The `dot2texi` package

Kjell Magne Fauske
http://www.fauskes.net/

Version 3.0

1 Introduction

The `dot2texi` package allows you to embed graphs written the DOT description language directly in your document. The dot2tex[1][2] tool is used to transform the output from Graphviz to LaTeX code using either the TikZ and PGF package, or the PSTricks package. The generated code can then be included directly in you document. This package can automate the process if shell escape is enabled.

This package is derived from the dottex[3] and gnuplottex[4] packages written by Lars Kotthoff.

2 Requirements

To run `dot2texi` the following packages are required:

- `xkeyval` version 2.3 or later required.
- `moreverb`
- PGF or PSTricks

In addition, the following external tools are required:

- Graphviz
- dot2tex (Version 2.7.0 or later recommended). The dot2texi package assumes that dot2tex is installed somewhere on the system path.

For automatic creating of code, TeX must be configured to allow calling of external programs. This feature is a potential security risk and is therefore usually disabled by default. You can enable this feature by specifying an option to TeX when compiling the document. For `pdflatex` the option is `--shell-escape`. On some systems the option is called `--enable-write18`.

[1] Available from: `http://www.fauskes.net/code/dot2tex/`
[2] and `http://www.ctan.org/tex-archive/help/Catalogue/entries/dot2tex.html`
[3] `http://www.ctan.org/tex-archive/help/Catalogue/entries/dottex.html`
[4] `http://www.ctan.org/tex-archive/help/Catalogue/entries/gnuplottex.html`

3 Support

If you have any questions or comments, send an email to `kjellmf@gmail.com` or use the mailing list available at `http://groups.google.com/group/dot2tex-users/`.

4 Usage

The package is loaded by writing `\usepackage{dot2texi}` in the document preamble.

5 Package options

The following package options are recognized:

`[`⟨***shell***⟩`]` Use shell escape to automatically generate TEXcode using do2tex. This is the default behavior.

`[`⟨***noshell***⟩`]` Disable shell escape. Note that you can locally enable and disable shell escape for each `dot2tex` environment.

`[`⟨***forceshell***⟩`]` Force shell escape. Will run `dot2tex` on graphs even if they locally use the `[`⟨*noshell*⟩`]` option.

`[`⟨***miktex***⟩`]` MikTeX compatibility mode.

`[`⟨***debug***⟩`]` Invoke `dot2tex` in debug mode.

`[`⟨***autosize***⟩`]` Invoke `dot2tex` with the `--autosize` option. Will preprocess the graph using the preview package and adjust node and edge label sizes so that they fit the LaTeX output.

`[`⟨***outputdir=dir***⟩`]` Set a directory where the generated graph code will be stored. The default is to put the files in the current directory. The *outputdir* value should have a trailing slash to ensure that it is interpreted as a directory. Example:

```
\usepackage[outputdir={docgraphs/}]
...
```

5.1 Output format

`[`⟨***tikz***⟩`]` Use the tikz output format. This is the default output format.

`[`⟨***pgf***⟩`]` Use the pgf output format.

`[`⟨***pstricks***⟩`]` Use the pstricks output format.

5.2 Graph layout

Set the default graph layout tool

`[`⟨***dot***⟩`]` Hierarchical layout

`[`⟨***neato***⟩`]` Spring model layot

`[`⟨***circo***⟩`]` Circular layout

`[`⟨***fdp***⟩`]` Spring model layout

`[`⟨***twopi***⟩`]` Radial layout

6 Macros

`\setoutputdir` Set a directory where the generated graph code will be stored. Does the same as the `[`⟨*outputdir*⟩`]` package option. Useful if you want to organize your graphs in different directories. Example:

```
\documentclass{report}
\usepackage{dot2texi}
\begin{document}
...
\chapter{Chapter A}
\setoutputdir{chapA/}
....
\chapter{Chapter B}
\setoutputdir{chapB/}
...
\end{document}
```

7 The `dot2tex` environment

`dot2tex` The `dot2texi` package defines the `dot2tex` environment. The contents of the environment will be written to file during compilation. If shell escape is enabled the `dot2tex` tool and Graphviz will then be run on the saved file. This process generates LaTeX code that will be included automatically during compilation.

7.1 Environment options

Most of the package options can also be used in the dot2tex environment. They will then locally override the package options.

`[`⟨***shell***⟩`]` Enable shell escape for the current graph.

`[`⟨***noshell***⟩`]` Disable shell escape for the current graph.

Output formats:

`[`⟨***tikz***⟩`]` Use the tikz output format.

`[`⟨***pgf***⟩`]` Use the pgf output format.

`[⟨`***pstricks***`⟩]` Use the pstricks output format.

