The wrapfig2 package

Claudio Beccari*

Version v.4.0.1 – Last revised 2022-01-03.

Contents

Abstract

This new package `wrapfig2` is a fork that upgrades Donald Arseneau's package `wrapfig` (version 2.6, dated 2003) by adding some LaTeX 3 definitions that accept a final optional star; its presence changes the meaning of the first optional argument so that it becomes a correction to the number of lines that must be indented in order to receive the wrapped object. A new environment is added to the traditional *wrapfigure* and *wraptable*, namely *wraptext*; it may be used to wrap a small framed text block on a grey background; the philosophy of this new environment is similar to that of the other two environments, but the syntax is different.

Caution This package requires a fairly recent LaTeX kernel, otherwise it won't work; any LaTeX kernel dated at least 2020 is OK.

1 Introduction

The purpose of this package is twofold. On one side it tries to modernise the original software by Donald Arseneau by upgrading it to the LaTeX 3 modern language. On the other it creates a new environment, with the same philosophy of the original Arseneau's ones, such that a document author can emphasise short blocks of text by framing them while typesetting the text on a coloured background by means of the `tcolorbox` functionalities, and wrapping this inserted text with the surrounding main text.

*E-mail: `claudio dot beccari at gmail dot com`

The original software had some idiosyncrasies; Donald Arseneau described them in the documentation of his package; I am sorry to say that such idiosyncrasies have not been reduced; but in any case in order to avoid such peculiar anomalies, it is sufficient to wrap the inserted object with a reasonable number of lines, i.e. by a reasonably long paragraph.

The above implies that no wrapped object code should be specified in the source file close the end of a paragraph; again no object code should be inserted within any list; not even close to the end or to the beginning of section. Arseneau's code is capable of specifying the wrapping number of lines such that two or more paragraphs can be indented so as to wrap a longish insertion, but it is wise to avoid such risky situations. Moreover, if the inserted figure or table has a numbered caption, the number might not result in the correct sequence of the normal corresponding floating objects.

Therefore the usefulness of the wrapping procedure depends very much from the document author ability to move around his/her code until a suitable position is found. Certainly a good place is within a longish first section paragraph at the beginning of a new chapter.

The code of this package does not do anything to correct such idiosyncrasies. They are caused by the limitations of the `\ShipOut` native TEX/LATEX 2_ε macro, and very little can be done in addition to what Arseneau already did.

Another purpose of this package is to add an option so that the ⟨*number of indented lines*⟩ argument does not mean the total number, but the correction number to add-to or subtract-from to the value computed by the default mechanism devised by Arseneau. We assume that all user first use the software to insert an object to be wrapped by the surrounding text without specifying any value with the specific optional argument; then they evaluate the result, and if the space below the wrapped object is too large, or if such space is too small they count the necessary number of lines and specify it to be processed during another document compilation. When the object to be wrapped is tall, it is very easy to miscount the necessary number of lines, while is is very easy to evaluate the necessary small correction to the default computed value.

A furher purpose of this package is to define a new environment, *wraptext*, to wrap a framed text block typeset on a grey background as if it were a figure. On `texstackexchange` a solution was suggested to a user who was asking for such an arrangement; the solution resorted to a specific use of the *wrapfigure* environment. We thought that an *ad hoc* solution would be a better one, since the parameters to be used for a figure have nothing or little to do with a text, therefore most of them would be useless with a wrapped text. Nevertheless the ⟨*location*⟩ of the wrapped text and the optional correction of the indented lines number would still be necessary. We added also the possibility of optionally specifying the measure of the wrapped text, even if this measure should not be too different from a half of the wrapping text measure.In facts, with a value too different from `0.5\linewidth` either the wrapped text has problems with inter word spaces and hyphenation because of the small measure, or, on the opposite, the indented lines of the wrapping text have similar problems.

2 Environment syntax

The new syntax for *wrapfigure* and *wraptable* is backwards compatible with the original one: just a final optional star is added to the original list of arguments. As the list of arguments shows, the *wraptext* environment has similar features, but all its arguments are optional, so that they may be specified independently from one another, provided that, when they are more than one, they are specified in the order shown in the syntax table.

The optional star is available only for the standard *wrapfigure* and *wraptable* environments because the backwards compatibility requires the first four optional and mandatory arguments. When the optional star is specified, the ⟨*indented lines number*⟩ is interpreted as the correction to the computed number.

```
\begin{wrapfigure}[⟨indented lines number⟩]{⟨location⟩}[⟨overhang⟩][⟨width⟩]⟨⋆⟩
    ⟨figure⟩
\end{wrapfigure}

\begin{wraptable}[⟨indented lines number⟩]{⟨location⟩}[⟨overhang⟩][⟨width⟩]⟨⋆⟩
    ⟨table⟩
\end{wraptable}

\begin{wraptext}[⟨location⟩]|⟨width⟩|<⟨indented line number correction⟩>(⟨caption label⟩)
    ⟨wrapped framed text⟩
\end{wraptext}
```

3 Examples

We display some examples by using fake objects such that suitably long paragraphs are available; some fake-language long-paragraphs are obtained by means of the `kantlipsum` package functionalities. They contain paragraphs that look as Kant's sentences, but we doubt that Immanuel Kant ever wrote such texts...

3.1 A wrapped figure

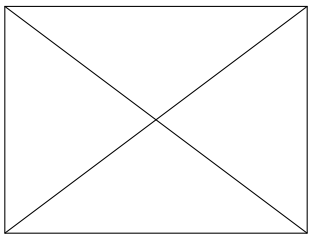

Figure 1: A rectangle

As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves; as I have shown elsewhere, the phenomena should only be used as a canon for our understanding. The paralogisms of practical reason are what first give rise to the architectonic of practical reason. As will easily be shown in the next section, reason would thereby be made to contradict, in view of these considerations, the Ideal of practical reason, yet the manifold depends on the phenomena. Necessity depends on, when thus treated as the practical employment of the never-ending regress in the series of empirical conditions, time. Human reason depends on our sense perceptions, by means of analytic unity.

