April 14, 2012

MyCV*

Author:
Andrea Ghersi

Abstract

This LaTeX class provides a set of functionality for writing *curriculum vitæ* with different layouts. To achieve this goal, it adopts a different approach with respect to the other c.v. classes or packages.
Basically, the idea is that a user can write some custom configuration directives, by means of which is possible both to produce different c.v. layouts and quickly switch among them.
In order to process such directives, this class uses a set of lists, provided by the package *etextools*. A basic support for using the *TikZ* decorations is also provided.

* This file has version number 1.5.3 -- documentation dated April 13, 2012 -- last revised April 14, 2012

CONTENTS

1 FUNDAMENTALS

"Computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty"

-- Knuth [1973]

1.1 INTRODUCTION

The main goal of this class (*MyCV*) is to give support for creating *curriculum vitæ* (CV) with different layouts, allowing easy switching among them. The class also provides a basic support for using the *TikZ* decorations and defines a bunch of commands for handling the contents of a CV, even though this is not its primary goal. On CTAN archives, there are available various CV packages more *contents-oriented*, as it were, and they may be used together with this class, providing missing contents functionality.
Probably, the choice of the class's name was not so appropriate but, at the beginning, I did not plan to publish it and chose the simplest and most obvious name. I realized that only when I wrote these notes, but by then it was a bit too late.
Before starting to describe *MyCV* more in details, I have to say it was my first class in LaTeX and, although I tried to do my best, the lack of experience probably brought me to make some choice not so opportune as I hoped. It goes without saying that any advice or constructive criticism is greatly appreciated.

1.2 CLASS FILES

The class *MyCV* is composed by five files. A short brief of each one is given here:

- mycv.cls

  it is the main file and, basically, handles the class options (section 2.2) as well as the inclusion of all other files (the remaining four);

- mycv_base.def

  it contains all the commands and definitions dealing with the layout components of a CV (section 1.3): it is the core-system file;

- mycv_style.def

  it contains the default style commands (subsection 2.3.2) provided by this class: if the default style is not used, this file will not be included by *mycv.cls*;

- mycv_dec.def

  it contains the decoration commands (subsection 2.3.3): if decorations are not enabled, this file will not be included by *mycv.cls*;

- mycv_misc.def

  it contains some miscellaneous commands and definitions.

1.3 LAYOUT COMPONENTS

This class considers a *curriculum vitæ* as logically divided into three main components: *header*, *body* and *footer*. For each of these ones, a list, that basically contains some sub-components, is being associated; files with the CV contents are also considered sub-components[1]. For these reasons, we can actually say that *MyCV* uses a sort of *list-driven* approach.

1.3.1 Main components

MyCV recognizes the following three lists that, for all intends and purposes, are a concrete representation of the main logical components:

- **headerlayoutlist**;
- **bodylayoutlist**;
- **footerlayoutlist**.

It is mandatory, for the correct behavior of the class, to not change the above list names. In the case a component is not required, the relative list may be omitted: for example, if a CV does not have a footer component, the list *footerlayoutlist* is not strictly necessary. What follows is an example of a list definition:

```
\def\headerlayoutlist{sub-component1,sub-component2,[...]}
```

1.3.2 Sub-components

We previously said that *MyCV* is based on three main components (*header*, *body* and *footer*) and that each of these ones are represented by a list. A list (therefore a main component), in turn, may have one or more sub-component, separated by a comma, which are identified as follows:

- `Main[Header|Body|Footer]PageBegin`;
- `Main[Header|Body|Footer]PageEnd`;
- `Sub[Header|Body|Footer]PageBegin`;
- `Sub[Header|Body|Footer]PageEnd`;
- `filename` with the (partial) CV contents.

Both "`Main[...]PageBegin`" and "`Sub[...]PageBegin`" are *minipages*; the difference is that the former have a default width of 100% of the *textwidth* macro, while that value is 44-45% for the latter (it depends on the components type).
A *filename* sub-component may either directly be the name of a file or a macro (variable): depending on the case, the syntax slightly changes.

Sub-components options

Each sub-component, *filename* included, may have associated options, with colons as separators, so that the syntax is something like:

`sub-component:option1:option2:[...]`.

