A LaTeX Package for changing the page grid and MVL *

Arthur Ogawa (mailto:ogawa@teleport.com), 1.0rc5

November 12, 2004

This file embodies the `ltxgrid` package, the implementation and its user documentation.

The distribution point for this work is ftp://ftp.teleport.com/users/ogawa/macros/latex/contrib/supported/ltxgrid..., which contains fully unpacked, prebuilt runtime files and documentation.

The `ltxgrid` package was commissioned by the American Physical Society and is distributed under the terms of the LaTeX Project Public License, the same license under which all the portions of LaTeX itself is distributed. Please see http://ctan.tug.org/macros/latex/base/lppl.txt for details.

To use this document class, you must have a working TeX installation equipped with LaTeX 2_ε and possibly pdftex and Adobe Acrobat Reader or equivalent.

To install, retrieve the distribution, unpack it into a directory on the target computer, into a location in your filesystem where it will be found by LaTeX; in a TDS-compliant installation this would be: `texmf/tex/macros/latex/ao/.`

To use, read the user documentation `ltxgrid.pdf`.

Contents

*This file has version number 1.0rc5, last revised 2001/07/26. For version number and date, search on "1.0rc5" in the .dtx file, or see the end of the 00readme.txt file.

1 Processing Instructions

The package file `ltxgrid.sty` is generated from this file, `ltxgrid.dtx`, using the DOCSTRIP facility of LaTeXvia `tex ltxgrid.ins`. The typeset documentation that you are now reading is generated from the same file by typesetting it with LaTeX or pdftex via `latex ltxgrid.dtx` or `pdflatex ltxgrid.dtx`.

1.1 Build Instructions

You may bootstrap this suite of files solely from `ltxgrid.dtx`. Prepare by installing LaTeX 2_ε (and either tex or pdftex) on your computer, then carry out the following steps:

1. Within an otherwise empty directory, typeset `ltxgrid.dtx` with LaTeX or pdflatex; you will obtain the typeset documentation you are now reading, along with the installer `ltxgrid.ins`, and the file `00readme.txt`.

   Note: you will have to run LaTeX twice, then `makeindex`, then LaTeX again in order to obtain a valid index and table of contents.

2. Now typeset `ltxgrid.ins`, thereby generating the package file `ltxgrid.sty`.

3. Install `ltxgrid.sty` by moving it to a location in your filesystem where they will be found by LaTeX.

1.2 Bill of Materials

Following is a list of the files in this distribution arranged according to provenance.

1.2.1 Primary Source

One single file generates all.

```
%ltxgrid.dtx
%
```

1.2.2 Generated by `latex ltxgrid.dtx`

Typesetting the source file under LaTeX generates the readme and the installer.

```
%00readme.txt   ltxgrid.ins
%
```

1.2.3 Generated by `tex ltxgrid.ins`

Typesetting the installer generates the package files.

```
%ltxgrid.sty
%
```

1.2.4 Documentation

The following are the online documentation:

```
%ltxgrid.pdf
%
```

1.2.5 Auxiliary

The following are auxiliary files generated in the course of running LaTeX:

```
%ltxgrid.aux ltxgrid.idx ltxgrid.ind ltxgrid.log ltxgrid.toc
%
```

2 Code common to all modules

The following may look a bit klootchy, but we want to require only one place in this file where the version number is stated, and we also want to ensure that the version number is embedded into every generated file.

Now we declare that these files can only be used with LaTeX 2_ε. An appropriate message is displayed if a different TeX format is used.

```
%<*doc|ltxgrid>
\NeedsTeXFormat{LaTeX2e}[1995/12/01]%
%</doc|ltxgrid>
```

As desired, the following modules all take common version information:

```
%<ltxgrid>\ProvidesFile{ltxgrid.sty}%
%<*doc>
\expandafter\ProvidesFile\expandafter{\jobname.dtx}%
%</doc>
```

The following line contains, for once and for all, the version and date information. By various means, this information is reproduced consistently in all generated files and in the typeset documentation.

```
%<*doc|ltxgrid>
 [2001/07/26 1.0rc5 page grid package]% \fileversion
%</doc|ltxgrid>
```

3 The driver module `doc`

This module, consisting of the present section, typesets the programmer's documentation, generating the `.ins` installer and `00readme.txt` as required.

Because the only uncommented-out lines of code at the beginning of this file constitute the `doc` module itself, we can simply typeset the `.dtx` file directly, and there is thus rarely any need to generate the "doc" DOCSTRIP module. Module delimiters are nonetheless required so that this code does not find its way into the other modules.

The `\end{document}` command concludes the typesetting run.

```
%<*doc>
```

3.1 The Preamble

The programmers documentation is formatted with the `ltxdoc` class with local customizations, and with the usual code line indexing.

```
\documentclass{ltxdoc}
\RequirePackage{ltxdocext}%
\RequirePackage[colorlinks=true,linkcolor=blue]{hyperref}%
\ifx\package@font\@undefined\else
```

```
\expandafter\expandafter
\expandafter\RequirePackage
\expandafter\expandafter
\expandafter{%
              \csname package@font\endcsname
            }%
\fi
\CodelineIndex\EnableCrossrefs
```

3.1.1 Docstrip and info directives

We use so many DOCSTRIP modules that we set the `StandardModuleDepth` counter to 1.

```
\setcounter{StandardModuleDepth}{1}
```

The following command retrieves the date and version information from this file.

```
\expandafter\GetFileInfo\expandafter{\jobname.dtx}%
```

3.2 The installer file

The installer `ltxgrid.ins` appears here. If you have retrieved the standard distribution of this package, the installer file is already on your filesystem. If you are bootstrapping, the first typesetting of the `.dtx` file will cause the installer to be generated.

The following modules are used to direct DOCSTRIP in generating the external files:

Module	File	Description
doc	`ltxgrid.drv`	driver for programmer's documentation
ltxgrid,ltxgrid-krn	`ltxgrid.sty`	this package
ltxgrid-krn		the portion of this package suitable for inclusion within another p

```
\begin{filecontents}{ltxgrid.ins}
%% This file will generate documentation and runtime files
%% from ltxgrid.dtx when run through LaTeX or TeX.
\input docstrip
\preamble

This is a generated file;
altering it directly is inadvisable;
instead, modify the original source file.
See the URL in the file 00readme.txt.

Copyright notice.

   These files are distributed
   WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

\endpreamble
\keepsilent
  \generate{%
   \file{ltxgrid.drv}{\from{ltxgrid.dtx}{doc}}%
```

```
47    \file{ltxgrid.sty}{%
48     \from{ltxgrid.dtx}{ltxgrid,ltxgrid-krn}%
49    }%
50   }%
51 \ifToplevel{
52 \Msg{*************************************************************}
53 \Msg{*}
54 \Msg{* To finish the installation, please move}
55 \Msg{*    ltxgrid.sty}
56 \Msg{* into a directory searched by TeX;}
57 \Msg{* in a TDS-compliant installation:}
58 \Msg{* texmf/tex/macros/latex/ao/.}
59 \Msg{*}
60 \Msg{* To produce the documentation,
61 \Msg{*       run ltxgrid.dtx through LaTeX.}
62 \Msg{*}
63 \Msg{* Happy TeXing}
64 \Msg{*************************************************************}
65 }
66 \endbatchfile
67 \end{filecontents}
```

Note that, because all of the files generated by the installer are part of the standard distribution, it will be necessary to run the installer only when bootstrapping (or, of course, during development). Note, too, that it is rare to generate the `doc` module because it suffices to simply typeset the `.dtx` file itself.

3.3 The "Read Me" File

As promised above, here is the contents of the "Read Me" file. That file serves a double purpose, since it also constitutes the beginining of the programmer's documentation. What better thing, after all, to have appear at the beginning of the typeset documentation?

A good discussion of how to write a ReadMe file can be found in Engst, Tonya, "Writing a ReadMe File? Read This" *MacTech* October 1998, p. 58.

Note the appearance of the `\StopEventually` command, which marks the dividing line between the user documentation and the programmer documentation.

The usual user will not be asked to do a full build, not to speak of the bootstrap. Instructions for carrying these processes begin the programmer's manual.

```
68 \begin{filecontents*}{00readme.txt}
69 \title{%
70  A \LaTeX\ Package for changing the page grid and MVL%
71  \thanks{%
72   This file has version number \fileversion,
73   last revised \filedate.%
74   % For version number and date,
75   % search on "\fileversion" in the .dtx file,
76   % or see the end of the 00readme.txt file.
77  }%
78 }%
79
80 \author{%
81 Arthur Ogawa (\texttt{mailto:ogawa@teleport.com}),
```

```
\fileversion\\Copyright (C) 1999, 2000 Arthur Ogawa
}%
\maketitle

This file embodies the \classname{ltxgrid} package,
the implementation and its user documentation.

The distribution point for this work is
\url{ftp://ftp.teleport.com/users/ogawa/macros/latex/contrib/supported/ltxgrid...},
which contains fully unpacked, prebuilt runtime files and documentation.

The \classname{ltxgrid} package was commissioned by the American Physical Society
and is distributed under the terms of the \LaTeX\ Project Public License,
the same license under which all the portions of \LaTeX\ itself is distributed.
Please see \url{http://ctan.tug.org/macros/latex/base/lppl.txt} for details.

To use this document class, you must have a working
\TeX\ installation equipped with \LaTeXe\
and possibly pdftex and Adobe Acrobat Reader or equivalent.

To install, retrieve the distribution,
unpack it into a directory on the target computer,
into a location in your filesystem where it will be found by \LaTeX;
in a TDS-compliant installation this would be:
\file{texmf/tex/macros/latex/ao/.}

To use, read the user documentation \file{ltxgrid.pdf}.

\tableofcontents

\section{Processing Instructions}

The package file \file{ltxgrid.sty}
is generated from this file, \file{ltxgrid.dtx},
using the {\sc docstrip} facility of \LaTeX
via |tex ltxgrid.ins|.
The typeset documentation that you are now reading is generated from
the same file by typesetting it with \LaTeX\ or pdftex
via |latex ltxgrid.dtx| or |pdflatex ltxgrid.dtx|.

\subsection{Build Instructions}

You may bootstrap this suite of files solely from \file{ltxgrid.dtx}.
Prepare by installing \LaTeXe\ (and either tex or pdftex) on your computer,
then carry out the following steps:
\begin{enumerate}
\item
Within an otherwise empty directory,
typeset \file{ltxgrid.dtx} with \LaTeX\ or pdflatex;
you will obtain the typeset documentation you are now reading,
along with
the installer \file{ltxgrid.ins},
and the file \file{00readme.txt}.

```

```
Note: you will have to run \LaTeX\ twice, then \file{makeindex}, then
\LaTeX\ again in order to obtain a valid index and table of contents.
\item
Now typeset \file{ltxgrid.ins},
thereby generating the package file \file{ltxgrid.sty}.
\item
Install \classname{ltxgrid.sty}
by moving it to a location
in your filesystem where they will be found by \LaTeX.
\end{enumerate}
\end{filecontents*}
```

3.4 The Document Body

Here is the document body, containing only a `\DocInput` directive—referring to this very file. This very cute self-reference is a common `ltxdoc` idiom.

```
\begin{document}%
\expandafter\DocInput\expandafter{\jobname.dtx}%
% ^^A\PrintChanges
\end{document}
%</doc>
```

4 Using this package

Once this package is installed on your filesystem, you can employ it in adding functionality to LaTeX by invoking it in your document or document class.

4.1 Invoking the package

In your document, you can simply call it up in your preamble:

```
%\documentclass{book}%
%\usepackage{ltxgrid}%
%\begin{document}
%⟨your document here⟩
%\end{document}
%
```

However, the preferred way is to invoke this package from within your customized document class:

```
%\NeedsTeXFormat{LaTeX2e}[1995/12/01]%
%\ProvidesClass{myclass}%
%\LoadClass{book}%
%\RequirePackage{ltxgrid}%
%⟨class customization commands⟩
%\endinput
%
```

Note that this package requires the features of the `ltxutil` package, available at ftp://ftp.teleport.com/users/ogawa/macros/latex/contrib/supported/ltxutil/.

Once loaded, the package gives you acccess to certain procedures, usually to be invoked by a LaTeX command or environment, but not at the document level.

4.2 Changing the page grid

This package provides two procedures, `\onecolumngrid`, `\twocolumngrid`, that change the page grid (it can be extended to more columns and to other page grids).

They differ from standard LaTeX's `\onecolumn` and `\twocolumn` commands in that they do not force a page break. Also, upon leaving a multiple-column grid, the columns are balanced. In other respects they work same.

They differ from the grid-changing commands of Frank Mittelbach's `multicol` package in that they allow floats of all types (single- and double column floats, that is) and preserve compatability with the `longtable` package.

