdown` format, too. The `kramdown` format is an extended Markdown syntax. As input formats it accepts HTML and `kramdown` besides standard Markdown. [kra12] These additional features of `kramdown` might be useful for future approaches. To generate lists bibliographies for scripts, papers or books.

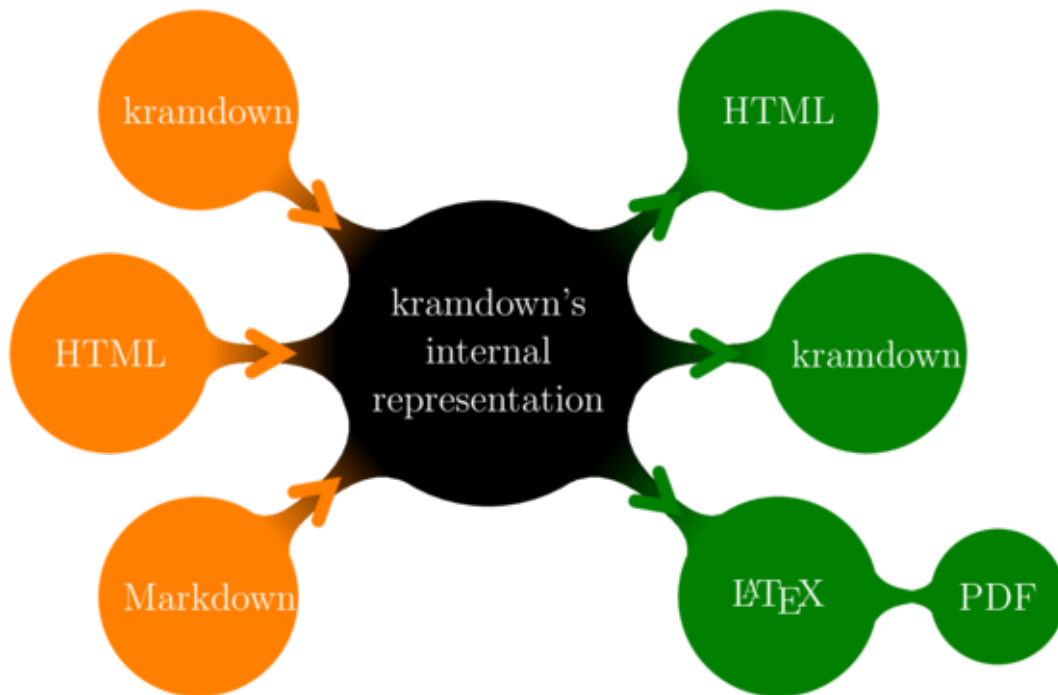


Figure 3.3: Overview about kramdowns converting options. [kra12]

3.3.2 Twitter Bootstrap Framework

In 2011 Twitter released Bootstrap¹. Bootstrap is a collection of methods and script for creating front-ends of websites. It contains templates for site structuring, tables, text, buttons, menus, forms, lists and some other often used elements of websites. Additionally some functions are supported with JavaScript. Bootstrap is written in HTML, JavaScript, CSS and LESS. LESS is a dynamic stylesheet language and extends CSS. That allows the use of variables, functions, nested selectors and operators.[les12] Bootstrap is completely free of charge and open source. [boo12]

Bootstrap evolved at Twitter while working on several projects with different libraries. The projects became inconsistent and a high administrative effort was needed. Some developers at Twitter lead by Mark Otto worked on something to document and share common design within the company. Twitter determined that this toolkit could be more than an intern helper tool. So they rounded it up with all common function which are needed for modern web development and released it on GitHub. [mar12]

¹Blog post by Mark Otto: <https://dev.twitter.com/blog/bootstrap-twitter>

3.3.3 HTML5

Twitter Bootstrap supports HTML5. HTML is the markup language for displaying content in a web browser. HTML5 is the latest revision of the HTML standard. It's an open format developed by the World Wide Web Consortium (W3C). HTML5 is a working draft since 2008. In 2011 the HTML Working Group at W3C advanced HTML5 to Last Call. So right now HTML communities all over the world are asked to confirm the standard. It is estimated that HTML5 will reach W3C Recommendation by 2014.[\[htm11\]](#) Although it is not yet finished, it is already widely used.

