Templateme Documentation!

Release 0.0.6

['Rafal Kobel']

Sep 17, 2019

Contents

Templateme is a simple console program which helps you create new projects. It has a template base which allow you to create a new project just like you do it in your favourite IDE. Thanks configurable variables you can configure your project before it exactly create.

This document helps you understand how work templateme and how you can prepare your customer configurable templates for your new projects. It also will teach you how you can distribute and share your own templates to help other users in creating their programs.

CHAPTER 1

Short Informations

1.1 Instalation

There are two different methods to install `TemplateMe`. You can install it from PyPi repository or download sources from github and try to install it. Please choose one of the methods and install it on you computer.

1.1.1 from PyPi repository

To install TemplateMe from repository you can use pip program.

```
pip3 install templateme
```

1.1.2 from sources

Another option is to install program from github sources. To do this just get all necessary files using git or by github's zip file and call following command:

```
python3 setup.py install
```

1.2 Usage

The TemplateMe works as console application. It allows you to create new project from predefined templates. To see list of all available elements call:

```
templateme --list
```

In case that you want to prepare new project for your favourite language you need to choose one of the templates and write in your terminal:

```
templateme -t <template_name> -o <your_project_name>
```

Where `<template_name>` is the one of the templates, listed in previous step and `<your_project_name>` is a name of folder with your created project. After that, the program will ask you to write a few variables that are necessary to build templates.

For more information about application please visit the documentation or try:

```
templateme --help
```

1.3 Author

Rafal Kobel <rafalkobel@rafyco.pl>

CHAPTER 2

Usage of TemplateMe

2.1 Quick start

Create new project in defined place..

The TemplateMe works as console application. It allows you to create new project from predefined templates. To see list of all available elements call:

```
templateme --list
```

In case that you want to prepare new project for your favourite language you need to choose one of the templates and write in your terminal:

```
templateme -t <template_name> -o <your_project_name>
```

Where `<template_name>` is the one of the templates, listed in previous step and `<your_project_name>` is a name of folder with your created project. After that, the program will ask you to write a few variables that are necessary to build templates.

CHAPTER 3

Configuration

CHAPTER 4

How to create custom templates?

CHAPTER 5

Distribution custom templates

CHAPTER 6

Code Documentation

This chapter described all modules, classes and functions present in TemplateMe package. This information can be usefull for developers which want to make applications based on TemplateMe.

6.1 module templateme

Template Me - module with template for your favourite projects.

`templateme.`**`get_version`**`()`

Get version of protemp package.

6.2 module templateme.arguments

Module to manage template's arguments.

`class` `templateme.arguments.`**`Argument`**`(`*name*, *required=True*, *default=''*, *question=None*, *description=None*`)`

One of argument.

`static create_from_dict``(`*dictionary*`)`

Create argument from dictionary element.

`is_set`

True if set, False otherwise.

`name`

Name of argument.

`question`

String of question about value.

`value`

Value of argument.

exception templateme.arguments.**ArgumentError**
Argument error.

class templateme.arguments.**ArgumentsContainer**(*args*)
Manager with information about arguments.

add_values(*list_of_values*)
Add value of element.

add_values_from_list(*list_of_arguments*)
Add values from list of 2-elements lists.

all
List of all elements.

get_argument(*key*)
Get argument by key.

input_missing()
Ask about all missing arguments.

missing_args
Return all argument that is not set.

update(*arguments*)
Update values of arguments.

templateme.arguments.**empty_args**()
Crate ArgumentContainer for empty arguments list.

6.3 module templateme.configuration

Configuration files for TemplateMe!

The module contains class which allows configuration terminal console and all default option in TemplateMe.

class templateme.configuration.**Configuration**(*debug=False*)
Configuration class.

debug
Check if configuration is from debug.

get_val(*key*, *section='global'*, *default=None*)
Get value of option.

localizations
Localization of all place where application should look for templates.

6.4 module templateme.console

Create new project in defined place..

