

# The `wrapfig2` package

Claudio Beccari\*

Version v.4.0 – Last revised 2021-12-21.

<b>Contents</b>			
<b>1 Introduction</b>	<b>1</b>	3.2 A wrapped table . . . . .	4
<b>2 Environment syntax</b>	<b>3</b>	3.3 A wrapped text . . . . .	5
<b>3 Examples</b>	<b>3</b>	<b>4 Remarks</b>	<b>6</b>
3.1 A wrapped figure . . . . .	3	<b>5 Other floating objects</b>	<b>7</b>
		<b>6 The code</b>	<b>8</b>

## Abstract

This new package `wrapfig2` is a fork that upgrades Donald Arseneau’s package `wrapfig` (version 2.6, dated 2003) by adding some L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X kernel, otherwise it won’t work; any L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 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 the solution is to wrap the inserted object with a sufficient number of lines, i.e. by a sufficiently long paragraph.

The above implies that no wrapped object code should be inserted by 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 order with 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 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  $\text{\TeX/L\TeX}$   $2_\epsilon$  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  *$\langle$ number of indented lines $\rangle$*  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 any user first uses the software to insert an object to be wrapped by the surrounding text without specifying any value with the specific optional argument; then evaluates the result, and if the space below the wrapped object is too large, or if such space is too small s/he counts the necessary number of lines and specifies 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 third 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 `textstackexchange` 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  *$\langle$ location $\rangle$*  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: with a value too different from  $0.5\text{\linewidth}$  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

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.

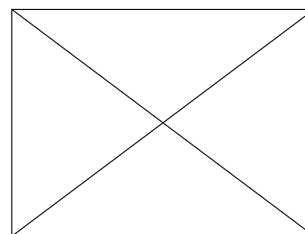


Figure 1: A rectangle

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 indentation of the wrapping text. Nevertheless the software can compute the object width if a zero value is specified or if the mandatory `<width>` parameter is completely omitted; this second possibility is a feature of this package, that uses a L<sup>A</sup>T<sub>E</sub>X 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}{1}
\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 this poor one, with the last caption line overlapping the wrapping text.

First	Second	Let us suppose that the noumena have nothing to do with
Third	Fourth	necessity, since knowledge of the Categories is a posteriori.

Table 2: A small table. 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.

If

### 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  $\langle width \rangle$  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  $\langle width \rangle$  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  $\langle wrapfigure \rangle$  environment.

## 4 Remarks

The syntax of the original environments  $\langle wrapfigure \rangle$  and  $\langle wraptable \rangle$  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  $\langle line number \rangle$  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  $\langle placement \rangle$  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 indentation, since the mechanism does not consider the inter paragraph spacing that  $\text{\TeX}$  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  $\langle overhang \rangle$  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  $\langle width \rangle$  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, for the redefined environments  $\langle wrapfigure \rangle$  and  $\langle wraptable \rangle$ , this braced argument is optional.

## 5 Other floating objects

Pictures and textual arrays may be floated by means of the standard  $\langle figure \rangle$  and  $\langle table \rangle$  environments. But other floating objects may be defined by means of the `float` package, just to name one, or classes. 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  $\langle wrapfigure \rangle$ , by using instead:

```
\begin{wrapfloat}{figure}[\langle line number \rangle]{\langle placement \rangle}%
  [\langle overhang \rangle]{\langle width \rangle}{\star}
  \langle image \rangle
\end{wrapfloat}
```

Another  $\langle object \rangle$  may be wrapped by using:

```

\begin{wrapfloat}{\langle object name \rangle}[\langle line number \rangle]{\langle placement \rangle}%
  [\langle overhang \rangle]{\langle width \rangle}\langle \star \rangle
  \langle object \rangle
\end{wrapfloat}

