# The package yagusylo*

Le TEXnicien de surface†

February 27, 2009

**Abstract**

This package enables you to obtain a symbol whithout loading the package which usually provides it in order to avoid name clashes.

It could be considered as an extension of pifont gone technicolor. It's behaviour is controlled by keys using xkeyval.

The English documentation for the user is available in `yagusylo-en.pdf`.

**Résumé**

Cette extension permet d'obtenir un symbole sans avoir à charger l'extension qui le fournit habituellement. Cela permet d'éviter des conflits de noms.

On peut la considérer comme une extension de pifont en technicolor. Son comportement est controlé à l'aide de clés grâce à xkeyval.

La documentation en français pour l'utilisateur est `yagusylo-fr.pdf`.

## Contents



*This document corresponds to the file yagusylo v1.2, 2009/02/26, 50th anniversary edition.

†le.texnicien.de.surface@wanadoo.fr

# 1 The code

We begin with the usual salutations and then load some packages

```
\RequirePackage{xifthen}
\RequirePackage{suffix}
\RequirePackage{xargs}
```

xargs requires xkeyval and xifthen requires etex, calc, ifthen, and ifmtarg so no need to ask for them again but they are all available from now on.

Two macros the behaviour of which will be controlled by options on the document body level. For the moment, no info, no warning.

```
\newcommand*{\Y@GINFO}[2][]{}
\newcommand*{\Y@GWARNING}[3]{}
```

The first global key controls the symfam

```
\define@choicekey*+[Y@G]{yagusylo.sty}{symfam}[\Y@G@SymFamChoice\nr]%
 {marvosym,fourier,wasysym,bbding,dingbat,ark,ifsym,ifsymgeo,%
  ifsymgeonarrow,ifsymgeowide,ifsymclock,ifsymweather,pifont}%
 [pifont]%
 {\Y@GINFO{You choose \Y@G@SymFamChoice}}%
 {\PackageError{yagusylo}%
   {The symbols family ``\Y@G@SymFamChoice'' is not yet known.}
   {I don't know the family you required.\MessageBreak
    If it's not a typo you may consider to contact me
    (TdS)\MessageBreak to obtain support for it.}}
```

the boolean key `color` will control the loading of xcolor thereafter

```
\define@boolkey[Y@G]{yagusylo.sty}[Y@G@]{color}[false]{}
```

the boolean key `configfile` will control the inputing of the configuration file, if it exists,

```
\define@boolkey[Y@G]{yagusylo.sty}[Y@G@]{configfile}[false]{}
```

The keys `info` and `onerror` control the behaviour of yagusylo

```
\define@choicekey*+[Y@G]{yagusylo.sty}{info}[\Y@G@Info\nr]%
  {mute,normal,verbose}[normal]{}{}
\define@choicekey*+[Y@G]{yagusylo.sty}{onerror}[\Y@G@OnError\nr]%
  {nice,tough}[tough]{}{}
```

It is possible to pass options to xcolor with the following key

```
\define@key[Y@G]{yagusylo.sty}{XcolorOptions}[!Y@G!]%
  {\def\Y@GXcolorOptions{#1}}
```

It's now time to set the default values of the global keys, read the options given by the user and relax the macro `\nr` which wont be mentioned again.

```
\ExecuteOptionsX[Y@G]{symfam,color,XcolorOptions,info,onerror,configfile}
\ProcessOptionsX[Y@G]\relax
\let\nr\relax
```

Depending of the value given to the key `color` we load xcolor or not. If the value of `XcolorOptions` is the default `!Y@G!` xcolor is loaded without any option, if not xcolor is loaded with the options provided by the user. The control of the validity of those options will be done by xcolor itself.

```
\ifY@G@color
\ifthenelse{\equal{\Y@GXcolorOptions}{!Y@G!}}
  {\RequirePackage{xcolor}}
```

```
  {\RequirePackage[\Y@GXcolorOptions]{xcolor}
    \PackageInfo{yagusylo}{Package xcolor loaded with options:
      \Y@GXcolorOptions}}
\define@key[Y@G]{yagusylo.sty}{symcolor}[red]{\def\Y@GSymColor{#1}}
\setkeys[Y@G]{yagusylo.sty}{symcolor}
\newcommand\Y@G@Couleur[1]{\color{#1}}
\else
\define@key[Y@G]{yagusylo.sty}{symcolor}[]{}
\newcommand\Y@G@Couleur[1]{\relax}
\def\Y@GSymColor{\relax}
\fi
```

We don't want the global options `color`, `info` and `onerror` to be changed afterwards so we disable them.

```
\disable@keys[Y@G]{yagusylo.sty}{color,onerror,info}
```

`\setyagusylokeys` The macro requires an argument, if it is a `*` the default settings of the global keys are redone, in the other case, the user is meant to provide legal pairs of key-value.

```
\newcommand\setyagusylokeys[1]{%
  \ifthenelse{\equal{#1}{*}}%
  {\setkeys[Y@G]{yagusylo.sty}{%
      symfam,symcolor,%
      leadtype,symplace,sympos,boxwidth,before,after,%
      head,tail}}%
  {\setkeys[Y@G]{yagusylo.sty}{#1}}}
```