`[⟨`***format=outputformat***`⟩]` Set output format. Allowed values are `tikz|pgf|pstricks`.

Graph layout:

`[⟨`***dot***`⟩]` Hierarchical layout

`[⟨`***neato***`⟩]` Spring model layot

`[⟨`***circo***`⟩]` Circular layout

`[⟨`***fdp***`⟩]` Spring model layout

`[⟨`***twopi***`⟩]` Radial layout

`[⟨`***prog=layouttool***`⟩]` Set program to use for graph layout. Allowed values are `dot|neato|circo|fdp|twopi`.

Files:

`[⟨`***outputdir=dir***`⟩]` Locally override the `[⟨`*outputdir*`⟩]` package option value.

`[⟨`***file=filename***`⟩]` Set the base name of the generated `dot` and `tex` file. The name is generated automatically, but this option lets you override the default file name. Example:

```
\begin{dot2tex}[file=mygraph]
...
\end{dot2tex}
```

Compiling the above code will generate the files `mygraph.dot` and `mygraph.tex`. Note that the `dot` and `tex` extensions are added automatically. Here is another example:

```
\begin{dot2tex}[file=graphs/minimal]
...
\end{dot2tex}
```

The above code will generate the files `minimal.dot` and `minimal.tex` in the `graphs` directory.

Note: If the file name contains spaces or other special characters use:

```
\begin{dot2tex}[file="name with spaces"]
...
```

Note: If the `[⟨`*outputdir*`⟩]` option is set, its value will be prepended.

Options:

`[⟨`***options=opts***`⟩]` Pass additional command line options to `dot2tex`. See the dot2tex documentation[5] for available options.

Note. If opts contains an equal sign,=, you have to put opts inside curly brackets. Example:

[5] `http://www.fauskes.net/code/dot2tex/documentation/`

```
\begin{dot2tex}[options=--graphstyle "scale=0.25"]
graph G {
...
}
\end{dot2tex}
```

The above code will not work because opts i parsed incorrectly. Instead you have to write:

```
\begin{dot2tex}[options={--graphstyle "scale=0.25"}]
graph G {
...
}
\end{dot2tex}
```

[⟨***autosize***⟩] Run `dot2tex` with the `--autosize` option.

[⟨***noautosize***⟩] Locally override the package [⟨*autosize*⟩] option.

[⟨***codeonly***⟩] Run `dot2tex` with the `--codeonly` option. The generated code will then not be wrapped inside a `tikzpicture` or `pspicture` environment. Note that this option requires `dot2tex` version 2.7.0 or later.

[⟨***graphstyle=style***⟩] Set the `<<graphstyle>>` template variable. For the default templates the style value will be inserted as:

```
...
\begin{tikzpicture}[<<graphstyle>>]
...
\end{tikzpicture}
..
```

Use this option to set styles that affect all of the graph. Example:

```
...
\begin{dot2tex}[graphstyle={scale=0.5,transform shape}]
...
\end{dot2tex}
...
```

The above style value will scale down the graph to half its size.

Note: Use curly braces around the style value to avoid confusing the xkeyval parser.

[⟨***mathmode***⟩] Run `dot2tex` with tex mode set to math. This means that labels are interpreted as math.

[⟨***scale=factor***⟩] Scale the graph by a factor.

Note: This is an experimental feature. Currently writing `scale=0.5` is equivalent to `graphstyle={scale=0.5,transform shape}`. Will not work for the pstricks output format or if the `codeonly` option is used.

[⟨***straightedges***⟩] Run `dot2tex` with the `--straightedges` option.

[⟨***styleonly***⟩] Run `dot2tex` with the `--styleonly` option. Only works with the `tikz` output format. Uses only styles when drawing nodes. No draw or shape option is added.

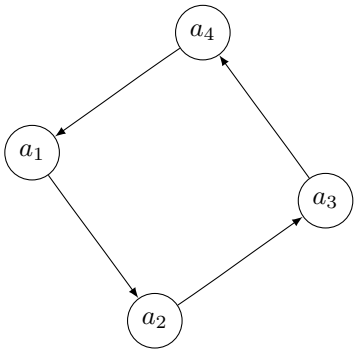
[⟨***tikzedgelabels***⟩] Run `dot2tex` with the `--tikzedgelabels` option.

Warning: All of the options are passed to dot2tex as command line options. If an option is given multiple times, the last one will win. One side effect of this is that you can't use both the `scale` and `graphstyle` at the same time. Example:

```
...
\begin{dot2tex}[scale=0.5,graphstyle={shorten >=5pt}]
...
\end{dot2tex}
...
```

In the above code the `graphstyle` option will win. If you interchange the options, the `graphstyle` option will have no effect.

