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

Diplom Thesis Informatics

TODO: ? Framework for connecting Trello to other services and applications

Sebastian Engel

11th September 2012

Referees

Prof. Dr. Torsten Grust
(Informatik)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Name Zweitgutachter
( )
Medizinische Fakultät
Universität Tübingen

Engel, Sebastian:
Framework for connecting Trello to other services and applications
Diplom Thesis Informatics
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 fo 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 webservices 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

Thanks to Prof. Grust and M.Sc. Tom Schreiber for the oppurtunity to work on this interesting project. A great thank you to my lectors Alena Dausacker and Jennifer Proehl. Thanks to Moritz Uhlig for introducing me to HTML in the first place. Thanks to Sabrina Pfeffer, Martin Lahl, Philipp Wolter, Thomas Zappe and Steffen Zietkowski for the support during my studies. Thanks to my parents for the financial support and the patience.

Contents

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 Ruby . . . 6

2.2.1 RubyGems . . . 7

2.3 JSON . . . 8

3 Trello API wrapper **11**

3.1 Handling of date and time . . . 12

4 Applications **13**

4.1 Command Line Interface . . . 13

4.2 Export to HTML . . . 14

4.2.1 Markdown . . . 15

4.2.2 Twitter Bootstrap Framework . . . 18

4.2.3 HTML5 . . . 19

4.2.4 Templating with ERB . . . 19

4.3 Sync Google Calendar . . . 23

4.4 Export to iCalendar 26
4.5 Sync to Joomla 28
4.6 Backup 28
4.6.1 Export 28
4.6.2 Import 28
4.6.3 Member import 29
4.6.4 Close all boards 29

5 Conclusion **31**

6 Outlook **33**
6.1 Trello Alfred Extension 33
6.2 Native applications 34

Bibliography **35**
Index 39

List of Figures

2.1 A Trello board. 3
2.2 A opened card in Trello. 10

3.1 Connections between Trello, the API wrapper and the actual features. [rub][htm][joo12][goo12a] 11

4.1 The browser view of the HTML converted from the Markdown in listing 4.5 17
4.2 Overview about kramdowns converting options. [kra12] 18

6.1 Alfred Extension for Trello: This command would add a card with the name *Visit the Reichstag* to the board called *Berlin*. . 34

List of Listings

2.1 GET request using RestClient. 5
2.2 POST request using RestClient. 6
2.3 POST request with open-uri. 6
2.4 Using the gem *gemname* 7
2.5 JSON example. 8
3.1 Response of the token request. 12
4.1 Example usage of a script with CLI. 13
4.2 Definition of a command-line option 13
4.3 Output of the `-h` option. 14
4.4 Example for a `html.rb` call. 15
4.5 Example for a text written in Markdown. 16
4.6 Listing 4.5 converted to HTML. 16
4.7 Recognised tags in ERB. 20
4.8 Ruby method in ERB template. 20
4.9 Ruby method in ERB template. 20
4.10 Generating HTML without a templating engine. 21
4.11 Generating HTML without a templating engine. 21
4.12 Generating HTML with ERB. 22
4.13 Initialisation of the Google Calendar API connection. 23
4.14 Response of the token request. 24
4.15 Adding a new event to Google Calendar. 24
4.16 iCalendar example. 27

4.17 Checking if the file has the MIME type “application/zip” 29

Nomenclature

API	Application Programming Interface
CLI	Command Line Interface
CMS	Content Management System
ERB	embedded Ruby
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
MVC	Model View Controller
REST	Representational State Transfer
RFC	Request For Comments
RSS	Really Simple Syndication
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTC	Universal Time Coordinated
XHTML	Extensible Hyper Text Markup Language

XML	Extensible Markup Language

Chapter 1

Introduction

Die Arbeit gliedert sich dazu wie folgt: Die Grundlagen von BlaBlaBla werden in Kapitel 2 erarbeitet. ... Eine Diskussion und ein kurzer Ausblick im Kapitel **??** beschliesen diese Arbeit.

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

TODO: !

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 Software[1]. It is a collaboration tool to manage projects, launched in 2011[2].

Figure 2.1: A Trello board.