Whatever the value of `onerror` key, yagusylo will give, at the end of its loading, a summary of the global setup which is enforced at the beginning of the document:

```
\AtEndOfPackage{%
\PackageInfo{yagusylo}{^^J%
============================================================^^J%
^^J%
****************** YAGUSYLO GLOBAL SETUPS ******************^^J%
^^J%
==================== GENERAL OPTIONS =====================^^J%
^^J%
option ``info'' ......... is ``\Y@G@Info''^^J%
option ``onerror'' ...... is ``\Y@G@OnError''^^J%
option ``color'' ........ is \ifY@G@color``true''\else``false''\fi^^J%
option ``configfile'' ... is \ifY@G@configfile``true''\else``false''\fi^^J%
^^J%
-------------------- CHANGEABLE OPTIONS --------------------^^J%
^^J%
\ifY@G@color
option ``symcolor'' ....... is ``\Y@GSymColor''^^J\fi
option ``symfam'' ......... is ``\Y@G@SymFamChoice''^^J%
option ``leadtype'' ....... is ``\Y@Gyagfillleadtype''^^J%
option ``symplace'' ....... is ``\Y@Gyagfillsymplace''^^J%
option ``sympos'' ......... is ``\Y@Gyagfillsympos''^^J%
option ``boxwidth'' ....... is ``\Y@Gyagfillboxwidth''^^J%
option ``before'' ......... is ``\Y@Gyagfillbefore''^^J%
option ``after'' .......... is ``\Y@Gyagfillafter''^^J%
option ``head'' ........... is ``\the\Y@Glinehead''^^J%
option ``tail'' ........... is ``\the\Y@Glinetail''^^J%
```

```
:::::::::::::::::: OPTIONS FOR YAGENUMERATE ::::::::::::::::::^^J%
option ``firstitemnum'' ... is ``\number\Y@GEnumFirstItemNum''^^J%
option ``enumlength'' ..... is ``\number\Y@GEnumLength''^^J%
\ifY@G@color
option ``symcolor'' ....... is ``\Y@GEnumSymColor''^^J\fi
option ``symfam'' ......... is ``\Y@G@EnumSymFamChoice''^^J%
------------------------------------------------------------^^J%
^^J%
The known patterns for environment ``yagenumerate'' are:^^J%
``piwcr'' (default), ``piwcs'', ``pibcr'', and ``pibcs''.^^J%
^^J%
============================================================%
\@gobbletwo}%
```

Always at the end of the loading, we redefine the info and warning/error commands:

```
\ifthenelse{\equal{\Y@G@Info}{verbose}}
  {\renewcommand*{\Y@GINFO}[2][0]{\PackageInfo{yagusylo}{#2}}}{}%
\ifthenelse{\equal{\Y@G@Info}{normal}}
  {\renewcommand*{\Y@GINFO}[2][0]{%
      \ifthenelse{#1>0}{\PackageInfo{yagusylo}{#2}}{}}{}%
  \ifthenelse{\equal{\Y@G@OnError}{tough}}
  {\renewcommand*{\Y@GWARNING}[3]{\PackageError{yagusylo}{#1}{#3}}}{}
  \ifthenelse{\equal{\Y@G@OnError}{normal}}
  {\renewcommand*{\Y@GWARNING}[3]{\PackageError{yagusylo}{#1}{#3}}}{}
  \ifthenelse{\equal{\Y@G@OnError}{nice}}
  {\renewcommand*{\Y@GWARNING}[3]{\PackageWarning{yagusylo}{#1#2}}}{}}
```

This is the first utility macro which is not really more than an abbreviation. `Y@G` is the prefix of all internal names.

```
\newcommand\Y@G@U@FamilyDef[1]{\fontencoding{U}\fontfamily{#1}}
```

And then two other shortcuts.

```
\newcommand\Y@Gif[3]{%
  \ifthenelse{\equal{#1}{#2}}{\Y@G@U@FamilyDef{#3}}{}}
\newcommand\Y@GifE[4]{%
  \ifthenelse{\equal{#1}{#2}}{\Y@G@U@FamilyDef{#3}#4}{}}
```

The internal macro.

```
\newcommand\Y@Gyagding[3]{{%
    \Y@G@GetSymb{#1}\selectfont\Y@G@Couleur{#3}\char#2}}
```

`\yagding` This user-level macro is defined taking advantage of the features of `\newcommandx` of xargs which enable us to have an optional argument before and another after the mandatory one. The default values of 1st and 3rd arguments ensure that the produced ding will be chosen in current default symfam with current default colour, if there is any.

```
\newcommandx\yagding[3][1=\Y@G@SymFamChoice,3=\Y@GSymColor]{{%
    \Y@G@GetSymb{#1}\selectfont\Y@G@Couleur{#3}\char#2}}
```