These commands must be issued in vertical mode (conceivably via a `\vadjust`) such that they are ultimately present in the MVL, where they can do their work. Because they do not work in LaTeX's left-right mode, they are unsuitable at the document level. Furthermore, packaging a grid command in a `\vadjust`, although possible, will probably not acheive satisfactory page layout.

Page grid commands are not intended to be issued unnecessarily: only the first of two successive `\onecolumngrid` commands is effective; the second will be silently ignored.

`\onecolumngrid` You command LaTeX to return to the one-column grid with the `\onecolumngrid` command. If you are already in the one-column grid, this is a no-op. The one-column grid is considered special of all page grids, in that no portion of the page is held back (in `\pagesofar`); all items that might go on the current page (with the exception of floats and footnotes) are on the MVL.

`\twocolumngrid` You command LaTeX to return to the two-column grid with the `\twocolumngrid` command. If you are already in the two-column grid, this is a no-op.

These two commands should be issued by a macro procedure that can ensure that TeX is in outer vertical mode.

4.3 Changing the MVL

This package also provides commands to modify the main vertical list (MVL) in a safe way. The scheme here is to structure, insofar possible, TeX's MVL as follows:

box or boxes
penalty
glue

This should be a familiar sequence. It is the prototype sequence for a vertical list, and is followed when TeX breaks paragraphs into lines, and when TeX generates a display math equation.

If you (as a macro programmer) wish to modify the value of the penalty or glue item, you can use one of the MVL-altering commands to do so. Certain operations are implemented here; you can make up your own.

Note that these commands must be issued in vertical mode, perhaps via a `\vadjust` or a `\noalign`. They can work directly if you are in inner mode (say within a parbox or a minipage).

`\removestuff` You instruct LaTeX to remove both the penalty and the glue item with this command.

`\addstuff` You issue the `\addstuff{`⟨*penalty*⟩`}{`⟨*glue*⟩`}` command to add a penalty, glue, or both. If you do not wish to add one or the other, the corresponding argument should be nil. Note that the effect of `\addstuff` is to stack the penalties and glue

items. Therefore, the lesser of the two penalties takes effect, and the two glue items add together.

`\addstuff` is limited because once applied, it cannot be applied again with correct results.

`\replacestuff` The `\replacestuff` command is syntactically the same as `\addstuff`, but works differently: the existing penalty and glue are replaced or modified.

The specified penalty is not inserted if the existing penalty is greater than 10000 (that is, in case of a `\nobreak`), otherwise, the lower (non-zero) of the two penalties is inserted.

If the specified glue has a larger natural component than the existing glue, we replace the glue. However, if the specified glue's natural component is negative, then the existing glue's natural component is changed by that amount.

`\replacestuff` can be applied mutiple times bceause it retains the list structure in the canonical form.

Note that we treat two penalties specially (as does TEX): a penalty of 10000 is considered a garbage value, to be replaced if found. This is the signal value that TEX inserts on the MVL replacing the penalty that caused the page break (if the page break occurred at a penalty). Also, a penalty of zero is indistinguishable from no penalty at all, so it will always be replaced by the given value.

Therefore, it is highly recommended to never set any of TEX's penalty parameters to zero (a value of, say, 1, is practically the same), nor should a skip parameter be set to zero (instead, use, say, 1sp). Also, to prevent a pagebreak, do not use a penalty of 10000, use, say 10001 instead.

You can define your own construct that modifies the MVL: Define a command, say, `\myadjust`, as follows:

```
%\def\myadjust#1{\noexpand\do@main@vlist{\noexpand\@myadjust{#1}}\@tempa}%
%
```

that is, `\myadjust` invokes `\do@main@vlist`, passing it the procedure name `\@myadjust` along with the arguments thereof pre-expanded. Next, define the procedure `\@myadjust`:

```
%\def\@myadjust#1{⟨meddle with the MVL⟩}%
%
```

when `\@myadjust` executes, you will be in the output routine (in inner vertical mode) and the MVL will be that very vertical list.

5 Compatability with LATEX's Required Packages

Certain packages, usually ones written by members of the LATEX Project itself, have been designated "required" and are distributed as part of standard LATEX. These packages have been placed in a priviledged position vis á vis the LATEX kernel in that they override the definitions of certain kernel macros.

Compatability between `ltxgrid` and these packages is complicated by a number of factors. First is that `ltxgrid` alters the meaning of some of the same kernel macros as certain of the "required" packages. Second is that fact that certain of the "required" packages of LATEX are incompatible with each other.

Examples of the first kind are the `ftnright`, `multicol`, and `longtable` packages. The `ltxgrid` package is not compatible with `multicol`, but if you are using

ltxgrid, you do not need to use ftnright or multicol anyway. The ltxgrid package does however attempt to be compatible with longtable.

Among the "required" packages that are mutually incompatible are multicol and longtable, the incompatibility arising because both packages replace LaTeX's output routine: if one package is active, the other must not be so. This state of affairs has remained essentially unchanged since the introduction of the two as LaTeX2.09 packages in the late 1980s.

The reason that ltxgrid can remain compatible with longtable is due to the introduction of a more modern architecture, the "output routine dispatcher", which allows all macro packages access to the safe processing environment of the output routine, on an equal footing. The relevant portions of the longtable package are reimplemented in ltxgrid to take advantage of this mechanism.

Timing is critical: the ltxgrid package will be incompatible with any package that redefines any of the kernel macros that ltxgrid patches—if that package is loaded *after* ltxgrid.

Hereinafter follows some notes on specific LaTeX packages.

5.1 ftnright

Frank Mittelbach's ftnright package effects a change to LaTeX's \twocolumn mode such that footnotes are set at the bottom of the right-hand column instead of at the foot of each of the two columns.

Note that it overwrites three LaTeX kernel macros: \@outputdblcol, \@startcolumn, and \@makecol. Fortunately none of the three are patched by ltxgrid, so that compatability is not excluded on this basis.

At the same time, it changes the meaning of \footnotesize, the macro that is automatically invoked when setting a document's footnote into type. One might well argue that it is an error for the meaning of \footnotesize to be determined by a package such as ftnright, that indeed such a choice should be made in the document class, or in a file such as bk10.clo.

To avoid being tripped up by this misfeature in ftnright, it is only necessary to reassert our meaning for \footnotesize later on, after ftnright has been loaded.

Note that ftnright inserts code that demands that LaTeX's flag \if@twocolumn is true, that is, it will complain if deployed in a \onecolumn document. It is therefore necessary for any other multicolumn package to assert that flag in order to avoid this package's complaint. It is an interesting question exactly why this package has this limitation. After all, a one-column page grid is just a degenerate case of the two column.

5.2 longtable

David Carlisle's longtable package sets tables that can be so long as to break over pages. According to its author, it uses the same override of LaTeX's output routine as Frank Mittelbach's multicol package. By implication, then, it has a hard incompatability with the latter.

The longtable package also performs a check of whether the document is in \twocolumn mode, and declines to work if this is the case. It is not clear, however, that there is any true incompatability present if so. It's just that David did not see

any reason anyone would want to set such long tables in a multicolumn document, hence the check.

There does not appear to be any indication that `longtable` would work less well under `ltxgrid` than under standard LaTeX's `\twocolumn` mode. Therefore, this `ltxgrid` patches `longtable` (if loaded) so as to provide compatability. In the course of which, `longtable` becomes more robust (`longtable` has mumerous bugs and incompatabilities of long standing, some of which are repaired by `ltxgrid`).

One problem remains, namely that, if a `longtable` environment breaks over columns and thereby inserts its special headers and footers at that break, and those columms are then balanced (due to a return to the one-column page grid), then those inserted rows will remain, and may no longer fall at the column break. This will, of course look wrong.

The only way to fix this problem is to avoid doing column balancing in the way I have implemented here; such an enhancement to this package is possible.

5.3 multicol

Frank Mittelbach's `multicol` package provides a page grid with many columns, albeit denies the placement of floats in individual columns.

It esablishes its own `\output` routine, which is the reason it runs afoul of the `longtable` package. On the other hand, `ltxgrid` specifically allows for the case where a package installs its own `\output` routine, so there is no incompatability on that basis.

Still, it is pointless to use `multicol` if you are using `ltxgrid`, since both packages provide multicolumn page layouts. Therefore, `multicol` is not supported by `ltxgrid`.

5.4 ltxgrid

It has been pointed out that one of the disadvantages of adopting the `ltxgrid` package is that it does alter the LaTeX kernel. Any package that itself alters the LaTeX kernel may be incompatible with `ltxgrid`, and new packages (destined perhaps to become part of the successor to LaTeX 2_ε) may break `ltxgrid`.

The consequence is that packages introduced in future, and future changes to LaTeX may be incompatible with `ltxgrid`. This is, of course, true. The development plan for `ltxgrid` is that when such packages and LaTeX kernel changes come about, the burden will be on `ltxgrid` to change in a way that provides for continued compatability with those packages and LaTeX kernel changes.

6 How `ltxgrid` places footnotes

In conventional multicolumn layouts, a footnote will appear at the bottom of the column in which it is called out. The `ltxgrid` package implements this conventional layout choice by default. However, other choices are possible (a la `ftnright`, whose compatability with `ltxgrid` has not been tested).

One unusual feature of `ltxgrid`'s default implementation must be mentioned, though, namely the case in a two-column page grid, where a footnote is followed by a temporary change to the one-column page grid (e.g., for a wide equation). In such a case, the material above the wide material is split into two columns, and

a footnote whose callout appears in the right-hand column will nonetheless be set at the base of the left column.

This arrangement was chosen because it ensures that the footnotes at the bottom of any page will appear in numerical order. It can be argued that this choice is "incorrect", but be that as it may, the `ltxgrid` package does not foreclose on other arrangements for the footnotes. The package can be adapted to accomodate any page design desired.

7 Limitations in `ltxgrid`'s default column balancing method

In a multicolumn page grid, when encountering a page that is not completely full, it is customary to set the material in balanced columns (typically with the last column no longer than any of the others). Such a case also crops up when temporarily interrupting the multicolumn grid to set material on the full width of the page: the material on the page above the break is customarily set in balanced columns.

An awkward case arises when we have already set one or more complete columns of type before encountering the need to balance columns. In this subset of cases, the default in `ltxgrid` is to do an operation I call "re-balancing": the material on the page so far is pasted back together into a single column, and new, balanced column breaks are calculated.

This scheme typically works fine, but it has a significant vulnerability: any discardable items trimmed at the original column break is lost, never to be retrieved. Consequently, after re-balancing, an element like, say, a section head can fail to have the correct amount of whitespace above.

This problem is due to an unfortunate optimization in TEX, wherein a certain class of nodes is trimmed from the top of main vertical list upon returning from the output routine: any penalty, glue, or leader node falls in to this class of discardable nodes, and trimming proceeds until a non-discardable node (such as a box, or rule) is encountered. It gets better: a third class of nodes is transparent to this trimming process; they are neither discarded nor do they halt the process of trimming: mark nodes and all whatsits fall into this class of transparent nodes; they are quietly passed over during trimming.

An alternative approach for TEX to take would have been, rather than discarding the node entirely, to simply mark it as discarded. (Implementors of NTS, please note!) Then, upon shipping out, such nodes would not make it into the DVI. TEX's optimization, driven by the small computer architectures current when it was developed, does save mem, but at the cost of revisiting page breaks in a reliable way.

FIXME: how to fix a column break in the above case? Widetext?

8 Implementation of package

Special acknowledgment: this package uses concepts pioneered and first realized by William Baxter (mailto:web@superscript.com) in his SuperScript line of commercial typesetting tools, and which are used here with his permission. His thorough understanding of TEX's output routine underpins the entire `ltxgrid` package.

8.1 Beginning of the ltxgrid DOCSTRIP module

Requires the underpinnings of the ltxkrnext package.

```
%<*ltxgrid>
\def\package@name{ltxgrid}%
\expandafter\PackageInfo\expandafter{\package@name}{%
 Page grid for \protect\LaTeXe,
 by A. Ogawa (ogawa@teleport.com)%
}%
\RequirePackage{ltxutil}%
%</ltxgrid>
```

8.2 Banner

Credit where due.

```
%<*ltxgrid-krn>
\typeout{%
 ltxgrid: portions licensed from W. E. Baxter (web@superscript.com)%
}%
```

8.3 Sundry

Here are assorted macro definitions.

\lineloop The document-level command \lineloop sets numbered lines until the specified count is reached. This command is mainly used to construct test documents.

```
\newcounter{linecount}
\def\lineloop#1{%
 \loop
 \ifnum\c@linecount<#1\relax
  \global\advance\c@linecount\@ne
  \par
  \hb@xt@\hsize{%
   \ifnum\c@linecount<100 0\fi\ifnum\c@linecount<10 0\fi\number\c@linecount
   \vrule depth2.5\p@
   \leaders\hrule\hfil
  }%
  \penalty\interlinepenalty
 \repeat
}%
```

8.4 Mark Components

Override LaTeX's mark macros to allow more components.