Before HTML5 there were two popular standards: HTML4 and XHTML1. XHTML1 defines a XML serialisation for HTML4. With HTML5 there is only one language called HTML. This language can be written in HTML syntax and XML syntax. [\[htm12d\]](#) The W3C realised early the importance of smartphone in the future. They guided the development of HTML5 with the consideration of being able to run on low-powered devices. The market share of HTML5 enabled devices is still rising. [\[sma11\]](#) Smartphone sales even beat PC sales in 2012. Thus, it is increasingly important that web apps work well on small devices. To ensure this HTML5 is used. The foundation created here can therefore be used as a basis for the future.

HTML5 introduces several new elements. Some of them are used in `html.rb`. At first there is the `<header>` element. A header is an area at the top of a web page. It is used for navigational aids, logos or a search bar. [\[htm12a\]](#) At the bottom is the `<footer>` element. The `<footer>` element can be used as a side wide footer and as a footer of sections. [\[htm12b\]](#) Here it is just used as side wide footer as preparation for data like licencing information, imprint and informations about the author. The cards are enclosed in `<article>` elements. Every single card is represented as an `<article>` element. That makes sense because typically a card is an article, especially if the user wants the card to be displayed on a web page. The W3C writes as requirements for a `<article>` element that it has to be self-contained. [\[htm12c\]](#) A card is a collection of information which is ordered by the several types of data.

This foundation of generating HTML out of Trello can be used in various ways. It's predestinated for website with recurring kinds of information. Every business card style web page can definitely be managed with this script. From very small websites through to whole blogs could build on it.

3.3.4 Templating with ERB

Every card is embedded in the same HTML structure. Templates specify this HTML without the actual data. Instead of the data there is just a wildcard and a few control structures. Templating systems are used to organise the source code in operationally-distinct layers. The design is completely handled

in the template file. But the control structure is partially in the template file, too. For a true separation of presentation of the data models and the logic components template engines, however, are unsuitable and there are additional concepts such as Model View Controller required (MVC).

ERB is a templating system for Ruby. It's part of the Ruby standard library. ERB accepts every string as a template, no matter if it is stored in a file, a database or some other kind of storage. ERB mainly used for generating HTML files. It is also able to generate any other kind of structured text, like RSS feeds and XML. [erb11] [erb12]

While generating the HTML ERB copies the plain text portions of the template file directly to the resulting document. The parts which ERB has to process are marked with certain tags listed in listing 3.7

```

1 <% Ruby code -- inline with output %>
2 <%= Ruby expression -- replace with result %>
3 <## comment -- ignored -- useful in testing %>
4 % a line of Ruby code -- treated as <% line %>
5 %% replaced with % if first thing on a line and %
   processing is used
6 <%% or %%> -- replace with <% or %> respectively

```

Listing 3.7: Recognised tags in ERB.

Only the first two tags are used here. The optimum would be if only <%= Ruby expression -- replace with result %> would be used. Because that would imply that there are no control structures in the template file, just wildcards for data.

An real example of an ERB template is listing 3.8.

```

1 <small><%= getDate(card['due'], format='de') %></small>

```

Listing 3.8: Ruby method in ERB template.

A Ruby method is used as wildcard here. When processing the template file the method will be executed with the given variables and the result will be copied in the HTML file. But there are control structures, too. The block around the line in listing 3.8 i showed in listing 3.9

```

1 <% if card['due'] %>
2   <small><%= getDate(card['due'], format='de') %></small>
3 <% end %>

```

Listing 3.9: Ruby method in ERB template.

The *if* construct is embedded with the <% ... %> tag for ERB. Thats because it's not a wildcars which is replaced with actual content. This tag is just for control structures.