```

If the floating  $\langle placement \rangle$  codes have to be used, another floating object with the desired  $\langle object name \rangle$  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 it 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 L<sup>A</sup>T<sub>E</sub>X kernel. From the L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 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  $\langle width \rangle$  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*.

```

1 \NeedsTeXFormat{LaTeX2e}[2020-01-01]
2 \ProvidesPackage{wrapfig2}%
3 [2021-11-30 v.4.0 Wrap text around figures, tables, framed text blocks]
4 \@ifpackageloaded{etoolbox}{\RequirePackage{etoolbox}}
5 \ifcsdef{c@WF@wrappedlines}{\endinput}{}
6 \ifcsdef{fpeval}{\RequirePackage{xfp}}
7 \RequirePackage{xparse}
8

```

Next we define some dimensions, boxes, token registers, T<sub>E</sub>X counters, and alias names. The `WF@correctlines@switch` T<sub>E</sub>X numeric register (not a L<sup>A</sup>T<sub>E</sub>X counter) is going to be used a boolean switch; if its value is zero, it 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.

```

9 \newdimen\wrapoverhang \wrapoverhang\z@
10 \newdimen\WF@size
11 \newcounter{WF@wrappedlines}
12 \newbox\WF@box
13 \newbox\NWF@box
14 \newtoks\WF@everypar
15 \newif\ifWF@float
16 \newcount\WF@correctlines@switch
17 \newdimen\insertwidth
18 \let\@parshape\parshape
19 \let\WF@@everypar\everypar
20

```

In what follows we are going to use very often the functionalities of the **xparse** package that are mostly already included into the L<sup>A</sup>T<sub>E</sub>X kernel; but, since we use one of this functionalities that have not migrated to the L<sup>A</sup>T<sub>E</sub>X kernel, we load it, as they remain available with that package; nevertheless we specified a L<sup>A</sup>T<sub>E</sub>X format date insuring that the L<sup>A</sup>T<sub>E</sub>X 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 user have available a definitely older T<sub>E</sub>X system installation, s/he should upgrade it, or must avoid using this **wrapfig2** package and use the original one; if s/he needs to wrap text, s/he 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 *wratable* environments are very simple by means of the underlying *wrapfloat* environments.

```

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

```

```

23  {\endwrapfloat}
24
25  \NewDocumentEnvironment{wraptable}{o m o G{Opt}}{%
26    {\wrapfloat{table}[#1]{#2}[#3]{#4}}%
27    {\endwrapfloat}
28

```

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 text. 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 *optional braced 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.

It is true that some of the input parameters specified to the opening command with L<sup>A</sup>T<sub>E</sub>X 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 of T<sub>E</sub>X count registers are *local*, while assignment of the L<sup>A</sup>T<sub>E</sub>X named counters, through the `\setcounter` macro and its siblings, are *global*) so that we can use their values within the closing commands.

The `\NWFB@box` has been assigned at the beginning; remember that L<sup>A</sup>T<sub>E</sub>X 3 registers of any kind are not limited in number as they were some years ago even with L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>. 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` L<sup>A</sup>T<sub>E</sub>X 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.

```

29 \NewDocumentEnvironment{wraptext}%
30     {0{1}    D||{0.5\columnwidth} D<>{0}    D(){figure}}%
31 {%                                                    Open environment
32     \insertwidth=#2\relax
33     \def\textplacement{#1}%
34     \def\textcorrection{#3}%
35     \def\WF@caption@label{#4}%
36     \begin{lrbox}\NWF@box%                This box is to contain the framed text
37         \minipage{\insertwidth}%
38         \tcolorbox
39     }{%                                                    Close environment
40     \end{tcolorbox}\end{minipage}\end{lrbox}%
41     \edef\NWF@wli%                \NWF@wli is a macro, not a counter
42     {%
43         \fpeval{%
44             round((\ht\NWF@box+\dp\NWF@box)/\baselineskip,0)+2+\textcorrection
45         }%
46     }%
47     \ifhmode\unskip\else\leavevmode\noindent\fi
48     \bgroup\edef\x{\egroup\noexpand\wrapfloat{\WF@caption@label}[\NWF@wli]%
49         {\textplacement}{\the\insertwidth}}\x
50     \box\NWF@box %                Output framed box containing text
51     \endwrapfloat
52     \ignorespaces
53 }
54

```

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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 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 test, 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` to take care of examining 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