There is the concept of so called *boards* which contents 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 asigned members, a due date, some

[1] Official Fog Creek Software website: http://www.fogcreek.com
[2] The 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 everone who's working on a project can see whats 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. Thats 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]

Authentification

Though the scripts which are used here need access to private boards in Trello there has to be any kind of authentification. 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. Whith this key the user can generate a token. This token will be send 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 expriations 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 a GET URL of a RESTful web service looks like this:

`https://api.trello.com/1/cards/4fc8dd3e1b9ecf0c3571902f? key='PRIVATEKEY&token=TOKEN`

This is a GET request to get a specific card with the id `4fc8dd3e1b9ecf0c3571902f`. 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. This GET request with the RestClient gem in Ruby looks like 2.1.

```
1 member = RestClient.get('https://api.trello.com/1/cards/4
    fc8dd3e1b9ecf0c3571902f?key='+$key+'&token='+$token)
2 pp JSON.parse(member)
```

Listing 2.1: GET request using RestClient.

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.

```
response = RestClient.post(
  'https://api.trello.com/1/boards',
  :name => board['name'],
  :desc => board['desc'],
  :key => $key,
  :token => $token
)
```

Listing 2.2: POST request using RestClient.

```
uri = URI('https://api.trello.com/1/boards')
req = Net::HTTP::Post.new(uri.path)

req.set_form_data(
  'name' => board['name'],
  'desc' => board['desc'],
  'key'=>$key,
  'token'=>$token
)

Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.
   scheme == 'https') do |http|
  response = http.request(req)
end
```

Listing 2.3: POST request with open-uri.

Listing 2.2 and listing 2.3 show the very same API call. But 2.2 is realised with RestClient and listing 2.3 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 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 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.

2.2.1 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 theres 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/.

2.3 JSON

All the responses to Trello API calls use JSON[3]. It is a subset of the JavaScript programming language. Despite its relation to JavaScript it's language independent. JSON is da 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 an 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 suqred brackets. The values are separated by commas. Both can be arbitrary nested. At every point one of my script 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`. [jso12a]

```
{
    "id": "4eea4ffc91e31d1746000046",
    "name": "Example Board",
    "desc": "This board is used in the API examples",
    "lists": [{
        "id": "4eea4ffc91e31d174600004a",
        "name": "To Do Soon"
    }, {
        "id": "4eea4ffc91e31d174600004b",
        "name": "Doing"
    }, {
        "id": "4eea4ffc91e31d174600004c",
        "name": "Done"
    }]
}
```

Listing 2.5: JSON example.

In listing 4.6 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 4.6 starts with a curly bracket. That means the uppermost structure is an object. Here are a few key/value pairs like `"id":`

[3]RFC document for JSON: The application/json Media Type for JavaScript Object Notation (JSON) http://www.ietf.org/rfc/rfc4627.txt

`"4eea4ffc91e31d1746000046"`. The key `"list"` has an array as value. So their value is in suqared brakets. Each element of the array is an object again. To process the received JSON Ruby needs to parse it first. For that purpose there is the a gem simply called *JSON*. It parses the JSON with the

```
JSON.parse(receivedJson)
```

command, where `receivedJson` is a variable that contains the plain JSON received from the Trello API with a GET request. Aufter the parsing the JSOn objects are now represented as Ruby Hashes and the JSON arrays as Ruby arrays. To post content to the Trello API it has to be in JSON, too. So there is another method of the JSON class in Ruby which generates JSON from Ruby hashes and arrays.

```
JSON.generate(hashBoards)
```

