Creating a

Markua parser

in Perl 5

by Gábor Szabó

Creating a Markua Parser in Perl 5

A case study of Test Driven Development in Perl

Gábor Szabó

This book is for sale at http://leanpub.com/markua-parser-in-perl5

This version was published on 2019-04-21

Also By Gábor Szabó

Contents

Preface

Markua is a Markdown-inspired language for books. It was developed and being used by LeanPub. This eBook is written in Markdown.

The eBook follows the development of a partial Markua parser in Perl 5. It starts off by setting up the Continuous Integration environment on Travis-CI and Appveyor. The Cover coverage monitoring with Coveralls. Then we start developing the parsers step-by-step using unit tests to make sure we implement each feature and that we don't break any feature that was already working properly.

The eBook, just as the parser is a work in progress. You'll get frequent updates of the eBook, with clear explanation which part is new, so you can focus on reading that part.

Changes

v0.04

Remove the newlines from the paragraph text to fit the expectatoon of Markua.

v0.03 2018-03-29

- New sections:
  * Generate test expectations for the parser
  * Parse bulleted list
  * Parse numbered list
  * Release the Markua::Parser to CPAN
  * Add attributes to Markua - round 1
  * Test coverage report with Devel::Cover for the Markua Parser

V0.02 2018-03-28

- New sections:
  * Consider everything not recognizable as a paragraph in Markua
  * Markua resources: Include files

v0.01 2018-03-24

- First release

Error reporting

While I make all the effort to make both the eBook and the code I write error and bug-free, I am quite sure there will be plenty of both of those. I welcome your error reporting on the GitHub page of the Markua Parser[1] or in e-mail to gabor@szabgab.com as you feel appropriate.

[1]https://github.com/szabgab/perl5-markua-parser

Markua Parser

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Implementing a Markua Parser in Perl 5

Markua[2] is a Magical Typewriter. It is a Markdown[3]-inspired format to write books. It was created by Peter Armstrong[4] and used by LeanPub[5] for writing books.

In this project I am going to create a Markua parser in Perl 5, or at least I start doing it and will implement enough of it so I can start writing the Perl Maven articles in Markua. That will allow me to easily include Perl Maven articles in an eBook published on LeanPub. For example to create the eBook of the Perl Maven Tutorial[6] or to create the Markua Parser in Perl 5 eBook[7]

The first few articles are available to everyone. The rest of the articles are only available to Perl Maven Pro[8] subscribers and to readers of the Markua Parser in Perl 5[9] eBook.

- Start writing the Markua parser in Perl - h1 tag * Creating Makefile.PL and a CPAN distribution for the Markua Parser * Travis-CI for the Markua Parser project * Add test coverage reporting with Coveralls to Markua Parser in Perl * Enable Appveyor (CI on Windows) for the Perl 5 Markua Parser * Process the 6 headers of Markua * Refactor test cases of the Markua Parser in Perl 5 * Collecting errors while parsing Markua in Perl 5 - disregarding empty rows * Get Coveralls to notify when test-coverage shrinks * Consider everything not recognizable as a paragraph in Markua * Markua resources: Include files * Generate test expectations for the Markua parser * Parse bulleted list in Markua * Parse numbered list in Markua * Release the Markua::Parser to CPAN * Add attributes to Markua - round 1 * Test coverage report with Devel::Cover for the Markua Parser

Start writing the Markua parser in Perl

Markua[10] is a Magical Typewriter. It is a Markdown[11]-inspired format to write books. It was created

[2]https://leanpub.com/markua/
[3]https://en.wikipedia.org/wiki/Markdown
[4]https://twitter.com/peterarmstrong
[5]https://leanpub.com/
[6]https://leanpub.com/perl-maven/
[7]https://leanpub.com/markua-parser-in-perl5
[8]https://perlmaven.com/pro
[9]https://leanpub.com/markua-parser-in-perl5
[10]https://leanpub.com/markua/
[11]https://en.wikipedia.org/wiki/Markdown

by Peter Armstrong[12] and use by LeanPub[13] for writing books.

In this project I am going to create a Markua parser in Perl 5, or at least I start doing it and will implement enough of so I can start writing the Perl Maven articles in Markua. That will allow me to easily include Perl Maven articles in an eBook published on LeanPub. For example to create the eBook of the Perl Maven Tutorial[14].