The TemplateMe works as console application. It allows you to create new project from predefined templates. To see list of all available elements call:

```
templateme --list
```

In case that you want to prepare new project for your favourite language you need to choose one of the templates and write in your terminal:

```
templateme -t <template_name> -o <your_project_name>
```

Where `<template_name>` is the one of the templates, listed in previous step and `<your_project_name>` is a name of folder with your created project. After that, the program will ask you to write a few variables that are necessary to build templates.

`templateme.console.`**`examine_save`**(*template*, *project_name*, *force*)
Check if template exists and you can save it. If not, ask about confirmation to do this.

`templateme.console.`**`main`**(*argv=None*, *debug=False*)
Main function for command line program.

@param argv: Option parameters @type argv: list

`templateme.console.`**`print_template`**(*template*)
Print information about template.

6.5 module templateme.containers

6.6 module templateme.containers.abstract

Module with abstract classes.

class `templateme.containers.abstract.`**`TMPElement`**(*path*, *template*, *project_name='project'*)
Class with one of files in template.

classmethod load_txt()
Load text.

print_element()
Print path and source of element.

save(*path*, *project_name='project'*)
Save element in project.

save_path
Path to save element in output directory.

text
Element text.

class `templateme.containers.abstract.`**`TMPSource`**(*manager*)
Class manage one of the containers.

classmethod get_all_templates()
Get list of all templates.

get_template(*name*)
Get one of templates by id.

templates
List of all templates.

class `templateme.containers.abstract.`**`Template`**(*name*, *manager*, *manifest=None*)
Class which describe template.

add_ignored(*args*)
Add ignore pattern.

args
Arguments with manifest updated file.

description
Return long description.

elements
List of all template's elements.

classmethod examine_save(*path*, *force=False*)
Check if template can be save.

include_templates
List of included template.

name
Name of template.

print_elements()
Print elements on the screen.

save(*path*, *project_name=None*, *force=False*)
Save template in path.

short_description
Return short description.

exception templateme.containers.abstract.**TemplateError**
Class describe template error.

6.7 module templateme.containers.path

Module for template from path.

class templateme.containers.path.**PathElement**(*path*, *localization*, *template*)
Class with template's element file from path.

load_txt()
Load information from file.

class templateme.containers.path.**PathSource**(*manager*, *path*)
Class with path source from directory.

get_all_templates()
Return all templates from directory.

class templateme.containers.path.**PathTemplate**(*path*, *name*, *manager*)
Class with template from path.

exception templateme.containers.path.**TemplateError**
Problem with template.

6.8 module templateme.containers.resource

Module for template from resource.

class templateme.containers.resource.**ResourceElement**(*path*, *template*, *localization*, *name*)

Class with template's element file from path.

load_txt()

Load information from file.

class templateme.containers.resource.**ResourceSource**(*manager*, *source_dir='templates'*)

Class with path source from directory.

get_all_templates()

Return all templates from directory.

class templateme.containers.resource.**ResourceTemplate**(*name*, *manager*, *package_localization='templates'*)

Class with template from path.

6.9 module templateme.manager

Module of template's manager.

class templateme.manager.**TMPManager**(*name='Project'*, *debug=False*)

Template manager.

get_all_templates()

Return all of available templates from all of containers.

get_template(*name*)

Return template by name.

render_template_txt(*txt*, *template*)

Rendering template to text format.

exception templateme.manager.**TMPManagerError**

Manager error.

6.10 module templateme.manifest

Module to parsing manifest file.

class templateme.manifest.**Manifest**(*data*, *template*)

Manifest parsing class.

static create_from_file(*path*, *template*)

Create manifest object from file.

static create_from_string(*text*, *template*)

Create manifest object from string.

exception templateme.manifest.**ManifestError**

Manifest error.

CHAPTER 7

Changelog

7.1 versions 0.1.*

7.1.1 version 0.0.6

- Prepare better tests for console application
- More options to create arguments for template
- Arguments you can add from command application

7.1.2 version 0.0.5

- Documentation in Sphinx
- Fix problem with templates in PyPi package

7.1.3 version 0.0.3

- Projects urls

7.1.4 version 0.0.2

- Ask if add files to existing project
- Custom templates in home location

7.1.5 version 0.0.1

- Templates to cpp and Python
- Simple implementation of templates
- Console program to create new program

Python Module Index

t

Index