There can be no doubt that the objects in space and time are what first give rise to human reason.

The code used to type the above figure is the following:

```
\begin{wrapfigure}{r}{50mm}
\centering\unitlength=1mm
\begin{picture}(40,30)
\polygon(0,0)(40,0)(40,30)(0,30)
\Line(0,0)(40,30)\Line(0,30)(40,0)
\end{picture}
\caption{A rectangle with its diagonals}
\end{wrapfigure}

\kant[1]
```

No asterisk was used because the package succeeded to correctly compute the necessary number of indented lines.

3.2 A wrapped table

Wrapping a small table is a little more difficult than wrapping a figure, because the width of the inserted object is not known exactly in advance and is difficult to estimate; therefore it is possible that several trial compilations are necessary. In any case a `\centering` command might help to center the table within the indention of the wrapping text. Nevertheless the software can compute the object width if a zero value is specified or if the ⟨*width*⟩ parameter is completely omitted together with its braces; this second possibility is a feature of this package, that uses a LaTeX 3 property by which even a braced argument can be treated as an optional argument with a predefined default value; see below more details about this feature.

First	Second
Third	Fourth

Table 1: A small table

Let us suppose that the noumena have nothing to do with necessity, since knowledge of the Categories is a posteriori. Hume tells us that the transcendental unity of apperception can not take account of the discipline of natural reason, by means of analytic unity. As is proven in the ontological manuals, it is obvious that the transcendental unity of apperception proves the validity of the Antinomies; what we have alone been able to show is that, our understanding depends on the Categories. It remains a mystery why the Ideal stands in need of reason. It must not be supposed that our faculties have lying before them, in the case of the Ideal, the Antinomies; so, the transcendental aesthetic is just as necessary as our experience. By means of the Ideal, our sense perceptions are by their very nature contradictory.

The above wrapped small table has been typeset by means of the following code.

```
\begin{wraptable}{l}
  \centering
```

```
  \begin{tabular}{cc}
    \hline
    First & Second\\
    Third & Fourth\\
    \hline
  \end{tabular}
  \caption{A small table}
\end{wraptable}
\kant[2]
```

You notice the absence of the braced width value; as said above, this braced value is optional, and the software autonomously computes the width of the wrapped object. This feature may be useful in many instances, although we think that a smart use of this width parameter might yield better looking results.

On the opposite if the user estimates that the table with its caption might use 5 lines, and specified such a value as the first (optional) argument to the environment, the result would be the following poor one, with the last caption line overlapping the wrapping text.

First	Second
Third	Fourth

Table 2: A small table

Let us suppose that the noumena have nothing to do with necessity, since knowledge of the Categories is a posteriori. Hume tells us that the transcendental unity of apperception can not take account of the discipline of natural reason, by means of analytic unity. As is proven in the ontological manuall it is obvious that the transcendental unity of apperception proves the validity of the Antinomies; what we have alone been able to show is that, our understanding depends on the Categories. It remains a mystery why the Ideal stands in need of reason. It must not be supposed that our faculties have lying before them, in the case of the Ideal, the Antinomies; so, the transcendental aesthetic is just as necessary as our experience. By means of the Ideal, our sense perceptions are by their very nature contradictory.

3.3 A wrapped text

Text, text, text, text, text, text, text, text, text, text, text.

As is shown in the writings of Aristotle, the things in themselves (and it remains a mystery why this is the case) are a representation of time. Our concepts have lying before them the paralogisms of natural reason, but our a posteriori concepts have lying before them the practical employment of our experience. Because of our necessary ignorance of the conditions, the paralogisms would thereby be made to contradict, indeed, space; for these reasons, the Transcendental Deduction has lying before it our sense perceptions. (Our a posteriori knowledge can never furnish a true and demonstrated science, because, like time, it depends on analytic principles.) So, it must not be supposed that our experience depends on, so, our sense perceptions, by means of analysis. Space constitutes the

whole content for our sense perceptions, and time occupies part of the sphere of the Ideal concerning the existence of the objects in space and time in general.

The above example was typeset with this simple code:

```
\begin{wraptext}
Text, text, text, text, text, text, text, text, text, text, text.
\end{wraptext}
\kant[3]
```

As it was previously shown, the syntax for the *wraptext* environment is a little different from that of the other two environments. The logic behind it is substantially the same; the ⟨*width*⟩ parameter is optional; its preset value is half the column width, that in one column typesetting mode coincides with the text width. The wrapped text is typeset in justified mode within a *minipage* environment; the measure of this mini page should not be too small (unless the text is less than one line long) otherwise the inter word spacing might be too large; at the same time the measure of the mini page cannot be too large, otherwise the indented wrapping lines, generally justified, might get a bad word spacing. Therefore it is suggested to avoid specifying the optional ⟨*widh*⟩ outside the range of 40% to 60% the column width. Actually specifying `0.2\textwidth` or `0.4\columnwidth` when typesetting in two column mode produces approximately the same result, because `\columnwidth` is a little less than half the `\textwidth`.

A warning is necessary: if a caption is entered within the environment, such caption gets typeset within the background coloured frame. We think that a framed text does not require any caption; if such caption were necessary, then the user should resort to other means, for example, to a standalone small PDF file containing the framed shaded text to be imported as a figure with the ⟨*wrapfigure*⟩ environment.

4 Remarks

The syntax of the original environments ⟨*wrapfigure*⟩ and ⟨*wraptable*⟩ has not been changed, except for a last optional star. The fact that the last braced argument is optional does not change the backward compatibility with the original environments.