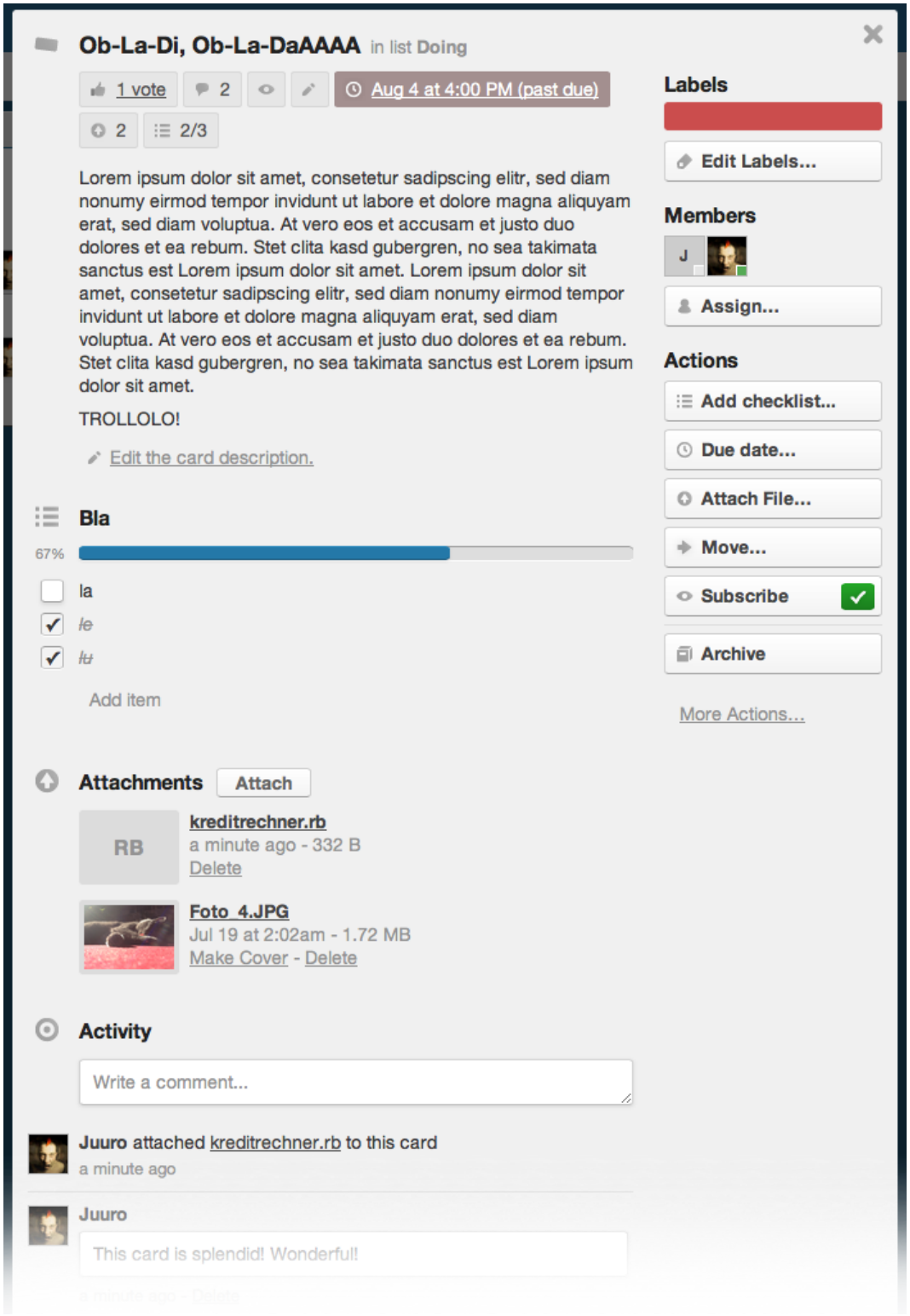

Figure 2.2: A opened card in Trello.

Chapter 3

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 whith 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.