We remain bound by the weakness of LaTeX's scheme in that one cannot emulate the action of TEX whereby material with marks can be inserted in the middle of a vertical list such that the marks are reliably calculated. If we did that, \@themark would no longer be utilized.

A more robust scheme involves placing all marks (component and value) into a list (using global scoping, i.e., \gdef), and using \@@markto place an index on that list into the MVL. Then, e.g., \@@botmarksignifies the place where that list is to be cut, and the \botmark of any component is the value of the last element

of the cut list having the given component. The \firstmark and \topmark can likewise be defined relative to \@@firstmark and \@@topmark, except in the latter case, we want the first following the cut instead of the last preceding the cut.

The limitation of this scheme is its demands upon TEX's mem. The list of marks would need to be trimmed back to, effectively, \topmark at the beginning of every page.

This approach is not yet part of the extended LaTeX kernel.

\@@mark
\@@topmark
\@@firstmark
\@@botmark
\@@splitfirstmark
\@@splitbotmark

Remember primitives under a new set of names.

```
\let\@@mark\mark
\let\@@topmark\topmark
\let\@@firstmark\firstmark
\let\@@botmark\botmark
\let\@@splitfirstmark\splitfirstmark
\let\@@splitbotmark\splitbotmark
```

8.4.1 Procedures that expose the component data structure

This portion of the code exposes the internal representation of the mark components. If we wish to add more components, we will have to revise these macro definitions: \@themark, \nul@mark, \set@mark@netw@, \set@marktw@, \set@markthr@@, \get@mark@@ne, \get@mark@tw@, \get@mark@thr@@, \get@mark@f@ur.

\@themark

FIXME: is it safer to eliminate \@themark in favor of a message that evaluates \@@botmark?

Note: these definitions expose the data structure of mark components.

```
\def\@themark{{}{}{}{}}%
\def\nul@mark{{}{}{}{}\@@nul}%
```

\set@mark@netw@
\set@marktw@
\set@markthr@@

These procedures insert the new value of a particular mark component into the given argument. They expose the data structure of mark components.

```
\def\set@mark@netw@#1#2#3#4#5#6#7{\gdef#1{{#6}{#7}{#4}{#5}}\do@mark}%
\def\set@marktw@#1#2#3#4#5#6{\gdef#1{{#2}{#6}{#4}{#5}}\do@mark}%
\def\set@markthr@@#1#2#3#4#5#6{\gdef#1{{#2}{#3}{#6}{#5}}\do@mark}%
```

\get@mark@@ne
\get@mark@tw@
\get@mark@thr@@
\get@mark@f@ur

These procedures retreive the value of a particular mark component. They expose the data structure of mark components.

```
\def\get@mark@@ne#1#2#3#4#5\@@nul{#1}%
\def\get@mark@tw@#1#2#3#4#5\@@nul{#2}%
\def\get@mark@thr@@#1#2#3#4#5\@@nul{#3}%
\def\get@mark@f@ur#1#2#3#4#5\@@nul{#4}%
```

8.4.2 Procedures that do not expose the component data structure

\mark@netw@
\marktw@
\markthr@@

These procedures insert the new value of a particular mark component into \@themark, then execute \do@mark. They constitute the implementation layer for mark components one, two, and three. An analogous procedure for component four could be defined; call it \markf@ur.

```
\def\mark@netw@{\expandafter\set@mark@netw@\expandafter\@themark\@themark}%
\def\marktw@{\expandafter\set@marktw@\expandafter\@themark\@themark}%
\def\markthr@@{\expandafter\set@markthr@@\expandafter\@themark\@themark}%
```

\do@mark
\do@@mark

Access procedures \mark(AKA \@@mark). The \do@mark procedure is used when a mark is being put down into the MVL; \do@@mark when this happens in the output routine.

```
\def\do@mark{\do@@mark\@themark\nobreak@mark}%
\def\do@@mark#1{%
 \begingroup
  \let@mark
  \@@mark{#1}%
 \endgroup
}%
```

\let@mark
\nobreak@mark

The procedure that makes \csnames robust within a mark. Use \appdef and \robust@ to extend the list.

```
\def\let@mark{%
 \let\protect\@unexpandable@protect
 \let\label\relax
 \let\index\relax
 \let\glossary\relax
}%
\def\nobreak@mark{%
 \@if@sw\if@nobreak\fi{\@ifvmode{\nobreak}{}}{}%
}%
```

8.4.3 Using mark components

These procedures use the component mark mechanism to implement a mark component that remembers the current environment (used in page makeup) and the the two mark components left over from the original LaTeX. The fourth component is presently unused.

\mark@envir

The third mark component's access procedures. The \mark@envir and \bot@envir commands are a good model of how to write access procedures for a new mark component.

```
\def\mark@envir{\markthr@@}%
\def\bot@envir{%
 \expandafter\expandafter
 \expandafter\get@mark@thr@@
 \expandafter\@@botmark
              \nul@mark
}%
```

\markboth
\markright
\leftmark
\rightmark

Set procedures for legacy components.

```
\def\markboth{\mark@netw@}%
\def\markright{\marktw@}%
```

Retrieval procedures for legacy mark components. The procedure for retrieving the first component from \botmark and the second component from \firstmarkhave names in LaTeX; they are called, respectively, \leftmark and \rightmark.

It is possible to retrieve the components of \topmark as well: use \saved@@topmark.

```
\def\leftmark{%
 \expandafter\expandafter
 \expandafter\get@mark@@ne
```

```
 \expandafter\saved@@botmark
              \nul@mark
}%
\def\rightmark{%
 \expandafter\expandafter
 \expandafter\get@mark@tw@
 \expandafter\saved@@firstmark
              \nul@mark
}%
```

8.5 Output Super-routine

We want to change LaTeX's output routine, but do not wish to remain vulnerable to interference from such "required" packages as `multicol` (authored by Frank Mittelbach) and `longtable` (authored by David P. Carlisle), which swap in their own output routines when the respective package is active.

The better mechanism, used here, is due to William Baxter (web@superscript.com), who has allowed his several ideas to be used in this package.

In what follows, we effectively wrap up the old LaTeX output routine inside a new, more flexible "super routine". When the output routine is called, the "super routine" acts as a dispatcher. If the old routine is needed, it is called.

If a package attempts to substitute in their own output routine, they will effectively be modifying a token register by the name of `\output`. The primitive `\output` is now known by a different name, which should no longer be necessary to use.

Usage note: to make a visit to the output routine employing the dispatcher, enter with a value of `\outputpenalty` that corresponds to a macro. Defining as follows:

```
%\@namedef{output@10000}{⟨your code here⟩}%
%
```

by convention, your output routine should void out `\box\@cclv`.

In rewriting LaTeX's output dispatcher in a much simpler form, we also avoid the sin of multiple `\shipout`s within a single visit to the output routine.

Conceptually, we divide visits to the output routine into two classes. The first involves natural page breaks (at a `\newpage` or when `\pagetotal`> `\pagegoal`) and usually resulting in `\box\@cclv` either being shipped out or salted away (e.g., each column in a multicolumn layout). We might call this class the "natural output routines"; the `\outputpenalty` will never be less than −10000. Furthermore, we ensure that `\holdinginserts` is cleared when calling such routines.

The other class involves a forced visit to the output routine via a large negative penalty (< −10000). They do not generally result in a `\shipout` of `\box\@cclv`: they may be dead cycles. We provide a mechanism (call it a "one-off" output routine) that allows us to specify certain processing to be done when TeX reaches the current position on the page.

One-off output routines themselves fall into two divisions, ones that process `\box\@cclv`, and ones that work on the main vertical list (MVL). The former are typified by changes to the page grid, perhaps even column balancing. The latter involve the insertion of penalties or glue and the processing of floats.

The natural output routine is a single procedure. We have not introduced multiple natural output routines based on the `\outputpenalty` because TEX does not support such a thing: TEX sometimes lays down a penalty whose value is the sum of other penalties. Because of this, we cannot depend on the value of `\outputpenalty` in such areas.

We do introduce flexibility in the form of a mechanism for patching into the natural output routine. Three hooks are offered, allowing a procedure to prepare for the upcoming visit to the output routine, access to `\box\@cclv`, and after shipping out (or otherwise committing the material to the page).

Environments, commands, and even packages can install their own procedures into these hooks. For instance, if the longtable package is loaded, it will install its procedures, but those procedures will punt if the page break being processed does not actually fall within a longtable environment.

`\primitive@output` Here we remember the TEX primitive `\output` and its value, and then proceed to take over the `\csname` of `\output`, making it a `\toks` register and installing the old value of the output routine.

```
\let\primitive@output\output
```

`\output` Grab the tokens in `\the\output` (but without the extra set of braces). The value of `\toks@` must remain untouched until loaded into the appropriate token register; this is done a few lines below.

```
\long\def\@tempa#1\@@nil{#1}%
\toks@
\expandafter\expandafter
\expandafter{%
\expandafter \@tempa
             \the\output
             \@@nil
             }%
\newtoks\output
\output\expandafter{\the\toks@}%
```

`\dispatch@output` We now install our own output routine in place of the old one, which is still available as `\the\output`.

The output routine is simply the procedure `\dispatch@output`. It either dispatches to a procedure based on a particular value of `\outputpenalty` or it executes `\the\output` tokens.

```
\primitive@output{\dispatch@output}%
\def\dispatch@output{%
 \let\par\@@par
 \expandafter\let\expandafter\@tempa\csname output@\the\outputpenalty\endcsname
 \outputdebug@sw{%
  \saythe\badness
  \saythe\outputpenalty
  \saythe\holdinginserts
  \say\thepagegrid
  \saythe\pagegrid@col
  \saythe\pagegrid@cur
 %\say\bot@envir
  \saythe\insertpenalties
 %\say\@@topmark
```

```
%\say\saved@@topmark
%\say\@@firstmark
%\say\saved@@firstmark
 \say\@@botmark
%\say\saved@@botmark
 \saythe\pagegoal
 \saythe\pagetotal
 \saythe{\badness\@cclv}%
 \expandafter\@ifx\expandafter{\csname output@-\the\execute@message@pen\endcsname\@tempa}{%
  \say\@message@saved
 }{%
  \expandafter\say\csname output@\the\outputpenalty\endcsname
 }%
 \say\@toplist
 \say\@botlist
 \say\@dbltoplist
 \say\@deferlist
 {\tracingall\scrollmode
  \showbox\@cclv
  \showbox\@cclv@saved
  \showbox\pagesofar
  \showbox\footbox
  \showbox\footins@saved
  \showbox\footins
  \showlists
 }%
}{}%
\@ifnotrelax\@tempa{\@tempa}{\the\output}%
}%
\@ifxundefined{\outputdebug@sw}{%
 \@booleanfalse\outputdebug@sw
}{}%
```

8.6 Further thoughts about inserts

The only safe way to deal with inserts is to either set `\holdininserts` or to commit to using whatever insert comes your way: you cannot change your mind once you see a non-void `\box\footins`, say.

Therefore all output routine processing must proceed with `\holdinginserts` set until you are sure of the material to be committed to the page. At that point, you can clear `\holdinginserts`, spew `\box\@cclv`, put down the appropriate penalty, and exit, with the knowledge that TEX will re-find the same pagebreak, this time visiting the output routine with everything, including inserts, in their proper place. This technique applies to split elements (screens, longtable, index) as well as to manufactured pages (float pages and clearpage pages).

Therefore, the output routine must not make assumptions about whether `\holdinginserts` should be cleared; instead this must be left to the one-off output routines or the natural output routine.

If we are manufacturing pages ("float page processing"), and if `\pagegoal` is not equal to `\vsize`, then inserts are at hand, and our criterion should take into account the insert material, even though we cannot measure its height based on the size of `\box\footins` (because `\holdinginserts` is set, you see).

It would be better to take the complement of \floatpagefraction and use that as a standard for the looseness of the page. Since \pagegoal reflects the inserted material, the criterion becomes the difference of the aggregate height of the floats and the \pagegoal versus this "page looseness" standard.

As a check, consider what happens if we bail out: \@deferlist has never been touched, so it requires no attention. Also, \holdinginserts has never been cleared, so inserts require no attention. So we only have to ensure that marks are preserved, which is already taken care of by the message handler mechanism.

If we are doing ordinary page cutting, then the scheme would be to detect whether we are within a screen (or longtable as may be), do the adjustment to the page height, and return, but this time with \holdinginserts cleared. Upon reentering the output routine, we may or may not be within the screen environment, but we are now sure to have a final page break, and we can commit this material (by shipping out or by saving it out as a full column).

In the above, the first of the two visits to the output routine is a dead cycle and requires propagation of marks, but nothing else.