The alternative to using a templating system is to write the HTML at the same time and in the same file when the data is processed. That would result in a very confusing file which produces equally confusing HTML code. If the developer makes sure that the HTML code is well-structured the source code in the script looks even messier. That's because character escape codes like `\t` and `\n` have to be inserted manually in the source code. Otherwise the templating system takes this task. An example of such a confusing mixing of HTML, character escape codes and Ruby is shown in 3.10.

```

1 htmlSite << "</strong></span></p>
2   \t\t\t\t\t<div style=\"text-align: left; padding-left: 5
3     px;\"><span style=\"font-size: xx-small;\">
4 htmlSite << description
5 htmlSite << "</span></div>
6   \t\t\t\t\t<div style=\"text-align: left;\"><span style=\"
7     font-weight: normal; font-size: small;\">
8   \t\t\t\t\t<ul>
9 if element.attachments != []
10   attachments.each do |attachment|
11     name = attachment.name
12     url = attachment.url
13     htmlSite << "\t\t\t\t\t\t\t\t\t\t<li><a href=\"\"
14       htmlSite << url
15       htmlSite << "\">
16       htmlSite << name
17       htmlSite << "<a/></li>
18   end
19 end

```

Listing 3.10: Generating HTML without a templating engine.

To represent the list of cards with the title in Ruby there is the Ruby class `webpage`. It is defined in listing 3.11.

```

1 class Webpage
2   def initialize( title )
3     @title = title
4
5     @cards = [ ]
6   end
7
8   def add_card( card )
9     @cards << card
10  end
11
12  def get_binding
13    binding

```



```
14   end
15 end
```

Listing 3.11: Generating HTML without a templating engine.

There are three methods. The `initialize(title)` method generates the actual instance of the class. The instances modeled after this method contain the given title and an empty array for the cards. The `add_card(card)` method simply adds a new card to the `@cards` array. The last method is `get_binding`. It generates a `Binding` object of the current local variables.

```
1 templateFile = File.open("templateHtml.html.erb", "rb")
2 template = templateFile.read
3
4 rhtml = ERB.new(template)
5
6 webpage = Webpage.new( @htmlTitle )
7
8 cardsFull.each do |card|
9   webpage.add_card(card)
10 end
11
12 html = rhtml.result(webpage.get_binding)
13
14 fileHtml = File.new("index.html", "w+")
15 fileHtml.puts html
16 fileHtml.close()
```

Listing 3.12: Generating HTML with ERB.

In listing 3.12 the template data is set up. At first in line 1 the template file is opened in the next line read and saved in the `template` variable. So the template is saved as string in `template`. In line 4 with `rhtml` an instance of ERB is created. After that in line 6 the `Webpage` class is used. One instance with the given title is generated. In line 8 each card is added to the instance of `Webpage`. Finally in line 12 the `Binding` object of `webpage` is created. With the ERB method `result` the data in the `Binding` object and the template come together. This is the step where the wildcards in the template file get filled with the actual data of the `Binding` object. The resulting HTML code is saved in the string variable `html` and `html` is saved to the file `index.html` in line 14.

3.4 One way sync to Google Calendar

3.5 Export to iCalendar

3.6 One way sync to Joomla

3.6.1 For every card an article

3.6.2 All cards in one article

3.7 One way sync to WordPress

3.8 Backup

3.8.1 Export

3.8.2 Import

Filename option

The `-n` (or `-name`) argument for this script stands for the filename of the backup file which contains the exported Trello data. With `-n` the user can specify a file to import. While processing the script first checks if the user has passed this argument. If not, it aborts. If the `-n` argument is given, the script proves if the file is a ZIP file. For that it doesn't use the filename but the MIME type of the file.

```

1 if 'file -Ib #{@filename}'.gsub(/;.*\n/, "") != "
   application/zip"
2   puts "ERROR: The backup\index{Backup} file has to be a
      ZIP\index{ZIP} file!"
3   abort
4 end

```

Listing 3.13: Checking if the file has the MIME type “application/zip”

In line 1 the `file -Ib #{@filename}` is a bash call for receiving the MIME type of a file. Ruby executes it and with the `gsub`-Method it cuts the MIME part out of the received string. This shell script part in a ruby file is a bit dirty. But only for this small case it would be elaborately to use a separate gem.

TODO: What's a MIME type?

3.8.3 Member import

3.8.4 Close all boards

`closeboards.rb` is more of a helper tool for developers. While developing for Trello emerge several Testaccounts, which get crowded with test boards now and then. The execution of this script will close all boards in the specified Trello account. The CLI isn't employed here because it would be too dangerous. If accidentally the wrong account would be indicated all the boards were gone.