The `\Y@G@GetSymb` macro selects the correct font setting from the symbolic name of the symfam. There is no check mechanism for the argument is passed through the key-value mechanism and so checked at giving time.

```
\newcommand\Y@G@GetSymb[1]{%
  \Y@GifE{#1}{marvosym}{mvs}{\fontseries{m}\fontshape{n}}%
```

```
  \Y@Gif{#1}{fourier}{futs}%
  \Y@Gif{#1}{wasysym}{wasy}%
  \Y@Gif{#1}{bbding}{ding}%
  \Y@Gif{#1}{dingbat}{dingbat}%
  \Y@Gif{#1}{ark}{ark}%
  \Y@Gif{#1}{ifsym}{ifsym}%
  \Y@Gif{#1}{ifsymgeo}{ifgeo}%
  \Y@GifE{#1}{ifsymgeonarrow}{ifgeo}{\fontshape{na}}%
  \Y@GifE{#1}{ifsymgeowide}{ifgeo}{\fontshape{w}}%
  \Y@Gif{#1}{ifsymclock}{ifclk}%
  \Y@Gif{#1}{ifsymweather}{ifwea}%
  \Y@Gif{#1}{pifont}{pzd}}
```

`\defdingname` Next user-level macro relies on the `\newcommandx` to enable 3 optional arguments, the last one placed after the mandatory ones. The 4th argument is used to build the name of the internal macro with `\edef` or `\xdef` depending of the value of the 2nd (optional) argument which defaults to `local`.

This macro issues an info message if `info` key has value `verbose`.

To facilitate the use of the 2nd argument we provide a shortcut for the 1st one. It's because you have to give the 1st explicity to be able to give the 2nd explicity as well.

```
\newcommandx\defdingname[5][1=*,2=local,5=\Y@GSymColor]{%
  \Y@GINFO{You define ``#4'' with symbols family ``#1'' and color
    ``#5''}%
```

First we treat the default, and commonest (?), case for 1st argument

```
  \ifthenelse{\equal{#1}{*}}
```

in which case we do the same for the 2nd argument

```
  {\ifthenelse{\equal{#2}{local}}
```

and then we look at the last, about color.

```
    {\ifthenelse{\equal{#5}{*}}
      {\expandafter\edef\csname Y@G@@ #4\endcsname
        {\yagding[\Y@G@SymFamChoice]{#3}[\noexpand\Y@GSymColor]}}
      {\expandafter\edef\csname Y@G@@ #4\endcsname
        {\yagding[\Y@G@SymFamChoice]{#3}[#5]}}}%
```

Here the ding name will be globally defined.

```
    {\ifthenelse{\equal{#2}{global}}
      {\ifthenelse{\equal{#5}{*}}
        {\expandafter\xdef\csname Y@G@@ #4\endcsname
          {\yagding[\Y@G@SymFamChoice]{#3}[\noexpand\Y@GSymColor]}}
        {\expandafter\xdef\csname Y@G@@ #4\endcsname
          {\yagding[\Y@G@SymFamChoice]{#3}[#5]}}}%
```

And we come here in case the 2nd argument is neither `local` nor `global`.

```
      {\Y@GWARNING{Value #2 not recognised} {.\MessageBreak The possible
          values are ``local'' (default) and ``global''.\MessageBreak I
          will assume you wanted ``local''} {The possible values are
          ``local'' (default) and ``global''}}}}%
```

We use the same pattern of nested `\ifthenelse` when the 1st argument is explicitly provided.

```
  {\ifthenelse{\equal{#2}{local}}
```

```
    {\ifthenelse{\equal{#5}{*}}
      {\expandafter\edef\csname Y@G@@ #4\endcsname
        {\yagding[#1]{#3}[\noexpand\Y@GSymColor]}}
      {\expandafter\edef\csname Y@G@@ #4\endcsname
        {\yagding[#1]{#3}[#5]}}}%
    {\ifthenelse{\equal{#2}{global}}
      {\ifthenelse{\equal{#5}{*}}
        {\expandafter\xdef\csname Y@G@@ #4\endcsname
          {\yagding[#1]{#3}[\noexpand\Y@GSymColor]}}
        {\expandafter\xdef\csname Y@G@@ #4\endcsname
          {\yagding[#1]{#3}[#5]}}}%
      {\Y@GWARNING{Value #2 not recognised} {.\MessageBreak The possible
          values are ``local'' (default) and ``global''.\MessageBreak I
          will assume you wanted ``local''} {The possible values are
          ``local'' (default) and ``global''}}}}}
```

`\yagding+` First appearance of `\WithSuffix` from suffix. The default font encoding is still `U` but it can be given explicitly, enabling the user to pick something in a font with different encoding. Here once again, `*` has a special meaning for 3rd and 4th arguments.

```
\WithSuffix\newcommandx\yagding+[6][1=U,6=\Y@GSymColor]{{%
    \fontencoding{#1}\fontfamily{#2}%
    \ifthenelse{\equal{#3}{*}}{}{\fontseries{#3}}%
    \ifthenelse{\equal{#4}{*}}{}{\fontshape{#4}}%
    \selectfont\Y@G@Couleur{#6}\char#5}}

\WithSuffix\newcommand\yagding*[1]{%
  \ifthenelse{\isnamedefined{Y@G@@ #1}}
  {\csname Y@G@@ #1\endcsname}
  {[\string? #1 \string?]%
    \Y@GWARNING{The name #1 is not defined}
    {.\MessageBreak
      Or you forgot to define it\MessageBreak
      or you defined it inside an alien group.\MessageBreak
      In any case you used it}{}}}
```

