May 20, 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 *TikZ* decorations is also provided.

* This file has version number 1.5.6 -- documentation dated April 13, 2012 -- last revised May 20, 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 basic support for using the *TikZ* decorations and defines a bunch of commands for managing the contents of a CV, though that 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. Before starting to describe the class more in details, it goes without saying that any advice or constructive criticism is greatly appreciated.

1.2 CLASS FILES

The class *MyCV* is composed by six 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, except for *mycv_version.def*;

- 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 file;

- mycv_style.sty

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

- mycv_dec.sty

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

- mycv_misc.def and mycv_version.def

  they respectively contain some miscellaneous commands and the version string.

1.3 LAYOUT COMPONENTS

This class considers a *curriculum vitæ* as logically divided into three main components: *header*, *body* and *footer*. To each of these ones, a list, that basically contains some sub-components, is associated; files with the CV contents are also considered sub-components. 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's 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 be the name of a file or a macro with it: 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 already 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 separator at all, but this is not recommended if only 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 beginning or ending of a sub-component (*filename* included). A command may have a sequence of arguments, separated by "=", each of which can either be *optional* or *mandatory*. In total, the class 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 of side by side (that is the default behavior). This option only exists for "*PageEnd" sub-components. Example: SubBodyPageEnd:/pagebreak/.

- *macroname

  filename@

  ⟨*macroname*⟩ is a macro expanding to the name of a file (with the CV contents), while ⟨*filename*⟩ is directly the name itself (the only *non-alphanumeric* characters allowed there are "_" and "-"). 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 the following requirements:

```
\RequirePackage{kvoptions} % for class options with key-value format
\RequirePackage{etextools} % for lists and other useful tools
\RequirePackage{ifthen}    % for the \ifthenelse command
\RequirePackage{xstring}   % for string utilities
\RequirePackage{svn-prov}  % for file info extracted from SVN
\RequirePackage{hyperref}  % for hypertext links and other stuff
```

If the default style is used, by means of the class option "*style*" (section 2.2), this class requires (in addiction to *hyperref* and *svn-prov*):

```
\RequirePackage{xparse}    % for commands with multiple default arguments
\RequirePackage{pifont}    % for the 'ding' style (itemize environment)
\RequirePackage{titlesec}  % for title format and spacing
\RequirePackage{fancyhdr}  % for custom headers and footers
\RequirePackage{xcolor}    % for colors
\RequirePackage{calligra}  % for the calligra font
\RequirePackage{times}     % for the 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 (in addiction to *svn-prov*, *xparse* and *xstring*):

```
\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 (with or without the extension ".sty") containing the style commands. By default, the file *mycv_style.sty*, provided by the class itself, is that used. It is also possible to not use any style 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; the same goes for the decoration commands, which need the class option "*withDec*".
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 the one 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 update of the document.

▷ **\myitemize**

a list environment that uses the *ding* style.

2.3.3 Decorations

MyCV provides some commands for *TikZ* decorations. The support provided is not complete at all (on the other hand *TikZ* has a huge amount of functionality), but it 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, shows the decoration ⟨*decoration-type*⟩, while if 0 does not. The *starred* version of the command uses the shading technique and the last argument is the background shading color.
The *not starred* version does not consider the argument ⟨*shading-type*⟩ (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, shows the decoration ⟨*decoration-type*⟩, while if 0 does not. ⟨*decoration-type*⟩ was tested with the following decorations: "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*⟩.

▷ \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 SOME EXAMPLES

This section gives some *minimal* examples and does 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** and **SPL** (*model-layouts.tex*). We opt to share the *header* and *footer* components, so we also create a second file named *model-common.tex*, such as in listing 2.1.

Listing 2.1: model-common.tex

```
% ----------------------
% the shared header list
% ----------------------
\def\headerlayoutlist{%
   MainHeaderPageBegin:<postcmd:vspace=10pt>,
      % --------------------------------- left header
      SubHeaderPageBegin:<precmd:hfill>,
         % header file (1)
         header_title@,
      SubHeaderPageEnd:<postcmd:hfill>,
      % --------------------------------- right header
      SubHeaderPageBegin,
         % header file (2)
         header_contacts@,
      SubHeaderPageEnd,
   MainHeaderPageEnd%
}

% ----------------------
% the shared footer list
% ----------------------
\def\footerlayoutlist{footer_sign@}
```

2.4.1 Double page layout

Here we deal with the layout components specific for the **DPL**, as showed in listing 2.2.

Listing 2.2: DPL model (part of 'model-layouts.tex')

```
\def\bodylayoutlist{% the DPL's body list
   % ---------------------------------------------------------
   % 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%
}
```

2.4.2 Single page layout

As far as the **SPL**, we do not need to use the `*PageBegin` components, but it is sufficient to directly include the files with the contents. The resulting code is showed in listing 2.3.

Listing 2.3: SPL model (part of 'model-layouts.tex')

```
\def\bodylayoutlist{% the SPL's body list
   % ---------------------------------------------------
   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>
   % ---------------------------------------------------
}
```

2.4.3 Main file

Since we both have the components for the double and single page layouts, we can proceed writing the main file (*mycv-example-main.tex*) that picks and use them.
We start by setting up some options for the class, such as those related to the decorations and language support, as well as the name of the model (layout) we mean to register. Besides, we opt to store the **CV** contents files in the current directory (that is not the default one where the class searches for the contents files), so there is need to specify its path with the option "*cntdir*".
In listing 2.4 we take the **DPL** as an example, but switching to the **SPL** would just be a matter of changing the "*mdlname*" option from **verDPL** to **verSPL**.

