

dtxgen generate template for LaTeX self-extracting .dtx file

doc generated from the script with `gendoc`

bash script, version=1.10

Synopsis

```
dtxgen [options] basename.[sty,cls]
```

Options

<code>-h, --help</code>	Print short help and exit.
<code>-H, --Help</code>	Print full documentation via less and exit.
<code>-V</code>	Print the script's version and exit.
<code>-s, --short=X</code>	Set short, one-liner, class or package description to X. By default, the string <i>A new LaTeX class</i> or <i>A new LaTeX package</i> will be used.
<code>-v, --version</code>	Set the initial version; by default 1.00 (without initial v) will be used
<code>-d, --date</code>	Set the initial version's date. By default, the current date will be used. The date should be entered in the <code>yyyymmdd</code> format, although it will be stored the LaTeX way: <code>yyyy-mm-dd</code>
<code>-m, --mail</code>	Set the author's email address. Alternatively, you can set a default value in the environment variable <code>EMAIL</code> ; if you do so and still use this option, the option's value will have priority
<code>-n, --name</code>	Set the author's name (first name, followed by surname). Alternatively, you can set a default value in the environment variable <code>NAME</code> ; if you do so and still use this option, the option's value will have priority.
<code>-c, --class=X</code>	For class templates only: inserts a <code>\LoadClass {X}</code> , so that the new class will start with the properties of the <code>X</code> class. The default is <code>article</code> .
<code>-f, --format=X</code>	Set the Latex format to be used for compilation to X. The default is <code>pdflatex</code> , but you may need another format, like <code>xelatex</code> or <code>lualatex</code> .
<code>-b, --body=X</code>	Use class or package file X as a source for the code for the new class or package, instead of the demo code. X will be looked up with <code>kpsewchich</code> .
<code>-i, --history</code>	Create a section History instead of the standard Change History section. The standard Change History allows very detailed reports, but most people confine themselves to global remarks about changes between versions, appear- ing at one place in the document. The <code>--_history</code> option provides such a straightfor- ward history section, formatted in a <code>longtable</code> environment.

Description

dtxgen creates a template for a self-extracting .dtx file, based on the model described by [Joseph Wright](#). It is useful for those who plan to create a new Documented LaTeX Source (.dtx) file. It is also useful if you already have a .dtx file, but want it to be self-extracting and like the tools that come with it for maintaining the file.

Usage example

The simplest possible example, creating a file `demo.dtx`, is:

```
dtxgen demo.sty
```

It contains some very simple code between two comment lines:

```
% ===== Start of your own code =====  
% ===== End of your own code =====
```

which you will have to replace with your own code, plus a simple documentation for this code, starting with:

```
% \section{Usage}
```

and ending before:

```
% \StopEventually{^^A
```

which you will have to replace with documentation for your own code.

Everything in the documentation before `\section{Usage}` is generated automatically, and it includes the authors name, probably you, which is taken from the environment variable `NAME` and the authors email address, taken from the environment variable `EMAIL`. Both can be overruled with the options `--name` and `--mail` respectively.

If you already have a working style file, you can create a self-extracting version of it. For example, you could make a self-extracting version of `fancyhdr` with:

```
dtxgen --body=fancyhdr.sty demo.sty
```

which generates a `demo.dtx` from which you can generate `demo.sty` which works just like `fancyhdr.sty`.

Maintaining your .dtx file

Once you have your template, say `demo.dtx`, you'll have to add code to it (unless you already added it by using the `--body` option, and you will need to add documentation. In order to help you with this, **dtngen** has included a few tools in the `.dtx` file.

To use these, run:

```
tex demo.dtx
```

which generates 5 new files:

```
README.md demo.sty demo.log etc.uue make.sh
```

Now you can run:

```
sh make.sh
```

which generates a Makefile and another shell script, `storeback.sh`, plus the documentation for your package, `demo.pdf`.

Now, you can edit code and documentation in `demo.dtx`, as well as your `README.md` file and your `storeback.sh` script and even your Makefile and `storeback.sh`, depending on any extra files you need for your documentation like figures or example files.

After your edits, you can regenerate the documentation by running `make`, install your package or clean up your directory. In all cases, your edits are stored back in `demo.dtx`. Try:

```
make help
```

to review your options.

Author

Wybo Dekker

Copyright

Released under the [GNU General Public License](#)

Functions used:

excheck

synopsis: `excheck executable1 [executable2...]`
description: check if all needed execs are there and getopt is GNU
`apt-file find -x /pdfcrop$`

handle_options

synopsis: handle_options "\$@"
description: handle the options.
globals used: Myname Version
globals set: args short date mail name loadclass format
returns: the number of remaining arguments