Chapter 4

Conclusion

Chapter 5

Outlook

5.1 Trello Alfred Extension

Alfred [\[alf12\]](#) is a small Mac application which simplifies the way one can search the web or access all sorts of applications. It consists just of a input field which one can access with a keystroke combination. It's like an extended Spotlight (on Mac) or Windows Search (on Windows). Developers can write extensions to access other webservices and applications with Alfred. It's even possible to run scripts with Alfred. With that possibility given it's perfect for accessing Trello while working in a fast and easy way.

There are three commands to add or read cards with this extension:

1. `trello board-name` will return the card-names and statuses of this board.
2. `trello board-name list-name` will return card-names and statuses of this list in this board.
3. `trello board-name text for a new card` will add a new card with the specified text to the first list of this board.
4. `trello board-name list-name text for a new card` will add a new card with the specified text to this list of this board.

If you enter `trello Berlin Visit the Reichstag` in Alfred the extension looks for a board called *Berlin*. If it finds nothing it looks for *Berlin Visit* and so on. So your board names shouldn't end with an imperative. The thought behind this operating principle is that it's very unlikely that a board name ends with an imperative and that imperatives are often used for card titles because cards are sort of a command.

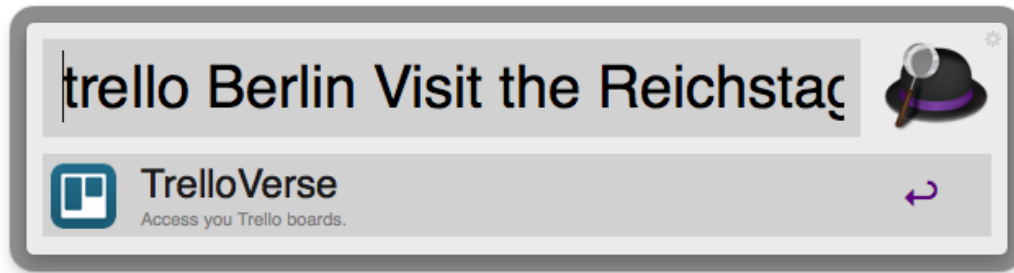


Figure 5.1: Alfred Extension for Trello: This command would add a card with the name *Visit the Reichstag* to the board called *Berlin*.

If you omit the text after the board name the extension will show you all card names of this board and its statuses.

Sometimes there are several boards with similar board names. In this case the extension will pick the “last” match. So if you have two boards called *Berlin* and *Berlin sightseeing* the extension will pick *Berlin sightseeing*. This approach makes sense because if the extension would pick the first match, in this case *Berlin*, it wouldn’t be possible to access *Berlin sightseeing*. In the case that one wants to access *Berlin* and add a new card beginning with *sightseeing* one has to put this board name between tick marks.

TODO: Code this and verify the practicability.

5.2 Native applications

Although Trello is an extremely good web-app, I’m of the opinion that a native application is always the better solution. The first reason is because it’s a dedicated app and so it’s integrated with the operating system. Especially for todo-applications it’s an advantage that they can access the systems notification system, or that they could completely vanish in the background so they don’t bother the user while working. There are mobile applications for iOS [tre12a] and Android [tre12c] by Trello itself. But there’s no Mac, Windows or Linux application.

A native application would even speed up the Alfred extension because the application could cache the data. So there hasn’t to be an actual HTTP request for every command by the Alfred extension. And if a HTTP request is necessary the user hasn’t to wait because the application will handle the command in the background.

