ETeX Documentation

V0.1

RosiePuddles

Contents

1 Preface

This package is designed to allow the user to generate LaTeXfiles and associated pdf files in a more user friendly way. Please note, however, this package is still currently heavily in development, and things will go wrong. Any bugs can be reported on the issues page of the GitHub repository. You can request any features you cannot find and want adding to the package. Having said that, I hope you find this package useful and fairly easy to use as intended.

2 Main Classes

2.1 Document

```
class Document(*args, **kwargs) -> None
```

The `Document` class is the main class used in ETeX. It handles all tex codegeneration, and contains all information about the document as well as the actual contents themself.

2.1.1 generate_TeX function

```
Document.generate_TeX(self, _compile: bool = True, **kwargs) -> str
```

The `generate_TeX` function is used to firstly generate the .tex file which is then compiled and the resulting .pdf file is opened. The parameter `_compile` is set to `False` by default. If it is changed to `True`, then only the .tex file will be generated, but not compiled. For the `kwargs`, you can pass in `debug=True` to see the logs from pdflatex as it compiles the .tex file. The output .tex file name will be a formatted version of the value for the title given on instantiation of a new instance of the `Document` class. Any of the following characters are removes:

- $
- %
- /
- \

Bullet points are also formatted and turned into underscores. The resulting formatted filename is then used for all of the resulting output files.

2.1.2 add function

```
Document.add(self, item: _main) -> None:
        self.contains.append(item)
```

The `add` function adds a class that inherits from the class `_main`[1] to the list of contents inside an instance of the `Document` class. The function is used to add items into the document.

[1]See section 2.5.1 for a full list of classes that directly or indirectly inherit from the _main class.

2.1.3 new_section function

```
Document.add(self, title: str, _type: int = 0) -> None:
        _type = _type % 3
        self.contains.append(_section(title, _type))
```

The `new_section` function is used to add a new section to a `Document` class instance. The `_type` argument is used to identify the type of section with 0 being a section, 1 being a subsection, and 2 being a subsubsection.

2.2 Text

```
class Text(self, text: str, align: str = None) -> None
```

The `Text` class is the class used for the handling of text inside of ETeX. The class contains some general string formatting features allowing for **bold**, *italic*, highlighted, and underlined text inside of the document. To read more on this see section 2.2.1. The text can also be aligned to either the left, center, or right using the `align` argument. This will only apply to the current `Text` class instance and will not be applied to any subsequent instances of the class.

2.2.1 String formatting

To format a string in ETeX, you use the * and ~ characters. The following table shows the formatting character and the relevant format.

Formatting character	Associated formatting
*	**Bold**
*	*Italic*
~	Highlight
~~	Underline

2.3 List

```
class List(self, list_type: str = 'numbered', items: list = None) -> None
```

The `List` class is used to created lists inside of ETeX. These list can be either a numbered list or a bullet point list through the use of the `list_type` argument[2]. The list can also be initialised with items already inside of it, so long as the items inherit from the `_main` class[3]. The list can also be left empty upon initialisation and later on have items added to it using the `add` function.

2.3.1 List types

To change the type of list, you can use the `list_type` argument, which takes in a string of wither `numbered` or `bullet`, which correspond to a numbered list, or a bullet point list.

[2]See section 2.3.1 for list types

[3]See section 2.5.1 for a full list of classes that directly or indirectly inherit from the _main class.

2.3.2 add function

```
List.add(self, item: _main) -> None:
        self.items.append(item)
        self.add_super(item.packages)
```

The add function adds the given item to the end of the list instance's list. The item has to inherit from the `_main` class to be added. The second line of the function is part of the process of ensuing all the required packages are declared in the preamble of the .tex document.

2.4 Group

```
class Group(self, items: list = None)
```

The `Group` class is a holding class used for storing other classes. The primary use for this class is alongside lists. When an item is added to a list it is added as a new item, however if the user wants to add several different classes to a list as the same point they can put all the items into a `Group` class and add that to the list.

2.5 _main

The `_main` class is the base class for all other classes the user interfaces with and provides several important functions and alterations to base functions that are used throughout.

2.5.1 Child classes

This section provides a list of all the different child classes of the `_main` class:

- `Text`
- `Footnote`
- `Columns`
- `Equation`
- `List`
- `Group`
- `line`
- `plot`
- `coordinates`
- `axis`
- `Code`
- `Chemical`
- `ChemEquation`

2.5.2 generate_TeX method

```
_main.generate_TeX(self, *args, **kwargs) -> str
```

The `generate_TeX` method generates raises an exception if run. All classes that inherit from `_main` will overwrite this method with their own method to generate their unique LaTeXcode.

3 Maths Classes

4 Plotting Classes

5 Chemistry Classes