Set up Git and GitHub repository

On my local disk created a new directory called "perl5-markua-parser", and in it a README.md file which is a readme file written in Markdown format for GitHub to display nicely.

```
$ mkdir perl5-markua-parser
$ cd perl5-markua-parser

# Created README.md using vim
```

The README.md file:

```
# Markua Parser

[Markua](https://leanpub.com/markua/) is a Markdown-inspired format to write books.

This module implements parsing (part of) the Markua specification.
```

Set it up as a local git repository and committed the first change:

```
$ git init
$ git add README.md
$ git commit -m "start with a readme"
```

Then I've created a new repository on GitHub called perl5-markua-parser[15], I've told my local git repository about the remote repository, and pushed out the first changes.

```
$ git remote add origin git@github.com:szabgab/perl5-markua-parser.git
$ git push -u origin master
```

commit[16].

[12]https://twitter.com/peterarmstrong
[13]https://leanpub.com/
[14]https://www.indiegogo.com/projects/updating-the-perl-maven-tutorial/reft/775728/pmarticle0302
[15]https://github.com/szabgab/perl5-markua-parser
[16]https://github.com/szabgab/perl5-markua-parser/commit/605d7df604a819748a3ab393c44e19f59a478183

Create constructor and test it

Before we start writing the parser, let's create the skeleton of the module with a constructor and a test-case for them. I've created a directory called "lib/Markua" and a file called "Parser.pm" in it.

```
$ mkdir -p lib/Markua
```

```perl
package Markua::Parser;
use strict;
use warnings;

sub new {
    my ($class) = @_;
    my $self = bless {}, $class;
    return $self;
}


1;
```

For details read getting started with classic Perl OOP[17] or constructor in core Perl[18].

The corresponding test was saved in the new 't' directory we just created:

```
$ mkdir t
```

```perl
use strict;
use warnings;

use Test::More;
use Markua::Parser;

plan tests => 1;

my $m = Markua::Parser->new;
isa_ok $m, 'Markua::Parser';
```

Nothing fancy. Just checking if the generated object is an instance of the class.

We can run the tests by typing in

[17]https://perlmaven.com/getting-started-with-classic-perl-oop
[18]https://perlmaven.com/core-perl-oop-constructor

```
$ prove -l
```

```
$ git add .
$ git commit -m "create module with constructor and test it"
```

commit[19]

Start parsing

Before writing the parser, let's write a simple test-case for it. In the 't' directory I've created a subdirectory called 'input' where we are going to store the sample input files.

```
$ mkdir t/input
```

In there I've created a simple Markua file:

```
# Heading One
```

The parser is expected to create a Perl data structure.

I've also created a directory called 't/dom' that will contain the expected data structures in JSON format. (DOM stands for Document Object Model.)

```
$ mkdir t/dom
```

In there I've placed the first such JSON file:

```
[
    {
        "tag" : "h1",
        "text" : "Heading One"
    }
]
```

In the test file we load two modules, Path::Tiny[20] for easy reading of the JSON file and JSON::MaybeXS[21] to parse the JSON string.

[19]https://github.com/szabgab/perl5-markua-parser/commit/532b1b17c3520a1a9508e26c460d9e42b9c6a5d4
[20]https://metacpan.org/pod/Path::Tiny
[21]https://metacpan.org/pod/JSON::MaybeXS

```
use JSON::MaybeXS qw(decode_json);
use Path::Tiny qw(path);
```

the test code itself is another 2 lines:

```
my $result = $m->parse_file('t/input/heading1.md');
is_deeply $result, decode_json( path('t/dom/heading1.json')->slurp_utf8 );
```

In the first line we use the not yet implemented `parse_file` method that receives the path to the Markua file and returns the data structure. Or so it will do once we implement it. The second line uses the `is_deeply` function from Test::More to compare the data structure generated by the Markua parser to the expected data structure that was read in from the JSON file and converted to a Perl data structure by `decode_json`.

The full test file is here:

```
use strict;
use warnings;

use Test::More;
use JSON::MaybeXS qw(decode_json);
use Path::Tiny qw(path);
use Markua::Parser;

plan tests => 2;

my $m = Markua::Parser->new;
isa_ok $m, 'Markua::Parser';

my $result = $m->parse_file('t/input/heading1.md');
is_deeply $result, decode_json( path('t/dom/heading1.json')->slurp_utf8 );
```

Then finally the implementation of the parser itself uses Path::Tiny to read in the Markua source file and then uses regexes to parse the lines. Very simple, but works for the first test case:

```perl
package Markua::Parser;
use strict;
use warnings;
use Path::Tiny qw(path);

sub new {
    my ($class) = @_;
    my $self = bless {}, $class;
    return $self;
}

sub parse_file {
    my ($self, $filename) = @_;
    my @entries;
    for my $line (path($filename)->lines_utf8) {
        if ($line =~ /^# (\S.*)/) {
            push @entries, {
                tag => 'h1',
                text => $1,
            };
        }
    }
    return \@entries;
}


1;
```

The `parse_file` method expects two paramers. The instance object represnting the current parser and the name of the file to be parsed.