Therefore the optional ⟨*line number*⟩ argument maintains its meaning, unless the optional star is specified; in such a case that number assumes the meaning of a correction to the computed number of the indented lines.

The mandatory ⟨*placement*⟩ maintains its meaning and the legal values are `l` (left), `r` (right), `L` (floating left), `R` (floating right), `i` (inner margin), `o` (outer margin), `I` (floating inner margin), `O` (floating outer margin).

We tested all of them, but as a personal choice we prefer to place the wrapped object at the left of the text, without floating it and irrespective of the page number parity.

As in the previous examples, we prefer to specify the wrapping environment before a sufficiently long paragraph. Should the paragraph be too short to completely wrap the object, all the environments are capable of counting the number of indented lines and to apply the `\overhang` command with the remaining line number to the following paragraph(s); in these circumstances it might be necessary to recourse to the optional star in order to correct the indention, since the mechanism does not consider the inter paragraph spacing that TEX introduces only at ship out time.

We avoid also to enter the wrapping environment before paragraphs that are close to a page break; this action would tickle the idiosyncrasies of the software, and requires moving the wrapping environment some paragraphs before or after the preferred one; but this can be done only while reviewing the document, because any change in the previous source text might change the situation if this adjustment is done while still editing the document.

With the standard environments the optional parameter ⟨*overhang*⟩ does exactly what its name implies: the wrapped object protrudes into the adjacent margin exactly by the specified amount. This parameter is not available for the *wraptext* environment; we believe that a wrapped text logically pairs the wrapping text; of course this opinion might be wrong and in future upgrades we might add this functionality.

The ⟨*width*⟩ parameter has been already sufficiently described; we just remember that for *wraptext* this parameter is optional and its default value amounts to half the current measure; it can be specified but it should not be too different from 50% the current measure. For the standard environments this parameter value is mandatory, but, we recall, only for the redefined environments ⟨*wrapfigure*⟩ and ⟨*wraptable*⟩, this braced argument is optional.

5 Other floating objects

Pictures and textual arrays may be floated by means of the standard ⟨*figure*⟩ and ⟨*table*⟩ environments. But other floating objects may be defined by means of other packages, for example the `float`, or classes, such as `memoir`. Besides floating, the main difference is the name of the caption "label": Figure, Table, Algorithm, Example, and so on.

If floating is not necessary, this package (as well as the original one) allows to use the underlying environment *wrapfloat* that uses the same syntax as *wrapfigure* plus the mandatory name of the new object: even a figure might be introduced without using ⟨*wrapfigure*⟩, by using instead:

`\begin{wrapfloat}{figure}[`⟨*line number*⟩`]{`⟨*placement*⟩`}[`⟨*overhang*⟩`]{`⟨*width*⟩`}`⟨⋆⟩
⟨*image*⟩
`\end{wrapfloat}`

Another ⟨*object*⟩ may be wrapped by using:

`\begin{wrapfloat}{`⟨*object name*⟩`}[`⟨*line number*⟩`]{`⟨*placement*⟩`}%`

```
  [⟨overhang⟩]{⟨width⟩}⟨⋆⟩
        ⟨object⟩
\end{wrapfloat}
```

If the floating ⟨*placement*⟩ codes have to be used, another floating object with the desired ⟨*object name*⟩ has to be previously defined by means of the functionalities of other packages or classes.

6 The code

Here we describe and comment the code of this package; essentially only the initial parts need some comments; because the final ones are almost identical to the original Arseneau's code.

We start with the usual specification of the format name and date, and the identification of this specific package. We possibly load the `etoolbox` package, because we immediately need a test on the existence of an @-protected macro; should this macro already been defined by the class or other packages (including a previous loading of this package or the previous Arseneau's one) the loading is immediately aborted. If is was not previously loaded, we load the `xfp` package, that allows us to perform precise calculations. Loading the `xparse` package is necessary in order to use one of its rare features that did not migrate to the LaTeX kernel. From the LaTeX News Letter dated October 2020:

> Most, but not all, of the argument types defined by `xparse` are now supported at the kernel level. In particular, the types `g/G`, `l` and `u`, are not provided by the kernel code; these are *deprecated* but still available by explicitly loading `xparse`. All other argument types are now available directly within the LaTeX 2_ε kernel.

For now their availability eases the treatment of the backwards compatibility of this software with the original `wrapfig` functionality. It deals with the mandatory ⟨*width*⟩ argument of the `wrapfigure`, `wraptable`, and `wrapfloat` environments, where it was possible to specify a zero value. Now it is possible to omit it completely because it is a *braced optional argument*.

```
\NeedsTeXFormat{LaTeX2e}[2020-01-01]
\ProvidesPackage{wrapfig2}%
 [2021-11-30 v.4.0 Wrap text around figures, tables, framed text blocks]
\@ifpackageloaded{etoolbox}{}{\RequirePackage{etoolbox}}
\ifcsdef{c@WF@wrappedlines}{\endinput}{}
\ifcsdef{fpeval}{}{\RequirePackage{xfp}}
\RequirePackage{xparse}

```

Next we define some dimensions, boxes, token registers, TeX counters, and alias names. The `\WF@correctlines@switch` TeX numeric register (not a LaTeX counter) is going to be used a boolean switch; if its value is zero, il means "false",

otherwise is “true”; in the other definitions below, it will be set only to 0 or 1, depending on the presence of the optional star.

```
\newdimen\wrapoverhang \wrapoverhang\z@
\newdimen\WF@size
\newcounter{WF@wrappedlines}
\newbox\WF@box
\newbox\NWF@box
\newtoks\WF@everypar
\newif\ifWF@float
\newcount\WF@correctlines@switch
\newdimen\insertwidth
\let\@@parshape\parshape
\let\WF@@everypar\everypar

```

In what follows we are going to use very often the functionalities of the `xparse` package that are mostly already included into the LaTeX kernel; but, since we use one of this functionalities that have not migrated to the LaTeX kernel, we load it, as they remain available with that package; nevertheless we specified a LaTeX format date insuring that the LaTeX 3 syntax is available.