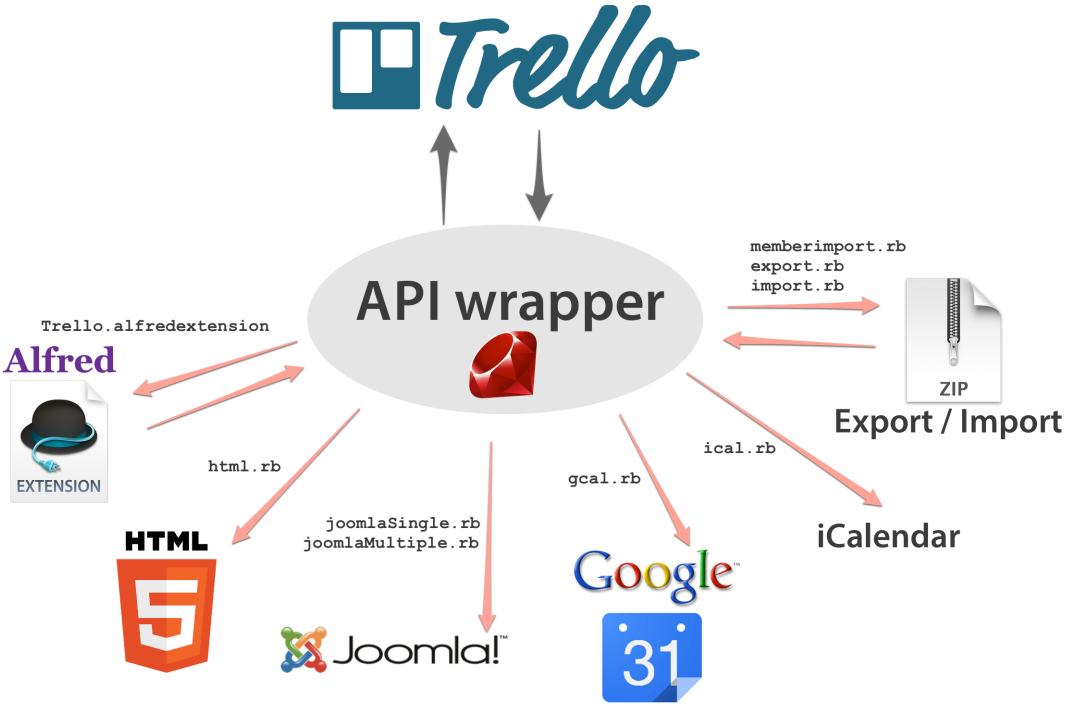

Figure 3.1: Connections between Trello, the API wrapper and the actual features. [rub][htm][joo12][goo12a]

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 cummulate 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.

3.1 Handling of date and time

The cards in Trello may have set due dates. A due date is a date at which the creator of the card thinks it should be done. The due date is represented by the Trello API as an ISO 8601 formatted string. The timezone of the date is UTC. To ensure that the correct time is displayed always the date has to be adapted to the local time. That's perfeormed with the code in listing 3.1.

```
1 fdate = Time.iso8601(date).getlocal
```

Listing 3.1: Response of the token request.

It parses the given date string in ISO 8601 format to a Ruby Time object. The `getlocal` is the important part here. This function of the Time class determines server's time zone and readjusts the time accordingly. For this function working as intended it's important that the correct time zone is set on the used server.

TODO: Describe getDate

Chapter 4

Applications

4.1 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 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 4.1: Example usage of a script with CLI.

An example usage of a script with CLI would look like Listing 4.1. The `-c` is an comman-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 4.2: Definition of a command-line option

Listing 4.2 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.

OptionParse 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 ist automatically generated by the definitions of the command-line options like in Listing 4.2. 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 4.3: Output of the `-h` option.

Listing 4.3 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.

4.2 Export to HTML

Used libraries:

- `erb`
- `json`

- `rest_client`
- `pp`
- `kramdown`

The `html.rb` script exports the data of one ore 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 4.4 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 4.1 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 4.4: 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 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.

4.2.1 Markdown

Markdown is a small lightweight plain text formatting snytax, 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]

```
### iCloud:

1.   Shared Photo Streams Now you can *share* just the **
   photos** you want, with just the people you choose.
2.   Reminder

---------------------------------

Here is an example of AppleScript:

    tell application "Foo"
        beep
    end tell

![Apple logo](http://upload.wikimedia.org/wikipedia/
   commons/f/fa/Apple_logo_black.svg "Apple logo")
```

Listing 4.5: Example for a text written in Markdown.

```
<h3>iCloud:</h3>

<ol>
  <li>Shared Photo Streams Now you can <em>share</em>
      just the <strong>photos</strong> you want, with
     just the people you choose.</li>
  <li>Reminder</li>
</ol>

<hr>

<p>Here is an example of AppleScript:</p>

<pre>
  <code>tell application "Foo"
  beep
  end tell</code>
</pre>

<p><img alt="Apple logo" title="Apple logo" src="http
   ://upload.wikimedia.org/wikipedia/commons/f/fa/
```

```
Apple_logo_black.svg"></p>
```

Listing 4.6: Listing 4.5 converted to HTML.

Listing 4.5 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 4.6 in HTML. The appearance, of course, depends on the used CSS on the respective websites. The appearance in Trello is like in figure 4.1.

