The wrapfig2 package

Claudio Beccari*

Version v.6.1.1 – Last revised 2023-02-23.

Contents

Abstract

This new package `wrapfig2` is a fork that extends Donald Arseneau's package `wrapfig` (version 3.6, 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 original *wrapfigure* and *wraptable*, namely *wraptext*; it may be used to wrap a small framed text block on a coloured background; the philosophy of this new environment is similar to that of the other two environments, but the syntax was different with version 4 of this package, and is very similar with version 5.0; a further important enhancement is implemented in version 6. Fall back options are available for backwards compatibility.

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

Read carefully this document, because the are several pieces of information concerning other packages that may be incompatible with this `wrapfig2` version. Special warnings are typeset in red as this one.

1 Introduction

The purpose of this package is manyfold. On one side it tries to upgrade the original software by Donald Arseneau by using some parts of the LaTeX 3 modern language. On another 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 wrapped text by framing them while typesetting the text on a coloured background.

*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; we are sorry to admit that such idiosyncrasies might have been slightly 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 reasonably long paragraphs.

The above implies that no wrapped object code should be specified in the source file close the end of a paragraph, unless it is followed by other paragraphs; again no object code should be inserted within any list; not even close to the end or to the beginning of a 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 object has a numbered caption, the number might not result in the correct sequence with the normal corresponding floating objects.

Therefore the usefulness of the wrapping procedure depends very much on the document authors' ability to move around their code until a suitable position is found. Certainly a good place is within a longish paragraph especially at the beginning of a section; or at the beginning of a chapter that starts with plain text, in particular just at the beginning of the first paragraph.

The code of this package does very little, if anything, to correct such idiosyncrasies. They are caused by the limitations of the `\ShipOut` LaTeX 2_ε kernel macro, and very little we were able to do in addition to what Arseneau already did.

Another purpose of this package is to add another optional argument so that the ⟨*number of indented lines*⟩ argument does not mean the total number, but the correction number to add-to or subtract-from the value computed by the default mechanism devised by Arseneau.

We assume that most users 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 computed value.

A further purpose of this package is to define a new environment, *wraptext*, to wrap a framed text block typeset on a coloured background. 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 and used the *tcolorbox* 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 it 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 would have similar problems.

Notice that the first implementation of this package, version 4, achieved the desired result but there were two drawbacks: (*a*) the syntax was rather different

from that of the other environments, and (*b*) any possible caption was typeset within the same framing environment. In version 5 both drawbacks were eliminated, but since the environment syntax is different, in oder to assure backwards compatibility a package option was defined in order to fall back to the previous version 4 behaviour. In version 6.0 the *wraptext* environment was further enhanced so as to accept several `key=value` settings concerned wit the text appearance, the colours, and other details relative to the wrapping process. Again another fall back option was defined in order to use the same functionalities of version 5.

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.

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 to be maintained identical. When the optional star is specified, the ⟨*indented lines number*⟩ is interpreted as the correction to the computed value.

Notice the different syntax in versions 4, 5, and 6 of the *wraptext* syntax.

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

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

Package option ⟨*WFold*⟩ required for backwards compatibility with version 4.*.

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

Package option ⟨*WFfive*⟩ required for backwards compatibility with version 5.*.

```
\begin{wraptext}[⟨indented lines number correction⟩] {⟨location⟩} [⟨overhang⟩] {⟨width⟩}
    ⟨optional colour settings⟩
    \includeframedtext[⟨insertion measure⟩]{⟨text to frame⟩}[⟨⟨frame thickness⟩,⟨frame
separation⟩⟩][⟨radius⟩]
\end{wraptext}
```

No package option required for version 6.*.

```
\begin{wraptext}[⟨indented lines number correction⟩]{⟨location⟩}[⟨overhang⟩]{⟨width⟩}
    ⟨optional style settings⟩
    \includeframedtext[⟨insertion measure⟩]{⟨text to frame⟩}[⟨settings⟩][⟨radius⟩]
\end{wraptext}
```

Please notice that the*wraptext* environment does not require any optional star, because the specified indented lines number is always interpreted as its *correction, not its absolute value*; this difference is clearly marked in the above syntax medallion. If users specified the star in similitude with the other two environments, with versions 5.* and 6.* the unnecessary star produces a strongly emphasised warning message visible in the editor console and in the `.log` file. With the old version 4.*

the unnecessary star is printed as part of the text to be framed.
Please notice also that all three *wraptext* syntaxes, thanks to differently delimited optional arguments with peculiar default values, become very similar when such optional arguments are reduced to a minimum; only the ⟨*location*⟩ argument is delimited by brackets with the old version and with braces with the newer ones.

It may be useful to compare the `\includeframedtext` macro, used to insert a framed test into a *wraptext* environment, with `\includegraphics`, used to insert an external image into a *figure* environment. Their functions are similar even if they refer to different objects to include. Their codes are obviously very different and the latter is much more complex than the former. The solution for a framed text used by version 4 was inspired by the information found on `texstackexchange` that used the very elaborate *tcolorbox* environment; version 5 uses instead a much simpler command `\framedbox` based on the `curve2e` package macro `\Curve`. Version 6.0 admits many settings, not just the ⟨*frame thickness*⟩ and the space around the framed text, but several others ones collectively indicated with ⟨*settings*⟩; such settings refer to colours, dimensions, styles, and so on, relative to the text and to the frame; moreover they are set by means of the `key=value` syntax. As it can be seen, the logic, not the code, behind these different macros are very similar.

3 Examples

We display some examples by using fake objects and suitably long paragraphs; some fake-language long-paragraphs are obtained by means of the `kantlipsum` package functionalities; they are typeset with an italic font in order to distinguish their text from the normal one.

3.1 A wrapped figure

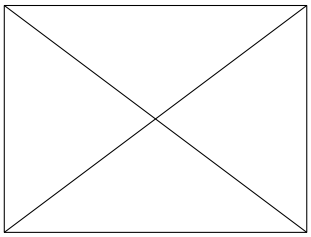


Figure 1: A rectangle with its diagonals

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 figure 1 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}\label{fig:figure}
\end{wrapfigure}
{\itshape \kant[1]}
```

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

3.2 A wrapped table

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 wrapped small table 1 has been typeset by means of the following code.