We create an empty array called `@entries` that will hold the parsed DOM.

Then we ue the `lines_utf8` method of the <hl>Path::Tiny</a> object to read in all the lines of the Markua file and go over line-by-line using a `for` loop.

In the `/^# (\S.*)/` regex[22] the leading `^` forces the regex to look for a match at the beginning of the sting. `# ` then tells it to match those two character immediately after the beginning of the string. Whatever is matched by the rge within the pair of parentheses `()` will be saved in the variable `$1`. In the regex inside the parentheses `\S` means any non-white-space character, `.` means any character (except of newline) and `*` tells the dot to match `0` or more so in other words the regex inside the parentheses will match any string of any length, it just has to start with something visible. (So there can't be `2` spaces after the initial `#`.)

[22]https://perlmaven.com/regex

I am not sure if this is the correct regex for the specification of Markua, for that I'd need to read it more thoroughly, but for now it works for us and it satisfies our test. We can always improve it later.

If the regex matches we create an reference to a hash with the name of the tag `h1` and the value or "text" of it which the text that followed the `#`. We take the anonymous hash and push[23] it (append it) to the `@entries` array.

At the end we return a reference[24] to the `@entries` array.

```
$ git add .
$ git commit -m "first parsing of an h1 tag"
$ git push
```

commit[25]

To be continued

In the meantime go and support the crowdfunding campaign[26].

Creating Makefile.PL and a CPAN distribution for the Markua Parser

When you start writing a project, especially when it is in-house, creating a CPAN distribution might not be high on your priorities, but having the capability will simplify the use of various tools and services. So I recommend that you prepare your module/application in a similar way.

We need to create a file called Makefile.PL that holds the list of dependencies of our code and a few other instructions how to install it. For simple Perl-only modules (that don't include code in C or XS or in some other language), the Makefile.PL is quite simple and standard.

[23]https://perlmaven.com/manipulating-perl-arrays
[24]https://perlmaven.com/array-references-in-perl
[25]https://github.com/szabgab/perl5-markua-parser/commit/491850ef6a6c7b5a79ef436dd407e497a5a2b2c5
[26]https://www.indiegogo.com/projects/updating-the-perl-maven-tutorial/reft/775728/pmarticle0302

```perl
use strict;
use warnings;
use ExtUtils::MakeMaker;

WriteMakefile(
    NAME         => 'Markua::Parser',
    AUTHOR       => q{Gabor Szabo <szabgab@cpan.org>},
    VERSION_FROM => 'lib/Markua/Parser.pm',
    ABSTRACT     => 'Parsing Markua documents and creating DOM',
    ( $ExtUtils::MakeMaker::VERSION >= 6.3002
        ? ( 'LICENSE' => 'perl' )
        : () ),
    PL_FILES  => {},
    PREREQ_PM => {
        'Path::Tiny'      => 0.072,
        'JSON::MaybeXS'   => 1,
    },
    TEST_REQUIRES => {
          'Test::More'    => 1.001014,
    },
);
```

For detailed explanation see the Makefile.PL of ExtUtils::MakeMaker[27] and the packaging with Makefile.PL[28] articles.

The first time you run it with `perl Makefile.PL` you'll get a warning:

```
WARNING: Setting VERSION via file 'lib/Markua/Parser.pm' failed
```

In order to fix this we add the version number to the 'lib/Markua/Parser.pm' file:

```perl
our $VERSION = 0.01;
```

The whole file can be seen here:

[27] https://perlmaven.com/makefile-pl-of-extutils-makemaker
[28] https://perlmaven.com/packaging-with-makefile-pl

```
package Markua::Parser;
use strict;
use warnings;
use Path::Tiny qw(path);

our $VERSION = 0.01;

sub new {
    my ($class) = @_;
    my $self = bless {}, $class;
    return $self;
}

sub parse_file {
    my ($self, $filename) = @_;
    my @entries;
    for my $line (path($filename)->lines_utf8) {
        if ($line =~ /^# (\S.*)/) {
            push @entries, {
                tag => 'h1',
                text => $1,
            };
        }
    }
    return \@entries;
}


1;
```