`\defdingname+` And now the "grown-up" version of `\defdingname` which mixes the syntax of both `\defdingname` and `\yagding+`.

```
\WithSuffix\newcommandx\defdingname+[8] [1=U,2=local,8=\Y@GSymColor]{%
  \ifthenelse{\equal{#2}{local}}%
  {\ifthenelse{\equal{#8}{*}}%
    {\expandafter\edef\csname Y@G@@ #7\endcsname
      {\yagding+[#1]{#3}{#4}{#5}{#6}[\noexpand\Y@GSymColor]}}%
    {\expandafter\edef\csname Y@G@@ #7\endcsname
      {\yagding+[#1]{#3}{#4}{#5}{#6}[#8]}}}%
  {\ifthenelse{\equal{#2}{global}}%
    {\ifthenelse{\equal{#8}{*}}%
      {\expandafter\xdef\csname Y@G@@ #7\endcsname
        {\yagding+[#1]{#3}{#4}{#5}{#6}[\noexpand\Y@GSymColor]}}%
      {\expandafter\xdef\csname Y@G@@ #7\endcsname
        {\yagding+[#1]{#3}{#4}{#5}{#6}[#8]}}}%
    {\Y@GWARNING{Value #2 not recognised} {.\MessageBreak The possible
        values are ``local'' (default) and ``global''.\MessageBreak I
        will assume you wanted ``local''} {The possible values are
```

```
        ‘‘local’’ (default) and ‘‘global’’}}}}
```

The `\Y@Gfill` macro does the internal work when one of the `\yagfill(*/+)` is called. There is generally no error control for the arguments are passed by keys.

It is heavily indebted to the code of the `\Pifill` of pifont.

```
\newcommand{\Y@Gfill}[7]%
{{\leavevmode
  \dimendef\Y@Glongi=255\relax
  \dimendef\Y@GBoxActualWidth=254\relax
  \ifthenelse{\equal{#2}{l}}{\let\Leaders=\leaders}{}%
  \ifthenelse{\equal{#2}{x}}{\let\Leaders=\xleaders}{}%
  \ifthenelse{\equal{#2}{c}}{\let\Leaders=\cleaders}{}%
  \ifthenelse{\equal{#3}{a}}
  {\def\LaBoite{\hbox{\makebox{\hspace{#6}#1\hspace{#7}}}}}
  {%
```

Now we can check the required length against the actual length of the material to be typeset.

```
    \settowidth{\Y@Glongi}{#1}%
    \ifdim#5>\Y@Glongi%
    \setlength{\Y@GBoxActualWidth}{#5}%
    \else
    \setlength{\Y@GBoxActualWidth}{\Y@Glongi}%
    \Y@GINFO{The choosen value for ‘‘boxwidth’’ is too short,\MessageBreak
      I will use the natural width of the symbol: \the\Y@GBoxActualWidth}\fi
    \ifthenelse{\equal{#3}{n}}
    {\ifthenelse{\(#4<0\)\or\(#4>1000\)}
      {\Y@GWARNING{I don’t understand what you want}
        {.\MessageBreak I assume the default}
        {You should read the documentation [[SECyagfillboxwidth]].}%
        \def\LaBoite{\hbox{\makebox[\Y@GBoxActualWidth][c]{#1}}}}%
      {\setlength{\Y@Glongi}{\Y@GBoxActualWidth-\Y@Glongi}%
        \setlength{\Y@Glongi}{\Y@Glongi/1000*#4}%
        \def\LaBoite{\hbox{\makebox%
            [\Y@GBoxActualWidth][l]{\hspace*{\Y@Glongi}#1\hfill}}}}}%
    {\def\LaBoite{\hbox{\makebox[\Y@GBoxActualWidth][#3]{#1}}}}%
  \Leaders\LaBoite\hfill\kern\z@}}
```

We then define the keys governing the behaviour of `\yagfill(*/+)`.

```
\define@choicekey*+[Y@G]{yagusylo.sty}{leadtype}[\Y@Gyagfillleadtype\nr]%
{l,c,x}[l]%
{\Y@GINFO{Key ‘‘leadtype’’ is ‘‘\Y@Gyagfillleadtype’’}}%
{\PackageError{yagusylo}%
  {Possible choices for key ‘‘leadtype’’: l (default), c or x}
  {I don’t know the type you required.\MessageBreak
   Read the doc, please}}
\define@choicekey*+[Y@G]{yagusylo.sty}{symplace}[\Y@Gyagfillsymplace\nr]%
{c,r,l,a,n}[c]%
{\Y@GINFO{Key ‘‘symplace’’ is ‘‘\Y@Gyagfillsymplace’’}}%
{\PackageError{yagusylo}%
  {Possible choices for key ‘‘symplace’’: c (default), l, r, a or n}
  {I don’t know the place you required.\MessageBreak
   Read the doc, please}}
```

```
\define@key[Y@G]{yagusylo.sty}{sympos}[0]{%
  \ifthenelse{\(#1<0\)\or\(#1>1000\)}
  {\PackageError{yagusylo}
    {Your choice ``#1'' for key ``sympos'' is out of bound}
    {You should chose a interger between 0 and 1000.\MessageBreak
      You should read the documentation}}
  {\Y@GINFO{Keys ``sympos'' is ``#1''}%
    \def\Y@Gyagfillsympos{#1}}}
\define@key[Y@G]{yagusylo.sty}{boxwidth}[0.2in]{%
  \ifthenelse{\dimtest{#1}<{0pt} \or \dimtest{#1}={0pt}}
  {\PackageError{yagusylo}
    {Your choice ``#1'' for key ``boxwidth'' is out of bound}
    {You should chose a positive length.\MessageBreak
      You should read the documentation}}
  {\Y@GINFO
    {You've asked for ``#1'' as box width.\MessageBreak
      In any case the box will be at least\MessageBreak
      as large as the symbol itself}}%
  \def\Y@Gyagfillboxwidth{#1}}
\define@key[Y@G]{yagusylo.sty}{before}[0pt]{%
  \ifthenelse{\dimtest{#1}<{0pt}}
  {\PackageError{yagusylo}
    {Your choice ``#1'' for key ``before'' is out of bound}
    {You should chose a non-negative length.\MessageBreak
      You should read the documentation}}
  {\Y@GINFO{Key ``before'' is ``#1''}}%
  \def\Y@Gyagfillbefore{#1}}
```