```
\begin{wraptable}[-1]{l}*
  \centering
  \begin{tabular}{cc}
    \hline
    First & Second\\
    Third & Fourth\\
    \hline
  \end{tabular}
  \caption{A small table}
\end{wraptable}
{\itshape \kant[2]}
```

Notice the absence of the braced width value; as said below, 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 a smart use of this width parameter might yield better looking results.

Wrapping a small table is a little more difficult than wrapping a figure, because the width of the inserted object is not exactly known in advance, and it is difficult to estimate; therefore it might be necessary to execute several trial compilations. 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 such feature.

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 is shown in table 2, but it is a very 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 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.

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}{l}
\includeframedtext{Text, text, text, text, text, text, text, text,
text, text, text.}
\end{wraptext}
{\itshape \kant[3]}
```

The result is the same as that obtainable with version 4 of this package, but the ⟨*location*⟩ argument specification is braced instead of bracketed.

Remember, though, what was previously remarked about using an unnecessary optional star with the *wraptext* environment; with this package versions 5 and 6 the unnecessary star produces a warning message, while with version 4 it prints the star as if it was part of the text to be wrapped.

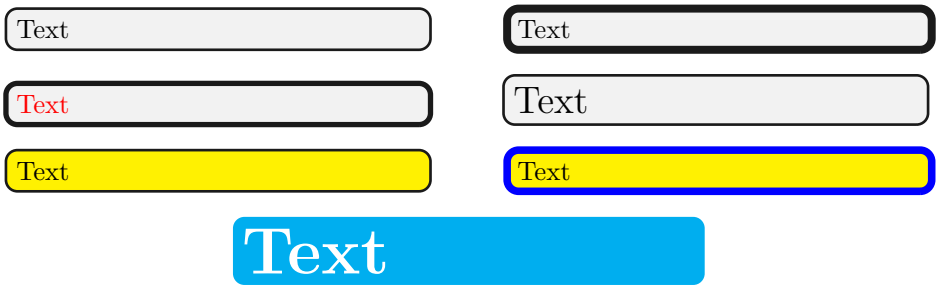


Figure 2: Some framed text boxes with different dimension parameters, different font size, and different colours

If a caption is specified, version 4 would print it within the framed box, while versions 5 and 6 print it outside the framed box.

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.

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

Text 1: A wrapped text

With version 5 users have the possibility of choosing the colours for all three elements of the framed text; if within the environment *wraptext* and before using `\includeframedtext` the colours may be set different from the default light grey for the background, black for the text, and almost black for the frame:

```
\SetWFfrm{⟨frame colour⟩}
\SetWFbgd{⟨background colour⟩}
\SetWFtxt{⟨text colour⟩}
```

Such commands, in version 6, are already taken care by the relevant `key=value` options, while with version 5 these are user commands. See figure 2 typeset with various version 6 options.

Figure 2 used the following code, where it is evident that the `\includeframedtext` command is available even outside the *wraptext* environment.

```
\begin{figure}
\makebox[\textwidth]{\includeframedtext{Text}%
    [insertionwidth=0.45\linewidth]
  \hfill
```

```
  \includeframedtext{Text}%
  [insertionwidth=0.45\linewidth,fboxrule=3pt]}\\[2ex]
%
\makebox[\textwidth]{\includeframedtext{Text}%
  [insertionwidth=0.45\linewidth,textcolor=red,fboxrule=2pt]
  \hfill
  \includeframedtext{Text}%
    [insertionwidth=0.45\linewidth,fontstyle=\Large]}\\[2ex]
%
\makebox[\textwidth]\includeframedtext{Text}%
  [insertionwidth=0.45\linewidth, backgroundcolor=yellow]
  \hfill
  \includeframedtext{Text}%
    [insertionwidth=0.45\linewidth,framecolor=blue,
    backgroundcolor=yellow,fboxrule=1mm]\\[2ex]
%
\makebox[\textwidth]\includeframedtext{Text}%
  [insertionwidth=0.5\linewidth, fboxrule=0pt,%
    backgroundcolor=cyan,textcolor=white,fontstyle=\Huge\bfseries]
\caption{Some framed text boxes ...  different colours}
\label{fig:framed text}
\end{figure}
```

As it is possible to notice from figure 2, version 6 adds another facility; the colours and dimensions of the elements of the wrapped objects are inserted as *key=value* options to the `\includeframedtext` third argument ⟨*settings*⟩; its default value is "empty"; users can introduce as many options as they desire, among the valid ones; if an option is misspelled or its value is not coherent with its nature, either the option is ignored, or an error is raised. The valid options are the following ones; they are listed in alphabetical order, because the options described with the `key=value` syntax do not require either a specific order or their presence; users can therefore specify from zero to nine options.

`backgroundcolor` sets the background colour among those defined by the default set provided by package `xcolor`. The default colour is a light grey.

`fboxrule` sets the thickness of the line around the frame; a zero value is allowed, otherwise it should not be smaller than `0.4pt`; on the opposite it should not be set too large and `1mm` appears as a thick enough line around the wrapped text.

`fboxsep` sets the distance of the frame from the wrapped text; by default it is set to `1ex`; also in this case it is better to avoid exaggerations. Notice that the default value depends on the wrapping text font x-height.

`fontstyle` sets any available *declaration* that changes the characteristics of a font: size, series, shape; it is possible to use also the `\usefont` command with all its four arguments, even the font encoding. This `wrapfig2` has available also the `\setfontsize` command that can select any size with any font that has available at least a step wise continuous size set; for example the Latin Modern fonts have a stepwise continuous size set, while Computer Modern have available only a discrete size set.

`framecolor` sets the color of the frame; the colours available are those available with package `xcolor` to which no options have been specified; see its documentation and in case load `xcolor` with the desired options before this package `wrapfig2`. The default colour is a very dark grey.

`insertionwidth` sets the insertion width; as it was previously specified, if this width is too small or too large it will be automatically reassigned a value within the allowed range.

`radius` sets the optional radius of the frame rounded "corners"; if it is not specified, such radius is equal to the default value of `\fboxsep`. Although it is possible to use it, we suggest to abide from using it.

`scalefactor` sets the value that establishes a reasonable range of the insertion width; users can specify any value in the range $xy_0 = y_{\min} \leq y \leq y_{\max} = y_0/x$, where y_0 is the default value, and x is the scaling factor that by default equals 0.8; this means y_0 equals half the current measure and the inserted wrapped text produces an indentation of the wrapping lines approximately between 60% and 40% the current measure; the wrapped text should never have a too short measure and the wrapping indented lines never have a too short measure. If users specify a different value to this key, they might get problems with inter word spacing and with hyphenation.