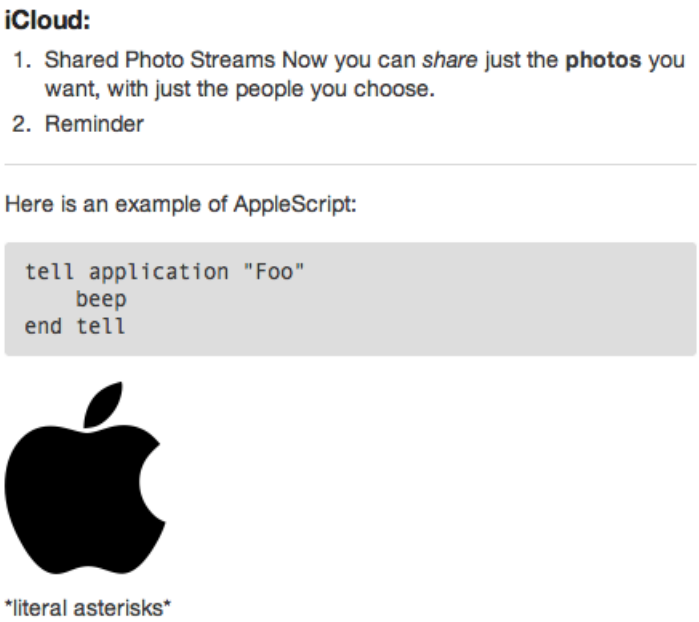

Figure 4.1: The browser view of the HTML converted from the Markdown in listing 4.5

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 ist, 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 4.2 decribes 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.

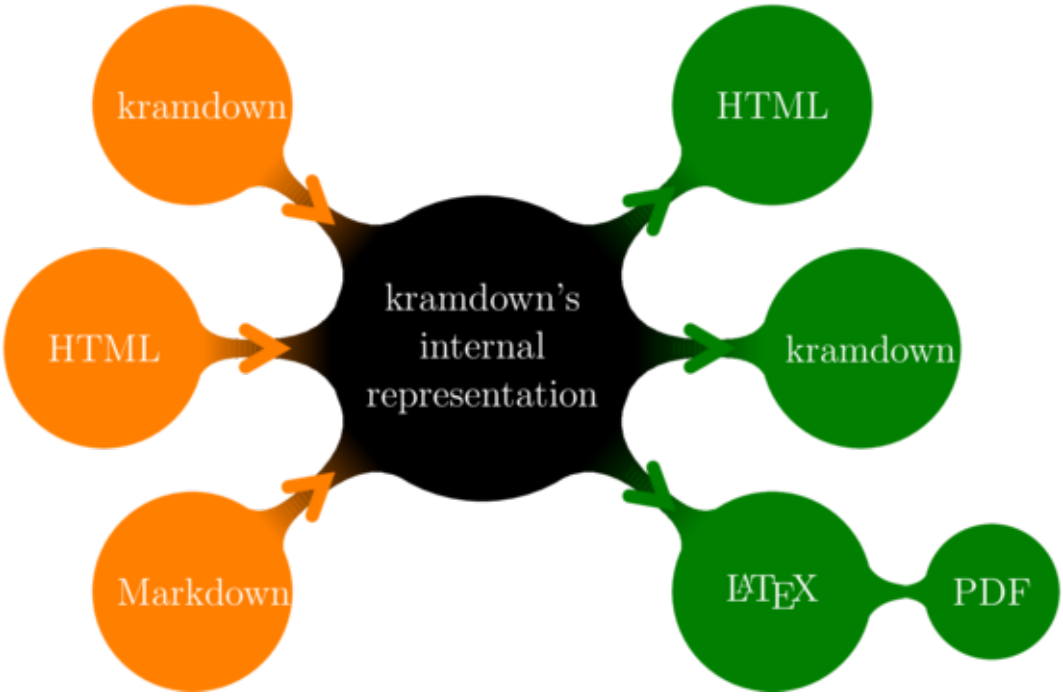

Figure 4.2: Overview about kramdowns converting options. [kra12]

4.2.2 Twitter Bootstrap Framework

In 2011 Twitter released Bootstrap[1]. 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. Additionaly 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 varables, 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 rouned it up with all common function which are needed for modern web development and released it on GitHub. [mar12]

[1]Blog post form the lead developer about the launch of Twitter Bootstrap: https://dev.twitter.com/blog/bootstrap-twitter

4.2.3 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 starndard. 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 og 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 the are usen 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 bootom 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 typicalle 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 is 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 recuring 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.

4.2.4 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 4.7

```
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 4.7: 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 4.8.

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

Listing 4.8: 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 4.8 i showed in listing 4.9

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

Listing 4.9: 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 4.10.

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

Listing 4.10: 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 4.11.

```
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 4.11: 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 4.12: Generating HTML with ERB.

In listing 4.12 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.