The key `after` has a special behaviour. If it has the default value, the length `\Y@Gyagfillafter` is made equal to the length `\Y@Gyagfillbefore`, else it is checked and an error is issued if it is negative.

```
\define@key[Y@G]{yagusylo.sty}{after}[!Y@G!]{%
  \ifthenelse{\equal{#1}{!Y@G!}}%
  {\def\Y@Gyagfillafter{\Y@Gyagfillbefore}}%
  {\ifthenelse{\dimtest{#1}<{0pt}}
      {\PackageError{yagusylo}
        {Your choice ``#1'' for key ``after'' is out of bound}
        {You should chose a non-negative length.\MessageBreak
          You should read the documentation}}
      {\Y@GINFO{Key ``after'' is ``#1''}}%
      \def\Y@Gyagfillafter{#1}}}
```

And now we set the keys just defined.

```
\setkeys[Y@G]{yagusylo.sty}{leadtype,symplace,sympos,boxwidth,before,after}
```

`\yagfill` Here comes the first user macro to fill with dings. If the optional argument is given it must be a list of key-value pairs.

```
\newcommand\yagfill[2][!Y@G!]{%
\begingroup
\ifthenelse{\equal{#1}{!Y@G!}}{}%
  {\setkeys[Y@G]{yagusylo.sty}{#1}}%
  \Y@Gfill{\yagding{#2}}%
  {\Y@Gyagfillleadtype}{\Y@Gyagfillsymplace}%
  {\Y@Gyagfillsympos}{\Y@Gyagfillboxwidth}%
```

```
  {\Y@Gyagfillbefore}{\Y@Gyagfillafter}%
\endgroup}
```

And then its friends,

\yagfill* first the starred version,

```
\WithSuffix\newcommand\yagfill*[2][!Y@G!]{%
\begingroup
\ifthenelse{\equal{#1}{!Y@G!}}%
  {}%
  {\setkeys[Y@G]{yagusylo.sty}{#1}}%
\Y@Gfill{\yagding*{#2}}%
        {\Y@Gyagfillleadtype}{\Y@Gyagfillsymplace}%
        {\Y@Gyagfillsympos}{\Y@Gyagfillboxwidth}%
        {\Y@Gyagfillbefore}{\Y@Gyagfillafter}%
\endgroup}
```

\yagfill+ then the plussed version.

```
\WithSuffix\newcommand\yagfill+[2][!Y@G!]{%
\begingroup
\ifthenelse{\equal{#1}{!Y@G!}}{}%
  {\setkeys[Y@G]{yagusylo.sty}{#1}}%
\Y@Gfill{#2}%
        {\Y@Gyagfillleadtype}{\Y@Gyagfillsymplace}%
        {\Y@Gyagfillsympos}{\Y@Gyagfillboxwidth}%
        {\Y@Gyagfillbefore}{\Y@Gyagfillafter}%
\endgroup}
```

Some lengths and some keys

```
\newlength{\Y@Glinehead}
\newlength{\Y@Glinetail}
\define@key[Y@G]{yagusylo.sty}{head}[0.5in]{%
  \setlength{\Y@Glinehead}{#1}}
\define@key[Y@G]{yagusylo.sty}{tail}[0.5in]{%
  \setlength{\Y@Glinetail}{#1}}
\setkeys[Y@G]{yagusylo.sty}{head,tail}
```

which can be set up with the next macro

```
\newcommandx\setyagline[2][2=!Y@G!]{%
  \ifthenelse{\equal{#2}{!Y@G!}}%
  {\setkeys[Y@G]{yagusylo.sty}{head=#1,tail=#1}}%
  {\setkeys[Y@G]{yagusylo.sty}{head=#1,tail=#2}}}
```