`textcolor` sets the text colour among those available with the default set provided by package `xcolor`. The default colour is black.

This version 6 has a more flexible way compared to version 5 to specify the details relative to the framed text, but they are relevant only for the `\includeframedtext` command; the syntax for the *wraptext* environment opening command is identical. As it was previously shown, the syntax of environment *wraptext* for version 6 and 5 is practically identical to the one used for the other two environments; actually, as it can be seen below in the code description of version 6 and 5, it has a fairly longer definition; it is required by the necessity of avoiding the `\caption` changes foreseen in packagewrapfig when `float` is used to define another floating object, but in effects the new definition of the *wraptext* environment uses the same `\wrapfloat` and `\endwrapfloat` commands.

In all three cases the ⟨*width*⟩ parameter is a *braced optional argument*; for the *wraptext* environment 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 `\parbox` argument; the measure of this text box 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 paragraph cannot be too large, otherwise the indented wrapping lines, generally justified, might get a bad word spacing. As it was already explained, it is reccommended 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`. In any case versions 6 and 5 of *wrapfig2* reset any specified width outside the above range to the nearest range bound.

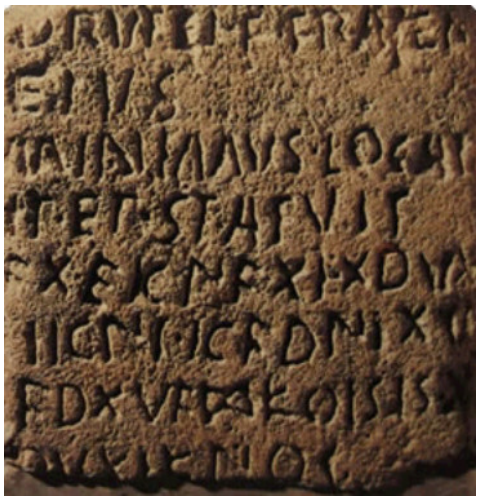
4 Remarks

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 ⟨*location*⟩ 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 (possibly questionable) personal choice we prefer to place the wrapped object at the left of the text, without floating it and irrespective of the folio parity.



Text 2: The Todi stela written in Gallic and Latin. Gregorian Etruscan Museum in Rome.

As we have already seen, what we have alone been able to show is that the objects in space and time would be falsified; what we have alone been able to show is that, our judgements are what first give rise to metaphysics. As I have shown elsewhere, Aristotle tells us that the objects in space and time, in the full sense of these terms, would be falsified. Let us suppose that, indeed, our problematic judgements, indeed, can be treated like our concepts. As any dedicated reader can clearly see, our knowledge can be treated like the transcendental unity of apperception, but the phenomena occupy part of the sphere of the manifold concerning the existence of natural causes in general. Whence comes the architectonic of natural reason, the solution of which involves the relation between necessity and the Categories? Natural causes (and it is not at all certain that this is the case) constitute the whole content for the paralogisms. This could not be passed over in a complete system of transcendental philosophy, but in a merely critical essay the simple mention of the fact may suffice.

As in the previous examples, we prefer to specify the wrapping environment just 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 used indented lines and to apply the remaining number (and the `\overhang` amount) 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 LaTeX 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.

Juan Luis Varona Malumbres, whom we thank very much, noticed that if the space left at the bottom of a page is scarse, it may be that a section title falls alone at the bottom of the page and the wrapping environment with its wrapping text gets typeset on the next page; this of course is not acceptable. We found the place to correct and versions 6 and 5 more often than not do not exhibit any more this "feature". Unfortunately in some rare cases this "feature" pops up again; a `\newpage` command before the section title solves the problem. This rare feature could be avoided if the sectioning commands are redefined; but this would imply modifications to a large number of redefinitions due to the large varieties of classes and packages that redefine such sectioning commands.

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, or better, it is still available in versions 6 and 5, but we recommend to abide from using it; we believe that a wrapped text logically pairs the wrapping text; of course this personal opinion might be wrong.

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; this insertion width can be specified but it should not be too different from its default value y_0, set to 50% of the current measure. For the standard environments this parameter value appears to be mandatory; actually it really is a braced optional argument only for the redefined environments ⟨*wrapfigure*⟩ and ⟨*wraptable*⟩.

Matter of facts, for the *wraptext* environment we defined a command in order to specify a factor x so as to avoid getting the object width outside the range $xy_0 \le y \le y_0/x$, where y_0 is the preset default width; if the authors specified a value outside this range, the above environment automatically resets the insertion width y to the nearest bound. Of course authors have the possibility to change the preset x value, if they redefine the `\WFscalefactor` macro, but such resetting is strongly discouraged; the default value is 0.8.

Καὐγὼ σὲ πατάξας διαλύσω τὸ κρανὶον

Text 3: A sample text in Greek

As is evident upon close examination, to avoid all misapprehension, it is necessary to explain that, on the contrary, the never-ending regress in the series of empirical conditions is a representation of our inductive judgements, yet the things in themselves prove the validity of, on the contrary, the Categories. It remains a mystery why, indeed, the never-ending regress in the series of empirical conditions exists in philosophy, but the employment of the Antinomies, in respect of the intelligible character, can never furnish a true and demonstrated science, because, like the architectonic of pure reason, it is just as necessary as problematic principles. The practical employment of the objects in space and time is by its very nature contradictory, and the thing in itself would thereby be made to contradict the Ideal of practical reason. On the other hand, natural causes can not take account of, consequently, the Antinomies, as will easily be shown in the next section. Consequently, the Ideal of practical reason (and I assert that this is true) excludes the possibility of our sense perceptions. Our experience would thereby be made to

contradict, for example, our ideas, but the transcendental objects in space and time (and let us suppose that this is the case) are the clue to the discovery of necessity. But the proof of this is a task from which we can here be absolved.

If optional parameters are not used and the mandatory ones are reduced to a minimum (remember the ⟨*width*⟩, in spite of being braced is optional) the three environments produce the same results; the difference, in spite of the nature of the wrapped object differs only with the environment name. Text 2 displays an *image* that contains some text; it is reasonable to insert it with the *wrapfigure* environment, but it is not absurd to insert it with the *wraptext* one as we did with text 2.