4.3 Sync Google Calendar

Used libraries:

- `erb`
- `json`
- `rest_client`
- `pp`
- `google/api_client`

Google Calendar is a free web service by Google for time-management. The service can be enabled in several calendar applications such as Apple Calendar (it was called iCal prior to Mac OS X 10.8) and Microsoft Outlook. Even all important mobile operating systems support it. Google Calendar it one of the most popular calendar web services. One advantage over other offferings is the excellent integration with all the other Google services which of most are very popular, too.

Google provides an API to access Calendar. There is even an API wrapper for Ruby made by Google. But ether it is very buggy or the documentation is poorly written. Some parameters the documentation says are available to send an API call at aren't actually available. Google grants normal developers a courtesy limit of 10,000 requests per day. Developers who need more requests per day for their application have to negotiate with Google and to contract about a higher request rate.

```
1  client = Google::APIClient.new
2  client.authorization.client_id = ' '
3  client.authorization.client_secret = ' '
4  client.authorization.scope = 'https://www.googleapis.com/
       auth/calendar'
5  client.authorization.refresh_token = ' '
6  client.authorization.access_token = ' '
7
8  result = client.authorization.fetch_access_token!
9  client.authorization.access_token = result['access_token'
       ]
10
11 service = client.discovered_api('calendar', 'v3')
```

Listing 4.13: Initialisation of the Google Calendar API connection.

Authetication with Google is much more complicated than with Trello. Listing 4.13 shows the initialisation of the Google Calendar API connection. At first the project has to be registered in the Google APIs Console. [goo12c] There the developer can get the `client_id` and the `client_secret`. The `scope` depends on the Google APi the developer wants to use. Here it is `https://www.googleapis.com/auth/calendar` of course. [goo12b] To get the `access_token` this URL must be called:

```
https://accounts.google.com/o/oauth2/auth?scope=https%3A%2F%2Fwww.
googleapis.com%2Fauth%2Fcalendar&redirect_uri=https%3A%2F%2Foauth2-
login-demo.appspot.com%2Fcode&response_type=code&client_id=CLIENTID.
apps.googleusercontent.com&access_type=offline
```

If the request succeeds the response is as noted in listing 4.14.

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in":3920,
  "token_type":"Bearer",
  "refresh_token":"1/xEoDL4iW3cxlI7yDbSRFYNG01kVKM2C-259
     HOF2aQbI"
}
```

Listing 4.14: Response of the token request.

The access token is the actual important value. But it expires after about an hour typically. If it is used after the expiration date the API will respond with an error. beacuse of that it's important to store the refresh token. Otherwise, the user has to enter his Google login data every time an access token expires. If the application looses the refresh token, the API calls will no longer work. No new access tokens can be generated. The user, or in this case the developer, has to obtain a refresh token manually again. [goo12d] Unfornately, how long the refresh tokens are valid is unknown.

For the sync to Google Calendar only cards with a due dated are considered. At first the script instructs the API wrapper to get all cards. Sadly the Trello API provides no filter method to get only the cards with a due date. The next step is to check if the cards have due dates. If a card has a due date the script checks if this card is already added as an event in Google Calendar. To do so the Google library looks after this card on the basis of its id. If it's not already in Google Calendar the script has to add a new event to Google Calendar.

```
event = {
  'summary' => card['name'],
  'description' => card['desc'],
  'location' => card['id'],
```