$\langle placement \rangle$  argument.

```

55
56 \NewDocumentEnvironment{wrapfloat}{m o m o G{\z@}}%
57 {\def\@captype{#1}\WF@wr[#2]{#3}{#4}{#5}}%
58 {\ifdim\hsize>\z@
59   \par\hrule\@width\hsize\@height\z@ %   force width with invisible rule
60   \else
61     \unskip \egroup \box\z@ %               or close hbox
62   \fi
63 \egroup %                               close the vtop box; its width now is known
64 \WF@floatstyhook %                       support for float.sty
65 \def\width{\wd\WF@box}%
66 \setlength\wrapoverhang{\WF@ovh}%
67 \xdef\WF@ovh{\the\wrapoverhang}%         save until wrapping
68 \ifdim\ht\WF@box>\topskip \ht\WF@box\z@ \fi%   too high, set flag
69 \ifdim\ht\WF@box<.5\p@ %                   too short, move up
70   \global\setbox\WF@box\vtop{\vskip-1.4ex\unvbox\WF@box}%
71 \fi
72 \global\WF@size=%                         compute total box height with \fpeval
73 \fpeval{\ht\WF@box+\dp\WF@box+1.5\baselineskip+\tw@intextsep}\p@
74 \aftergroup\WF@startfloating %             use even when not really floating
75 \unless\ifWF@float
76   \ifhmode
77     {\unskip \parfillskip\z@skip \par \vskip-\parskip}%
78     \aftergroup\noindent
79   \fi
80 \fi
81 \global\@ignoretrue
82 }
83

```

The working macro `\WF@wr` is defined with the  $\text{\LaTeX 3}$  language; it grabs all the optional and mandatory arguments in a single step, contrary to  $\text{\LaTeX 2}_\epsilon$  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  $\epsilon\text{\TeX}$  typesetting program extensions, that have been included in the  $\text{pdf}\text{\LaTeX}$ ,  $\text{Xe}\text{\LaTeX}$  and  $\text{Lua}\text{\LaTeX}$  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  $\langle overhang \rangle$  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  $\text{\LaTeX 3}$  `\fpeval` computing macro 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 mandatory #4  $\langle width \rangle$  parameter (actually a  $\langle optional\ braced\ parameter \rangle$ ) may be specified to be Opt; in any case Opt is the default parameter value; if so, the object is treated at its natural width, by boxing it into an hbox and using this hbox width as the working width

```

84 \NewDocumentCommand\WF@wr{o m o m s}{%
85   \xdef\WF@wfname{wrap\@capttype\space}%
86   \unless\ifvoid\WF@box
87     \WFClear \WF@collision
88   \fi
89   \xdef\WF@place{\string'\@car#2r\@nil}%
90   \ifnum\lccode\WF@place=\WF@place
91     \global\WF@floatfalse
92   \else
93     \global\WF@floattrue
94   \fi
95   \ifx\parshape\WF@fudgeparshape
96     \unless\ifWF@float\WF@collision\fi
97   \else
98     \ifx\par\@par
99       \ifnum\@parshape>\z@\WF@conflict\fi
100    \else
101      \WF@conflict
102    \fi
103    \fi
104    \IfValueTF{#1}%           save optional line number or correction
105      {\gdef\WF@wli{#1}}%
106      {\gdef\WF@wli{0}}%
107 %
108 \IfValueTF{#3}%           save optional overhang
109   {\gdef\WF@ovh{#3}}%
110   {\gdef\WF@ovh{\z@}}%
111 %
112 \global\setbox\WF@box\vtop\bgroup \setlength\hsize{#4}%       set width
113 \ifdim\hsize>\z@
114   \@parboxrestore
115 \else
116   \setbox\z@\hbox\bgroup
117   \let\wf@caption\caption
118   \let\caption\wf@caption
119   \ignorespaces
120 \fi
121 \IfBooleanTF{#5}%         if the asterisk is present set the numerical switch
122   {\global\WF@correctlines@switch=\@ne}%
123   {\global\WF@correctlines@switch=\z@}%
124 \global\@ignoretrue
125 }
126