Bibliography

- [alf12] Alfred app. <http://www.alfredapp.com/>, 08 2012.
- [boo12] Twitter bootstrap. <http://twitter.github.com/bootstrap/>, 08 2012.
- [erb11] An introduction to erb templating. <http://www.stuartellis.eu/articles/erb/>, 02 2011.
- [erb12] Class: Erb (ruby 1.9.3). <http://ruby-doc.org/stdlib-1.9.3/libdoc/erb/rdoc/ERB.html>, 08 2012.
- [goo12] <http://www.google.com>, 08 2012.
- [htm] W3c html5 logo. <http://www.w3.org/html/logo/>.
- [htm11] W3c invites broad review of html5. <http://www.w3.org/2011/05/html51c-pr.html>, 05 2011.
- [htm12a] 4.4 sections — html standard. <http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-footer-element>, 08 2012.
- [htm12b] 4.4 sections — html standard. <http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-footer-element>, 08 2012.
- [htm12c] 4.4 sections — html standard. <http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-article-element>, 08 2012.
- [htm12d] Html5 differences from html4. <http://www.w3.org/TR/html5-diff/>, 03 2012.
- [joo12] www.joomla.de. <http://www.joomla.de/>, 08 2012.
- [jso12] Json. <http://www.json.org/>, 08 2012.
- [kra12] kramdown. <http://kramdown.rubyforge.org/>, 06 2012.

- [les12] Less the dynamic stylesheet language. <http://lesscss.org/>, 08 2012.
- [mar04] Daring fireball: Markdown. <http://daringfireball.net/projects/markdown/>, 12 2004.
- [mar12] Bootstrap in a list apart no. 342 · deep thoughts by mark otto. <http://www.markdotto.com/2012/01/17/bootstrap-in-a-list-apart-342/>, 01 2012.
- [oau12] OAuth community site. <http://oauth.net/>, 08 2012.
- [res12] Restful web services: The basics. <https://www.ibm.com/developerworks/webservices/library/ws-restful/>, 08 2012.
- [rub] Iap: Caffeinated crash course in ruby. <http://sipb.mit.edu/iap/ruby/>.
- [rub00] [ruby-talk:02773] re: More code browsing questions. <http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/2773>, 05 2000.
- [rub12a] About ruby. <http://www.ruby-lang.org/en/about/>, 08 2012.
- [rub12b] Class: Optionparser (ruby 1.9.3). <http://ruby-doc.org/stdlib-1.9.3/libdoc/optparse/rdoc/OptionParser.html>, 08 2012.
- [rub12c] Rubyforge: Rubygems: Projektinfo. <http://rubyforge.org/projects/rubygems/>, 08 2012.
- [rub12d] Rubygems manuals. <http://docs.rubygems.org/>, 08 2012.
- [sma11] One billion html5 phones to be sold worldwide in 2013. <http://www.strategyanalytics.com/default.aspx?mod=pressreleaseviewer&a0=5145>, 12 2011.
- [tre12a] App store - trello. <http://itunes.apple.com/us/app/trello/id461504587?mt=8>, 08 2012.
- [tre12b] Getting started — trello documentation. <https://trello.com/docs/gettingstarted/index.html>, 08 2012.
- [tre12c] Trello android app available for download! — trello blog. <http://blog.trello.com/trello-android-app-available-for-download/>, 08 2012.
- [tre12d] Trello documentation — trello documentation. <https://trello.com/docs/index.html>, 08 2012.

Index

AJAX, [4](#)
Alfred, [21](#), [22](#)
 Extension, [21](#), [22](#)
Android, [4](#), [22](#)
API, [4](#), [5](#), [9](#)

Backup, [15](#)
Bootstrap, [15](#)

CLI, [9](#)
Command-Line Interface, [9](#)
CSS
 [3](#), [15](#)

ERB, [15](#)
Export, [15](#)

Fog Creek Software, [3](#)

Google
 Calendar, [15](#)

HTML
 [5](#), [4](#), [15](#)

iCalendar, [15](#)
Import, [15](#)
iOS, [4](#), [22](#)

Joomla, [15](#)
JSON, [5](#)

Linux, [22](#)

Mac, [21](#), [22](#)
Markdown, [12](#)
MIME, [16](#)

Ruby, [6](#)
RubyGems, [6](#)

Templating, [15](#)
Trello, [3–5](#), [9](#), [21](#), [22](#)
Twitter, [15](#)

Windows, [22](#)
 Search, [21](#)
WordPress, [15](#)

ZIP, [16](#)

Statement of authorship

I certify that I have prepared this thesis independently and that I'm using only the tools mentioned here. All passages that have been taken from other works are characterized as borrowing. This thesis has been submitted in identical or similar form in any other program as an examination.

Place, Date

Signature