```
5     'start' => {
6       'dateTime' => getDate(card['due'], format='iso8601'),
7       'timeZone' => 'Europe/Berlin'
8     },
9     'end' => {
10      'dateTime' => getDate(card['due'], format='iso8601'),
11      'timeZone' => 'Europe/Berlin'
12    }
13  }
14
15  insertevent = client.execute(:api_method => service.
        events.insert,
16                  :parameters => {'calendarId' => 'primary'
                      },
17                  :body => JSON.generate(event),
18                  :headers => {'Content-Type' => '
                      application/json'})
```

Listing 4.15: Adding a new event to Google Calendar.

At first a new `event` object has to be created. That happens in listing 4.15 in lines 1 to 13. A hash is used for that. The received information about a card from Trello is stored in a hash. So to get the information `card['KEYWORD']` is used, where `KEYWORD` stands for the respective name of the cards field that is needed. In the location field of an event the script stores the card id. That's not exactly the correct kind of data to put in a location field. But to determine if a card is already represented as an event in Google Calender the script has to use any unique identifier. Since there is no location for Trello cards anyway, the location field can be used. Sadly Google doesn't provide any *hidden* fields which developers can use for such purposes.

From line 15 is where the actual request happens. The `service.events.insert` tells the Google API which method the developer wants to adress. [goo12d] In this case it's the method to insert a new event to an existing calendar. In line 16 the developer has to specify the id of the calendar in which the new event should apper. Clearly `primary` is not an id. `primary` stands for the primary calendar. In Google Calendar the user can create several calendars for different purposes. One for work, one for private stuff and so forth. So `primary` is a valid value, too. In the following line the body is being sent. To do that the `event` object created before has to be converted to a JSON string. This is achieved by the `generate` method of the JSON class. [jso12b] In the last line of listing 4.15 the Google API is told that the body of this request is formatted as JSON.

If the event is already inserted in the calendar the script checks if the card's summary, description or due date have changed since the last sync to Google

Calendar. But before it checks back if this event really is the representation of the actual card. The Google API provides no method to search for events with a special location. So it isn't possible to search in the location of an event. The script looks in all fields of an event for the id of the corresponding Trello card. To ensure that the currently handled card is the same the currently handled event is the representation for, the script compares the location field of the event and the card id. If the comparison matches it runs almost the same code as in listing 4.15. But this time the Google Calendar API method which is used for updating the event is called `service.events.update`.

The third possible scenario is an orphaned event in Google Calendar. If a user in Trello removes the due date of a card or the card itself, the event in Google Calendar is no longer valid. To solve this problem the script loads all cards with due dates and all events from Google Calendar. All card ids are saved in an array and all location field entries in another. Now the array with the card ids from Trello is subtracted from the array with the location fields. The resulting array contains just events which don't have acorresponding card with a due date in Trello. After that the script checks if the remaining location field entries are in the format of Trello card ids. But the only indicators that can be used are the length of the string – a Trelloc ard id has 24 characters – and if it contains only numbers and letters. TBut there's a problem with this approach. If there are other events which are not inserted by this script which have accidently location fields with 24 characters containing only numbers and letters, they will be deleted. It's not very likely that this happens. Names of cities or other typical used date in the location field don't have the length of 24 characters. GPS positions contain dots and are shorter, too. But the possibility of a match exists. The solution would be to use a dedicated calendar only for the use with this script. Or to be very careful with adding new events manually. Of course it's possible to enable this function completely. But in this case there will remain in Trello delted cards as orphaned events in Google Calendar. They have to be deleted manually.

4.4 Export to iCalendar

Used libraries:

- `icalendar`
- `date`
- `json`
- `rest_client`
- `pp`

iCalendar is a popular format to exchange calendar data of all sorts. Originally it was introduced in 1998 by Lotus and Microsoft in RFC 2445. [**?**] Meanwhile it is supported by many applications which work with events of any kind. iCalendar is the defacto standard in this field. Hence it has great compatibility with many programs.

TODO: iCalendar format