Run the tests and generate the distribution

The following sequence of command will

- Check if all the prerequisites are met and generate `Makefile`. * Rearrange the files in the `blib` subdirectory in the same structure as they will be after installation. * Run the tests * Generate the `MANIFEST` that lists all the file that need to be included in the distribution. (This is based on the `MANIFEST.SKIP` file, but we did not need it for the simple case. * Generate the tar.gz file that can be uploaded to PAUSE or distributed in another way.

```
$ perl Makefile.PL
$ make
$ make test
$ make manifest
$ make dist
```

gitignore generated files

The above process generates a few files and directories that don't need to be in version control. The best approach is to add their names to the `.gitignore` file so git will ignore them.

This is what I had to create:

```
$ git add .
$ git commit -m "create Makefile.PL"
```

commit[29]

Travis-CI for the Markua Parser project

It is very useful to write unit-tests for your project, but it is a bit annoying that you have to remember running it every time before you push out a new request. Especially if you'd like to run your tests on multiple versions of Perl and maybe with different values of certain environment variables.

Travis-CI[30] is a cloud-based Continuous Integration system that will run your tests on every commit on as many versions of Perl as you like and with as many environment variables as you configure.

Not only that, but every time someone sends you a pull-request the tests will run on the PR as well, so both the contributor and you will know if the PR would break anything that worked before.

Not only that, but Travis-CI is free for Open Source projects on GitHub.

For simple cases it is quite easy to set up Travis-CI. It involves two steps:

- Tell Travis-CI to monitor your project * Create the Travis Configuration file .travis.yml

Tell Travis-CI to monitor your project

Visit Travis-CI[31], log in with your GitHub account and let it sync the list of your GitHub repositories.

If you have already done this earlier, then for a new GitHub project you might need to manually ask Travis to sync the list.

[29]https://github.com/szabgab/perl5-markua-parser/commit/79f7a57fd459144a0720e99abeae7191a622ee1f
[30]https://travis-ci.org/
[31]https://travis-ci.org/

For this visit your Profile[32] and click on the **Sync account**. After a few seconds you'll be able to locate the entry of your project.

Look at the relevant switch which is currently off:

<img src="img/travis-markua-parser-off.png">

and turn it on:

<img src="img/travis-markua-parser-on.png">

Create the Travis Configuration file .travis.yml

For every language Travis supports there are instructions on how to set them up. So there are instructions for Perl-based projects on Travis[33].

Basically you need to create a file called `.travis.yml` in the root of your Git repository with the following content listing the versions of perl you'd like to be used by Travis:

The file is in YAML[34] format.

```
$ git add
$ git commit -m "add travis configuration file"
$ git push
```

commit[35]

Once you push the file to GitHub it will trigger Travis-CI and that will start a build. Because we requested the build on 4 different versions of Perl, Travis will spun-off 4 virtual machines and run the builds and the tests in parallel.

Here are the Travis-CI build results[36] for my first build of this project.

A screenshot of that page: <img src="img/travis-markua-parser-first-success.png">

You will also receive an e-mail confirmation with the first successful build. Such as this one:

<img src="img/travis-markua-parser-first-success-email.png">

From now on every push to GitHub will trigger the build. If any one of the builds fail you'll get an e-mail notification so you don't have to do anything, just keep coding.

[32]https://travis-ci.org/profile/
[33]https://docs.travis-ci.com/user/languages/perl/
[34]https://perlmaven.com/yaml
[35]https://github.com/szabgab/perl5-markua-parser/commit/87a69da4c3c8ec459a6cb0554f577694f996eb1a
[36]https://travis-ci.org/szabgab/perl5-markua-parser/builds/348268435

Travis badge

It is quite customary that Open Source developers will add a Travis badge to their project. On the results page on Travis you'll see a badge (it probably will say "build unknown"). If you click on it, you'll get a popup like this allowing you to select the appropriate code for the badge.