```

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.

```

127 \def\wf@caption{\relax%           redefine \wf@caption in case \hsize is zero
128   \ifdim\hsize>\z@
129     \let\caption\wf@@caption
130   \else
131     \unskip \egroup \hsize\wd\z@ \@parboxrestore \box\z@%      empty \box0
132   \fi
133   \caption
134 }
135
```

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

```

136 \def\WF@startfloating{%
137   \WF@everypar\expandafter{\the\everypar}\let\everypar\WF@everypar
138   \WF@@everypar{\ifvoid\WF@box\else\WF@floathand\fi \the\everypar
139   \WF@wrapand
140 }}
141
```

The following macro is for floating wrapping environments.

```

142 \def\WF@floathand{%
143   \ifx\parshape\WF@fudgeparshape
144     \WF@fltmes
145   \else
146     \ifx\par\@par
147       \ifnum\@parshape=\z@
148         \ifdim\hangindent=\z@
149           \setbox\z@\lastbox \begingroup
150           \@par \WF@everypar{}\WF@putfigmaybe
151           \endgroup %           after this group start wrapping
152           \unless\ifvoid\z@ %   replace indentation
153             \box\z@
154         \fi
155       \else
156         \WF@fltmes
157       \fi
158     \else
159       \WF@fltmes

```

```

160     \fi
161     \else
162         \WF@fltmes
163     \fi
164 \fi}
165

```

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

```

166 \def\WF@putfigmaybef{%
167 \ifinner
168     \vskip-\parskip \global\WF@floatfalse
169     \let\pagetotal\maxdimen %          kludge flag for "not top of page"
170 \else %                                outer page
171     \@tempdima\pagedepth %             save page depth
172     {\advance\parskip\@tempdima\vskip-\parskip}%    back up to base line
173     \penalty\interlinepenalty %        update page parameters
174     \@tempdimb\pagegoal \advance\@tempdimb-\pagetotal %    room left on page
175     \ifdim \@tempdimb<\z@ %            page already full
176         \global\WF@floatfalse
177         \unless\ifdim-\@tempdimb>\pageshrink
178             \pagebreak
179         \fi
180     \else
181         \ifdim\WF@size>\@tempdimb%      box too high does not fit in \@tempdimb
182             \ifWF@float
183                 \dimen@.5\baselineskip
184             \else
185                 \dimen@ 2\baselineskip
186             \fi
187             \ifdim\pagestretch>\dimen@ \dimen@\pagestretch \fi
188             \ifdim\pagefilstretch>\z@ \dimen@\@tempdimb \fi
189             \ifdim\pagefillstretch>\z@ \dimen@\@tempdimb \fi
190             \advance\dimen@.5\baselineskip
191             \ifdim\dimen@>\@tempdimb %    stretch page contents
192                 \global\WF@floatfalse \pagebreak
193             \fi
194         \else %                          box fits in \@tempdimb
195             \global\WF@floatfalse
196         \fi
197     \fi
198     \vskip\@tempdima\relax %              return erased page depth
199 \fi
200 \noindent
201 \ifWF@float
202     \WF@fltmes
203 \else %                                place insertion here
204     \WF@info{Put \WF@wfname here:}%
205     {\ifodd
206         \if@twoside\c@page\else\@ne\fi %    assign l/r to i/o placement