```
1  BEGIN:VCALENDAR
2  CALSCALE:GREGORIAN
3  METHOD:PUBLISH
4  PRODID:iCalendar-Ruby
5  VERSION:2.0
6  BEGIN:VTIMEZONE
7  TZID:Europe/Berlin
8  BEGIN:STANDARD
9  DTSTART:20060811T073001
10 RRULE:FREQ=YEARLY;BYMONTH=10;BYDAY=-1SU
11 TZNAME:GMT+01:00
12 TZOFFSETFROM:+0200
13 TZOFFSETTO:+0100
14 END:STANDARD
15 END:VTIMEZONE
16 BEGIN:VEVENT
17 CATEGORIES:FAMILY
18 DESCRIPTION:503965e018a5fec561e66af1
19 DTEND;TZID=Europe/Berlin:20120828T160000
20 DTSTAMP:20120829T053947
21 DTSTART;TZID=Europe/Berlin:20120828T160000
22 SEQUENCE:1
23 SUMMARY:Buy a mountainbike.
24 TRANSP:TRANSPARENT
25 UID:2012-08-29T05:39:47+02:00_400189826@juuroair
26 URL:https://trello.com/card/buy-a-mountainbike/4
      ffa4c5ce75c29032a88ea31/5
27 END:VEVENT
```

Listing 4.16: iCalendar example.

Listing 4.16 shows an example of a file in the iCalendar format.

TODO: actual export

In comparison to the Google sync, the script is much more simple. There is no need for a sync opportunity and there is no API to work with. There is only one file in the correct format to be created. That's much faster than working with two APIs and perform several API calls. The server has to compute less data and there's less traffic to the APIs while updating. But of course

the iCalendar file must be available on a server. With the Google solution there is no need for an own web server. The another bigger problem with the webserver is, that the URL is accesible for everybody. If the calendar includes critical data this is a disqualifier for the user or the company. Although it is possible to provide the server with password protection, this would only be more complicated. Either the calendaring software doesn't support .htaccess or any other web authentication technology orthe user has to enter his login information periodically. The only way to use it safely with critical data is to use it in an intranet. There won't be any need for additional protcetion. But for the domestic use and for most small companies the solution with Google should be sufficient. Google even provides an iCalendar feed itself.

4.5 Sync to Joomla

TODO: What's Joomla? -¿ Mambo...
TODO: Ruby and DBs (MySQL)?
TODO: no API -¿ writing directly in the DB
TODO: Single article ¡-¿ multiple articles
TODO: Category page
TODO: Custom CSS in Joomla

4.6 Backup

TODO: Saving JSON in files.
TODO: Just one file.
TODO: zip(py)
TODO: Use temporary space of the OS. Why?

4.6.1 Export

4.6.2 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 scipt proofes if the file is a ZIP file. For that it soesn'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 4.17: 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 seperate gem.

TODO: What's a MIME type?

TODO: Problem adding members

TODO: Import problems with comments, votes, subscriptions.

4.6.3 Member import

TODO: Solution with memberimport.rb.

4.6.4 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 5

Conclusion

TODO: Conclusion?

Chapter 6

Outlook

6.1 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 constist just of a input field which one cann 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 acessing 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 nothign 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.

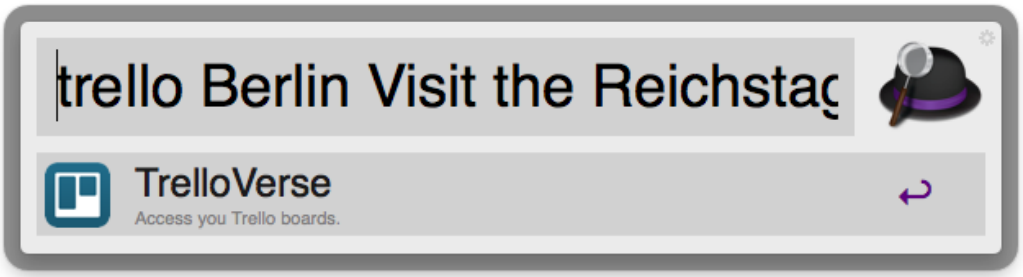

Figure 6.1: Alfred Extension for Trello: This command would add a card with the name *Visit the Reichstag* to the board called *Berlin*.

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 would 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 betweet tick marks.

TODO: Code this and verify the practicability.

6.2 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 operationg 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 theres 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 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.

[goo12a] http://www.google.com, 08 2012.

[goo12b] Frequently asked questions - google data apis — google developers. https://developers.google.com/gdata/faq#AuthScopes, 08 2012.

[goo12c] Google apis console. https://code.google.com/apis/console/, 08 2012.

[goo12d] Google calendar api - google apps platform — google developers. https://developers.google.com/google-apps/calendar/, 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/html5lc-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.

[jso12a] Json. http://www.json.org/, 08 2012.

[jso12b] Json implementation for ruby. url-http://flori.github.com/json/doc/index.html, 04 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, 31, 32
    Extension, 31, 32
Android, 4, 32
API, 4, 8, 11, 12

Backup, 27
Bootstrap, 18

CLI, 13
Command-Line Interface, 13
CSS, 18

ERB, 19
Export, 27

Fog Creek Software, 3

Google
    Calendar, 23

HTML, 18
    4, 19
    5, 4, 19

iCalendar, 26
Import, 27
iOS, 4, 32
ISO
    8601, 12

JavaScript, 18
Joomla, 27
JSON, 8

LESS, 18
Linux, 32

Mac, 31, 32
Markdown, 15
MIME, 27, 28

RSS, 20
Ruby, 6, 7
RubyGems, 7

Templating, 19
timezone, 12
Trello, 3, 4, 8, 11, 31, 32
Twitter, 18

W3C, 19
Windows, 32
    Search, 31
WordPress, 27
World Wide Web Consortium, 19

XHTML, 19
XML, 19, 20

ZIP, 27

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