\yagline and are used by

```
\newcommand{\yagline}[2][!Y@G!]%
{\begingroup
  \ifthenelse{\equal{#1}{!Y@G!}}{}%
  {\setkeys[Y@G]{yagusylo.sty}{#1}}%
  \par\noindent\hspace{\Y@Glinehead}%
  \Y@Gfill{\yagding[\Y@G@SymFamChoice]{#2}}%
     {\Y@Gyagfillleadtype}{\Y@Gyagfillsymplace}%
     {\Y@Gyagfillsympos}{\Y@Gyagfillboxwidth}%
     {\Y@Gyagfillbefore}{\Y@Gyagfillafter}%
  \hspace{\Y@Glinetail}\kern\z@\par
\endgroup}
```

and his friends

```
\WithSuffix\newcommand\yagline*[2][!Y@G!]%
{\begingroup
  \ifthenelse{\equal{#1}{!Y@G!}}{}%
  {\setkeys[Y@G]{yagusylo.sty}{#1}}%
  \par\noindent\hspace{\Y@Glinehead}%
  \Y@Gfill{\yagding*{#2}}%
     {\Y@Gyagfillleadtype}{\Y@Gyagfillsymplace}%
     {\Y@Gyagfillsympos}{\Y@Gyagfillboxwidth}%
     {\Y@Gyagfillbefore}{\Y@Gyagfillafter}%
  \hspace{\Y@Glinetail}\kern\z@\par
\endgroup}
\WithSuffix\newcommand\yagline+[2][!Y@G!]%
{\begingroup
  \ifthenelse{\equal{#1}{!Y@G!}}{}%
  {\setkeys[Y@G]{yagusylo.sty}{#1}}%
  \par\noindent\hspace{\Y@Glinehead}%
  \Y@Gfill{#2}%
     {\Y@Gyagfillleadtype}{\Y@Gyagfillsymplace}%
     {\Y@Gyagfillsympos}{\Y@Gyagfillboxwidth}%
     {\Y@Gyagfillbefore}{\Y@Gyagfillafter}%
  \hspace{\Y@Glinetail}\kern\z@\par
\endgroup}
```

We need a macro to check the number of nested levels we are at in the list-type environments. The six arguments are: ItemLevel, YagitemizeMaxDepth, ItemMark, setyagitemize, yagitemize, the number of the section in the documentation of this package.

The warning or error — depending of the key `onerror` — is issued once when the level reached has number YagitemizeMaxDepth.

```
\newcommand{\Y@GLevelTest}[6]{%
  \ifthenelse%
  {\cnttest{\value{Y@G#1}}<{\value{Y@G#2}}}%
  {\setcounter{Y@G#3}{\value{Y@G#1}}}%
  {\setcounter{Y@G#3}{\value{Y@G#2}}%
    \ifthenelse{\cnttest{\value{Y@G#1}}={\value{Y@G#2}}}
    {\Y@GWARNING{Too deeply nested for your setup}
      {.\MessageBreak
        I keep on using the last symbol.\MessageBreak
        You could have a look at your last\MessageBreak
        ``#4''\MessageBreak First ``#5'' too many}%
      {You could have a look at your last\MessageBreak
        ``#4''\MessageBreak First ``#5'' too many.\MessageBreak
        You should read the documentation [[#6]].}}{}}}
```

We need some global counters.

```
\newcounter{Y@GItemLevel}
\newcounter{Y@GItemMark}
\newcounter{Y@GYagitemizeMaxDepth}
```

`yagitemize` The first list-like environment is a yagusylo version of itemize, hence its name.

```
\newenvironmentx{yagitemize}[3][1=\Y@G@SymFamChoice,3=\Y@GSymColor]
{\stepcounter{Y@GItemLevel}%
  \ifthenelse{\equal{#2}{*}}
```

The second argument is *.

```
  {\Y@GLevelTest{ItemLevel}{YagitemizeMaxDepth}{ItemMark}
    {setyagitemize}{yagitemize}{5.1.2}%
    \edef\numero{\roman{Y@GItemMark}}%
    \begin{list}{%
        \yagding%
        [\csname Y@G@symfam@niveau\numero\endcsname]%
        {\csname Y@G@symfam@numero\numero\endcsname}%
        [\csname Y@G@symfam@couleur\numero\endcsname]}{}}
```

The second argument is explicitly given.

```
    {\begin{list}{\yagding[#1]{#2}[#3]}{}}%
    {\end{list}\addtocounter{Y@GItemLevel}{-1}}
```

`\setyagitemize` A macro to set up the `yagitemize` environment. The argument of `\setyagitemize` must be a list of triples separated by periods. Each triple defines a pattern with values separated by comas: firstly the symfam, secondly the number and thirdly the color.It uses `\Y@Gsetyagitemize`

```
\newcommand{\setyagitemize}[1]{%
  \setcounter{Y@GYagitemizeMaxDepth}{0}%
  \Y@Gsetyagitemize #1.Y@G@.\@nil}
```

which is defined here with the help of `\Y@Gsetyagitemizeaux`.

```
\def\Y@Gsetyagitemizeaux #1\fi{%
  \fi \Y@Gsetyagitemize #1}%
\def\Y@Gsetyagitemize #1.#2{%
  \def\Y@Gfairemotif ##1,##2,##3\@Y@Gnil{%
    \edef\Y@Gpremier{##1}\edef\Y@Gdeuxieme{##2}\edef\Y@Gtroisieme{##3}%
  }
  \ifx#2\@nil\relax
  \else
  \Y@Gfairemotif#1\@Y@Gnil
  \stepcounter{Y@GYagitemizeMaxDepth}%
  \edef\numero{\roman{Y@GYagitemizeMaxDepth}}%
  \expandafter\edef\csname Y@G@symfam@niveau\numero\endcsname{\Y@Gpremier}
  \expandafter\edef\csname Y@G@symfam@numero\numero\endcsname{\Y@Gdeuxieme}
  \expandafter\edef\csname Y@G@symfam@couleur\numero\endcsname{\Y@Gtroisieme}
  \Y@Gsetyagitemizeaux #2\fi}
```