The natural output routine

Here is the portion of the output routine that fields cases not handled by the dispatcher.

The default is to ship out a page and then look around for more material that might constitute a "float page". However, because \holdinginserts is normally set, this output routine must first have a dead cycle and come back again with \holdinginserts cleared. Then, after shipping out, it puts down a message that will manufacture zero or more float pages, finally terminating with a procedure that commits floats to a new unfinished page.

To accomodate special processing, we execute hooks whose name is based on the value of the "envir" mark component. The default is "document", ensured by an initial mark of that value; the associated procedures are all nil. Any unknown envir value will "\relax out".

The code \move@insert@sw tells whether we are on our first visit to the output routine (with \holdinginserts still set), or our second (with \holdinginserts cleared). The output routine will toggle the setting.

The commands \hold@insertions and \move@insertions respectively clear and set the state of \move@insert@sw, so this procedure effectively clears \holdinginserts just long enough to pick up the insertions. Important: any output routine that clears \holdinginserts must guarentee that it is restored on the subsequent visit to the output routine. Or, to put it another way, if an output routine detects that \holdinginserts is cleared, it should take it upon itself to restore it before exiting.

The branch with \holdinginserts set is executed first; the other branch follows on practically immediately thereafter. In the first branch, we simply execute the appropriate hook and then execute a dead cycle.

In the branch with \holdinginserts cleared, the procedure builds up the current column, which is now complete, with \@makecol, then dispatches to the shipout routine associated with the current page grid, \output@column@. At the end, it triggers the execution of an output routine to prepare the next column (or page).

8.7 Natural output routine

\output Here is what has become of the output routine of LaTeX. It is of necessity divided into phases, \output@holding is executed upon first encountering the natural page-breaking point, while inserts are being held. The second phase, \output@moving, is set in motion by the first: here the same material (in most cases) will be processed with \holdinginserts cleared.

```
\output={\toggle@insert\output@holding\output@moving}%
```

The procedure \output@holding is our first cycle through the output routine; \holdinginserts is still set. We give the current environment a heads up (it is through this means that longtable sets its running header and footer), then we execute a dead cycle, which should propagate marks.

One corner case that can crop up is the presence of a single unbreakable chunk whose size is larger than \vsize. Doing a dead cycle under such circumstances will not find the same breakpoint as this time (remember we threw in a \mark node). Instead, we attempt to remove the excess height of the material, so we can continue to propagate marks.

The corner case is at hand if the natural size of \box\@cclv exceeds \pagegoal and the contents cannot be shrunk to fit.

```
\def\output@holding{%
\csname output@init@\bot@envir\endcsname
%\vbadness\@M
%\vfuzz\maxdimen
\@if@exceed@pagegoal{\unvcopy\@cclv}{%
  \setbox\z@\vbox{\unvcopy\@cclv}%
\outputdebug@sw{{\tracingall\scrollmode\showbox\z@}}{}%
\dimen@\ht\@cclv\advance\dimen@-\ht\z@
\dead@cycle@repair\dimen@
}{%
\dead@cycle
}%
}%
\def\@if@exceed@pagegoal#1{%
 \begingroup
  \setbox\z@\vbox{#1}%
 \dimen@\ht\z@\advance\dimen@\dp\z@
  \outputdebug@sw{\saythe\dimen@}{}%
  \@ifdim{\dimen@>\pagegoal}{%
  \setbox\z@\vbox{\@@mark{}\unvbox\z@}%
   \splittopskip\topskip
   \splitmaxdepth\maxdepth
   \vbadness\@M
   \vfuzz\maxdimen
   \setbox\tw@\vsplit\z@ to\pagegoal
  \outputdebug@sw{{\tracingall\scrollmode\showbox\tw@\showbox\z@}}{}%
  \setbox\tw@\vbox{\unvbox\tw@}%
  \@ifdim{\ht\tw@=\z@}{%
   \ltxgrid@info{Found overly large chunk while preparing to move insertions. Attempting repai
    \aftergroup\true@sw
  }{%
    \aftergroup\false@sw
  }%
```

```
 }{%
 \aftergroup\false@sw
 }%
\endgroup
}%
```

The procedure `\output@moving` is our second cycle through the output routine; `\holdinginserts` is now cleared, and `\insert`s will have been split off into their respective box registers, like `\footins`.

1. Set the values of `\topmark` and `\firstmark`.
2. If we got here because of a `\clearpage` command, remove the protection box that this mechanism has left on the MVL.
3. If the contents of `\box\@cclv` are non-trivial, commit it to the current page or ship it out as the case may call for.
4. If not, discard it (we are at the end of `\clearpage` processing).
5. Set various values, including the available space for setting type on the next column (`\@colroom`).

The processing for a non-trivial `\box\@cclv` are:

1. Execute the head procedure for the current environment.
2. Make up a column and ship it out (or commit it to the current page) via a procedure keyed to the current page grid.
3. Put down an interrupt for `\do@startcolumn@pen`: this will force a visit to the output routine for the purpose of committing floats to the next column.
4. Possibly put down an interrupt to continue `\clearpage` proccessing.
5. Execute the tail procedure for the current environment.

The processing for a trivial `\box\@cclv` are:

1. Void out `\box\@cclv` and give appropriate warning messages and diagnostics.
2. Put down the same interrupts as for the non-trivial case above.

```
\def\output@moving{%
 \set@top@firstmark
 \@ifnum{\outputpenalty=\do@newpage@pen}{%
  \setbox\@cclv\vbox{%
   \unvbox\@cclv
   \setbox\z@\lastbox
   \@ifdim{\ht\z@=\ht\@protection@box}{\box\lastbox}{\unskip}%
  }%
 }{}%
 \@cclv@nontrivial@sw{%
  \csname output@prep@\bot@envir \endcsname
  \@makecol\csname output@column@\thepagegrid\endcsname
  \protect@penalty\do@startcolumn@pen
```

```
  \clearpage@sw{%
   \protect@penalty\do@endpage@pen
  }{}%
  \csname output@post@\bot@envir \endcsname
 }{%
  {\setbox\z@\box\@cclv}%
 }%
 \set@colroom
 \global\@mparbottom\z@
 \global\@textfloatsheight\z@ %FIXME: this legacy LaTeX variable is set, but never queried!
}%
```

The procedure `\@cclv@nontrivial@sw` determines if this visit to `\output@moving` is a trivial one, which happens at the end of `\clearpage` processing and under some pathological circumstances. It emits a Boolean, so it is syntactically like `\true@sw`, albeit does not execute solely via expansion.

Note: the case where `\box\@cclv` is void comes up at the very beginning of the job, when typesetting a (full-page-width) title block in a two-column layout.

Note: the code that removes the last box and skip from the output is intended to detect the case where the output has whatit nodes followed by topskip and a protection box. This is what happens under normal circumstances at the end of `\clearpage` processing.

```
\def\@cclv@nontrivial@sw{%
\@ifx@empty\@toplist{%
\@ifx@empty\@botlist{%
\@ifvoid\footins{%
 \@ifvoid\@cclv{%
  \false@sw
 }{%
\setbox\z@\vbox{\unvcopy\@cclv}%
\@ifdim{\ht\z@=\topskip}{%
\setbox\z@\vbox{%
\unvbox\z@
\setbox\z@\lastbox\dimen@\lastskip\unskip
\@ifdim{\ht\z@=\ht\@protection@box}{%
\advance\dimen@\ht\z@
\@ifdim{\dimen@=\topskip}{%
\aftergroup\true@sw
}{%
\aftergroup\false@sw
}%
}{%
\aftergroup\false@sw
}%
}%
{%
\false@sw % Normal for \clearpage
}{%
\true@sw
}%
}{%
\@ifdim{\ht\z@=\z@}{%
\ltxgrid@info{Found trivial column. Discarding it}%
\outputdebug@sw{{\tracingall\scrollmode\showbox\@cclv}}{}%
```

```
\false@sw
}{%
\true@sw
}%
}%
 }%
}{%
\true@sw
}%
}{%
\true@sw
}%
}{%
\true@sw
}%
}%
```

`\protect@penalty` The procedure `\protect@penalty` is the utility procedure for invoking a one-off output routine. Such a routine can expect to find the protection box above it in `\box\@cclv`: it should remove that box.

Note that `\execute@message` does the same thing as `\protect@penalty`, but in a slightly different way.

We create a specially formulated box that will be universally used when a protection box is needed. In this way, we can always recognize when `\box\@cclv` is trivial: it will consist of whatsits followed by `\topskip` glue and the `\@protection@box`.

```
\def\protect@penalty#1{\protection@box\penalty-#1\relax}%
\newbox\@protection@box
\setbox\@protection@box\vbox to1986sp{\vfil}%
\def\protection@box{\nointerlineskip\copy\@protection@box}%
```

`\dead@cycle` `\dead@cycle@repair` The procedure `\dead@cycle` is defined separately as a utility which can be used by any output processing routine to emulate what takes place in the standard output routine.

Here, we have entered the output routine with `\holdinginserts` enabled, which means that we are not yet ready to ship out material, because the `\insert` registers are being held. We want to clear `\holdinginserts` and come back here with the same page break as before, whereupon we may properly proceed with page makeup.

To do this, we propagate marks, then spew the contents of `\box\@cclv` followed by the original output penalty that landed us here (but only if it is not 10000, the flag value for a pagebreak not at a penalty).

However, the natural output routine should do this only if `\box\@cclv` is nontrivial. A pathological case exists wherein a box of height greater than `\textheight` would cause an infinite loop involving the output routine. The procedure `\dead@cycle@repair`, attempts to catch this case and avoid the loop.

The test of the height of `\box\@cclv` is not the correct one, because this test will run afoul in the case where `\box\@cclv` contains nothing but an `\insert` node. What to do?

It is possible that the pathological case can be detected by looking at `\pagetotal`. If that quantity is zero, then `\box\@cclv` really is trivial.

In the procedure `\dead@cycle@repair`, if `\box\@cclv` is nontrivial, we execute `\dead@cycle`, otherwise it contains nothing but a mark, so we dispense with propagating marks and we simply spew out `\box\@cclv` without an accompanying mark. This has the effect of failing to propagate marks, but this problem is preferrable to the infinite loop, which in principle could crash even a robust operating system by filling up the file system.

If a document has such a large chunk, it should be fixed, so we give a message in the log.

You ask, "In what way does this infinite loop come about?" Good question!

The setup is a chunk in the MVL that is taller than `\textheight`. (Yes, it's that simple.) As soon as the previous page ships out, the MVL will contain a mark (propagated from the previous page) followed by that large chunk (call it the 'big bad box', albeit does not need to be a single box). The next visit to the output routine will be a natural page break, but TEX will select the juncture between the mark and the big bad box as the least-cost page break. Unless the test in `\dead@cycle` is done, the cycle is perpetuated when the macro reinserts the mark.

The crux matter is achieving, in a robust way, the goal of going from a `\holdinginserts` state to one where the insertions are moving.

```
\def\dead@cycle@repair#1{%
\expandafter\do@@mark
\expandafter{%
\@@botmark
}%
\unvbox\@cclv
\nointerlineskip
\vbox to#1{\vss}%
\@ifnum{\outputpenalty<\@M}{\penalty\outputpenalty}{}%
}%
\def\dead@cycle@repair@protected#1{%
\expandafter\do@@mark
\expandafter{%
\@@botmark
}%
\begingroup
  \unvbox\@cclv
  \setbox\z@\lastbox % Remove protection box
 \nointerlineskip
  \advance#1-\ht\@protection@box
 \vbox to#1{\vss}%
  \protection@box % Reinsert protection box
 \@ifnum{\outputpenalty<\@M}{\penalty\outputpenalty}{}%
\endgroup
}%
\def\dead@cycle{%
 \expandafter\do@@mark
 \expandafter{%
               \@@botmark
             }%
 \unvbox\@cclv
 \@ifnum{\outputpenalty<\@M}{\penalty\outputpenalty}{}%
}%
```

\output@init@document
\output@prep@document
\output@post@document

The default processing simply provides for insertion of held-over footnotes. At a natural page break, we are either at the bottom of a column or at the bottom of a page. In either case, the `\output@init@` processing adjusts for the height of the held-over footnotes and bails out. Upon our return, at `\output@prep@` time, the page break will accomodate the material; it is now actually inserted by concatenating it with the contents of `\footins`. The default processing for `\output@post@` is nil.

```
\def\output@init@document{%
 \@ifvoid\footbox{}{%
  \global\advance\vsize-\ht\footbox
  \global\advance\vsize-\dp\footbox
 }%
}%
\def\output@prep@document{%
 \@ifvoid\footbox{}{%
% {\tracingall\scrollmode\showbox\footbox\showbox\footins}%
  \setbox\footins\vbox{\unvbox\footbox\unvbox\footins}%
 }%
}%
\def\output@post@document{}%
```

\@opcol

The standard LaTeX procedure `\@opcol` is now completely obsoleted.

```
\let\@opcol\@undefined
```

\@makecol

The procedure `\@makecol` packages up a page along with all its insertions and floats. Therefore it is essential that it be executed with `\holdininserts` cleared.