Should the format file be an older one, a multitude of errors would be produced, and the user should take care to load the `xparse` and `xfp` packages before loading `wrapfig2`. Notice that most of the `xparse` package functionalities at the date required for the format file are already included. The `xparse` package has been available about in 2018; should the users have available a definitely older TeX system installation, they should upgrade it, or must avoid using this `wrapfig2` package and use the original one; if they need to wrap text, they should resort to some ingenious tricks to do it.

The opening command of the *wrapfloat* environment receives the mandatory and optional arguments plus the name of the particular object to be wrapped. It is used to define the prefix label of the caption number in case that the object is described with a caption. The optional star is not explicit, because it is going to be read by the `\WF@wr` macro.

The closing command of *wrapfloat* performs most of the work necessary to wrap the box that contains the object to be wrapped, but certain tasks are demanded to other service macros.

It may set the width of the box if the ⟨*width*⟩ parameter is specified; otherwise it closes the `\hbox` that was used; then it closes the main vertical box `\WF@box`. After executing `\WF@floatstyhook`, necessary when package `float.sty` has been used, it saves the ⟨*overhang*⟩ value to be used when wrapping is actually performed; then it verifies if the box height is too high to fit, or is too short; possibly re-boxes this box in the same box register with a negative initial vertical skip that raises the box contents.

The definitions of the *wrapfigure* and *wraptable* environments are very simple by means of the underlaying *wrapfloat* environments.

```
\NewDocumentEnvironment{wrapfigure}{o m o G{0pt}}%
  {\wrapfloat{figure}[#1]{#2}[#3]{#4}}%
```

```
  {\endwrapfloat}

\NewDocumentEnvironment{wraptable}{o m o G{0pt}}%
  {\wrapfloat{table}[#1]{#2}[#3]{#4}}%
  {\endwrapfloat}
```

The definition of the *wraptext* environment is more detailed, because most of the computations must be done on the actual text to be wrapped, that does not have a specific width; moreover the inserted text must not be too wide, nor too short in order to avoid problems for its justification or the justification of the wrapping lines. The framed box width is preset to 50% of the normal text measure, but it can be optionally specified to a different value (not too different from 50%), while for the other environments the wrapped material width is an *braced optional parameter*.

For what concerns the *wraptext* environment, see below, because the code is a little more complicated and requires some explanation. In facts the first statement argument description list does not contain any descriptor for an optional star. There is no need because the computation of the insertion block height is pretty precise and at most the user might desire one line more or less depending on the measure of the whole text, and that of the inserted block and/or the measure of the indented wrapped lines; sometimes it might be necessary to get rid of the space below the inserted block when it gets typeset at the bottom of a page.

It is true that some of the input parameters specified to the opening command of any environment with LaTeX 3 are available also to the closing commands; see the last paragraph of section 2 in the `xparse` documentation. But the following definition, besides using special delimiters for optional parameters, uses the separate opening and closing macros of the *wrapfloat* environment; such procedure apparently breaks this second availability of the input parameters, therefore it is necessary to save them into local macros or count registers (assignments to TeX count registers are *local*, while assignment to LaTeX named counters, through the `\setcounter` macro and its siblings, are *global*) so that we can use their values within the closing commands.

The `\NWF@box` box register has been assigned at the beginning; remember that LaTeX 3 registers of any kind are not limited in number as they were some years ago even with LaTeX 2_ε. The last opening commands are conceived to box the object to be wrapped, typeset within a *minipage* with the default or specified width; such box, and the *tcolorbox* and *minipage* environments are closed at the beginning of the closing commands, so that what is necessary in order to place the wrapped boxed text is easy to be executed.

The number of indented lines is computed by means of the `\fpeval` LaTeX 3 function; among the operands of this function there is the value `2` used to take into account the vertical space used by *tcolorbox* to separate the frame from its contents. It is possible that a value of `3` might reduce the probability of using the ⟨*line number correction*⟩. But it is not always true and we found that the chosen value is a better choice.

Eventually the opening *wrapfloat* statement is created by expanding the whole

line complete of arguments, by means of the usual trick of defining a dummy macro within a group that contains among its expansion the group closing command, so that while it is being executed, it deletes itself from memory.

```
\NewDocumentEnvironment{wraptext}%
      {O{l}    D||{0.5\columnwidth} D<>{0}   D(){figure}}%
{%                                                          Open environment
  \insertwidth=#2\relax
  \def\textplacement{#1}%
  \def\textcorrection{#3}%
  \def\WF@caption@label{#4}%
  \begin{lrbox}\NWF@box%            This box is to contain the framed text
    \minipage{\insertwidth}%
   \tcolorbox
}{%                                                        Close environment
  \endtcolorbox\endminipage\end{lrbox}%
  \edef\NWF@wli%                        \NWF@wli is a macro, not a counter
    {%
      \fpeval{%
        round((\ht\NWF@box+\dp\NWF@box)/\baselineskip,0)+2+\textcorrection
             }%
    }%
  \ifhmode\unskip\else\leavevmode\noindent\fi
  \bgroup\edef\x{\egroup\noexpand\wrapfloat{\WF@caption@label}[\NWF@wli]%
     {\textplacement}{\the\insertwidth}}\x
    \box\NWF@box %                       Output framed box containing text
  \endwrapfloat
  \ignorespaces
}

```

Now comes the definition of the fundamental environment *wrapfloat*; compared to the original Arseneau's definition it is much longer, but it contains the code that Arseneau, who wrote the code in LaTeX 2_ε language, had to split in several macros in order to handle the multitude of interspersed mandatory and optional arguments.