8 Examples



The minimal code required to generate the above graph is:

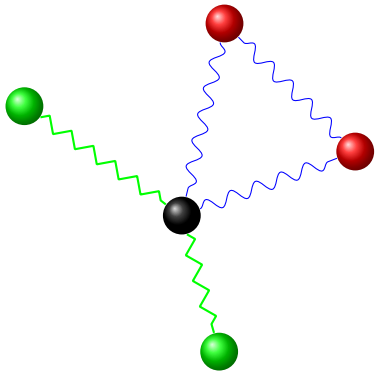
```
\documentclass{article}
\usepackage{dot2texi}

\usepackage{tikz}
\usetikzlibrary{shapes,arrows}

\begin{document}
\begin{dot2tex}[neato,mathmode]
  digraph G {
    node [shape="circle"];
    a_1 -> a_2 -> a_3 -> a_4 -> a_1;
    }
\end{dot2tex}

\end{document}
```

Now an example that uses several TikZ features:

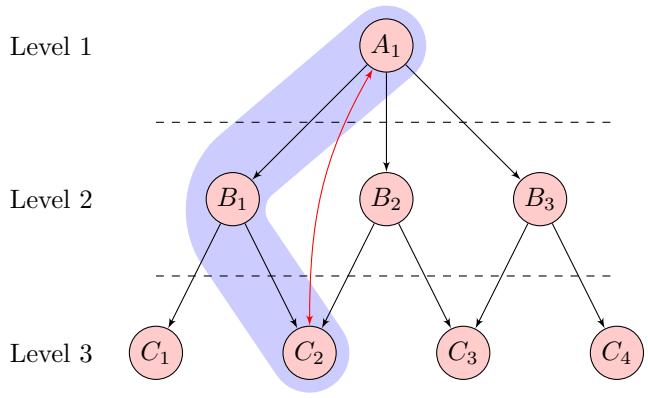


```
\documentclass{article}
\usepackage{dot2texi}

\usepackage{tikz}
\usetikzlibrary{shapes,arrows}
\usetikzlibrary{decorations.pathmorphing}

\begin{document}
\tikzstyle{ball} = [shape=circle, minimum size=.5cm]
\begin{dot2tex}[straightedges,fdp,styleonly]
graph G {
    node [style="ball, ball color =green", label=""];
    edge [style="decorate,decoration=zigzag, green,thick"];
    a_1 -- c -- a_2;
    c [style="ball, ball color=black"];
    edge [style="decorate,decoration=snake, blue"];
    node [style="ball, ball color = red", label=""];
    a_3 -- c -- a_4 --a_3;
}
\end{dot2tex}
\end{document}
```

The next example shows that the `[`⟨*codeonly*⟩`]` environment option is useful when you want to adjust and annotate the generated graph. Note that you manually have to wrap the code inside a figure environment, but this gives you a lot of flexibility. When using the `tikz` output format, you can later access the nodes.

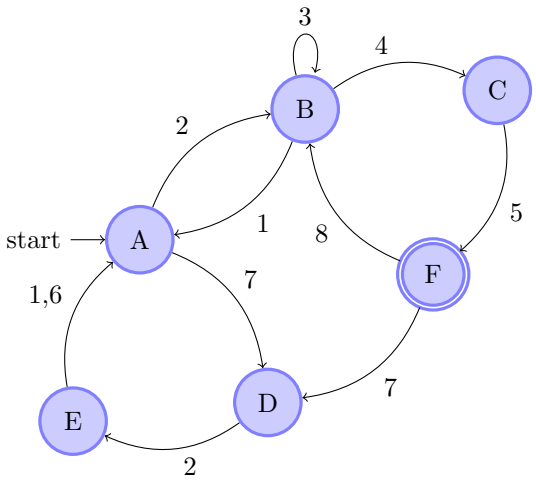



```
\documentclass{article}
\usepackage{tikz}
\usetikzlibrary{arrows,shapes}
\usepackage{dot2texi}
\begin{document}
% Define layers
\pgfdeclarelayer{background}
\pgfdeclarelayer{foreground}
\pgfsetlayers{background,main,foreground}

% The scale option is useful for adjusting spacing between nodes.
% Note that this works best when straight lines are used to connect
% the nodes.
\begin{tikzpicture}[>=latex',scale=0.8]
    % set node style
    \tikzstyle{n} = [draw,shape=circle,minimum size=2em,
                        inner sep=0pt,fill=red!20]
    \begin{dot2tex}[dot,tikz,codeonly,styleonly,options=-s -tmath]
        digraph G  {
            node [style="n"];
            A_1 -> B_1; A_1 -> B_2; A_1 -> B_3;
            B_1 -> C_1; B_1 -> C_2;
            B_2 -> C_2; B_2 -> C_3;
            B_3 -> C_3; B_3 -> C_4;
        }
    \end{dot2tex}
    % annotations
    \node[left=1em] at (C_1.west)  (l3) {Level 3};
    \node at (l3 |- B_1) (l2){Level 2};
    \node at (l3 |- A_1) (l1) {Level 1};
    % Draw lines to separate the levels. First we need to calculate
    % where the middle is.
    \path (l3) -- coordinate (l32) (l2) -- coordinate (l21) (l1);
    \draw[dashed] (C_1 |- l32) -- (l32 -| C_4);
    \draw[dashed] (C_1 |- l21) -- (l21 -| C_4);
    \draw[<->,red] (A_1) to[out=-120,in=90] (C_2);
    % Highlight the A_1 -> B_1 -> C_2 path. Use layers to draw
```

```
    % behind everything.
    \begin{pgfonlayer}{background}
        \draw[rounded corners=2em,line width=3em,blue!20,cap=round]
                (A_1.center) -- (B_1.west) -- (C_2.center);
    \end{pgfonlayer}
\end{tikzpicture}
\end{document}
```