Note that there is a corner case when in a multi-column grid, where the change back to one-column grid occurs just after a complete page ships out. We want to detect when `\@cclv` contains nothing but a `\mark`, but this is a TeX impossibility.

Note on `\@kludgeins`: we have removed this mechanism from LaTeX, because the implementation of `\enlargethispage` no longer requires it. Here, for consistency sake, we remove `\@makespecialcolbox`.

```
\def\@makecol{%
 \setbox\@outputbox\vbox{%
  \boxmaxdepth\@maxdepth
  \@tempdima\dp\@cclv
  \unvbox\@cclv
  \vskip-\@tempdima
 }%
 \xdef\@freelist{\@freelist\@midlist}\global\let\@midlist\@empty
 \@combinefloats
 \@combineinserts\@outputbox\footins
%\@ifvbox\@kludgeins{%
% \@makespecialcolbox
%}{%
  \set@adj@colht\dimen@
  \count@\vbadness
  \vbadness\@M
  \setbox\@outputbox\vbox to\dimen@{%
   \@texttop
   \dimen@\dp\@outputbox
   \unvbox\@outputbox
```

```
    \vskip-\dimen@
    \@textbottom
   }%
   \vbadness\count@
%}%
 \global\maxdepth\@maxdepth
}%
\let\@makespecialcolbox\@undefined
```

`\@combineinserts` We split out the procedure to add the `\footins` insertions to the packaged-up page. Any other non-trivial insertions should also be dealt with at this time.

```
\def\@combineinserts#1#2{%
 \setbox#1\vbox{%
  \unvbox#1%
% {\tracingall\scrollmode\showbox#2}%
  \vbox{%
   \@ifvoid#2{}{%
    \vskip\skip\footins
    \color@begingroup
    \normalcolor
    \footnoterule
    \nointerlineskip
    \box#2%
    \color@endgroup
   }{}%
  }%
 }%
}%
```

`\@floatplacement` In standard LaTeX, someone (DPC?) makes the assumption that `\@fpmin` can be assigned locally. This is no longer true now that we ship no more than one page per visit to the output routine. We apply a bandaid.

```
\appdef\@floatplacement{%
 \global\@fpmin\@fpmin
}%
```

`\pagebreak@pen` While we are in the way of registering certain penalty values, let us register the smallest one that will force a visit to the output routine. However, this penalty will not have an assciated macro: we wish to execute the natural output routine instead.

Note that this penalty is invoked by `\clearpage` and `\newpage`.

```
\mathchardef\pagebreak@pen=\@M
\expandafter\let\csname output@-\the\pagebreak@pen\endcsname\relax
```

8.8 Float placement

`\do@startcolumn@pen` The procedure `\do@startcolumn@pen` is executed as a one-off output routine just after a page is shipped out (or, in a multicolumn page grid, a column is salted away).

Its job is to either generate a "float page" (in reality a column) for shipping out, or to commit deferred floats to the fresh column, concluding with a dead cycle. In the former case, we accomodate split footnotes and other insertions (by comparing

\vsize and \pagegoal): the floats are spewed onto the page, whereupon LaTeX's output routine will place the footnotes and ship out, iterating the process once again.

Note that when this procedure is invoked, \box\@cclv still has within it the protection box, so we start by removing it. Note also that if there was a split insertion held over from the previous page, the insert node will be present in \box\@cclv, *prior to* the protection box. For this reason, we cannot just throw away that box, as we might be tempted to do.

FIXME: where else do we possibly inappropriately discard \box\@cclv?

Note that, because a column or page page had previously just been completed, we can assume that there is nothing of importance on the page, and because no message is being passed, we can preserve marks in a simple way.

A Note on terminology: In a single-column page grid, you might expect that we would execute the procedure \do@startpage. But this is not so. LaTeX has a confustion of long standing, in which the procedures that handle full-page width floats in a two-column page grid all have in their names the string 'dbl', which erroneously suggests having something to do with "double". It does not: when you see 'dbl', think "full page width".

```
\mathchardef\do@startcolumn@pen=10005
\@namedef{output@-\the\do@startcolumn@pen}{\do@startcolumn}%
\def\do@startcolumn{%
 \setbox\@cclv\vbox{\unvbox\@cclv\setbox\z@\lastbox\unskip}%
 \clearpage@sw{\@clearfloatplacement}{\@floatplacement}%
 \set@colroom
 \@booleanfalse\pfloat@avail@sw
 \begingroup
  \@colht\@colroom
  \@booleanfalse\float@avail@sw
  \@tryfcolumn\test@colfloat
  \float@avail@sw{\aftergroup\@booleantrue\aftergroup\pfloat@avail@sw}{}%
 \endgroup
 \fcolmade@sw{%
  \setbox\@cclv\vbox{\unvbox\@outputbox\unvbox\@cclv}%
% \csname float@column@\thepagegrid\endcsname
% \csname output@column@\thepagegrid\endcsname
  \outputpenalty-\pagebreak@pen % ask for a return visit, this time with insertions and all.
  \dead@cycle
 }{%
  \begingroup
   \let\@elt\@scolelt
   \let\reserved@b\@deferlist\global\let\@deferlist\@empty\reserved@b
  \endgroup
  \clearpage@sw{%
   \outputpenalty\@M
  }{%
   \outputpenalty\do@newpage@pen
  }%
  \dead@cycle
 }%
 \check@deferlist@stuck\do@startcolumn
 \set@vsize
}%
```

```
\def\@scolelt#1{\def\@currbox{#1}\@addtonextcol}%
\def\test@colfloat#1{%
 \csname @floatselect@sw@\thepagegrid\endcsname#1{}{\@testtrue}%
 \@if@sw\if@test\fi{}{\aftergroup\@booleantrue\aftergroup\float@avail@sw}%
}%
```

\@addtonextcol

We must adjust \@addtonextcol to take held-over inserts into account. Now that all deferred floats are queued up together (in order), we must have a way of differentiating them; this is done by the page grid-dependent procedure \@floatselect@sw@.

```
\def\@addtonextcol{%
 \begingroup
  \@insertfalse
  \@setfloattypecounts
  \csname @floatselect@sw@\thepagegrid\endcsname\@currbox{%
   \@ifnum{\@fpstype=8 }{}{%
     \@ifnum{\@fpstype=24 }{}{%
       \@flsettextmin
       \@reqcolroom \ht\@currbox
       \advance \@reqcolroom \@textmin
       \advance \@reqcolroom \vsize % take into account split insertions
       \advance \@reqcolroom -\pagegoal
       \@ifdim{\@colroom>\@reqcolroom}{%
         \@flsetnum \@colnum
         \@ifnum{\@colnum>\z@}{%
            \@bitor\@currtype\@deferlist
            \@if@sw\if@test\fi{}{%
              \@addtotoporbot
            }%
         }{}%
       }{}%
     }%
   }%
  }{}%
  \@if@sw\if@insert\fi{}{%
    \@cons\@deferlist\@currbox
  }%
 \endgroup
}%
```

\do@startpage@pen
\forcefloats@sw
\@sdblcolelt
\test@dblfloat
\@if@notdblfloat

Similar to \do@startcolumn, the procedure \do@startpage starts up a new page (not column) in a multi-column page grid. It is invoked after a page is shipped out in a multi-column page grid, and it commits full-page-width floats to the fresh page, possibly resulting in a float page. In implementation, it is similar to \do@startcolumn, except that it commits effectively via \@addtodblcol instead of \@addtonextcol. Note that this procedure will inevitably be followed by \do@startcolumn.

Some details of the procedure:

We begin by removing the protection box from \box\@cclv, then setting the values of the float placement parameters appropriately, and resetting \@colht, \@colroom, and \vsize to base values.

Next we attempt to compose a float page, a page consisting entirely of floats. If successful, we ship out the float page and lay down an interrupt that will send

us back here for another try.

If no float page is formed, we attempt to commit full-page-width floats to the text page, and return with a dead cycle. We are now ready to compose columns of text.

Note that all floats (both column floats and full-page-width floats) move through a single queue. To differentiate between the two, the width of the float is compared to `\textwidth`. This comparison is encapsulated in the macro `\@if@notdblfloat`, which should be used whenever such a determination must be made. This procedure returns a Boolean.

```
\mathchardef\do@startpage@pen=10006
\@namedef{output@-\the\do@startpage@pen}{\do@startpage}%
\def\do@startpage{%
 \setbox\@cclv\vbox{\unvbox\@cclv\setbox\z@\lastbox\unskip}%
 \clearpage@sw{\@clearfloatplacement}{\@dblfloatplacement}%
 \set@colht
 \@booleanfalse\pfloat@avail@sw
 \begingroup
  \@booleanfalse\float@avail@sw
  \@tryfcolumn\test@dblfloat
  \float@avail@sw{\aftergroup\@booleantrue\aftergroup\pfloat@avail@sw}{}%
 \endgroup
 \fcolmade@sw{%
  \global\setbox\pagesofar\vbox{\unvbox\pagesofar\unvbox\@outputbox}%
  \@combinepage
  \@combinedblfloats
  \@outputpage
  \global\pagegrid@cur\@ne
  \protect@penalty\do@startpage@pen
 }{%
  \begingroup
   \@booleanfalse\float@avail@sw
   \let\@elt\@sdblcolelt
   \let\reserved@b\@deferlist\global\let\@deferlist\@empty\reserved@b
  \endgroup
  \@ifdim{\@colht=\textheight}{% No luck...
   \pfloat@avail@sw{% ...but a float *was* available!
    \forcefloats@sw{%
     \ltxgrid@warn{Forced dequeueing of floats stalled}%
    }{%
     \ltxgrid@warn{Dequeueing of floats stalled}%
    }%
   }{}%
  }{}%
  \outputpenalty\@M
  \dead@cycle
 }%
 \check@deferlist@stuck\do@startpage
 \set@colht
%\set@colroom
}%
\def\@sdblcolelt#1{\def\@currbox{#1}\@addtodblcol}%
\def\test@dblfloat#1{%
 \@if@notdblfloat{#1}{\@testtrue}{}%
```

```
 \@if@sw\if@test\fi{}{\aftergroup\@booleantrue\aftergroup\float@avail@sw}%
}%
\def\@if@notdblfloat#1{\@ifdim{\wd#1<\textwidth}}%
\@booleanfalse\forcefloats@sw
```

`\@addtodblcol` The procedure `\@addtodblcol` is called into play at the beginning of each fresh page and operates on each deferred float, in the hopes of placing one or more such floats at the top of the current page.

We alter the procedure of standard LaTeX by putting failed floats into `\@deferlist` instead of `\@dbldeferlist`. Having done so, we must have a means of differentiating full-page-width floats from column-width floats. We assume that the latter will always be narrower than `\textwidth`.

In aid of detecting a stalled float flushing process, we set a Boolean if we encounter a qualified full-page-width float here. Any that qualify but fail the rest of the tests might still pass when reconsidered on an otherwise blank page.

```
\def\@addtodblcol{%
 \begingroup
  \@if@notdblfloat{\@currbox}{%
   \false@sw
  }{%
   \@setfloattypecounts
   \@getfpsbit \tw@
   \@bitor \@currtype \@deferlist
   \@if@sw\if@test\fi{%
    \false@sw
   }{%
    \@ifodd\@tempcnta{%
     \aftergroup\@booleantrue\aftergroup\float@avail@sw
     \@flsetnum \@dbltopnum
     \@ifnum{\@dbltopnum>\z@}{%
       \@ifdim{\@dbltoproom>\ht\@currbox}{%
        \true@sw
       }{%
        \@ifnum{\@fpstype<\sixt@@n}{%
         \begingroup
          \advance \@dbltoproom \@textmin
          \@ifdim{\@dbltoproom>\ht\@currbox}{%
           \endgroup\true@sw
          }{%
           \endgroup\false@sw
          }%
        }{%
         \false@sw
        }%
       }%
     }{%
      \false@sw
     }%
    }{%
     \false@sw
    }%
   }%
  }%
```

```
  {%
   \@tempdima -\ht\@currbox
   \advance\@tempdima
    -\@ifx{\@dbltoplist\@empty}{\dbltextfloatsep}{\dblfloatsep}%
   \global \advance \@dbltoproom \@tempdima
   \global \advance \@colht \@tempdima
   \global \advance \@dbltopnum \m@ne
   \@cons \@dbltoplist \@currbox
  }{%
   \@cons \@deferlist \@currbox
  }%
 \endgroup
}%
```

\@tryfcolumn
\@wtryfc
\@xtryfc
\@ztryfc

Whenever a page is shipped out, LaTeX automatically tries out a float column: a page containing nothing but floats (and, as we have added here, split footnotes).