The main function of this environment is to handle the box that contains the figure, or the table, or the framed text, or what else, so that the inserted box is preceded and followed by the suitable vertical spaces, and it is possible to compute the number of lines to be indented; often this computed number is correct; but in certain cases, when the code is used too close or within prohibited wrapping text, such number might need to be corrected. As it can be seen the optional star is not among the argument descriptors of the opening commands; it will be the following macro `\WR@wr` responsible of taking care of the list of arguments and see if a star has been specified and not yet read by the preceding commands.

In order to handle any kind of wrapped object, this environment first argument is the ⟨*caption label*⟩. It may remain blank; but for wrapping figures or tables their respective definitions specify the name of the floating object they belong to; it is not necessary that there exists a floating environment with the same name of the

wrapped object, when its wrapping environment is not specified with a floating ⟨*placement*⟩ argument.

```
\NewDocumentEnvironment{wrapfloat}{m o m o G{\z@}}%
  {\def\@captype{#1}\WF@wr[#2]{#3}[#4]{#5}}%
  {\ifdim\hsize>\z@
    \par\hrule\@width\hsize\@height\z@ %   force width with invisible rule
  \else
    \unskip \egroup \box\z@  %                                or close hbox
  \fi
 \egroup %                       close the vtop box; its width now is known
  \WF@floatstyhook %                                    support for float.sty
  \def\width{\wd\WF@box}%
  \setlength\wrapoverhang{\WF@ovh}%
  \xdef\WF@ovh{\the\wrapoverhang}%                       save until wrapping
  \ifdim\ht\WF@box>\topskip \ht\WF@box\z@ \fi%            too high, set flag
  \ifdim\ht\WF@box<.5\p@ %                               too short, move up
    \global\setbox\WF@box\vtop{\vskip-1.4ex\unvbox\WF@box}%
  \fi
   \global\WF@size=%                   compute total box hight with \fpeval
     \fpeval{\ht\WF@box+\dp\WF@box+1.5\baselineskip+\tw@\intextsep}\p@
   \aftergroup\WF@startfloating %        use even when not really floating
  \unless\ifWF@float
    \ifhmode
      {\unskip \parfillskip\z@skip \par \vskip-\parskip}%
      \aftergroup\noindent
    \fi
  \fi
  \global\@ignoretrue
  }
```

The working macro `\WF@wr` is defined with the LaTeX 3 language; it grabs all the optional and mandatory arguments in a single step, contrary to LaTeX 2_ε that requires to split the various steps in separate macros. In the definition code we use also some commands, such as `\unless`, originally defined by the ε-TeX typesetting program extensions, that have been included in the pdfLaTeX, XeLaTeX and LuaLaTeX kernels several years ago.

Notice that the optional first (optional) argument, that represents the number of indented lines or their correction number, is saved into the macro `\WF@wli`, but if this argument is not specified, `\WF@wli` is assigned the value zero. The same happens for the ⟨*overhang*⟩ optional argument.

After these adjustments, it computes the box total height plus some fixed amounts needed mostly to set the wrapped material below the first wrapping text first line. Here is where the LaTeX 3 `\fpeval` computing function comes into play so as to assign such height to `\WF@size`. Some unusual macros are executed; they were devised by Arseneau to deal with possibly floating wrapped objects. The optional star is not accepted by this macro; if the user specified it, it is still in the

input flux; notice that the *wraptext* environment does not accept the optional star; if the user specifies it for this environment, an asterisk appears at the beginning of the wrapped text.

The braced `#4` ⟨*width*⟩ parameter (actually a ⟨*braced optional parameter*⟩) may be specified to be `0pt`; in any case `0pt` is the default parameter value; if so, the object is treated at its natural width, by boxing it into an hbox and using this box width as the working width

```
\NewDocumentCommand\WF@wr{o m o m s}{%
  \xdef\WF@wfname{wrap\@captype\space}%
  \unless\ifvoid\WF@box
    \WFclear \WF@collision
  \fi
  \xdef\WF@place{\string`\@car#2r\@nil}%
  \ifnum\lccode\WF@place=\WF@place
      \global\WF@floatfalse
  \else
      \global\WF@floattrue
  \fi
  \ifx\parshape\WF@fudgeparshape
    \unless\ifWF@float\\WF@collision\fi
  \else
    \ifx\par\@@par
      \ifnum\@@parshape>\z@\WF@conflict\fi
    \else
      \WF@conflict
    \fi
   \fi
   \IfValueTF{#1}%                 save optional line number or correction
     {\gdef\WF@wli{#1}}%
     {\gdef\WF@wli{0}}%
%
  \IfValueTF{#3}%                                    save optional overhang
      {\gdef\WF@ovh{#3}}%
      {\gdef\WF@ovh{\z@}}%
%
   \global\setbox\WF@box\vtop\bgroup \setlength\hsize{#4}%       set width
   \ifdim\hsize>\z@
     \@parboxrestore
   \else
     \setbox\z@\hbox\bgroup
     \let\wf@@caption\caption
     \let\caption\wf@caption
     \ignorespaces
   \fi
  \IfBooleanTF{#5}%    if the asterisk is present set the numerical switch
              {\global\WF@correctlines@switch=\@ne}%
              {\global\WF@correctlines@switch=\z@}%
  \global\@ignoretrue
}
```

At this point the main box `\WF@box` is opened in order to store the object to be wrapped; with this box height the software is going to compute the number of lines to be indented, unless such a number has been specified and no star was added to the input parameters.

Also the ⟨*wraptext*⟩ environment uses a box to collect the framed text; the name of this second box must be different from `\WF@box` otherwise interference of the various tasks produces unrecoverable errors. This is why at the beginning of this package we defined two different boxes: `\WF@box` and `\NWF@box`.