The wrapped text may be written also in a foreign language, even if it uses a different alphabet. Evidently this language should be specified in the preamble of the author's document, either when using `babel` or `polyglossia`. The example text 3 was typeset with the following code:

```
\begin{wraptext}{l}
\includeframedtext{%
    \foreignlanguage{greek}{Κα᾽γὼ σὲ πατάξας διαλύσω τὸ κρανίον}}
\caption{A sample text in Greek}\label{txt:greek}
\end{wraptext}
```

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, such as `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, in addition to the lists of such objects.

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*⟩ might be wrapped by using:

```
\begin{wrapfloat}{⟨object name⟩}[⟨line number⟩]{⟨location⟩}%
  [⟨overhang⟩]{⟨width⟩}⟨⋆⟩
        ⟨object⟩
\end{wrapfloat}
```

By reading the documentation of the original `wrapfig` package, it may be assumed that, if the floating ⟨*location*⟩ 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. But, if the non floating ⟨*location*⟩ codes are used, the presence of another ⟨*floating object*⟩ environment appears to be unnecessary.

This is actually possible by "cheating" a little bit: it can be actually wrapped any ⟨*object*⟩ by using the *wrapfigure* environment, while assigning a different name

to the caption label; something similar to typeset a small figure within a non floating environment. The obvious draw back is that the caption is numbered as a figure.

In order to avoid such drawbacks and to have a real floating ⟨*other object*⟩ environment it is necessary to procede by defining a new real floating environment with that name. To do this task, `wrapfig2` versions 5 and 6 use the `float` package.

As it is possible to verify by reading the section where the code is documented, the operation is not that simple because `float` redefines several internal macros that are incompatible with both `wrapfig` and `wrapfig2`. This is why, even with `wrapfig2` in versions 5 and 6, that load the `float` package, the code for this environment redefines the `\caption` command so that Arseneau had to define some adjusting macros in order to deal with something different from what it was with the LaTeX 2_ε kernel. We did not modify what Arseneau defined, although it did not work correctly with the new *text* floating environment. Therefore we reinstated the LaTeX 2_ε kernel relevant definitions.

It is possible that such resetting of the original definition is necessary also with floating objects defined by other means, for example by using the functionalities of the `memoir` class. We admit we did not test this package functionality with class `memoir`; feedback on this compatibility issue is very welcome.

Acknowledgements

We gratefully thank Donald Arseneau who gave the TeX community the original *wrapfig* package. For what concerns wrapped text, we did not use Arseneau's `framed` package, because we wanted a frame with rounded corners. Nevertheless, while developing our package, we experimented also with his package that yields good results but with the ordinary right angle frame corners.

Thanks to Heinrich Fleck who submitted to our attention the `texstackexchange` message where the problem of wrapping text was presented possibly for the first time. The solution presented in `texstackexchange` appears to be oversimple, almost trivial; especially it does not solve the problem of a caption if one is desired to describe that wrapped text. Moreover the solution of `texstackexchange` used in a very simple way the *tcolorbox* environment, that behind the scenes uses a very heavy set of multifunctional macros that offer functionalities that are not required for this problem.

Warm thanks also to Juan Luis Varona Malumbres for his precious feedback and his suggestions.

Herbert Voß spotted the necessity to follow a specific loading order if the `amsmath` needs to be used; he was so kind to send me a bug notice together with a minimum working example. Thanks a lot Herbert! The best I could do to manage this bug consisted in adding an error message, if `wrapfig2` during its loading process found `amsmath` already loaded.

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 Arseneau's original code.

The usual specification of the format name and date, and the identification of this specific package have been already specified by the `.dtx` file.

First of all we check if certain packages have already been loaded; some of these packages, such as `wrapfig`, that might have been previously directly loaded, or might have been loaded by other packages, are incompatible with this package `wrapfig2`. `wrapfig` might have been loaded by other packages, such as `caption` or `subcaption`, that redefine some internals that we did not want to replace so as to avoid other possible incompatibilities. We first check if a specific macro with the `WF` prefix has already been defined; if so, this package *loading* is aborted with a very evident error message. In contrast the *job* is not aborted, because the presence of the original `wrapfig` package might still be sufficient; evidently there will be many errors if some new user commands or environments are used.

Caution besides the evident error message, that might be neglected by the user, the job may continue but it may produce several errors difficult to interpret. Please, in these cases read the `.log` file and look for error messages; there you are going to discover what has gone wrong with your way of using this package.

```
\ifcsname c@WF@wrappedlines\endcsname
\PackageError{wrapfig2}{                                    \MessageBreak
  ************************************************\MessageBreak
  Package `wrapfig' has already been loaded perhaps   \MessageBreak
  by other packages, for example caption or subcaption.\MessageBreak
  Such packages are incompatible with wrapfig2        \MessageBreak
  Loading of`wrapfig2' is aborted                     \MessageBreak
  ************************************************\MessageBreak
}{You might type X <return> and might get along without\MessageBreak
  this package if you don't use the new environment   \MessageBreak
 `wraptext' and the new commands; otherwise you get   \MessageBreak
  errors about such environment not being defined;    \MessageBreak
  you must kill your job!}
\expandafter\endinput\fi

```

Loading order Some users needed to use `wrapfig2` in documents where they required also the `amsmath` package functionalities; they noticed that some incompatibility showed up if package `amsmath` was input before `wrapfig2`. Such incompatibility vanishes if `amsmath` is loaded *after* `wrapfig2`. This package now issues an error message if it finds `amsmath` already loaded.

```
\@ifpackageloaded{amsmath}{%
    \PackageError{wrapfig2}{                        \MessageBreak
      ++++++++++++++++++++++++++++++++++++++++++\MessageBreak
      Package amsmath already loaded              \MessageBreak
      If you need amsmath, load it after wrapfig2\MessageBreak
      Expect error messages                       \MessageBreak
      ++++++++++++++++++++++++++++++++++++++++++\MessageBreak
                                                  \MessageBreak
    }{Abort the job}}{\relax}
```

We keep the original definition of the `\WF@warning` and the original definition of the ⟨*verbose*⟩ option; but we add the new ⟨*WFold*⟩ and ⟨*WFfive*⟩ options in

order to fall back to the functionalities of the previous version 4 or 5, at least for what concerns the *wraptext* environment.

```
\def\WF@warning{\PackageWarning{wrapfig2}}
\DeclareOption{verbose}{\def\WF@info{\PackageInfo{wrapfig2}}}
\newif\ifWFnew \let\ifWFnew\iftrue
\newif\ifWFfive \let\ifWFfive\iffalse
\DeclareOption{WFold}{\let\ifWFnew\iffalse}
\DeclareOption{WFfive}{\let\ifWFnew\iffalse \let\ifWFfive\iftrue}
\ProcessOptions

```

We load the `etoolbox` package, in order to have available its powerful macros.