The following four procedures employ certain macros to communicate between each other:

\fcolmade@sw, a boolean, says whether we were successful in making a float column.

\if@test, a \newif switch, says a float has failed some test.

\@deferlist, is the input to the process, a list, of deferred floats.

\@trylist, a list, stores the deferred floats to be tried out on the float column.

\@failedlist, a list of floats that have failed the selection for the float column.

\@flfail, a list of floats that have failed the second selection for the float column.

\@flsucceed, a list, the floats that have been successfully placed on the float column.

\@freelist, a list, receives any freed floats.

\@colht, a dimen, the available space for the column, including column floats and insertions (footnotes).

\@fpmin, a dimen, the required minimum height for the float column.

\@outputbox, a box, the output of the process.

\@fptop, \@fpsep, \@fpbot, glue, placed above, between, and below floats on the float column.

\@currtype, a count, used temporarily for the float's bits.

\@tempcnta, a count, used temporarily for the float's bits.

In \@tryfcolumn, we alter the criterion for a float page, because if footnotes are present at this point (presumably due to a split insertion) then \@fpminis no longer the right threshold to apply.

Note that we have changed \@tryfcolumn, \@xtryfc, and \@ztryfc syntactically so that the procedure to test for the float's being a column float versus a full-page-width float is passed in as an argument.

```
\def\@tryfcolumn#1{%
  \global\@booleanfalse\fcolmade@sw
  \@ifx@empty\@deferlist{}{%
    \global\let\@trylist\@deferlist
    \global\let\@failedlist\@empty
    \begingroup
      \dimen@\vsize\advance\dimen@-\pagegoal\@ifdim{\dimen@>\z@}{%
       \advance\@fpmin-\dimen@
```

```
      }{}%
      \def\@elt{\@xtryfc#1}\@trylist
    \endgroup
    \fcolmade@sw{%
      \global\setbox\@outputbox\vbox{\vskip \@fptop}%
      \let \@elt \@wtryfc \@flsucceed
      \global\setbox\@outputbox\vbox{\unvbox\@outputbox
        \unskip \vskip \@fpbot
      }%
      \let \@elt \relax
      \xdef\@deferlist{\@failedlist\@flfail}%
      \xdef\@freelist{\@freelist\@flsucceed}%
    }{}%
  }%
}%
\def\@wtryfc #1{%
  \global\setbox\@outputbox\vbox{\unvbox\@outputbox
    \box #1\vskip\@fpsep
  }%
}%
\def\@xtryfc#1#2{%
  \@next\reserved@a\@trylist{}{}% trim \@trylist. Ugly!
  \@currtype \count #2%
  \divide\@currtype\@xxxii\multiply\@currtype\@xxxii
  \@bitor \@currtype \@failedlist
  \@testfp #2%
  #1#2%
  \@ifdim{\ht #2>\@colht   }{\@testtrue}{}%
  \@if@sw\if@test\fi{%
   \@cons\@failedlist #2%
  }{%
   \begingroup
     \gdef\@flsucceed{\@elt #2}%
     \global\let\@flfail\@empty
     \@tempdima\ht #2%
     \def \@elt {\@ztryfc#1}\@trylist
     \@ifdim{\@tempdima >\@fpmin}{%
       \global\@booleantrue\fcolmade@sw
     }{%
       \@cons\@failedlist #2%
     }%
   \endgroup
   \fcolmade@sw{%
     \let \@elt \@gobble
   }{}%
  }%
}%
\def\@ztryfc #1#2{%
  \@tempcnta \count#2%
  \divide\@tempcnta\@xxxii\multiply\@tempcnta\@xxxii
  \@bitor \@tempcnta {\@failedlist \@flfail}%
  \@testfp #2%
  #1#2%
  \@tempdimb\@tempdima
```

```
  \advance\@tempdimb \ht#2\advance\@tempdimb\@fpsep
  \@ifdim{\@tempdimb >\@colht}{%
    \@testtrue
  }{}%
  \@if@sw\if@test\fi{%
    \@cons\@flfail #2%
  }{%
    \@cons\@flsucceed #2%
    \@tempdima\@tempdimb
  }%
}%
```

8.9 Clearing pages

Clearing the page is an elaboration of ending the page: it entails flushing all floats.

This package might make number of float flushing algorithms available, a very simple one that does not try to produce excellent pages, another that tries to make the best use of space, and a more complex one that tries to balance columns.

At the beginning of the page-clearing process, by definition all of the paragraph text involved is on the MVL and all floats have been encountered. There may be material in `\pagesofar`, and (in a multi-column page grid) any number of columns of the page have been composed. Also, there might be footnote material saved up in `\footbox`.

Because we did not want to perform multiple `\shipout`s per visit to the output routine, our multi-column page makeup will not compose multiple columns per visit. This implementation detail may not require alteration, but it is not a limitation that is truly necessary: it is only multiple `\shipout`s per visit that must be avoided.

The crux matter is how to continue with flushing floats even after the material in the MVL is exhausted. At that point, we must, upon completion of the output routine, insert into the MVL an interrupt that triggers the next step in the processing.

Therefore, after processing a `\do@startcolumn` interrupt, we must somehow force the completion of that column. This could be done by inserting a `\do@newpage@pen` interrupt.

And after processing a `\do@startpage@pen` interrupt, that results in `\@dbltopinsert`s, we must ensure that the multiple columns on the page get completed, so that the page itself finally gets shipped out. This part will proceed automatically given that `\do@startcolumn` processing completes successfully.

The process will not be complete until all deferred floats have been placed and shipped out, and all saved-up footnotes have been inserted.

Full-page-width floats can get out of order of column floats. This problem can be remedied by holding them all in the same list. We therefore stop using `\@dbldeferlist` entirely, and all of the procedures that formerly used it have been rewritten to use `\@deferlist` instead. When traversing the list, we apply a selector on the given box that determines whether it is a column-width or page-width float. This selector is different depending on the page grid.

When the `\@deferlist` is processed (by any means), we have to take care of the case where a float of one category is passed over but we are looking for a float of the other category. Here, we must terminate processing, to avoid disordering the floats. This we do by the usual means.

The system has a Boolean that says we are clearing pages: `\clearpage@sw`; if it is true, then at the tail of `\do@startcolumn` processing, we should put down a (`\vfil`?) `\do@newpage@pen` interrupt. This is because the MVL is now empty, so we have to force the columns to complete.

One potential very pathological case would be where there is one or more deferred floats that never successfully get placed: placing floats has stalled, and we will ship out blank pages indefinitely. How to detect this case?

First, `\do@startpage` will evidently be stalled if the following are all true: a) `\@tryfcolumn` and `\@sdblcolelt` both fail, b) there are deferred floats available for page placement, and c) the `\@colht`=`\textheight`, that is, the full page height is available for placement of column floats.

Second, `\do@startcolumn` will evidently be stalled if the following are all true: a) tryfcolumn fails, b) there are deferred floats available for column placement, and a) the `\@colroom`=`\textheight`, that is, the full page height is available for placement of column floats.

`\cleardoublepage` `\clearpage` `\newpage` `\newpage@prep`

The function of `\clearpage` is to end the current page with `\newpage` and then ship out additional pages until () inserts and (deferred) floats are exhausted.

The method involves setting the float placement parameters to completely permissive values and kicking out the current page (using a non-discardable penalty). A possibly short page will be shipped out, followed by any number of float pages. However these float pages, because using permissive float placement, will exhaust all inserts and deferred floats.

Bug Note: in the code for `\clearpage`, the first penalty we output is an unprotected `\pagebreak@pen`. I tried using a protected `\do@newpage@pen`, but that gave rise to a corner case where a blank page was output.

At present, the `\clearpage` procedure does the same as `\newpage`, except that `\clearpage@sw` is turned on, and the (discardable) `\newpage` is inevitably followed by the same procedures that are executed if a page is shipped out.

FIXME: it seems that better than `\pagebreak@pen` would be an unprotected penalty of a special value that would entail output routine processing consisting of the following steps: 3) `\unvbox\@cclv`, 1) set `\clearpage@sw` to `\true@sw`, 2) put down a protected `\do@startcolumn@pen`, 4) take a dead cycle.

The effect would be to liberalize float placement options for the current column as well as further columns that may be output as part of `\clearpage` processing. Of course, it would still be necessary to set `\clearpage@sw` again via an interrupt.

An optimization might be to clear `\clearpage@sw` as part of the same interrupt, but that would actually not work properly, because it is necessary for `\do@endpage` to possibly invoke furhter visits to the output routine before clearpage processing ceases.

```
\def\newpage@prep{%
  \if@noskipsec
    \ifx \@nodocument\relax
      \leavevmode
      \global \@noskipsecfalse
    \fi
  \fi
  \if@inlabel
    \leavevmode
    \global \@inlabelfalse
  \fi
```

```
  \if@nobreak \@nobreakfalse \everypar{}\fi
  \par
}%
\def \newpage {%
 \newpage@prep
 \do@output@MVL{%
  \vfil
  \penalty-\pagebreak@pen
 }%
}%
\def\clearpage{%
 \newpage@prep
 \do@output@MVL{%
  \vfil
  \penalty-\pagebreak@pen
  \global\@booleantrue\clearpage@sw
  \protect@penalty\do@startcolumn@pen
  \protect@penalty\do@endpage@pen
 }%
 \do@output@MVL{%
  \global\@booleanfalse\clearpage@sw
 }%
}%
\def\cleardoublepage{%
 \clearpage
 \@if@sw\if@twoside\fi{%
  \@ifodd\c@page{}{%
   \null\clearpage
  }%
 }{}%
}%
\@booleanfalse\clearpage@sw
```

`\do@endpage@pen` The penalty `\do@endpage@pen` simply dispatches to the page grid procedure that forces an end page. That procedure should test whether there is anything to ship out (say committed floats), then act accordingly. Note that as part of this work, it should `\unvbox\@cclv`, which has been left boxed up so it can be measured.

```
\mathchardef\do@endpage@pen=10007
\@namedef{output@-\the\do@endpage@pen}{%
 \csname end@column@\thepagegrid\endcsname
}%
```

`\do@newpage@pen` The penalty `\do@newpage@pen` allows a "non-discardable `\newpage`" command: a `\newpage` command that will not disappear at a pagebreak. This visit to the output routine will not be dispatched to an interrupt, rather the natural output routine will be executed, where it will remove the protection box.

Call this routine by executing `\protect@penalty\do@newpage@pen`.

```
\mathchardef\do@newpage@pen=10001
\expandafter\let\csname output@-\the\do@newpage@pen\endcsname\relax
```

`\@clearfloatplacement` The procedure `\@clearfloatplacement` sets all of the float placement parameters to completely permissive values. The standard values appear as comments.

```
\def\@clearfloatplacement{%
```

```
 \global\@topnum     \maxdimen % \c@topnumber
 \global\@toproom    \maxdimen % \topfraction\@colht
 \global\@botnum     \maxdimen % \c@bottomnumber
 \global\@botroom    \maxdimen % \bottomfraction\@colht
 \global\@colnum     \maxdimen % \c@totalnumber
%\global\@fpmin      \z@       % \floatpagefraction\@colht
 \global\@dbltopnum  \maxdimen % \c@dbltopnumber
 \global\@dbltoproom \maxdimen % \dbltopfraction\@colht
 \global\@textmin    \z@       % \@colht\advance \@textmin -\@dbltoproom
 \global\@fpmin      \z@       % \dblfloatpagefraction\textheight
 \let\@testfp\@gobble
 \appdef\@setfloattypecounts{\@fpstype16\advance\@fpstype\m@ne}%
}%
```

`\@doclearpage` The `\@doclearpage` procedure is now obsoleted, as is `\@makefcolumn`, which it invoked.

```
\let\@doclearpage\@undefined
\let\@makefcolumn\@undefined
```

`\clr@top@firstmark` `\set@top@firstmark` `\@outputpage` We want accurate values of `\topmark` and `\firstmark`, but we must deal with the fact that there are many different ways of contributing material to the page. Only upon the first contribution to the page is the value of `\topmark` accurate. However, with `\firstmark` we must potentially examine each contribution because the first mark on the page may happen to fall in the last piece of material contributed.

To begin, we define the procedure that initializes the macros to appropriate flag values.

```
\def\clr@top@firstmark{%
 \global\let\saved@@topmark\@undefined
 \global\let\saved@@firstmark\@empty
 \global\let\saved@@botmark\@empty
}%
\clr@top@firstmark
```

Note that the flag value for `\saved@@topmark` is `\@undefined`, just as one would expect. But that for `\saved@@firstmark` and `\saved@@botmark` is `\@empty`.