At the end, the counter `Y@GYagitemizeMaxDepth` has value the number of levels defined which is also the deepest level of nesting available before a warning — if `onerror` is `nice` — or an error.

```
\newcounter{Y@GYagitemizeStarMaxDepth}
\WithSuffix\newcommand\setyagitemize*[1]{%
  \setcounter{Y@GYagitemizeStarMaxDepth}{0}%
  \Y@Gsetyagitemizestar #1.Y@G@.\@nil}
\def\Y@Gsetyagitemizestaraux #1\fi{%
  \fi \Y@Gsetyagitemizestar #1}%
\def\Y@Gsetyagitemizestar #1.#2{%
  \ifx#2\@nil%
    \relax
  \else
    \def\motif{#1}%
    \stepcounter{Y@GYagitemizeStarMaxDepth}%
```

```
    \edef\numero{\roman{Y@GYagitemizeStarMaxDepth}}%
    \expandafter\edef%
       \csname Y@Gsymbol@listdepth\numero\endcsname{%
         \yagding*{\motif}}%
       \Y@Gsetyagitemizestaraux #2\fi}
\newcounter{Y@GItemStarLevel}
\newenvironment{yagitemize*}[1][!Y@G!]%
{\stepcounter{Y@GItemStarLevel}
  \ifthenelse{\equal{#1}{!Y@G!}}
  {\Y@GLevelTest{ItemStarLevel}{YagitemizeStarMaxDepth}{ItemMark}
    {setyagitemize*}{yagitemize*}{5.2.2}%
    \edef\Y@GActualSymbol{\csname
      Y@Gsymbol@listdepth\roman{Y@GItemMark}\endcsname}}
  {\edef\Y@GActualSymbol{\yagding*{#1}}}%
  \begin{list}{\Y@GActualSymbol}{}}
{\end{list}\addtocounter{Y@GItemStarLevel}{-1}}
\newcommand\yagnumber[3]{\protect\Y@Gyagding{#1}{\arabic{#2}}{#3}}
```

The boolean is used to control the redefinition of `\item`.

```
\newboolean{Y@GitemRedefined}
```

Here are the keys of the enum bunch.

```
\define@choicekey*+[Y@G]{y@genum}{symfam}[\Y@G@EnumSymFamChoice\nr]%
 {marvosym,fourier,wasysym,bbding,dingbat,ark,ifsym,ifsymgeo,%
  ifsymgeonarrow,ifsymgeowide,ifsymclock,ifsymweather,pifont}%
 [pifont]%
 {\Y@GINFO{You choose \Y@G@EnumSymFamChoice}}%
 {\PackageError{yagusylo}%
   {The symbols family ``\Y@G@EnumSymFamChoice'' is not yet known.}
   {I don't know the family you required.\MessageBreak
    If it's not a typo you may consider to contact me
    (TdS)\MessageBreak to obtain support for it.}}
\define@key[Y@G]{y@genum}{symcolor}[blue]{\def\Y@GEnumSymColor{#1}}
\define@key[Y@G]{y@genum}{firstitemnum}['254]{%
  \ifthenelse{\(#1<0\)\or \(#1>255\)}
  {\PackageError{yagusylo}
    {``#1'' is out of bound or not a number}
    {``firstitemnum'' requires a natural number\MessageBreak
    between 0 and 255}}%
  {\def\Y@GEnumFirstItemNum{#1}%
    \Y@GINFO{Key ``firstitemnum'' is
      ``\number\Y@GEnumFirstItemNum''}}}
\define@key[Y@G]{y@genum}{enumlength}[10]{%
  \ifthenelse{\(#1<1\)\or \(#1>255\)}
  {\PackageError{yagusylo}
    {``#1'' is out of bound or not a number}
    {``enumlength'' requires a natural number\MessageBreak
      between 1 and 255}}%
  {\def\Y@GEnumLength{#1}%
    \Y@GINFO{Key ``enumlength'' is ``\number\Y@GEnumLength''}}}
```

`\newenumpattern` The macro to create patterns for the `yagenumerate` environment. The macro where the pattern `foo` is kept is `\Y@GEnumPattern@foo`.

```
\newcommand*\newenumpattern[2]{%
  \ifthenelse{\isnamedefined{Y@GEnumPattern@#1}}
  {\PackageError{yagusylo}{Pattern ``#1'' already defined}{%
      You should choose another name\MessageBreak
      this version does NOT provide a ``renewenumpattern'' macro}}
  {{\renewcommand*{\Y@GINFO}[1]{}
      \expandafter\gdef\csname Y@GEnumPattern@#1\endcsname{%
        \setkeys[Y@G]{y@genum}{#2}}}
    \Y@GINFO[1]{Pattern ``#1'' defined with\MessageBreak
      symfam=\Y@G@EnumSymFamChoice\MessageBreak
      symcolor=\Y@GEnumSymColor\MessageBreak
      firstitetemnum=\number\Y@GEnumFirstItemNum\MessageBreak
      enumlength=\Y@GEnumLength\MessageBreak}}}
```