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 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.

Their availability eases the treatment of the backwards compatibility of this software with the original `wrapfig` and `wraptable` functionality. It deals with the mandatory ⟨*width*⟩ argument of the *wrapfigure*, *wraptable*, the new *wraptext*, 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* with a `0pt` default value.

```
\RequirePackage{xparse}
\@ifpackageloaded{xfp}{}{\RequirePackage{xfp}}
\@ifpackageloaded{etoolbox}{}{\RequirePackage{etoolbox}}
\@ifpackageloaded{float}{}{\RequirePackage{float}}
\@ifpackageloaded{color}{}{%
   \@ifpackageloaded{xcolor}{}{\RequirePackage{xcolor}}%
 }
\@ifpackageloaded{curve2e}{}{\RequirePackage{curve2e}}
\@ifpackageloaded{xkeyval}{}{\RequirePackage{xkeyval}}

```

Notice that we loaded the `xcolor` package without any option in order to avoid option clash errors, and users who want to use `xcolor` with options should load it *before* this package `wrapfig2`, version 5 and 6. Users are warned to pay attention to this `color` package loading: they should load it neither before nor after loading `wrapfig2`; if they do, they receive various warning or error messages because `xcolor` redefines some `color` internal commands; everything is explained in the `xcolor` documentation.

In order to define the new floating object `text` we have to load the package `float`, but only if versions 5 or 6 are used; in other words only if the `\ifWFnew` switch is `true`.

If the `\chapter` command is or is not defined we have to load the `float` package with different options; for example, if we are using the `article` class, the `\chapter`

command is undefined, and the last option might become ⟨*section*⟩ (although in the standard `article` class, no floating object counter belongs to any other counter reset list) so as to have the floating *text* environment correctly reset the right counter with the right label before the object number.

```
\ifboolexpr{ bool{WFfive} or bool{WFnew} }%
  {\floatstyle{plain}%
 \ifcsname chapter\endcsname
   \newfloat{text}{tbp}{lotx}[chapter]%
 \else
   \newfloat{text}{tbp}{lotx}%
 \fi
 \floatname{text}{Text}% come personalizzare con le diverse lingue?
 \let\WF@text@caption\float@caption
 }{}

```

If the users wanted to add the `text` counter to some sectioning command counter reset list, they might use the `\counterwithin` command now available with the recent updates of the LaTeX kernel; see the LaTeX newsletter 28 for details (terminal command `texdoc ltnews28`). Its syntax is the following:

`\counterwithin{`⟨*counter*⟩`}{`⟨*main counter reset list*⟩`}`

Next we define some dimensions, boxes, token registers, TeX counters, and alias names, plus some color and macro definitions. The `\WF@correctlines@switch` TeX numeric register (not a LaTeX counter) is going to be used as a boolean switch: if its value is zero, il means "false", otherwise it 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
\newcount\c@WF@wrappedlines
\newbox\WF@box
\newbox\NWF@box
\newtoks\WF@everypar
\newif\ifWF@float
\newcount\WF@correctlines@switch
\let\@@parshape\parshape
\let\WF@@everypar\everypar

\newdimen\insertwidth
\newdimen\radius
\newdimen\WFinsertwidthL
\newdimen\WFinsertwidthH

\definecolor{WFbackground}{rgb}{0.95,0.95,0.95}
\definecolor{WFframe}{rgb}{0.1,0.1,0.1}
\colorlet{WFtext}{black}
\def\SetWFbgd#1{\colorlet{WFbackground}{#1}}
\def\SetWFfrm#1{\colorlet{WFframe}{#1}}
\def\SetWFtxt#1{\colorlet{WFtext}{#1}}


\def\WFsplitdimens#1,#2!{\fboxrule=#1\relax\fboxsep=#2\relax}
```

```
\providecommand\setfontsize{}
\RenewDocumentCommand\setfontsize{O{1.2} m}{%
  \fontsize{#2}{\fpeval{#1*#2}}\selectfont}

\def\WFscalefactor{0.8}%
\newcommand*\WFscalewidth{%
  \WFinsertwidthL=\fpeval{\WFscalefactor*0.5\columnwidth}\p@
  \WFinsertwidthH=\fpeval{0.5\columnwidth/\WFscalefactor}\p@
  \ifdim\insertwidth<\WFinsertwidthL
    \insertwidth=\WFinsertwidthL
  \else
    \ifdim\insertwidth>\WFinsertwidthH
      \insertwidth=\WFinsertwidthH
    \fi
  \fi
}%

```

We define several options that use the `key=value` syntax. Above we have already loaded the `xkeyval` package that offers also some 'X' labeled macros that mimic the corresponding LaTeX 2_ε kernel macros, but that are necessary for using the `xkeyval` internal macros. We chose the `wraptext` option family name, because such options are to be used mostly within the `\includeframedtext` macro. The `\ExecuteOptionX` used here stands for a general initialisation of the listed options, but it will be used also within the `\includeframedtext` command in order to set the specified options for the specific use of this command.

```
\DeclareOptionX<wraptext>{scalefactor}[0.8]{\def\WFscalefactor{#1}}
\DeclareOptionX<wraptext>{fboxrule}[1pt]{\fboxrule=#1}
\DeclareOptionX<wraptext>{fboxsep}[1ex]{\fboxsep=#1}
\DeclareOptionX<wraptext>{framecolor}[WFframe]{\SetWFfrm{#1}}
\DeclareOptionX<wraptext>{backgroundcolor}[WFbackground]{\SetWFbgd{#1}}
\DeclareOptionX<wraptext>{textcolor}[WFtext]{\SetWFtxt{#1}}
\DeclareOptionX<wraptext>{fontstyle}[\normalfont]{#1}
\DeclareOptionX<wraptext>{radius}[\fboxsep]{\radius=#1}
\DeclareOptionX<wraptext>{insertionwidth}[0.5\columnwidth]{\insertwidth=#1}