Next, we define procedure `\set@top@firstmark`; it will be exercised everywhere material is contributed, capturing the mark values if appropriate.

```
\def\set@top@firstmark{%
 \@ifxundefined\saved@@topmark{\expandafter\gdef\expandafter\saved@@topmark\expandafter{\@@top
 \@if@empty\saved@@firstmark{\expandafter\gdef\expandafter\saved@@firstmark\expandafter{\@@fir
 \@if@empty\@@botmark{}{\expandafter\gdef\expandafter\saved@@botmark\expandafter{\@@botmark}}%
}%
```

When should `\set@top@firstmark` be called? A good candidate for a universal procedure for handling contributed material is the natural output routine; are any other calls needed?

Yes, in `\save@column` we must execute `\set@top@firstmark` because we are about to save away `\box\@cclv`, and we will never see its marks again (unless it is unboxed into the MVL), because TEX lets one access a box's marks only within an output routine that has put that box into `\box\@cclv`.

As soon as a page is shipped out, we initialize the two macros that hold the values of `\topmark` and `\firstmark`, respectively. LATEX has exactly one procedure

\@outputpage that does \shipout, which is as it should be: we tailpatch it, and the job is done.

```
\appdef\@outputpage{%
 \clr@top@firstmark
}%
```

8.10 Other interfaces to LaTeX

\@float
\@dblfloat
\@yfloat
\fps@
\fpsd@

The LaTeX kernel procedures \@float and \@dblfloat are treated on an equal footing. Each now takes environment-specific float placement defaults. If none are defined for the calling environment, we apply a default.

A parameter is passed that will set the width of text within the float, normally \columnwidth, and in the "dbl" version, \textwidth. However, an environment such as turnpage may change the meanings of these macros to allow turnpage floats.

```
\def\@float#1{%
 \@ifnextchar[{%}]{%Brace-matching klootch
  \@yfloat\width@float{#1}%
 }{%
  \@ifxundefined@cs{fps@#1}{%
   \edef\reserved@a{\noexpand\@yfloat\noexpand\width@float{#1}[\csname fps@\endcsname]}\reserv
  }{%
   \edef\reserved@a{\noexpand\@yfloat\noexpand\width@float{#1}[\csname fps@#1\endcsname]}\rese
  }%
 }%
}%
\def\@dblfloat#1{%
 \@ifnum{\pagegrid@col=\@ne}{%
  \@float{#1}%
 }{%
  \@ifnextchar[{%}]{%Brace-matching klootch
   \@yfloat\widthd@float{#1}%
  }{%
   \@ifxundefined@cs{fpsd@#1}{%
    \edef\reserved@a{\noexpand\@yfloat\noexpand\widthd@float{#1}[\csname fpsd@\endcsname]}\res
   }{%
    \edef\reserved@a{\noexpand\@yfloat\noexpand\widthd@float{#1}[\csname fpsd@#1\endcsname]}\r
   }%
  }%
 }%
}%
\def\@yfloat#1#2[#3]{%
 \@xfloat{#2}[#3]%
 \hsize#1\linewidth\hsize
 \minipagefootnote@init
}%
\def\fps@{tbp}%
\def\fpsd@{tp}%
\def\width@float{\columnwidth}%
\def\widthd@float{\textwidth}%
```

\end@float
\end@dblfloat
\end@@float
\check@currbox@count
\minipagefootnote@init
\minipagefootnote@here

LaTeX kernel procedures \end@float and \end@dblfloat have been changed to work alike; in particular, floats of both classes are deferred into the same queue.

This measure ensures that they will be placed in their original order, an aspect in which LaTeX is broken.

Note: when retrieving floats from the queues, we can differentiate those of the two categories by the width of the box.

Floats are processed via an output routine message, and are checked for sanity in re the float placement options. In the case of full-page-width floats, we ensure that the h and b float placement options are never asserted, because they make no sense.

Note that if we get to the end of the float box and still have pending footnotes, we put then out.

LaTeX Bug note: if a user types `\begin`table*[h], the float will never succeed in being placed! we try to catch such cases.

Note that the macro `\check@currbox@count` trys to catch cases where the float placement options are such that the float can never be placed.

```
\def\end@float{%
 \end@@float{%
  \check@currbox@count
 }%
}%
\def\end@dblfloat{%
 \@ifnum{\pagegrid@col=\@ne}{%
  \end@float
 }{%
  \end@@float{%
   \@boxfpsbit\@currbox{1}\@ifodd\@tempcnta{\global\advance\count\@currbox\m@ne}{}%
   \@boxfpsbit\@currbox{4}\@ifodd\@tempcnta{\global\advance\count\@currbox-4\relax}{}%
   \global\wd\@currbox\textwidth % Klootch
   \check@currbox@count
  }%
 }%
}%
\def\end@@float#1{%
 \minipagefootnote@here
%\minipagefootnotes
 \@endfloatbox
 #1%
 \@ifnum{\@floatpenalty <\z@}{%
  \@largefloatcheck
  \@cons\@currlist\@currbox
  \@ifnum{\@floatpenalty <-\@Mii}{%
   \do@output@cclv{\@add@float}%
  }{%
   \vadjust{\do@output@cclv{\@add@float}}%
   \@Esphack
  }%
 }{}%
}%
\def\check@currbox@count{%
 \@ifnum{\count\@currbox>\z@}{%
\count@\count\@currbox\divide\count@\sixt@@n\multiply\count@\sixt@@n
\@tempcnta\count\@currbox\advance\@tempcnta-\count@
\@ifnum{\@tempcnta=\z@}{%
   \ltxgrid@warn{Float cannot be placed}%
```

```
}{}%
 }{%
  % Is a \marginpar
 }%
}%
\providecommand\minipagefootnote@init{}%
\providecommand\minipagefootnote@here{}%
```

`\@specialoutput` The `\@add@float` procedure used to reside in standard LaTeX's `\@specialoutput`, which is no more.

Historical Note: `\@specialoutput` and Lamport's method of an output routine dispatcher is the genesis of our more powerful and refined way of using TeX's output routine to safely accomplish page makeup tasks. To it and to him we owe acknowledgement and thanks.

```
\let\@specialoutput\@undefined
```

`\@add@float` In the following, we do not need to execute `\@reinserts`, which was wrong anyway, as you cannot reliably recover insertions when they split (unless you have a way of reinserting the captured insertion ahead of the split-off part).

Now that full-page-width floats are being processed the same as column floats, we have to nip in here and cause them always to be deferred.

At the very end, the `\vsize` is adjusted for any newly committed float.

```
\def\@add@float{%
 \@pageht\ht\@cclv\@pagedp\dp\@cclv
 \unvbox\@cclv
 \@next\@currbox\@currlist{%
  \csname @floatselect@sw@\thepagegrid\endcsname\@currbox{%
   \@ifnum{\count\@currbox>\z@}{%
    \advance \@pageht \@pagedp
    \advance \@pageht \vsize \advance \@pageht -\pagegoal % do not assume \holdinginserts is c
%   \@ifvbox\@kludgeins{%
%    \@ifdim{\wd\@kludgeins=\z@}{%
%     \advance \@pageht \ht\@kludgeins
%    }{}%
%   }{}%
%   \@reinserts
    \@addtocurcol % Commit an h float
  }{%
%  \@reinserts
   \@addmarginpar
  }%
  }{%
   \@resethfps
   \@cons\@deferlist\@currbox
  }%
 }{\@latexbug}%
 \@ifnum{\outputpenalty<\z@}{%
  \@if@sw\if@nobreak\fi{%
   \nobreak
  }{%
   \addpenalty \interlinepenalty
  }%
 }{}%
```

```
 \set@vsize
}%
```

`\@reinserts` The `\@reinserts` procedure of standard LaTeX is now obsoleted.

```
\let\@reinserts\@undefined
```

`\@addtocurcol` We modify the `\@addtocurcol` procedure of standard LaTeX so that a float placed "here" may break over pages.

```
\def \@addtocurcol {%
   \@insertfalse
   \@setfloattypecounts
   \ifnum \@fpstype=8
   \else
     \ifnum \@fpstype=24
     \else
       \@flsettextmin
       \advance \@textmin \@textfloatsheight
       \@reqcolroom \@pageht
       \ifdim \@textmin>\@reqcolroom
         \@reqcolroom \@textmin
       \fi
       \advance \@reqcolroom \ht\@currbox
       \ifdim \@colroom>\@reqcolroom
         \@flsetnum \@colnum
         \ifnum \@colnum>\z@
           \@bitor\@currtype\@deferlist
           \if@test
           \else
             \@bitor\@currtype\@botlist
             \if@test
               \@addtobot
             \else
               \ifodd \count\@currbox
                 \advance \@reqcolroom \intextsep
                 \ifdim \@colroom>\@reqcolroom
                   \global \advance \@colnum \m@ne
                   \global \advance \@textfloatsheight \ht\@currbox
                   \global \advance \@textfloatsheight 2\intextsep
                   \@cons \@midlist \@currbox
                   \if@nobreak
                     \nobreak
                     \@nobreakfalse
                     \everypar{}%
                   \else
                     \addpenalty \interlinepenalty
                   \fi
                   \vskip \intextsep
                   \unvbox\@currbox %AO
                   \penalty\interlinepenalty
                   \vskip\intextsep
                   \ifnum\outputpenalty <-\@Mii \vskip -\parskip\fi
                   \outputpenalty \z@
                   \@inserttrue
                 \fi
```

```
              \fi
              \if@insert
              \else
                \@addtotoporbot
              \fi
            \fi
          \fi
        \fi
      \fi
    \fi
    \if@insert
    \else
      \@resethfps
      \@cons\@deferlist\@currbox
    \fi
}%
```

`\if@twocolumn` The `\newif` switch `\if@twocolumn` is entirely unused. However its access words are invoked by LaTeX's `\document` procedure, so we de-fang it.

```
\@twocolumnfalse
\let\@twocolumntrue\@twocolumnfalse
```

`\@addmarginpar` The procedure `\@addmarginpar` used to access `\if@twocolumn`, but that switch is not reliable; the better way is to use `\thepagegrid`. We establish a convention for a page-grid-oriented procedure, e.g., `\@addmarginpar@one`, that emits a boolean, telling this procedure whether to set the marginpar on the left or right.

```
\def\@addmarginpar{%
 \@next\@marbox\@currlist{%
  \@cons\@freelist\@marbox\@cons\@freelist\@currbox
 }\@latexbug
 \setbox\@marbox\hb@xt@\columnwidth{%
  \csname @addmarginpar@\thepagegrid\endcsname{%
   \hskip-\marginparsep\hskip-\marginparwidth
   \box\@currbox
  }{%
   \hskip\columnwidth\hskip\marginparsep
   \box\@marbox
  }%
  \hss
 }%
 \setbox\z@\box\@currbox
    \@tempdima\@mparbottom
    \advance\@tempdima -\@pageht
    \advance\@tempdima\ht\@marbox
 \@ifdim{\@tempdima >\z@}{%
   \@latex@warning@no@line {Marginpar on page \thepage\space moved}%
 }{%
   \@tempdima\z@
 }%
    \global\@mparbottom\@pageht
    \global\advance\@mparbottom\@tempdima
    \global\advance\@mparbottom\dp\@marbox
    \global\advance\@mparbottom\marginparpush
```

```
    \advance\@tempdima -\ht\@marbox
    \global\setbox \@marbox
                   \vbox {\vskip \@tempdima
                          \box \@marbox}%
    \global \ht\@marbox \z@
    \global \dp\@marbox \z@
    \kern -\@pagedp
    \nointerlineskip
  \box\@marbox
    \nointerlineskip
    \hbox{\vrule \@height\z@ \@width\z@ \@depth\@pagedp}%
}%
```

`turnpage` Any float (viz., `figure` or `table`) within the scope of this environment will be a turnpage float: It will be assumed to occupy an entire page (constitute a float page), the width will be `\textheight`, the height `\textwidth`, and the entire float will be presented rotated 90 degrees.

The implementation requires the services of the `\rotatebox` command, so we supply a dummy definition that explains things to the user.

```
\newenvironment{turnpage}{%
 \def\width@float{\textheight}%
 \def\widthd@float{\textheight}%
 \appdef\@endfloatbox{%
  \@ifxundefined\@currbox{%
   \ltxgrid@warn{Cannot rotate! Not a float}%
  }{%
   \setbox\@currbox\vbox to\textwidth{\vfil\unvbox\@currbox\vfil}%
   \global\setbox\@currbox\vbox{\rotatebox{90}{\box\@currbox}}%
  }%
 }%
}{%
}%
\def\rotatebox@dummy#1#2{%
 \ltxgrid@warn{You must load the graphics or graphicx package in order to use the turnpage env
 #2%
}%
\AtBeginDocument{%
 \@ifxundefined\rotatebox{\let\rotatebox\rotatebox@dummy}{}%
}%
```

8.11 One-off output routines

These procedures are executed in lieu of `\the\output` when the output penalty has the associated flag value.