If truth be told, each option has its own separator, so colons are not strictly necessary and, as a separator, any other symbol may be used. If wanted, it is also possible to not have any,

1 it may be a good practice, to make the best use of this class, to subdivide the contents of each cv section in different files, although this is not mandatory and any other choice may be made

but this is not recommended (just for a matter of clarity). Options for a sub-component are of different types, as listed below:

- <[pre | post]cmd:command1:command2:[...]>

  a sequence of commands is executed *before*/*after* the begin or end of a sub-component (*filename* included). A command may have a sequence of arguments, separated by "="; each of them can either be *optional* or *mandatory*. In total, *MyCV* recognizes four types of arguments:

  - arg (mandatory argument equivalent to {arg});
  - @arg (optional argument equivalent to [arg]);
  - !arg (optional argument equivalent to <arg>);
  - * (optional argument equivalent to *).

- /m[l | r]<value>/ (1)

  /endm[l | r]/ (2)

  changes the *left*/*right* margin of a text portion of a document, between option (1) and option (2); in a typical usage, these options are associated with different sub-components, such as *PageBegin and *PageEnd.
  Each time the option (1) is used, the option (2) is also required for ending the margin modification, except for the *filename* sub-component that automatically does that. Example (it moves the left margin to the right of 0.2in):

  ```
  SubBodyPageBegin:</ml0.2in/>
  [...]
  SubBodyPageEnd:</endml>.
  ```

- <width-value>

  sets the width of a sub-component in terms of *textwidth* percentage. This option only exists for "*PageBegin" sub-components. Example: SubBodyPageBegin:<0.48>.

- /pagesize<value>/

  sets the width of a sub-component, as the option above, but in terms of absolute reference (instead of *textwidth* percentage). Also this option only exists for "*PageBegin" sub-components. Example: SubBodyPageBegin:/pagesize5.5in/.

- /pagebreak/

  permits to break two contiguous sub-components, aligning them one above the other, instead side by side (that is the default behavior). This option only exists for "*PageEnd" sub-components. Example: SubBodyPageEnd:/pagebreak/.

- *varname

  filename@

  ⟨*varname*⟩ is a macro that expands to the name of a file (with the CV contents), while ⟨*filename*⟩ is directly the name itself. Example: *headerfile, where the macro *headerfile* is somewhere defined.

2 USAGE

"There are two ways to write error-free programs; only the third one works"

-- Alan J. Perlis

2.1 REQUIREMENTS

When *decorations* are not enabled and the *default style* is not used, *MyCV* has (apart from LaTeX 2ε) the following requirements:

```
\RequirePackage{kvoptions} % for options
\RequirePackage{etextools} % for lists and other useful tools
\RequirePackage{ifthen}    % for \ifthenelse command
\RequirePackage{pifont}    % for ding style (itemize environment)
\RequirePackage{xstring}   % for string utilities
\RequirePackage{svn-prov}  % for file info extracted from SVN
\RequirePackage{xparse}    % for commands with multiple default arguments
```

In addiction, if the default style is used, by means of the class option "*style*" (section 2.2), this class requires:

```
\RequirePackage{titlesec}  % for title format and spacing
\RequirePackage{fancyhdr}  % for custom footer
\RequirePackage{xcolor}    % for color
\RequirePackage{calligra}  % for calligra font
\RequirePackage{times}     % times font
\RequirePackage{marvosym}  % symbols - phone
\RequirePackage{amssymb}   % symbols - email
```

Finally, if decorations are enabled, by using the class option "*withDec*" (section 2.2), this class also requires:

```
RequirePackage{tikz} % for graphics
```

2.2 CLASS OPTIONS

MyCV can use any option supported by the *article* class, on which is based. In addiction, it provides the following options:

- language=<⟨*string*⟩>

  string language to pass to the *babel* package for the document (CV) language;

- cntdir=<⟨*dirname*⟩>

  sets the directory name where *MyCV* will search for files with the CV contents. The default one is "Contents";

▷ style=<⟨*filemane*⟩>

specifies the file name (without the extension ".tex" if any) with the style commands. By default, the file *mycv_style.def*, provided by the class itself, is that used. It is also possible to not use any file by specifying the value "none" as file name;

▷ mdlname=<⟨*name*⟩>

registers a name for the layout (model) intended to be used: in this way is possible, for example, to select the appropriate layout configuration file or a layout-specific portion of code;

▷ withDec

enables support for decorations (provided by the *TikZ* package);

2.3 CLASS COMMANDS

Here follows the complete list of the commands provided by *MyCV*. The style commands are only available if the class option "*style*" was used (section 2.2). The same goes for the decoration commands, which need the class option "*withDec*" to be used. In the following text of this section, when present, the form [...] (or <...>) indicates the default choice for an optional argument of a command.