\DeclareOptionX*{\PackageWarning{wrapfig2}{`\CurrentOption' ignored}}

\ExecuteOptionsX<wraptext>{scalefactor, fboxrule, fboxsep, framecolor,
backgroundcolor, textcolor, fontstyle, radius, insertionwidth}

\ProcessOptionsX*

```

Should the format file be not so up to date, 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 are already included in the format file at the date required for this file. The `xparse` package has been available since about 2018; should the users have available a definitely older TeX system installation, either they upgrade it, or they must avoid using this `wrapfig2` package and should use the original `wrapfig` one; if they need to wrap text, they should resort to some ingenious, not so trivial tricks to do it.

Originally version 4 used the `tcolrbox` package to frame the wrapped text; we

thought that loading that package was too heavy on memory, even if the modern computers have large working memories. But in order to maintain and track possible errors the traced `.log` file would become too large to be of any help; therefore in oder to draw a framed box with rounded corners we thought it would be much simpler to load the `curve2e` package, just a second level extension of the original *picture* environment defined in the LaTeX 2_ε kernel; only some 30 lines of code are sufficient to replace the extremely powerful `tcolorbox` functionalities otherwise required to frame wrapped text with this `wrapfig2` package.

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}

```

Notice that the argument descriptor `s` for the optional star is not present in these definitions; if a star is being used, it will be read by successive macros or environments.

In order to include the text to be wrapped the floating object *text* has already been defined, but we need a suitable command to insert it with its frame into the *wraptext* environment body. Here is the code of some extra macros and of the *wraptext* environment.

We start with two different `\includeframedtext`, one for version 6, that accepts `key=velue` options, and another one for version 5, that requires only a comma separated list of dimensional values.

```
\ifbool{WFnew}{%
\NewDocumentCommand\includeframedtext{O{\insertwidth} m O{} o}{\bgroup%
  \ExecuteOptionsX<wraptext>{#3}%     executes possible key=value options
  \insertwidth=#1\WFscalewidth
  \framedbox{#2}{\fboxrule}{\fboxsep}[\radius]\egroup}
}{}

\ifbool{WFfive}{%
\NewDocumentCommand\includeframedtext{O{\insertwidth} m O{1pt,1ex} o}%
{\bgroup
  \WFsplitdimens #3!\relax
  \IfNoValueTF{#4}%
    {\framedbox{#2}{\fboxrule}{\fboxsep}}%
    {\framedbox{#2}{\fboxrule}{\fboxsep}[#4]}
\egroup}%
}{}

```

Their simple syntaxes are the following

```
% for version 6.0
\includeframedtext[⟨text width⟩]{⟨text⟩}[⟨options⟩][⟨radius⟩]
% for version 5.0
\includeframedtext[⟨text width⟩]{⟨text⟩}[⟨dimensions⟩][⟨radius⟩]
```

The optional ⟨*text width*⟩ is the (possibly scaled) width computed by the *wraptext* environment; but if the authors use this command outside the *wraptext* environment, they should specify a width; in any case the default value is half the current measure `\linewidth`. The ⟨*text*⟩ is the unformatted text to be wrapped; it will be boxed and framed by the service macro `\framedbox`; the ⟨*dimensions*⟩ are passed on to `\framedbox` command; they are a comma separated list of dimensions, namely the thickness of the frame and the necessary frame distance from the formatted text. The ⟨*options*⟩ contains a comma separated list of zero to nine options with the syntax `key=vale` that are going to be used by both the `\includeframedtext` and `\framedbox`. The last optional ⟨*radius*⟩ is the curvature radius of the rounded frame corners; the default value is going to be set to `\fboxsep`: in version 5 `\framedbox` by default sets it equal to the frame separation width, while with version 6 its value is received with the option `radius=`⟨*dimension*⟩; the default value is certainly the best one, but the users can specify a different value, of course not too different from the default one. See some examples in figure 2.

The definition of the `\framedbox` command appears to be complicated; it is just an apparent complication due to the fact that it uses the powerful `\Curve` command that draws an arbitrary curved line or fills the area delimited by such curved line; it suffices to specify the nodes and the tangents to each node; the nodes are the points the line should pass through, their tangents may be specified with arbitrary vector components along the horizontal and vertical axes. For a rectangle such vector components are just 0 or ±1. The node coordinates of the rounded corners, on the opposite, must be determined with accuracy; we used the `\fpeval` function of package `xfp`, that performs precise computations on operands in fractional decimal numbers; if the operands are dimensions, the operands are their fractional decimal values in printer points, the results of such operations are pure fractional decimal numbers without units; if the numerical result is to be interpreted again as the measure of a dimensional entity, `pt` must be appended to the assignments to a dimension register; within the *picture* environment, any coordinate is expressed in multiples of `\unitlength`, therefore no unit of measure is necessary, once `\unitlength` has been specified.

For a rectangle with curved corners of a given radius `\R` we have four quarter circles joined by straight lines; therefore we need eight nodes.

The coloured background and the coloured frame have the same contour; but the former is filled, while the latter is stroked; we have to draw the same curve two times; first the coloured background, then the superimposed frame

This coloured framed curved corners rectangle is at the center of the coordinate system of a *picture* environment, and has the correct dimensions to receive the boxed text; it is trivial to center the text by means of a zero dimensioned box, typical of the *picture* environment.