```

```

207 \lccode'i'1\lccode'o'r\else \lccode'i'r\lccode'o'l%
208 \fi
209 \xdef\WF@place{\the\lccode\lccode\WF@place}%
210 }%
211 \hbox to\z@{%
212 \tempdima\wd\WF@box \tempdimb\WF@ovh
213 \advance\tempdima-\tempdimb \advance\tempdima\columnsep
214 \tempdimb\hsize \advance\tempdimb-\tempdima
215 \xdef\WF@adjlw{\the\tempdima}%
216 \ifnum '1=\WF@place %
217 \hss
218 \def\@tempa{\kern\columnsep}%
219 \else %
220 \tempdima\z@ %
221 \kern\tempdimb \kern\columnsep
222 \def\@tempa{\hss}%
223 \fi
224 \ifdim\tempdimb<\hsize
225 \xdef\WF@wrapil{\the\tempdima \the\tempdimb}%
226 \xdef\WF@adjtlm{\the\tempdima}%
227 \else
228 \xdef\WF@wrapil{\z@ \the\hsize}%
229 \xdef\WF@adjlw{\z@}\xdef\WF@adjtlm{\z@}%
230 \fi
231 \ifdim\pagetotal=\z@ %
232 \global\advance\WF@size-\intextsep
233 \else %
234 \setbox\WF@box\hbox{\lower\intextsep\box\WF@box}%
235 \fi
236 \dp\WF@box\z@
237 \box\WF@box
238 \@tempa
239 }%
240 \aftergroup\WF@startwrapping
241 \fi
242 }
243

```

Here comes the very important macro that counts the wrapping indented lines, so that wrapping is correct; of course the limitations of the  $\text{\TeX}$  and  $\text{\LaTeX}$  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 is the one that 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  $\text{\LaTeX}$  named counter, not of a  $\text{\TeX}$  numeric register; therefore special LaTeX commands, such as `\setcounter` or `\value`, have to be used in order to set or access the numerical value stored within the  $\text{\TeX}$  register associated



to the L<sup>A</sup>T<sub>E</sub>X counter name.

```

244 \def\WF@startwrapping{%
245   \ifnum\WF@wli=\z@ %                               no number was specified
246     \setcounter{WF@wrappedlines}%
247     {\fpeval{round(\WF@size/\baselineskip,0)}}%
248     \xdef\WF@wli{\value{WF@wrappedlines}}%
249   \else
250     \ifnum\WF@correctlines@switch>\z@ %               line number correction
251       \setcounter{WF@wrappedlines}
252       {\fpeval{round((\WF@size)/\baselineskip,0)+\WF@wli}}%
253       \xdef\WF@wli{\value{WF@wrappedlines}}%
254     \else
255       \setcounter{WF@wrappedlines}{\WF@wli}%         absolute number of lines
256       \global\advance\c@WF@wrappedlines\@ne
257     \fi
258   \fi
259   \ifnum\c@WF@wrappedlines>\@ne %                     fine tuning
260     \let\parshape\WF@fudgeparshape \let\WF@pspars\@empty \let\WF@par\par
261     \def\@setpar##1{\def\WF@par{##1}}\def\par{\@par}\let\@par\WF@mypar
262     \xdef\WF@restoretol{\tolerance\the\tolerance}\tolerance9999
263     \advance\linewidth-\WF@adjlw \advance\@totalleftmargin\WF@adjtln
264   \fi}
265

```

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 of the `dowhile` or `whiledo` L<sup>A</sup>T<sub>E</sub>X 3 functions. By now we did not afford this task, because first we would like to see if the overall software is reliable.

```

266 \def\WF@wrapand{%                                     for indenting one or more paragraphs
267   \ifnum\c@WF@wrappedlines<\tw@
268     \WF@finale
269   \else \begingroup %                                   create a parshape command
270     \@tempcnta\@ne \let\WF@wrapil\relax \gdef\WF@ps{}%
271     \@whilenum
272       \@tempcnta<\c@WF@wrappedlines\do{%               repeated indentation
273         \xdef\WF@ps{\WF@ps\WF@wrapil}\advance\@tempcnta\@ne
274       }%
275     \endgroup
276     \ifx\WF@pspars\@empty
277       \@parshape\c@WF@wrappedlines \WF@ps \WF@noil
278     \else %                                             use external 'parshape' values to modify my parshape
279       \WF@modps
280     \fi
281   \fi
282 }
283

```

This macro resets the paragraph properties and terminates the job.

```

284 \def\WF@mypar{\relax
285   \WF@par
286   \ifnum\@@parshape=\z@
287     \let\WF@pspars\empty % reset parshape
288   \fi
289   \global\advance\c@WF@wrappedlines-\prevgraf \prevgraf\z@
290   \ifnum\c@WF@wrappedlines<\tw@
291     \WF@finale
292   \fi
293 }
294