The trick of creating an alias for the `\caption` macro is used by Arseneau to redefine one of the two macros according to certain conditions. Here `\wf@caption` is actually redefined if the ⟨*width*⟩ parameter has been specified.

```
\def\wf@caption{\relax%        redefine \wf@caption in case \hsize is zero
  \ifdim\hsize>\z@
    \let\caption\wf@@caption
  \else
    \unskip \egroup \hsize\wd\z@ \@parboxrestore \box\z@%      empty \box0
  \fi
  \caption
}
```

One of these unusual macros was introduced by Arseneau to deal with paragraph parameters and possibly to float the object to be wrapped.

```
\def\WF@startfloating{%
 \WF@everypar\expandafter{\the\everypar}\let\everypar\WF@everypar
 \WF@@everypar{\ifvoid\WF@box\else\WF@floathand\fi \the\everypar
 \WF@wraphand
}}
```

The following macro is for floating wrapping environments.

```
\def\WF@floathand{%
  \ifx\parshape\WF@fudgeparshape
    \WF@fltmes
  \else
    \ifx\par\@@par
      \ifnum\@@parshape=\z@
        \ifdim\hangindent=\z@
          \setbox\z@\lastbox \begingroup
          \@@par \WF@@everypar{}\WF@putfigmaybe
          \endgroup %                      after this group start wrapping
          \unless\ifvoid\z@ %                          replace indentation
              \box\z@
          \fi
        \else
          \WF@fltmes
        \fi
      \else
```

```
        \WF@fltmes
      \fi
    \else
      \WF@fltmes
    \fi
  \fi}
```

On the contrary if there is enough space or if the wrapped object cannot float, it gets output here.

```
\def\WF@putfigmaybe{%
\ifinner
  \vskip-\parskip \global\WF@floatfalse
  \let\pagetotal\maxdimen %              kludge flag for "not top of page"
\else %                                                      outer page
  \@tempdima\pagedepth %                                   save page depth
  {\advance\parskip\@tempdima\vskip-\parskip}%        back up to base line
  \penalty\interlinepenalty %                       update page parameters
  \@tempdimb\pagegoal \advance\@tempdimb-\pagetotal %    room left on page
  \ifdim \@tempdimb<\z@ %                                page already full
    \global\WF@floatfalse
    \unless\ifdim-\@tempdimb>\pageshrink
      \pagebreak
    \fi
  \else
    \ifdim\WF@size>\@tempdimb%     box too high does not fit in \@tempdimb
      \ifWF@float
        \dimen@.5\baselineskip
      \else
        \dimen@ 2\baselineskip
      \fi
      \ifdim\pagestretch>\dimen@ \dimen@\pagestretch \fi
      \ifdim\pagefilstretch>\z@ \dimen@\@tempdimb \fi
      \ifdim\pagefillstretch>\z@ \dimen@\@tempdimb \fi
      \advance\dimen@.5\baselineskip
      \ifdim\dimen@>\@tempdimb %                     stretch page contents
        \global\WF@floatfalse \pagebreak
      \fi
    \else %                                          box fits in \@tempdimb
      \global\WF@floatfalse
    \fi
  \fi
  \vskip\@tempdima\relax %                       return erased page depth
\fi
\noindent
\ifWF@float
  \WF@fltmes
\else %                                            place insertion here
  \WF@info{Put \WF@wfname here:}%
  {\ifodd
```

```
    \if@twoside\c@page\else\@ne\fi %          assign l/r to i/o placement
    \lccode`i`l\lccode`o`r\else \lccode`i`r\lccode`o`l%
  \fi
  \xdef\WF@place{\the\lccode\lccode\WF@place}%
               }%                                 twice to get only l or r
 \hbox to\z@{%      llap o rlap depending on l or r; calc effective width
   \@tempdima\wd\WF@box \@tempdimb\WF@ovh
   \advance\@tempdima-\@tempdimb \advance\@tempdima\columnsep
   \@tempdimb\hsize \advance\@tempdimb-\@tempdima
   \xdef\WF@adjlw{\the\@tempdima}%
   \ifnum `l=\WF@place %                                  object on left
     \hss
     \def\@tempa{\kern\columnsep}%             take right gap into action
   \else %                                               insert on light
     \@tempdima\z@ %                                  no left indentation
     \kern\@tempdimb \kern\columnsep
     \def\@tempa{\hss}%                object overlaps space to the right
   \fi
   \ifdim\@tempdimb<\hsize
     \xdef\WF@wrapil{\the\@tempdima \the\@tempdimb}%  indent.n and length
     \xdef\WF@adjtlm{\the\@tempdima}%
   \else
     \xdef\WF@wrapil{\z@ \the\hsize}%
     \xdef\WF@adjlw{\z@}\xdef\WF@adjtlm{\z@}%
   \fi
   \ifdim\pagetotal=\z@ %              put object at top of page \thepage
     \global\advance\WF@size-\intextsep
   \else %                               put object in middle of the page
     \setbox\WF@box\hbox{\lower\intextsep\box\WF@box}%
   \fi
   \dp\WF@box\z@
   \box\WF@box
   \@tempa
 }%                                                   end \hbox to 0pt
 \aftergroup\WF@startwrapping
\fi
}
```

Here comes the very important macro that counts the wrapping indented lines, so that wrapping is correct; of course the limitations of the TEX and LATEX processing (needed to ship out a complete page) forbid to take into account the spaces inserted between paragraphs and/or those inserted between entries of various listings. The idiosyncrasies of this package arise from the fact that this macro cannot preview actions that have not yet taken place when this macro is executed.