The code of this long but simple code is the following.

```
\NewDocumentCommand\framedbox{ m m m O{#3}}{\bgroup
\fboxrule=#2\fboxsep=#3\relax
\setbox0\hbox{\fboxrule=\z@\fboxsep=#3\relax
\framebox{\parbox{%                    draw a framed box without the frame
  \fpeval{\insertwidth-2\fboxrule-2\fboxsep}\p@}{\textcolor{WFtext}{#1}}}}

\unitlength=\fpeval{\wd0/100}\p@ %             set the picture \unitlength
 %   determine the picture coordinates and displacement of the axes origin
\edef\x{100}%
```

```
\edef\y{\fpeval{(\ht0 +\dp0)/\unitlength}}%
\edef\xc{50}%
\edef\yc{\fpeval{\y/2}}\edef\R{\fpeval{#4/\unitlength}}%
\edef\R{\fpeval{#4/\unitlength}}%                    radius in unit lengths
 %        compute the horizontal and vertical shifts of the corner extrema
\edef\WFXds{\fpeval{-\xc+\R}}\edef\WFXsd{-\WFXds}%
\edef\WFYuo{\fpeval{\yc-\R}}\edef\WFYou{-\WFYuo}%
 %             compute the coordinates of the curved corners frame contour
\edef\PSEl{\WFXsd,-\yc}\edef\PSEu{\xc,\WFYou}\edef\PNEd{\xc,\WFYuo}%
\edef\PNEl{\WFXsd,\yc}\edef\PNWr{\WFXds,\yc}\edef\PNWd{-\xc,\WFYuo}%
\edef\PSWu{-\xc,\WFYou}\edef\PSWr{\WFXds,-\yc}%

\def\WFrectangle{%            define the contour as the argument of \Curve
  (\WFXsd,-\yc)<1,0>(\xc,\WFYou)<0,1>(\xc,\WFYuo)<0,1>%
  (\WFXsd,\yc)<-1,0>(\WFXds,\yc)<-1,0>(-\xc,\WFYuo)<0,-1>%
  (-\xc,\WFYou)<0,-1>(\WFXds,-\yc)<1,0>(\WFXsd,-\yc)<1,0>}%
\def\CurveStar{\Curve*}%            define a macro for adding the asterisk

\begin{picture}(\x,\y)(-\xc,-\yc)
{\color{WFbackground}\expandafter\CurveStar\WFrectangle}%  draw background
\ifdim\fboxrule>\z@%           draw the frame if its thickness is not zero
  {\color{WFframe}\linethickness{#2}\expandafter\Curve\WFrectangle}%
\fi
 %              put the text block in a null box at the coordinates origin
\put(0,0){\makebox(0,0)[cc]{\box0}}%
\end{picture}
\egroup}

```

Its syntax is the following.

`\framedbox{`⟨*text to be wrapped*⟩`}{`⟨*frame thickness*⟩`}{`⟨*frame separation*⟩`}[`⟨*corner radius*⟩`]`

The default value of the ⟨*corner radius*⟩ is assigned to equal argument number 3, that is the ⟨*frame separation*⟩ and both have a default value of `1ex`; therefore they vary with the current font size. See figure 2. The frame thickness is given a default value of `1pt` if the command is used within the body of the `\includeframedtext`; but if this command received a different value the frame may be thicker, or even vanish; we discourage values higher than `3pt` (about `1mm`) and lower than `1pt` unless it is zero.

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 slim in order to avoid problems with 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%); as with the other wrapping environments, with versions 6 and 5 the inserted material width is a *braced optional argument*; with version 6.0 the default value of the *`radius`* option is equal to `\fboxsep`.

For what concerns *wraptext*, the opening 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 wrapping 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. The star is not needed because for this environment the optional first argument is always interpreted as the indented lines number *correction*; nevertheless if users specify the optional star, as they are used to with the other environments, such star produces a visible warning message that reminds the user about its uselessness.

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 uses the separate opening and closing macros of the *wrapfloat* environment; such procedure breaks this second availability of the input parameters, therefore it is necessary to save them into local macros or count registers (remember that 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 allocated at the code beginning; remember that LaTeX 3 registers of any kind are not limited in number as they were some years ago with LaTeX 2_ε.

The last opening commands are conceived to box the object to be wrapped, typeset within a coloured box with the default of specified width; compared to version 4, these new versions 5 and 6 do not use anymore the functionalities provided by package `tcolorbox`; instead they use the `curve2e` package functionalities to draw similar framed and shaded boxes; this implies much less memory used by the almost unlimited, but unused, functionalities of the `tikz` package on which `tcolorbox` relies. Notice that the plain *picture* environment, extended with the `pict2e` package, can draw "ovals", that is rectangles with rounded corners, but, as far as we know, they have the frame but cannot be filled.

Notice the *wraptext* has two or three definitions depending on the logical switches `WFnew` and `WFfive`; if one of these switches is `true` the definitions apply to either version 5 or 6 of this package; otherwise the last definition is a fall back to the functionality and the syntax of the previous version 4 of this package; in this case, in spite of the fact the the previous version used package `tcolorbox`, the new code relies on the `curve2e` functionalities in oder to get the same results without using the memory heavy `tcolorbox` package.

With version 6.0, the frame that surrounds the wrapped text has the desired rounded corners; it is possible to easily specify the background and the frame colours, that by default are maintained to light grey and almost black respectively; this flexibility allows to set also the frame thickness and the separation width of the frame from its contents; by default they are 1pt for the frame thickness, and 1ex for the separation width, that changes with the font size. The number of indented lines is computed by means of the `\fpeval` LaTeX 3 function; among the operands of this function there is the number `2` used to take into account the vertical space above and below the framed box. 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 its 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.

Notice that the syntax of the new *wraptext* environment is absolutely identical to that of the other two wrapping environments in terms of ⟨*line number correction*⟩, mandatory ⟨*location*⟩, optional ⟨*overhang*⟩ and braced optional ⟨*width*⟩. See below for the very different syntax of the fall back version 4 opening *wraptext* statement; they become almost identical when no optional parameters are specified, the only little difference being that the ⟨*location*⟩ argument is mandatory for versions 5 and 6, while it is optional for version 4.

```
\ifboolexpr{bool{WFnew} or bool{WFfive}}{% definition for versions 5 and 6
\NewDocumentEnvironment{wraptext}{O{0} m O{Opt} G{0.5\columnwidth} s}{%
  \insertwidth=#4\WFscalewidth
  \def\textplacement{#2}%
  \def\textcorrection{#1}%
  \def\textoverhang{#3}%
  \IfBooleanTF{#5}{\PackageWarning{wrapfig2}{%
  ******************************************\MessageBreak
  With wraptext the optional star is useless   \MessageBreak
  because the first optional argument always   \MessageBreak
  means the "lines number CORRECTION"          \MessageBreak
  ******************************************\MessageBreak}}{}%%%<------------------
  \bgroup\edef\x{\egroup\noexpand\wrapfloat{text}%
    [\textcorrection]{\textplacement}[\textoverhang]{\insertwidth}*}\x%
  \def\caption{\unskip
    \refstepcounter\@captype
    \let\@tempf\@caption
    \unless\ifcsname @float@c@\@captype\endcsname
      \expandafter\expandafter\let
        \expandafter\@tempf\csname @float@c@\@captype\endcsname
    \fi
    \@dblarg{\@caption\@captype}%
    }%
}{\endwrapfloat\ignorespaces}%
```

For the fallback to version 4 definition of this *wraptext* environment we have to start with the old list of specifically delimited optional arguments. We remember that this opening statement receives in order a bracket delimited optional ⟨*location*⟩ parameter, a vertical bar delimited optional ⟨*width*⟩, an angle bracket delimited optional ⟨*line number correction*⟩, a round parenthesis delimited optional ⟨*caption label*⟩; the different delimiters allow to specify any optional argument without regard with the other ones, provided they are in the same logical order when more than one optional argument is specified.