2.3.1 Conditionals

▷ **\ifoption** {⟨*option*⟩} {⟨*true*⟩} {⟨*false*⟩}

\ifmodel {⟨*mdlname*⟩} {⟨*true*⟩} {⟨*false*⟩}

ifoption checks whether ⟨*option*⟩ was used, while *ifmodel* checks whether ⟨*mdlname*⟩ was registered in the class; then both commands use the appropriate ⟨*true*⟩ or ⟨*false*⟩ block of code.

2.3.2 Default style

▷ **\mysectionTitleFormat**

[⟨*titlerule-color-above*⟩] ⟶ [*myheadingscolor*]

[⟨*titlerule-color-below*⟩] ⟶ [*myheadingscolor*]

⟨*titlerule-color-above*⟩ is the color for the rule above a section name, while ⟨*titlerule-color-below*⟩ is for that below. *myheadingscolor* is the default color.

▷ **\mysectionTitleSpacing**

[⟨*left*⟩] ⟶ [0*pt*] [⟨*beforesep*⟩] ⟶ [0*pt*] [⟨*aftersep*⟩] ⟶ [5*pt*]

this command is just an alias for *\titlespacing{\section}{⟨left⟩}{⟨beforesep⟩}{⟨aftersep⟩}*. See the *titlesec* package for further information.

▷ **\mycfoot** {⟨*text*⟩}

adds ⟨*text*⟩ to the page footer. It may be useful, for example, to show information about the last document update.

2.3.3 Decorations

MyCV provides some commands for using the *TikZ* decorations. The support provided is not complete at all (on the other hand *TikZ* has a huge amount of functionality), but is enough for this class purposes. The only *TikZ* path supported is *rectangle*.

▷ **\mydecorationsPathmorphing**[*]

[⟨*show-decoration*⟩] ⟶ [1]

{⟨*decoration-type*⟩}

[⟨*decoration-color*⟩] ⟶ [*gray*]

<⟨*shading-type*⟩> ⟶ <*radial*>

<⟨*background-color*⟩> ⟶ <*white*>

⟨*show-decoration*⟩, if equals 1, does show the decoration ⟨*decoration-type*⟩, while if 0 does not. *Starred* version uses the shading technique, unlike the *not starred* one, and the last argument is the background shading color.
For *not starred* version, the argument ⟨*shading-type*⟩ is not considered (just for a matter of clarity, a "none" value may be used), and the last argument is simply the background color.
⟨*decoration-type*⟩ was tested with the following values: "shape", "straight", "zigzag", "random steps", "saw", "bent", "bumps", "coil", "snake" and "Koch snowflake".
⟨*shading-type*⟩ was tested with "radial" and "ball" shadings.

▷ **\mydecorationsShape**

[⟨*show-decoration*⟩] ⟶ [1] {⟨*decoration-type*⟩} [⟨*decoration-color*⟩] ⟶ [*gray*]

⟨*show-decoration*⟩, if equals 1, does show the decoration ⟨*decoration-type*⟩, while if 0 does not. ⟨*decoration-type*⟩ was tested with the following ones: "dart", "diamond", "rectangle" and "star".

▷ **\mydecorationsFading**

[⟨*path-fading*⟩] ⟶ [*north*]

{⟨*primary-color*⟩}

[⟨*color-gradient*⟩] ⟶ [80]

[⟨*secondary-color*⟩] ⟶ [*black*]

<⟨*opacity*⟩> ⟶ <1.0>

the resulting fill color is given by ⟨*primary-color*⟩, ⟨*color-gradient*⟩ and ⟨*secondary-color*⟩, which are composed as follows: ⟨*primary-color*⟩!⟨*color-gradient*⟩!⟨*secondary-color*⟩.

▷ **\mydecorationsSetPos[XTL | YTL | XBR | YBR]**

[⟨*coordinate-value*⟩] ⟶ [1*cm* | −1*cm* | −1*cm* | 1*cm*]

sets the position for the decoration in use. Since the decoration path is *rectangle*, it is sufficient to have the (x, y) coordinates of two points: the top-left and bottom-right. *XTL* stands for "`X-Top-Left`", *XBR* for "`X-Bottom-Right`" and so on.

▷ **\mydecorationsSetLineWidth**[*] [⟨*line-width*⟩] ⟶ [*tikz value*]

\mydecorationsSetSegmentAmplitude[*] [⟨*segment-amplitude*⟩] ⟶ [*tikz value*]

\mydecorationsSetSegmentLength[*] [⟨*segment-length*⟩] ⟶ [*tikz value*]

these commands may respectively be used for modifying the properties ⟨*line-width*⟩, ⟨*segment-amplitude*⟩ and ⟨*segment-length*⟩ for the decoration in use. *Starred* versions do not require any argument and reinitialize the properties to their default values.