```

These macros are to modify the paragraph settings.

```

295 \def\WF@modps{\begingroup
296   \afterassignment\@tempdimb \@tempdima\WF@pspars % a=indent.num, b= width
297   \advance\@tempdima-\WF@adjtlm \advance\@tempdimb\WF@adjlw
298   \let\WF@wrapil\WF@pspars
299   \edef\@tempb{\@@parshape\c@WF@wrappedlines
300     \WF@ps \the\@tempdima \the\@tempdimb}%
301   \expandafter\endgroup\@tempb
302 }
303
304 \let\@@setpar\@setpar
305 \def\WF@noil{\z@ \hsize}
306 \let\WF@pspars\empty
307
308 \def\WF@fudgeparshape{\relax
309   \ifnum\c@WF@wrappedlines<\tw@
310     \WF@finale
311   \else
312     \afterassignment\WF@fudgeparshapee \fam
313   \fi
314 }
315
316 \def\WF@fudgeparshapee{%
317   \ifnum\fam=\@ne \expandafter
318     \WF@parshapeeee
319   \else
320     \WF@conflict \@@parshape\fam
321   \fi
322 }
323
324 \def\WF@parshapeeee#1#2{%
325   \begingroup\delimitershortfall#1%
326   \nulldelimiterspace#2% \advance \nulldelimiterspace by \WF@adjlw
327   \edef\@tempa{\def\noexpand\WF@pspars{%
328     \the\delimitershortfall \the\nulldelimiterspace}}%
329   \expandafter\endgroup\@tempa \WF@wrapand
330 }

```

331

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

```

332 \def\WF@finale{%
333   \ifx\parshape\WF@fudgeparshape
334     \WF@restoretol \let\@setpar\@setpar \let\par\WF@@par
335     \advance\linewidth\WF@adjlw \advance\@totalleftmargin-\WF@adjtlm
336     \WF@info{Finish wrapping text}%
337     \ifx\par\@par
338       \def\@par{\let\par\@par\par}%
339     \else
340       \let\@par\WF@@par
341     \fi
342     \let\parshape\@parshape
343     \parshape=\ifx\WF@pspars\@empty
344       \z@
345     \else
346       \@ne \WF@pspars
347     \fi
348   \fi
349   \ifvoid\WF@box
350     \ifx\everypar\WF@everypar
351       \let\everypar\WF@@everypar \everypar\expandafter{\the\WF@everypar}%
352     \fi
353   \fi
354 }
355
```

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

```

356 \newcommand{\WFclear}{\par
357   \unless\ifvoid\WF@box
358     \vskip\bigskipamount \box\WF@box
359     \let\everypar\WF@@everypar \everypar\expandafter{\the\WF@everypar}%
360   \fi
361   \global\c@WF@wrappedlines\z@ \WF@finale
362   \global\WF@correctlines@switch\z@
363 }
364
```

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.

```

365 \begingroup
366 \toks0={\let\everypar\WF@@everypar
367   \everypar\expandafter{\the\WF@everypar}%
368   \let\parshape\@parshape
369   \let\@setpar\@setpar
370 }
371 \toks1=\expandafter{\@arrayparboxrestore}%
372 \toks2=\expandafter{\clearpage}%
373 \edef\@tempa{%

```

```

374 \def\noexpand\arrayparboxrestore{\the\toks0 \the\toks1}%
375 \def\noexpand\clearpage
376 {\noexpand\protect\noexpand\WFclear \the\toks2}}%
377 \expandafter
378 \endgroup\@tempa
379

```

Donald Arseneau classifies the following macro as the one that “pampers the RevTeX’s stupidity”.

```

380 \@ifundefined{capwidth}{\let\capwidth\hsize}{}%
381

```

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

```

382 \def\WF@conflict{\WF@warning
383 {\WF@wfname used inside a conflicting environment}}%
384

```

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

```

385 \def\WF@collision{\WF@warning{Collision between wrapping environments}}%
386