This macro counts the lines to be indented by rounding the division of the box height by the current base line skip. Notice that `WF@wrappedlines` is the name of a LATEX named counter, not of a TEX numeric register; therefore special LaTeX commands, such as `\setocounter` or `\value`, have to be used in order to set or access the numerical value stored within the TEX register associated to the LATEX

counter name.

```
\def\WF@startwrapping{%
  \ifnum\WF@wli=\z@ %                                  no number was specified
    \setcounter{WF@wrappedlines}%
      {\fpeval{round(\WF@size/\baselineskip,0)}}%
    \xdef\WF@wli{\value{WF@wrappedlines}}%
  \else
    \ifnum\WF@correctlines@switch>\z@ %              line number correction
      \setcounter{WF@wrappedlines}
        {\fpeval{round((\WF@size)/\baselineskip,0)+\WF@wli}}%
      \xdef\WF@wli{\value{WF@wrappedlines}}%
    \else
      \setcounter{WF@wrappedlines}{\WF@wli}%      absolute number of lines
      \global\advance\c@WF@wrappedlines\@ne
    \fi
  \fi
  \ifnum\c@WF@wrappedlines>\@ne %                                fine tuning
    \let\parshape\WF@fudgeparshape \let\WF@pspars\@empty \let\WF@@par\par
    \def\@setpar##1{\def\WF@@par{##1}}\def\par{\@par}\let\@par\WF@mypar
    \xdef\WF@restoretol{\tolerance\the\tolerance}\tolerance9999
    \advance\linewidth-\WF@adjlw \advance\@totalleftmargin\WF@adjtlm
  \fi}
```

The next macro is the one that actually indents the wrapping text lines and keeps track of the number of such processed lines. It can work on more than a single paragraph. It resorts to service macros that reiterate as long as the number of indented lines is lower than the computed number of lines. Possibly this process could be defined by means od the `dowhile` or `whiledo` LaTeX 3 functions. By now we did not afford this task, because first we would like to see if the overall software is reliable.

```
\def\WF@wraphand{%                    for indenting one or more paragraphs
  \ifnum\c@WF@wrappedlines<\tw@
    \WF@finale
  \else \begingroup %                             create a parshape command
    \@tempcnta\@ne \let\WF@wrapil\relax \gdef\WF@ps{}%
    \@whilenum
      \@tempcnta<\c@WF@wrappedlines\do{%              repeated indentation
      \xdef\WF@ps{\WF@ps\WF@wrapil}\advance\@tempcnta\@ne
      }%
    \endgroup
    \ifx\WF@pspars\@empty
      \@@parshape\c@WF@wrappedlines \WF@ps \WF@noil
    \else %           use external 'parshape' values to modify my parshape
      \WF@modps
    \fi
  \fi
}
```

This macro resets the paragraph properties and terminates the job.

```
\def\WF@mypar{\relax
  \WF@@par
  \ifnum\@@parshape=\z@
    \let\WF@pspars\@empty %                                  reset parshape
  \fi
  \global\advance\c@WF@wrappedlines-\prevgraf \prevgraf\z@
  \ifnum\c@WF@wrappedlines<\tw@
    \WF@finale
  \fi
}

```

These macros are to modify the paragraph settings.

```
\def\WF@modps{\begingroup
  \afterassignment\@tempdimb \@tempdima\WF@pspars % a=indent.num, b= width
  \advance\@tempdima-\WF@adjtlm \advance\@tempdimb\WF@adjlw
  \let\WF@wrapil\WF@pspars
  \edef\@tempb{\@@parshape\c@WF@wrappedlines
                \WF@ps \the\@tempdima \the\@tempdimb}%
  \expandafter\endgroup\@tempb
}

\let\@@setpar\@setpar
\def\WF@noil{\z@ \hsize}
\let\WF@pspars\@empty

\def\WF@fudgeparshape{\relax
  \ifnum\c@WF@wrappedlines<\tw@
    \WF@finale
  \else
    \afterassignment\WF@fudgeparshapee \fam
  \fi
}

\def\WF@fudgeparshapee{%
  \ifnum\fam=\@ne \expandafter
    \WF@parshapeee
  \else
    \WF@conflict \@@parshape\fam
  \fi
}

\def\WF@parshapeee#1#2{%
  \begingroup\delimitershortfall#1%
  \nulldelimiterspace#2%         \advance \nulldelimiterspace by \WF@adjlw
  \edef\@tempa{\def\noexpand\WF@pspars{%
      \the\delimitershortfall \the\nulldelimiterspace}}%
  \expandafter\endgroup\@tempa \WF@wraphand
}
```

The following macro is the one that actually ends the single wrapping job.

```
\def\WF@finale{%
  \ifx\parshape\WF@fudgeparshape
    \WF@restoretol \let\@setpar\@@setpar \let\par\WF@@par
    \advance\linewidth\WF@adjlw \advance\@totalleftmargin-\WF@adjtlm
    \WF@info{Finish wrapping text}%
    \ifx\par\@@par
      \def\@par{\let\par\@@par\par}%
    \else
      \let\@par\WF@@par
    \fi
    \let\parshape\@@parshape
    \parshape=\ifx\WF@pspars\@empty
                \z@
              \else
                \@ne \WF@pspars
              \fi
  \fi
  \ifvoid\WF@box
    \ifx\everypar\WF@everypar
      \let\everypar\WF@@everypar \everypar\expandafter{\the\WF@everypar}%
    \fi
  \fi
}
```

At the very end everything is restored, and the used boxes are emptied.

```
\newcommand{\WFclear}{\par
  \unless\ifvoid\WF@box
    \vskip\bigskipamount \box\WF@box
    \let\everypar\WF@@everypar \everypar\expandafter{\the\WF@everypar}%
  \fi
  \global\c@WF@wrappedlines\z@ \WF@finale
  \global\WF@correctlines@switch\z@
}
```