Here we define the 4 default patterns.

```
\newenumpattern{piwcr}{symfam=pifont,firstitemnum='254,enumlength=10}
\newenumpattern{piwcs}{symfam=pifont,firstitemnum='300,enumlength=10}
\newenumpattern{pibcr}{symfam=pifont,firstitemnum='266,enumlength=10}
\newenumpattern{pibcs}{symfam=pifont,firstitemnum='312,enumlength=10}
```

```
\newcommand{\Y@GSetYagEnumerate}[1]{%
  \ifthenelse{\isnamedefined{Y@GEnumPattern@#1}}%
  {\csname Y@GEnumPattern@#1\endcsname}%
  {\Y@GWARNING{The pattern ``#1'' is not know}
    {.\MessageBreak I select ``piwcr'' instead}
    {You should read the documentation [[SECyagpattern]].}%
    \csname Y@GEnumPattern@piwcr\endcsname}}
```

```
\define@key[Y@G]{y@genum}{enumpattern}[piwcr]{%
  \edef\Y@GEnumPatternChoice{#1}%
  \Y@GSetYagEnumerate{#1}}
```

```
\setkeys[Y@G]{y@genum}{symfam,symcolor,firstitemnum,enumlength}
```

```
\newcommand\setyagenumeratekeys[1]{%
  \ifthenelse{\equal{#1}{*}}%
  {\setkeys[Y@G]{y@genum}
    {symfam,symcolor,firstitemnum,enumlength,enumpattern}}%
  {\setkeys[Y@G]{y@genum}{#1}}}
```

yagenumerate

To create the following macros, I have resorted to allready written code from the LaTeX $2_\varepsilon$ kernel and the `pifont` package. I am also indebted to Arnaud Schmittbuhl on `fr.comp.text.tex` for invaluable pieces of advice.

```
\newcounter{Y@Gmaxitem}
\newcounter{Y@Gcitem}
\newenvironment{yagenumerate}[1][!Y@G!]
{%
  \ifthenelse{\equal{#1}{!Y@G!}}
  {\setkeys[Y@G]{y@genum}{symfam,symcolor,firstitemnum,enumlength}}
  {\ifthenelse{\equal{#1}{*}}
    {\setkeys[Y@G]{y@genum}{enumpattern=\Y@GEnumPatternChoice}}
    {\setkeys[Y@G]{y@genum}{#1}}}%
  \ifnum\@enumdepth>\thr@@\@toodeep\else
  \advance\@enumdepth \@ne
  \def\STOP{\PackageError{yagusylo}
    {at least one item too many}
```

```
    {you can't use more than ``\number\Y@GEnumLength'' items}}%
  \setcounter{Y@Gmaxitem}{\Y@GEnumFirstItemNum}
  \addtocounter{Y@Gmaxitem}{\Y@GEnumLength}
  \setcounter{Y@Gcitem}{\Y@GEnumFirstItemNum}
  \ifthenelse{\boolean{Y@GitemRedefined}}{}
  {\let\Y@Golditem\item
   \def\item{\stepcounter{Y@Gcitem}
     \ifnum\theY@Gcitem>\theY@Gmaxitem\expandafter\STOP
     \else\expandafter\Y@Golditem\fi}
   \setboolean{Y@GitemRedefined}{true}}
  \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
  \expandafter\def\csname p@enum\romannumeral\the\@enumdepth\endcsname{}%
  \expandafter\def\csname labelenum\romannumeral\the\@enumdepth\endcsname{%
    \csname theenum\romannumeral\the\@enumdepth\endcsname}%
  \expandafter\def\csname theenum\romannumeral\the\@enumdepth\endcsname{%
    \yagnumber{\Y@G@EnumSymFamChoice}%
    {enum\romannumeral\the\@enumdepth}{\Y@GEnumSymColor}}%
  \list{\csname label\@enumctr\endcsname}{%
    \@nmbrlisttrue
    \def\@listctr{\@enumctr}%
    \setcounter{\@enumctr}{\Y@GEnumFirstItemNum}%
    \addtocounter{\@enumctr}{-1}%
    \def\makelabel##1{\hss\llap{##1}}}
  \fi}{\endlist}
```

The following environment is a wrapper which enables to go back to a "classical" enumerate. It redefines `\item`

```
\newenvironment{notyagenum}
  {\let\item\Y@Golditem
```

and takes care of the boolean `Y@GitemRedefined` just in case a `yagenumerate` would be nested in the `enumerate`.

```
    \setboolean{Y@GitemRedefined}{false}}{}
```

If the config-file is asked for, we check that it exists before loading it.

```
\AtBeginDocument{%
  \ifY@G@configfile
  \InputIfFileExists{yagusylo.cfg}%
  {\PackageInfo{yagusylo}{Configuration file found and read\@gobble}}%
  {\PackageErrorNoLine{yagusylo}
    {No configuration file found}
    {Or yagusylo.cfg does not exist\MessageBreak
      or it is in some place unknown of TeX.}}%
  \fi}
```

# Change History

# Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.