<img src="img/travis-markua-parser-badge-selector.png">

I've pasted the Markdown example into the README.md file of the project:

```
# Markua Parser
[![Build Status](https://travis-ci.org/szabgab/perl5-markua-parser.svg?branch=master\
)](https://travis-ci.org/szabgab/perl5-markua-parser)

[Markua](https://leanpub.com/markua/) is a Markdown-inspired format to write books.

This module implements parsing (part of) the Markua specification.
```

```
$ git add .
$ git commit -m "add Travis badge"
$ git push
```

commit[37]

This push will trigger another build on Travis, but what is more interesting for us now is that of someone visits the GitHub page of the Perl 5 Markua Parser[38] project then they will see the up-to-date status of the project on Travis:

<img src="img/travis-markua-parser-success-badge.png">

Add test coverage reporting with Coveralls to Markua Parser in Perl

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Travis-CI

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

[37]https://github.com/szabgab/perl5-markua-parser/commit/3986c230ed82f7575789db7a0e1f00177c6fe656
[38]https://github.com/szabgab/perl5-markua-parser

Test Coverage

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Set up Coveralls for the Markua Parser

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Enable Appveyor (CI on Windows) for the Perl 5 Markua Parser

Travis-CI runs the test on a Linux box and I think it could also run on OSX, but I never tried that. Appveyor[39] on the other hand is a cloud-based Continuous Integration that runs your tests on MS Windows. (Actually I just saw an e-mail from them inviting me to try their Linux servers as well. For now let's stick to the Windows machines.)

The Markua parser should have no platform dependent part. So our could is expected to run on MS Windows as well. I only have an old Windows 7 at home and I'd avoid using it. Luckily Appveyor can provide the platform to test the code.

Enable Appveyor

Visit Appveyor[40], click on "Sign up for free". Here I've created my own account by typing in my name, e-mail address and a password.

Then I clicked on the "+ New Project" link, selected Github, I think back when I first did this I had to authenticate at GitHub at this point, but now it just lists all the project I have.

I searched for perl5-markua-parser and as I hovered over the name the "+ add" link appeared on the right hand side. Clicking that I told Appveyor to start monitoring the project.

Configure Appveyor

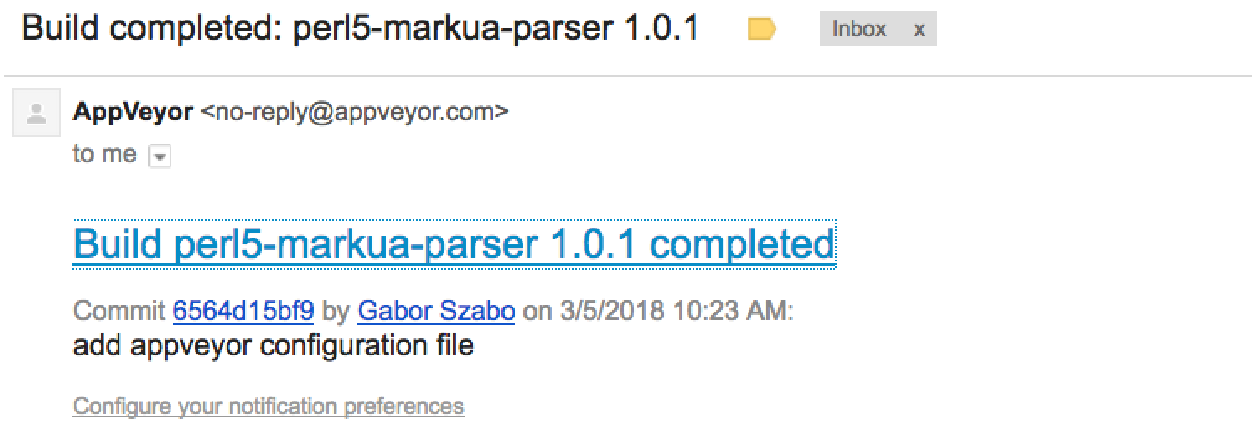
In order to tell Appveyor what to do you need to create a file called `appveyor.yml` or `.appveyor.yml` (with a leading dot). It needs to include some instructions installing Strawberry Perl[41], installing any prerequisites, and then running the tests with `gmake test`.

[39]https://www.appveyor.com/
[40]https://www.appveyor.com/
[41]http://strawberryperl.com/

```
$ git add .appveyor.yml
$ git commit -m "add appveyor configuration file"
$ git push
```

commit[42]

Once I pushed out the changes both Travis and Appveyor started to build the project. Coveralls was updated by Travis and we got an e-mail from Appveyor:


Build completed: perl5-markua-parser 1.0.1

AppVeyor <no-reply@appveyor.com>
to me

Build perl5-markua-parser 1.0.1 completed

Commit 6564d15bf9 by Gabor Szabo on 3/5/2018 10:23 AM:
add appveyor configuration file

Configure your notification preferences

Appveyor success e-mail

To tell the truth this is the first project where I manage to get Appveyor succeed in the first attempt. In many cases the project did not even started to run. The success is probably due to the fact that the project itself is really simple. So if Appveyor fails for you don't worry. It might take some tweaking in the appveyor.yml and in your code till the tests starts to pass.