Listing 2.4: mycv-example-main.tex

```
\documentclass[10pt,mdlname=verDPL,withDec,cntdir=.,language=english]{mycv}
\input{mycv-example-common}
\begin{document}
\cvdec\myrenderlayout\mycfoot{Last update: \today}
\end{document}
```

The file *mycv-example-common[.tex]*, showed in listing 2.5, selects the appropriate layout components (listings 2.2 or 2.3) to be included and, subsequently, processed by *\myrenderlayout*. In addiction, the file *mycv-example-common.tex* contains some decoration commands, which need to be used only if the option "*withDec*' was given to the class; this is the reason of the conditional command *\ifoption*.

Listing 2.5: mcv-example-common.tex

```
[...]
\ifoption{withDec}{%
   \newcommand{\cvdec}{
      \mydecorationsSetLineWidth[0.3mm]%
      \mydecorationsPathmorphing*{coil}<radial><lightgray>
   }
   [...]
}{\newcommand{\cvdec}{}}

% include layouts components
\input{model-layouts}
[...]
```

2.4.4 Layout notes

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 page layout, since LaTeX would automatically break the section contents. Unfortunately, with a double page layout the behavior is substantially different: this is because the class 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 page margins.
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 2.6 shows a practical example of what just discussed.

Listing 2.6: workaround example

```
% ------------------------------------------------------
% file with the section contents: i.e. <section_skills.tex>
% ------------------------------------------------------
% increases the counter 'cnt': it's defined outside this file
\stepcounter{cnt}

% selects the appropriate part of the file contents
\newcommand{\condblock}[2]{\ifthenelse{\value{cnt}<2}{#1}{#2}}

\condblock{skills section contents part A}%
   {skills section contents part B (the remaining part)}

% ------------------------------------------------------
% file with the DPL components: i.e. <model-layouts.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. As an alternative, it is possible to not use the `*PageBegin` and `*PageEnd` mechanism, delegating the double page layout to an external package (i.e. *multicols*). This kind of approach is showed in listings 2.7.

Listing 2.7: DPL2 model (part of 'model-layouts.tex')

```
\newcommand{\mcbegin}{\begin{multicols}{2}}
\newcommand{\mcend}{\end{multicols}}

\def\bodylayoutlist{%
    contents_partA@:<precmd:vspace=10pt:mcbegin:sectionnumber=1>,%
    [...]
    contents_partB@:<precmd:vspace=10pt:sectionnumber=6>:<postcmd:mcend>,%
}
```

Notice the usage of the commands *\mcbegin* and *\mcend*, which act as a wrapper for the *multicols* environment.

2.5 SPLIT FILE CONTENTS

This class uses a file based approach as far as the contents of a CV, that is to say it needs that the contents be into separated file/s (at least one) with respect to the main file.

As the class works, having the contents in just one single file is not as good as would be with multiple files (ideally a file for each section of the document), since multiple files allow to more easily customize the **CV** layout without directly acting on the contents. On the other hand, a multiple file approach may compel to use several files and not be convenient.
For these reasons, this class provides a **PERL** script named *mycv_split_contents.pl* (for a list of all the options provided, type *mycv_split_contents.pl -h* inside a shell environment) that can process a file and, according to what specified by the directives in the file itself, split its contents into several files (which will be those effectively used by the class). In this way is possible to directly manage one single file for the contents.
If requested, the script can also create a basic model file with the layout components: it has to be considered just as a skeleton and probably needs to be edited by hand afterward.
The directives that may be used inside a contents file have the following form:

```
###::filename:component:commands
```

filemane is the name of the file to create, *component* is the main component (*header*, *body* or *footer*) that the file represents, while *commands* are the relative commands associated to the component. Both *component* and *commands* are not mandatory, but may be useful, especially *component*, when is requested to generate a basic model file.
The script *mycv_split_contents.pl* may be called inside the main file as showed in listing 2.8.

Listing 2.8: Split contents example

```
% ------------------------------------------------------------------
% To compile this file is necessary to use the option '-shell-escape'
% to enable the 'write18' construct.
% ------------------------------------------------------------------
\documentclass[10pt,mdlname=verSPL,withDec,language=english]{mycv}
\immediate\write18{<dirpath>/mycv_split_contents.pl -i cv-contents.tex}
[...]
```

When the file *cv-contents.tex* (listing 2.9) is processed by the script, the files **.tex* are created in the directory "Contents" (the default one).

Listing 2.9: cv-contents.tex

```
% ----------------------------
% ###::header_title.tex::header
% ----------------------------
[...]
% ------------------------------------------------------------------
% ###::contents_partA.tex::body::<precmd:vspace=10pt:sectionnumber=1>
% ------------------------------------------------------------------
[...]
% ------------------------------------------------------------------
% ###::contents_partB.tex::body::<precmd:vspace=10pt:sectionnumber=2>
% ------------------------------------------------------------------
[...]
% ----------------------------
% ###::footer_sign.tex::footer
% ----------------------------
[...]
```

That's all, happy LaTeXing!

Andrea Ghersi