```

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

```

387 \def\WF@fltmes{%                                message for floats
388 \ifWF@float
389 \WF@info{\WF@wfname floats}%
390 \else
391 \WF@warning{Stationary \WF@wfname forced to float}%
392 \fi
393 }
394

```

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.

```

395 \let\WF@warning\@warning
396 \let\WF@info\@gobble
397

```

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

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.

```

398 \let\WF@floatstyhook\relax
399 %
400 \@ifundefined{newfloat}{\newfloat comes from somewhere besides
401 %                                float.sty
402 \@ifundefined{restylefloat}}%

```

```

403 \ifclassloaded{memoir}{%
404 \toks@=\expandafter\expandafter\expandafter
405 {\csname\string\newfloat\endcsname [{#1}]{#2}{#3}{#4}%
406 \newenvironment{wrap#2}{\wrapfloat{#2}}{\endwrapfloat}%
407 }% Mmmm; this might be wrong. Not tested
408 \edef\@tempa{\def\expandafter\noexpand\csname\string\newfloat\endcsname
409 [{#1}##2##3##4{\the\toks@}}%
410 \@tempa
411 }% end memoir support
412 }% other origins of \newfloat here?
413 }{% float.sty handler. Ops: Two versions for different versions
414 % Changing \floatstyle or \restylefloat changes \newfloat too.
415 \ifundefined{float@restyle}%
416 {% older float.sty
417 \toks@=\expandafter{\restylefloat{#1}% env. might be undefined
418 \@namedef{wrap#1}{%
419 \def\@cuptype{#1}\@nameuse{fst@#1}%
420 \def\WF@floatstyhook{\let\@currbox\WF@box \columnwidth\wd\WF@box
421 \global\setbox\WF@box\float@makebox}%
422 \@ifnextchar[\WF@wr{\WF@wr[]}}%
423 \expandafter\let\csname endwrap#1\endcsname \endwrapfigure
424 }%
425 \edef\@tempa{\def\noexpand\restylefloat##1{\the\toks@}}%
426 }{% newer float.sty: uses \float@restyle, and \float@makebox
427 % takes width arg
428 \toks@=\expandafter{\float@restyle{#1}% env. might be undefined
429 \@namedef{wrap#1}{\def\@cuptype{#1}\@nameuse{fst@#1}%
430 \def\WF@floatstyhook{\let\@currbox\WF@box
431 \global\setbox\WF@box\float@makebox{\wd\WF@box}}%
432 \@ifnextchar[\WF@wr{\WF@wr[]}}%
433 \expandafter\let\csname endwrap#1\endcsname \endwrapfigure
434 }%
435 \edef\@tempa{\def\noexpand\float@restyle##1{\the\toks@}}%
436 }%
437 \@tempa % perform redefinitions
438 %
439 }% end float.sty handler
440 }% end redefinitions of \newfloat
441
442 \ifcsname newfloatlist\endcsname% support ccaption.sty
443 \toks@=\expandafter\expandafter\expandafter
444 {\csname\string\newfloatlist\endcsname [{#1}]{#2}{#3}{#4}{#5}%
445 \@namedef{wrap#2}{\wrapfloat{#2}}%
446 \expandafter\let\csname endwrap#2\endcsname \endwrapfloat
447 }%
448 \edef\@tempa{%
449 \def\expandafter\noexpand\csname\string\newfloatlist\endcsname
450 [{#1}##2##3##4##5{\the\toks@}}%
451 \@tempa
452 \fi

```

453

We never described the package options; this code was present in Arseneau's code and we leave it here, with the necessary package name change. 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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>; therefore this is a residual of the old times when L<sup>A</sup>T<sub>E</sub>X 2.09 was still in use, more than 25 years ago...

```
454 \@ifundefined{DeclareOption}{\endinput}{%  
455 \def\WF@warning{\PackageWarning{wrapfig2}}%  
456 \DeclareOption{verbose}{\def\WF@info{\PackageInfo{wrapfig2}}}%  
457 \ProcessOptions  
458 \AtEndDocument{\WFclear}}%  
459  
460 \endinput
```