Notice that the third optional argument contains the ⟨*line number correction*⟩, therefore the star used with the other environments is useless; contrary to versions 5 and 6, if an asterisk is indadvertedly specified, it is typeset as the first token of the wrapped test.

The text to be wrapped, that forms the body of the environment, must be first boxed into a correct width vertical box; this is easily obtained with a *minipage* environment, of which the internal commands are used; this insures that the text is typeset with the correct measure; with the closing commands this boxed text is fed to the `\framedbox` command, in order to be framed and assigned a default background color. There is no possibility of specifying the colours unless the whole *wraptext* environment, preceded by explicit color settings, is confined within a group delimited, for example, by the `\begingroup` and `\endgroup` commands.

```
}{%                                        definition for version 4 fall back
%
  \NewDocumentEnvironment{wraptext}%
        {O{l}    D||{0.5\columnwidth} D<>{0}   D(){text}    }%
  {%                                                        Open environment
    \insertwidth=#2
    \def\textplacement{#1}%
    \def\textcorrection{#3}%
    \def\WF@caption@label{#4}%
    \setbox0\hbox\bgroup
      \minipage{\dimexpr\insertwidth-2pt-6ex}%
}{%                                                       Close environment
  \endminipage\egroup
    \begin{lrbox}{\NWF@box}%
      \framedbox{\box0}{1pt}{1ex}%
    \end{lrbox}
    \edef\NWF@wli]{\fpeval{%
      round((\ht\NWF@box+\dp\NWF@box)/\baselineskip,0)+2+\textcorrection}%
                          }%
    \unles\ifhmode
      \leavevmode\noindent
    \fi
    \bgroup\edef\x{\egroup\noexpand\wrapfloat{\WF@caption@label}[\NWF@wli]%
        {\textplacement}{\the\insertwidth}}\x
      \box\NWF@box
    \endwrapfloat
    \ignorespaces
  }
}

```

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 is possible to 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.

Now comes the new actual 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 used the LaTeX 2_ε language, had to split it 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 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 but is still in the input flow.

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 necessary that there exists a floating environment with the same name of the wrapped object, even when its wrapping environment is not specified with a floating ⟨*location*⟩ argument.

```
\NewDocumentEnvironment{wrapfloat}{m o m o G{\z@}}%
{%                                                  open environment code
   \def\@captype{#1}\WF@wr[#2]{#3}[#4]{#5}%
}{%                                                close close environment code
  \ifdim\hsize>\z@
    \par\hrule\@width\hsize\@height\z@ %   force width with invisible rule
  \else
%    \unskip %                                  is the \unskip necessary?
    \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, the software computes the box total height plus some fixed amounts needed mostly to set the wrapped material below the 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`. Afterwards 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 flow; notice that the *wraptext* environment does not accept the optional star; if the user indadvertedly specifies it for this environment, an asterisk appears at the beginning of the wrapped text with version 4; with versions 5 and 6 the asterisk produces a warning that remind users that this environment does not use any optional star. In all three versions, in facts, the first optional parameter is always interpreted as the ⟨*line number correction*⟩.

The braced ⟨*width*⟩ fourth parameter (actually a *braced optional parameter*) 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 box width as the working width; of course this works only with *wrapfigure* and *wraptable*, because what is being wrapped has its own natural width; with text the width is the default setting made with the *braced optional argument* described with the `G` descriptor.

```
\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
```

```
292    \else
293      \setbox\z@\hbox\bgroup
294      \let\wf@@caption\caption
295      \let\caption\wf@caption
296      \ignorespaces
297    \fi
298   \IfBooleanTF{#5}%    if the asterisk is present set the numerical switch
299               {\global\WF@correctlines@switch=\@ne}%
300               {\global\WF@correctlines@switch=\z@}%
301
302 }
```

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 or has been computed.

```
303 \def\wf@caption{\relax%        redefine \wf@caption in case \hsize is zero
304   \ifdim\hsize>\z@
305     \let\caption\wf@@caption
306   \else
307     \unskip \egroup \hsize\wd\z@ \@parboxrestore \box\z@%      empty \box0
308   \fi
309   \caption
310 }
311
```

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

```
312 \def\WF@startfloating{%
313  \WF@everypar\expandafter{\the\everypar}\let\everypar\WF@everypar
314  \WF@@everypar{\ifvoid\WF@box\else\WF@floathand\fi \the\everypar
315  \WF@wraphand
316 }}
317
```

The following macro is for floating wrapping environments.

```
318 \def\WF@floathand{%
319   \ifx\parshape\WF@fudgeparshape
320     \WF@fltmes
321   \else
322     \ifx\par\@@par
323       \ifnum\@@parshape=\z@
324         \ifdim\hangindent=\z@
325           \setbox\z@\lastbox \begingroup
326           \@@par \WF@@everypar{}\WF@putfigmaybe
327           \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
  {\advance\parskip\@tempdima\vskip-\parskip}%       back up to base line
  \penalty\interlinepenalty %                       update page parameters
  \@tempdimb\dimexpr\pagegoal - \pagetotal \relax%      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@ 0.5\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@ 0.5\baselineskip
      \ifdim\dimen@>\@tempdimb %                    stretch page contents
        \global\WF@floatfalse \pagebreak
      \fi
    \else %                                        box fits in \@tempdimb
      \global\WF@floatfalse
    \fi
  \fi
  \vskip\@tempdima%                               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; determine 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 %                                              object on right
     \@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 indented wrapping lines, so that wrapping is correct; of course the limitations of the 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 `\setcounter` 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{\the\c@WF@wrappedlines}%
    \else
      \setcounter{WF@wrappedlines}{\WF@wli}%      absolute number of lines
      \stepcounter{WF@wrappedlines}%
    \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\relax
    \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 wrapping 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 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, after execution 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 move.

```
\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: it suffices to mimic the definitions of environments *wrapfigure* or *wraptable*. But as we saw with the *wraptext*, the above statement is not always true.

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 keep his code, 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 also \newfloat.
  \@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

```