The link in the e-mail leads to the full report of the process: appveyor build report[43]

Appveyor Badge

There is a canonical link to the latest report[44], but even there I could not see any recommendation on how to add an Appveyor badge to my project. Luckily I already have it in a few of my projects so I copied it from one of them, tweaked it to refer to the current project and added this to the README.md file:

```
[![Build status](https://ci.appveyor.com/api/projects/status/github/szabgab/perl5-ma\
rkua-parser?svg=true)](https://ci.appveyor.com/project/szabgab/perl5-markua-parser/b\
ranch/master)
```

The new file is this:

[42]https://github.com/szabgab/perl5-markua-parser/commit/6564d15bf98c0fe5ee08d34fbc36bea92e4e0c29
[43]https://ci.appveyor.com/project/szabgab/perl5-markua-parser/build/1.0.1
[44]https://ci.appveyor.com/project/szabgab/perl5-markua-parser

```
# Markua Parser
[![Build Status](https://travis-ci.org/szabgab/perl5-markua-parser.svg?branch=master\
)](https://travis-ci.org/szabgab/perl5-markua-parser)
[![Coverage Status](https://coveralls.io/repos/github/szabgab/perl5-markua-parser/ba\
dge.svg?branch=master)](https://coveralls.io/github/szabgab/perl5-markua-parser?bran\
ch=master)
[![Build status](https://ci.appveyor.com/api/projects/status/github/szabgab/perl5-ma\
rkua-parser?svg=true)](https://ci.appveyor.com/project/szabgab/perl5-markua-parser/b\
ranch/master)

[Markua](https://leanpub.com/markua/) is a Markdown-inspired format to write books.

This module implements parsing (part of) the Markua specification.
```

Then the usual:

```
$ git add .
$ git commit -m "add Appveyor badge"
$ git push
```

commit[45]

If you visit the project now perl5-markua-parser[46] and scroll down where the content of the README.md is displayed then you will see the badges. (By the time you visit it, you might see additional badges. So don't be surprised.

Now that we have enabled Continuous Integration both on Linux and on Windows, and we also monitor our test coverage we can proceed with the actual code.

Process the 6 headers of Markua

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Prepare the test case

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

[45] https://github.com/szabgab/perl5-markua-parser/commit/d3708181fc8934d7c0d2e0bd2abbd58c702c2903
[46] https://github.com/szabgab/perl5-markua-parser

The implementation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Error handling? Incorrect Markua syntax?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Refactor test cases of the Markua Parser in Perl 5

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Collecting errors while parsing Markua in Perl 5 - disregarding empty rows

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Get Coveralls to notify when test-coverage shrinks

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Testing the mail

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

The missing test-case

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Consider everything not recognizable as a paragraph in Markua

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Test case

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Better error reporting

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Explaining the implementation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Increased test coverage

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Markua resources: Include files

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Tests case

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Improve the test reporting

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Implement the parser

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Test the parser

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Flexible expected errors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Tests pass successfully

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Commit the changes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Test coverage increase

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Generate test expectations for the Markua parser

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Generate pretty JSON

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

The result

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Parse bulleted list in Markua

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Implement parser

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Bulleted list with a dash (hyphen)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Fix the incorrect list to paragraph switching

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Remove unnecessary fields from the DOM of lists

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Parse numbered list in Markua

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Start with simple test casses

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Processing numbered lists

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Expected DOM

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

New commit

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Avoid forgetting to add test-case

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Release the Markua::Parser to CPAN

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Getting feedback

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

CPAN Testers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Changes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Add POD to the module

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Update Makefile.PL

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

MANIFEST and MANIFEST.SKIP

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Creating the distribution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Add attributes to Markua - round 1

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Mapping of extension to format and type

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Add attributes to resources

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Update the expected DOM

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Test and commit

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Improved test coverage?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Test coverage report with Devel::Cover for the Markua Parser

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Using Devel::Cover

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Test coverage report in HTML

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Branch Coverage

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Condition Coverage

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/markua-parser-in-perl5.

Commit

Index