ETeX Documentation
V0.1

RosiePuddles

Contents

1 Preface

This package is designed to allow the user to generate LaTeX files 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.
Please note that every class inherits from the Having said that, I hope you find this package useful and fairly easy to use as intended.
Please note that every class inherits from the `_main` class unless specified otherwise. Each class that inherits from `_main` may overwrite methods defined in the `_main` class. If a class does overwrite a predefined method this will be documented, otherwise there will be no specific documentation if the method is inherited.

2 Main Classes

This section is for the documentation of the classes contained within `ETeX`.
Each of the classes in this section, except for the `Document` class, inherit from the `_main` class[1]. As such please be aware that when looking for documentation on a certain method that the method may be inherited and documentation will be contained within the `_main` class section.

2.1 Document

```
class Document:
        def __init__(self, *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 method

```
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:

- $
- %
- /
- \

[1] See subsubsection 2.2.1 for a full list of classes that directly or indirectly inherit from the _main class.

Full stops 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 method

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

The `add` method adds items to the document. The added item must inherit from the `_main` class. Only items added to the `Document` class instance will be included in the final document.

2.1.3 new_section method

```
Document.new_section(self, title: str, _type: int = 0) -> None
```

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 _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.

2.2.1 Child classes

This section provides a list of all the different child classes of the `_main` class. This is split up into two parts. Those that directly inherit from the `_main` class, and those that inherit from the `_main` class through inheriting from the `_holder` class.
Classes that inherit from `_main`:

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

Classes that inherit from `_holder`:

- `Columns`
- `List`
- `Group`

2.3 _holder

```
class _holder(_main):
        def __init__(self, packages) -> None
```

The `_holder` class is a second base class that inherits from the `_main` class. The class adds the `add` method and allows for child classes to have class instances added to them. For a full list of classes that inherit from `_holder` see subsubsection 2.2.1.

2.3.1 add method

2.4 Text

```
class Text(_main):
        def __init__(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 subsubsection 2.4.1. 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.

2.4.1 Inbuilt 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

2.4.2 Extra formatting

Withing the text environment regular LaTeXcommands can be used. Some useful examples are given below:

- \verb|foo| produces text in a code-like font as seen below:
  `foo`
- $
  The $ character allows you to write inline maths equations such as the example below:
  $2x+y^3=-1$ → $2x + y^3 = -1$

2.5 List

```
class List(_holder):
        def __init__(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. The list can also be initialised with items already inside of it, so long as the items inherit from the `_main` class. The list can also be left empty upon initialisation and later on have items added to it using the `add` function.

2.5.1 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.

2.6 Group

```
class Group(_holder):
        def __init__(self, items: list = None) -> 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.

2.7 Columns

```
class Columns(_holder):
        def __init__(self, columns: int, items: list = None, unbalanced: bool =
        ↪  False) -> None
```

The `Columns` class is used to add columns to the document. It is similar to the `Group` class in that it stores classes to be contained within it's formatting. Only items added to the class will be put into columns. To make the columns unbalances, i.e. with the contents not spread out equally over all the columns, you can change the `unbalanced` argument to `True`.

3 Maths Classes

4 Plotting Classes

This section is for classes contained within `ETeX.maths.plots`. All classes inherit from `_main` unless stated otherwise.

4.1 Axis

```
class Axis(_main):
        def __init__(self, *args, **kwargs) -> None
```

The `Axis` class is the handler for all plots. It is centre justified. Within the `**kwargs` argument there are a large number of parameters that we can pass in. these are listed below:

- `title: str`
  This is the title of the axis and is positioned centre justified above the axis
- `width: int or float`
  This is the width of the axis. This is measured in cm.
- `height: int or float`
  This is the height of the axis. This is measured in cm.
- Min and max values:
  These correspond to the minimum and maximum x and y values on the axi. If none are specified the minimum or maximum values of the plots contained within the axis will be used instead. The following options are available:
  - `xmin: int or float`
  - `xmax: int or float`
  - `ymin: int or float`
  - `ymax: int or float`
- Axis labels:
  These correspond to the x and y axis labels. The following options are available:
  - `xlab: str`
  - `ylab: str`
- `samples: int`
  This is the number of samples used for plotting functions. By default it is set to 100.
- `showTickMarks: bool`
  This bool controls weather or not tick marks are shown on the x and y axes. This is set to `True` by default.
- `clip: bool`
  This bool controls weather or not the plots can be clipped to fit within the axis. This is set to `False` by default.

4.1.1 add_plot method

```
Axis.add_plot(self, new_plot: plot or coordinates) -> None
```

The `add_plot` method adds a plot to the current `Axis` instance. The plot must be an instance of either a `plot` or `coordinates` class.

4.2 Plot

```
class Plot(_main)
        def __init__(self, function: str, *args, **kwargs) -> None
```

The `Plot` class is used for plotting mathematically defined functions. These then have to be added to an `Axis` class to be shown. The class has several options for the presentation of the function, which are listed below:

- `domain: tuple`
  This controls the domain of the function. It must be a tuple with two values in in ascending order, for example (1,5).
- `color: str`
  This sets the colour of the plot. this colour must either be native to LaTeX or defined in the `DocumentSettings` class[2].

5 Chemistry Classes

[2] Soon to be added