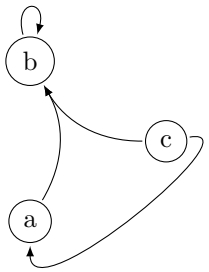
The next example uses the `automata` library to create a nice looking state machine:




```
\documentclass{article}
\usepackage{dot2texi}
\usepackage{tikz}
\usetikzlibrary{automata,shapes}
\begin{document}
\begin{tikzpicture}[
    every state/.style={draw=blue!50,very thick,fill=blue!20}]
\begin{dot2tex}[styleonly,codeonly,neato]
digraph G {
    d2ttikzedgelabels = true;
    node [style="state"];
    edge [lblstyle="auto",topath="bend left"];
    A [style="state, initial"];
    A -> B [label=2];
    A -> D [label=7];
    B -> A [label=1];
    B -> B [label=3,topath="loop above"];
    B -> C [label=4];
    C -> F [label=5];
    F -> B [label=8];
    F -> D [label=7];
    D -> E [label=2];
    E -> A [label="1,6"];
    F [style="state,accepting"];
```

```
}
\end{dot2tex}
\end{tikzpicture}
\end{document}
```

Another example of using the special `topath` attribute. Note the use of the `graphstyle` option to shorten the edges.




```
\documentclass{article}
\usepackage{dot2texi}

\usepackage{tikz}
\usetikzlibrary{shapes,arrows}

\begin{document}
\begin{dot2tex}[circo,graphstyle={shorten >=1pt,shorten <=1pt}]
digraph G {
    mindist = 0.5;
    node [shape="circle"];
    a -> b [topath="bend right"];
    c -> b [topath="bend left"];
    c -> a [topath="out=10,in=-90"];
    b -> b [topath="loop above"];
}
\end{dot2tex}

\end{document}
```

9 Tips and tricks

- Generating the graphs can be quite time consuming. The `shell` and `noshell` environment options are very useful when working with documents with many graphs.
- Use the `outputdir` package option to organize your files.

9.1 External files

Sometimes it is practical to keep a dot graph in a separate file, for instance when the graph is automatically generated. Dot2tex version 2.8 or later sup-

ports an inclusion mechanism similar to LaTeX's. If a graph contains only the line `\input{filename.dot}` it will load `filename.dot` and convert it. Here is an example:

```
...
\begin{dot2tex}
\input{externalgraph.dot}
\end{dot2tex}
..
```

10 Changelog

- Version 3.0 (2008-05-07)
  - Updated documentation examples to use PGF 2.00.
  - Added the experimental `scale` environment option.
  - Added the `graphstyle` environment option.
  - Added the `mathmode` environment option.
  - Added the `tikzedgelabels` environment option.
  - Added the `straightedges` environment option.
  - Added the `outputdir` option and `\setoutputdir` macro.
  - Added the `file` environment option.
- Version 2.0 (2007-12-09)
  - Updated documentation.
  - It is now not necessary for Windows users to specify the `miktex` option. Platform is now detected automatically.
  - Added the `forceshell` package option.
  - Added the `codeonly` environment option.
  - Added the `styleonly` environment option.
  - Added the `autosize` and `noautosize` option.
- Version 1.0. Initial release

Acknowledgements

Thanks to Lars Kotthoff for writing the excellent dottex and gnuplottex packages. The platform detection code is based on the ifplatform[6] package written by Will Robertson and Johannes Große.

Thanks to all users for valuable feedback and suggestions! A special thanks to Rolf Niepraschk for pointing me to the dottex and ifplatform packages.

[6] `http://www.ctan.org/tex-archive/help/Catalogue/entries/ifplatform.html`