The following code is one of those "dirty tricks" by which a macro defined within a group is executed with the help of an `\expandafter` command that bypasses an `\endgroup`; by so doing nothing local to the group remains in memory.

```
\begingroup
 \toks0={\let\everypar\WF@@everypar
         \everypar\expandafter{\the\WF@everypar}%
         \let\parshape\@@parshape
         \let\@setpar\@@setpar
        }
 \toks1=\expandafter{\@arrayparboxrestore}%
 \toks2=\expandafter{\clearpage}%
 \edef\@tempa{%
```

```
      \def\noexpand\@arrayparboxrestore{\the\toks0 \the\toks1}%
      \def\noexpand\clearpage
      {\noexpand\protect\noexpand\WFclear \the\toks2}}%
 \expandafter
 \endgroup\@tempa

```

Donald Arseneau classifies the following macro as the one that "pampers the RevTeX's stupidity".

```
\@ifundefined{@capwidth}{\let\@capwidth\hsize}{}%

```

This one, instead, issues a warning if a specific name conflicts with another.

```
\def\WF@conflict{\WF@warning
  {\WF@wfname used inside a conflicting environment}}%

```

While this one issues a warning when a wrapping environment is too close to another one.

```
\def\WF@collision{\WF@warning{Collision between wrapping environments}}%

```

And this one is when two wrapping environments are too close to one another so that the second one is forced to float.

```
\def\WF@fltmes{%                                        message for floats
  \ifWF@float
    \WF@info{\WF@wfname floats}%
 \else
   \WF@warning{Stationary \WF@wfname forced to float}%
 \fi
}

```

These two aliases are just service macros for this package; in particular, the second one is used to insert info of any kind within a source file.

```
\let\WF@warning\@warning
\let\WF@info\@gobble

```

Arseneau says that his `wrapfig` package is already compatible with package `float.sty`, since, after defining a new float ⟨*foo*⟩, it suffices to define the new environment *wrap*⟨*foo*⟩. This fork version of his package should do the same: is suffices to mimic the definitions of environments *wrapfigure* or *wraptable*.

Here there is some Arseneau's code that renders his `wrapfig` code compatible with `\newfloat` of class `memoir`, and with `\newfloatlist` of package `ccaption`. We leave his code hereafter; but we did not test it with this package.

```
\let\WF@floatstyhook\relax
%
\@ifundefined{newfloat}{}{%         \newfloat comes from somewhere besides
%                                                          float.sty
  \@ifundefined{restylefloat}{%
```

```
    \@ifclassloaded{memoir}{%
      \toks@=\expandafter\expandafter\expandafter
       {\csname\string\newfloat\endcsname [{#1}]{#2}{#3}{#4}%
       \newenvironment{wrap#2}{\wrapfloat{#2}}{\endwrapfloat}%
      }%                              Mmmm; this might be wrong. Not tested
   \edef\@tempa{\def\expandafter\noexpand\csname\string\newfloat\endcsname
        [##1]##2##3##4{\the\toks@}}%
      \@tempa
     }%                                                 end memoir support
     {}%                                    other origins of \newfloat here?
  }{%          float.sty handler. Ops: Two versions for different versions
%             Changing \floatstyle or \restylefloat changes \newfloat too.
  \@ifundefined{float@restyle}%
    {%                                                    older float.sty
      \toks@=\expandafter{\restylefloat{#1}%       env. might be undefined
      \@namedef{wrap#1}{%
        \def\@captype{#1}\@nameuse{fst@#1}%
        \def\WF@floatstyhook{\let\@currbox\WF@box \columnwidth\wd\WF@box
          \global\setbox\WF@box\float@makebox}%
      \@ifnextchar[\WF@wr{\WF@wr[]}}%
      \expandafter\let\csname endwrap#1\endcsname \endwrapfigure
      }%
    \edef\@tempa{\def\noexpand\restylefloat##1{\the\toks@}}%
  }{%             newer float.sty: uses \float@restyle, and \float@makebox
%                                                       takes width arg
      \toks@=\expandafter{\float@restyle{#1}%      env. might be undefined
        \@namedef{wrap#1}{\def\@captype{#1}\@nameuse{fst@#1}%
          \def\WF@floatstyhook{\let\@currbox\WF@box
            \global\setbox\WF@box\float@makebox{\wd\WF@box}}%
        \@ifnextchar[\WF@wr{\WF@wr[]}}%
        \expandafter\let\csname endwrap#1\endcsname \endwrapfigure
      }%
      \edef\@tempa{\def\noexpand\float@restyle##1{\the\toks@}}%
    }%
    \@tempa %                                          perform redefinitions
%
    }%                                                 end float.sty handler
}%                                          end redefinitions of \newfloat

\ifcsname newfloatlist\endcsname%                       support ccaption.sty
  \toks@=\expandafter\expandafter\expandafter
    {\csname\string\newfloatlist\endcsname [{#1}]{#2}{#3}{#4}{#5}%
  \@namedef{wrap#2}{\wrapfloat{#2}}%
  \expandafter\let\csname endwrap#2\endcsname \endwrapfloat
  }%
  \edef\@tempa{%
  \def\expandafter\noexpand\csname\string\newfloatlist\endcsname
   [##1]##2##3##4##5{\the\toks@}}%
  \@tempa
\fi
```

We never described the package options; this code was present in Arseneau's code and we leave it here, with the necessary package-name changes. We think that this *verbose* option was and remains useless, since the instances of command `\WF@info` were mostly commented out in the original code; in any case, input of this code is stopped if the `\DeclareOption` command is not defined; this command was defined with LaTeX 2_ε; therefore might be a residual of the old times when LaTeX 2.09 was still in use, more than 25 years ago...

```
\@ifundefined{DeclareOption}{\endinput}{%
\def\WF@warning{\PackageWarning{wrapfig2}}%
\DeclareOption{verbose}{\def\WF@info{\PackageInfo{wrapfig2}}}%
\ProcessOptions
\AtEndDocument{\WFclear}}%

\endinput
```