2.3.4 Miscellaneous

▷ **\mypdfauthor** {⟨*author*⟩}

\mypdftitle {⟨*title*⟩}

\mypdfsubject {⟨*subject*⟩}

these commands do nothing but register ⟨*author*⟩, ⟨*title*⟩ and ⟨*subject*⟩ information in the document properties of the pdf is being produced.

▷ **\mylang** {⟨*text*⟩} [⟨*language*⟩] ⟶ [*english*]

temporarily changes the language in use (*babel* package) to ⟨*language*⟩ for ⟨*text*⟩.

▷ **\myitemize**

a list environment that uses the *ding* style.

▷ **\mychangemargin** {⟨*left-margin*⟩} {⟨*right-margin*⟩}

mychangemargin environment changes the left and right margin of a portion of text. The environments *mychangemarginLeft* and *mychangemarginRight*, whose meaning is straight forward, are also available.

▷ **\myrenderlayout** [⟨*component*⟩] ⟶ [*a*]

processes and draws the layout component(s). The option value “h” is for the header component, “b” and “f”, respectively, for the body and footer ones, while “a” is for all components.

2.4 AN EXAMPLE

This section gives a *minimal* example and some considerations about the use of *MyCV* (the class permits to do much better with a little patience). This is done by creating two *curriculum vitæ* with the same contents, but different layouts: one **CV** will use a double page layout (abbreviated **DPL** from here forward), while the other will use a single page layout (**SPL**).
The sample code presented here can be found in the “Examples” directory shipped with the *mycv* bundle, which this document is part of, and that also contains files with the **CV** contents: these files are not listed in the present document, as they do not contain anything worth being mentioned for the purpose of these notes.
First and foremost, to keep the code organized, we need a file containing the layout components for the **DPL** (*model-dpl.tex*) and another one for the **SPL** (*model-spl.tex*). We opt for

having the header and footer components being shared, so we create a third file named *model-common.tex* such as this:

Listing 2.1: model-common.tex

```
% file with the common layout components: header and footer
\ifmodel{verDPL}{%
   \newcommand{\cvdec}{%
      mydecorationsSetLineWidth=@0.3mm:%
      mydecorationsPathmorphing=*=coil=!radial=!lightgray%
   }
}{\newcommand{\cvdec}{}}
\newcommand{\sectionnumber}[1]{\section{Section #1}}

\def\headerlayoutlist{%
   MainHeaderPageBegin:<postcmd:vspace=10pt>,
      % ---------------------------------------- left header
      SubHeaderPageBegin:<precmd:\cvdec:hfill>,
         header_title@,    % header file one
      SubHeaderPageEnd:<postcmd:hfill>,
      % ---------------------------------------- right header
      SubHeaderPageBegin,
         header_contacts@, % header file two
      SubHeaderPageEnd,
   % ----------------------------------------
  MainHeaderPageEnd
}

\def\footerlayoutlist{footer_sign@} % footer file
```

Notice that in listing 2.1, for the DPL, we used some decoration commands. Obviously, it is also possible to not use decorations (simply by not using the relative class option), but in that case the showed code would yield some compilation errors, since decoration commands would be called without being defined.
A possible solution makes the use of the conditional command *\ifoption*, so that the first part of the code would become something like in listing 2.2.

Listing 2.2: model-common-hint

```
\ifmodel{verDPL}{%
   \ifoption{withDec}{%
      [...]
   }{\newcommand{\cvdec}{}}
}{\newcommand{\cvdec}{}}
```

Now we can deal with the layout components specific for the DPL (*model-dpl.tex*) as in listing 2.3.

Listing 2.3: model-dpl.tex

```
\input{Models/model-common}
\def\bodylayoutlist{%
   % ------------------------------------------------------
   % moves the right margin to the left (text and title rules)
   % ------------------------------------------------------
   MainBodyPageBegin:<0.96>,
      % ------------------------------------------------------
      % the 2 directives below are just used as a trick to do the
      % same thing for the left margin (it is moved to the right)
```

```
      % ------------------------------------------------------
      SubBodyPageBegin,
      SubBodyPageEnd,
      % ----------------------------------------
      % left page (0.48 of textwidth)
      % ----------------------------------------
      SubBodyPageBegin:<0.48>,
         contents_partA@:<precmd:vspace=10pt:sectionnumber=1>,
         contents_partB@:<precmd:vspace=10pt:sectionnumber=2>,
         contents_partC@:<precmd:vspace=10pt:sectionnumber=3>,
      SubBodyPageEnd:<postcmd:hfill>,
      % ----------------------------------------
      % right page (0.48 of textwidth)
      % ----------------------------------------
      SubBodyPageBegin:<0.48>,
         contents_partA@:<precmd:vspace=10pt:sectionnumber=4>,
         contents_partB@:<precmd:vspace=10pt:sectionnumber=5>,
      SubBodyPageEnd,
      % ----------------------------------------
   MainBodyPageEnd%
}
```

As far as the **DPL**, we have done; we still have to deal with the layout components specific for the **SPL** (*model-dpl.tex*). In this case, we do not need to use `*PageBegin` components, but it is sufficient to directly include the files with the contents. The resulting code is showed in listing 2.4.

Listing 2.4: model-spl.tex

```
\input{Models/model-common}
\def\bodylayoutlist{%
   % ------------------------------------------------------
   contents_partA@:<precmd:vspace=10pt:sectionnumber=1>,
   contents_partB@:<precmd:vspace=10pt:sectionnumber=2>,
   contents_partC@:<precmd:vspace=10pt:sectionnumber=3>,
   contents_partA@:<precmd:vspace=10pt:sectionnumber=4>,
   contents_partB@:<precmd:vspace=10pt:sectionnumber=5>
   % ------------------------------------------------------
}
```

At this point, we both have the components for the double and single page layouts and we can proceed writing the main files (*mycv-example-dpl.tex* and *mycv-example-spl.tex*) that pick and use them.
We start by setting up some options for the *MyCV* class; we have chosen to store the **CV** contents files in the directory "Contents" (that is the default one where the class searches for the contents files), so there is not need to specify the directory path with the option "*cntdir*" (section 2.2).
The options we want to pass to the class are those related to the decorations support and language; in addiction, we pass the name of the layout (model) we mean to use.
Here we take the **DPL** as an example (listing 2.5), but switching to the **SPL** would just be a matter of changing the "*mdlname*" option from **verDPL** to **verSPL**.

Listing 2.5: mycv-example-dpl.tex

```
\documentclass[10pt,mdlname=verDPL,withDec,language=english]{mycv}
\input{mycv-example-common}
```

What remains to do is just to include the appropriate layout components file (2.3 or 2.4) and process them with the *\myrenderlayout* command, as showed in listing 2.6; since this portion of code is shared between *mycv-example-dpl.tex* and *mycv-example-spl.tex*, we need to use *\ifmodel* for selecting the right file to be included.

Listing 2.6: mcv-example-common.tex

```
[...]
\ifmodel{verSPL}{\input{Models/model-spl}}{\relax}
\ifmodel{verDPL}{\input{Models/model-dpl}}{\relax}

\begin{document}
\myrenderlayout % all components
[...]
\end{document}
```

2.5 SPLIT THE CONTENTS

When a double layout page is used, it may occur, for example, that a section is too long for a page: this would not be a problem with a single layout page, since LaTeX would automatically break the section contents. Unfortunately, with a double page layout the behavior is substantially different: this is because *MyCV* uses a *minipage-based mechanism* and a minipage is by itself not breakable. Thus, what happens is that part of the section contents comes out from the margins, without being displayed.
When a problem such as this occurs, a possible workaround is to manually break the section contents. This can be done by using a counter that keeps track of the number of times a same file is included: when the counter is equal 1, a part of the section contents is included in the left page, otherwise is the remaining one to be included in the right page. Listing 2.7 shows a practical example of what just discussed.

Listing 2.7: split-contents-example

```
% ------------------------------------------------------
% file with the section contents: i.e. <section_skills.tex>
% ------------------------------------------------------
% increase the counter 'acounter', that is defined outside this file
\stepcounter{acounter}
\newcommand{\conditionalblock}[2]{\ifthenelse{\value{acounter}<2}{#1}{#2}}
\textconditionalblock%
   {skills section contents part A}%
   {skills section contents part B (the remaining part)}
% ------------------------------------------------
% file with the DPL components: i.e. <model-dpl.tex>
% ------------------------------------------------
\def\bodylayoutlist{%
   SubBodyPageBegin:<0.48>, % left page
      % include part A in the left page
      section_skills@,
      [...]
   SubBodyPageEnd,
   SubBodyPageBegin:<0.48>, % right page
      % include part B in the right page
      section_skills@,
      [...]
   SubBodyPageEnd
}
```

Of course the proposed workaround is not the best we could wish for, since it requires manual operations. Unfortunately, at the moment, I would not know how to solve the problem with a brighter idea. Any advice or comment is greatly appreciated, either for the specific problem or for this work in general.

That's all, happy LaTeXing!

Andrea Ghersi