`output@-1073741824` The first one-off output routine handles the end of the job, wherein LaTeX executes `\@@end`, and breaks to the output with a penalty of "40000000 = $2^{32}/4$. We simply discard `\box\@cclv` and leave. This means that LaTeX is obligated to do `\clearpage` as part of its `\end{document}` processing, otherwise material will be lost.

```
\@namedef{output@-1073741824}{%"40000000
 \deadcycles\z@
%\showbox\@cclv
```

```
 \setbox\z@\box\@cclv
}%
```

`\save@column@pen` The one-off output routine associated with `\penalty\save@column@pen` will be called within a sequence of three such routines by `\execute@message`or its companion routine `\execute@message@insert`. This procedure must save away any the current page and preserve marks.

```
\mathchardef\save@column@pen=10016
\@namedef{output@-\the\save@column@pen}{\save@column}%
```

`\@cclv@saved` We take over the `\@holdpg` box register. Hereafter, we no longer use the `\@holdpg` box register, so let the world know. This should decisively break packages that assume standard LaTeX. Breaking decisively is preferred to quietly proceeding erroneously.

```
\let \@cclv@saved \@holdpg
\let \@holdpg \@undefined
```

`\save@column` The procedure `\save@column` does the actual work of saving away the material on the page. It is invoked both by `\save@column@pen` and by `\save@column@insert@pen`. We save `\box\@cclv` and the primitive `\@@topmark`.

```
\def\save@column{%
 \@ifvoid\@cclv@saved{%
\set@top@firstmark
\global\@topmark@saved\expandafter{\@@topmark}%
 }{}%
\global\setbox\@cclv@saved\vbox{%
  \@ifvoid\@cclv@saved{}{%
 \unvbox\@cclv@saved
 \marry@baselines
}%
\unvbox\@cclv
\lose@breaks
\setbox\z@\lastbox
}%
}%
\newtoks\@topmark@saved
```

`\prep@cclv` The procedure `\prep@cclv` is used by message handlers to set up their environment to ape that of the usual output routine, with the boxed-up page in `\box\@cclv`. Here, we retrieve the material from `\@cclv@saved`, where it was saved away by the one-off output routine associated with `\save@column@pen`.

```
\def\prep@cclv{%
 \setbox\z@\box\@cclv
 \setbox\@cclv\box\@cclv@saved
 \vbadness\@M
}%
```

`\save@column@insert@pen` The one-off output routine associated with `\penalty\save@column@insert@pen` is similar to that of `\save@column@pen` augmented with the processing of insertions. It is called by `\execute@message@insert` (i.e., at a grid change) and saves away the current page and preserves marks. In addition, it saves away any insertions that fall on the current page. As with the regular output routine, it executes in two phases, first with `\holdinginserts` set, then with it cleared.

```
\mathchardef\save@column@insert@pen=10017
\@namedef{output@-\the\save@column@insert@pen}{\toggle@insert\savecolumn@holding\savecolumn@mo
```

The procedure `\savecolumn@holding` is the first phase of saving a column with its inserts. This phase must detect and remedy the one circumstance that will confound our efforts to propagate marks. It is similar to `\output@holding`, except that we have to deal with the protection box, which must remain, because the messaging mechanism is being used.

If it appears that we have the pathological "Big Bad Box" case at hand, we use the `\dead@cycle@repair@protected` procedure instead of `\dead@cycle` to do our dead cycle.

```
\def\savecolumn@holding{%
\@if@exceed@pagegoal{\unvcopy\@cclv\setbox\z@\lastbox}{%
  \setbox\z@\vbox{\unvcopy\@cclv\setbox\z@\lastbox}%
  \outputdebug@sw{{\tracingall\scrollmode\showbox\z@}}{}%
\dimen@\ht\@cclv\advance\dimen@-\ht\z@
\dead@cycle@repair@protected\dimen@
}{%
\dead@cycle
}%
}%
```

The procedure `\save@column@moving` is the second phase of saving a column with its inserts. Now that `\holdinginserts` is cleared, we can look in the various `\insert` registers for our inserts (at present there is only one, `\footins`). if anything is there, we save it away and ask for another cycle (because it may have split).

Note that the message that is about to be executed had better deal properly with the contents of the `\footins@saved` box.

```
\def\savecolumn@moving{%
 \@cclv@nontrivial@sw{%
  \save@column
 }{%
  {\setbox\z@\box\@cclv}%
 }%
\@ifvoid\footins{}{%
\outputdebug@sw{{\tracingall\scrollmode\showbox\footins}}{}%
\global\setbox\footins@saved\vbox{\unvbox\footins@saved\marry@baselines\unvbox\footins}%
\protect@penalty\save@column@insert@pen
}%
}%
\newbox\footins@saved
```

`\save@message@pen` The one-off output routine associated with `\penalty\save@message@pen` saves away the message that has been passed. This procedure is penultimate in a sequence of one-off output routine calls; earlier ones have saved away the MVL and preserved marks, the last executes the message.

Note that we are passing tokens to TEX's primitive `\mark` mechanism, so we must ensure that they are not inappropriately expanded. We use the same mechanism for all such cases, namely `\let@mark`.

Note: we expect that `\box\@cclv`'s contents are well known: `\topskip`, protection box, and a `\mark`, the latter havin the message. But if we came here via `\penalty`10017, there might be an `\insert` node here as well, because a footnote

split. Because this procedure simply voids out \box\@cclv, such material would be lost. Perhaps we can repair things by manipulating the \insert mechanism temporarily.

```
\mathchardef\save@message@pen=10018
\@namedef{output@-\the\save@message@pen}{\save@message}%
\def\save@message{%
 \setbox\z@\box\@cclv %FIXME: what if \box\@cclv is not empty?
 \toks@\expandafter{\@@firstmark}%
 \expandafter\gdef\expandafter\@message@saved\expandafter{\the\toks@}%
 \expandafter\do@@mark\expandafter{\the\@topmark@saved}%
}%
\gdef\@message@saved{}%
```

\execute@message@pen The one-off output routine associated with \execute@message@pen simply executes the given message. It is last in a sequence of one-off output routine calls; earlier ones have saved all that require saving.

```
\mathchardef\execute@message@pen=10019
\@namedef{output@-\the\execute@message@pen}{\@message@saved}%
```

8.12 Output messages

Message handlers are procedures that execute output messages, tokens that are passed to the output routine for execution in an environment appropriate to page makeup.

How it works. We put down three large negative penalties, each of which will be handled by the output dispatcher (not the output routine), each penalty being protected by a removable, non-discardable item (i.e., a box). Either three or four invocations of one-off output routines are involved per message.

We make the last of the three protection boxes have a depth equal to the value of \prevdepth that was current when the procedure is called. This effectively restores \prevdepth.

In each case, the one-off output routine will remove the extraneous box we have inserted. And the second and third one-off routines will simply void \box\@cclv, because its contents are entirely artificial.

FIXME: not so! If \holdinginserts is cleared, that box may have an insert node; it must be preserved, too.

The first routine saves away the current column contents and remembers the \topmark for later use. There is a variant routine that first clears \holdinginserts, so that the message can handle any inserts present in the boxed-up page; this of course entails yet another visit to the output routine.

The penultimate routine saves away the tokens transmitted in via the \@@mark: the argument of the macro. These tokens are of course the very thing we wish to execute within the safety of the output routine. It also puts down a mark containing the \topmark tokens saved by the first routine. By this means, the mark, which we have clobbered, is restored.

The last routine simply executes the given tokens. In the course of doing this, it must take care of \box\@cclv, either by shipping it out, or by \unvboxing it onto the MVL.

\execute@message The procedure \execute@message simply calls the utility procedure \@execute@message with a penalty value for the standard treatment.

```
\def\execute@message{%
 \@execute@message\save@column@pen %Implicit #2
}%
```

`\execute@message@insert` The procedure `\execute@message@insert` is like `\execute@message` in all respects except that the penalty value is `\save@column@insert@pen`, which arranges for the message handler involved to deal with the page's insertions. At the same time, we prepare the `\footins` box so that these insertions can be dealt with.

Note: If more insertions are added to LaTeX (presumably via `\newinsert`), then they must be dealt with in a way entirely analogous to `\footins`.

```
\def\execute@message@insert#1{%
 \@execute@message\save@column@insert@pen{\setbox\footins\box\footins@saved#1}%
}%
```

`\@execute@message` The utility procedure `\@execute@message` is called by `\execute@message` and `\execute@message@insert`. We prepare by creating a `\vbox` containing all the needed nodes and proceed by simply `\unvbox`ing that box onto the MVL. We ensure that `\box\@cclv` is properly set up for the output message handler by always inserting `\prep@cclv` in advance of the argument.

Note that each one-off output routine is invoked effectively the same as `\protect@penalty`, except that the second invocation involves an additional `\mark` node, and the third a specially prepared protection box.

Note also that TeX's primitive `\mark` is called here without any expansion protection. This is the only place where it is called that way, but it's OK because those tokens have have been pre-expanded by procedures that call `\execute@message`. FIXME: all procedures calling `\execute@message` must pre-expand their tokens!

```
\long\def\@execute@message#1#2{%
 \begingroup
  \dimen@\prevdepth\@ifdim{\dimen@<\z@}{\dimen@\z@}{}%
  \setbox\z@\vbox{%
   \protect@penalty#1%
   \protection@box
   \toks@{\prep@cclv#2}%
   \@@mark{\the\toks@}%
   \penalty-\save@message@pen
%  \hbox{\vrule\@height\z@\@width\z@\@depth\dimen@}%
   \setbox\z@\null\dp\z@\dimen@\ht\z@-\dimen@
   \nointerlineskip\box\z@
   \penalty-\execute@message@pen
  }\unvbox\z@
 \endgroup
}%
```

`\do@output@cclv` The procedure `\do@output@cclv` provides access to message handlers at their simplest. The message will execute in the usual environment of the output routine, with the boxed-up page in `\box\@cclv`, and we assume that `\holdinginserts` remains set. This procedure must be invoked within main vertical mode; it is the obligation of the macro writer to ensure that this is the case.

```
\def\do@output@cclv{\execute@message}%
```

`\do@output@MVL` The procedure `\do@output@MVL`, like `\do@output@cclv`, is an interface for messages, but provides two additional services: the command may also be invoked in horizontal mode, and the message handler will execute with the MVL unboxed.

```
\def\do@output@MVL#1{%
 \@ifvmode{%
  \begingroup\execute@message{\unvbox\@cclv#1}\endgroup
 }{%
  \@ifhmode{%
   \vadjust{\execute@message{\unvbox\@cclv#1}}%
  }{%
   \@latexerr{\string\do@output@MVL\space cannot be executed in this mode!}\@eha
  }%
 }%
}%
```

`\lose@breaks` The purpose of this procedure is to get rid of all the extraneous `\penalty\@M` nodes that tend to build up in the MVL.

```
\def\lose@breaks{%
 \loopwhile{%
  \count@\lastpenalty
  \@ifnum{\count@=\@M}{% 10000 is a TeX magic number!
   \unpenalty\true@sw
  }{%
   \false@sw
  }%
 }%
}%
```

`\removestuff` `\removestuff` is a document-level command that removes the bottom skip glue item from the MVL.

```
\def\removestuff{\do@output@MVL{\unskip\unpenalty}}%
```

`\removephantombox` The procedure `\removephantombox` is a special-purpose message handler exclusively for preventing incorrect spacing above display math. It must be issued in horizontal mode within the phantom paragraph generated when display math starts up in vertical mode.

```
\def\removephantombox{%
 \vadjust{%
  \execute@message{%
   \unvbox\@cclv
   \setbox\z@\lastbox
   \unskip
   \unskip
   \unpenalty
   \penalty\predisplaypenalty
   \vskip\abovedisplayskip
  }%
 }%
}%
```

`\addstuff` `\addstuff` is a document-level command that adds penalty, glue, or both to the MVL. The penalty and glue items are rearranged so that all penalties nodes precede all the glue nodes, which is the canonical arrangement.

```
\def\addstuff#1#2{\edef\@tempa{\noexpand\do@output@MVL{\noexpand\@addstuff{#1}{#2}}}\@tempa}%
\def\@addstuff#1#2{%
 \skip@\lastskip\unskip
```

```
\count@\lastpenalty\unpenalty
\@if@empty{#1}{}{\penalty#1\relax}%
\@ifnum{\count@=\z@}{}{\penalty\count@}%
\vskip\skip@
\@if@empty{#2}{}{\vskip#2\relax}%
}%
```

`\replacestuff` `\replacestuff` is a document-level command similar to `\addstuff`; but it replaces penalty, glue, or both in the MVL. The penalty and glue items are rearranged so that all penalties nodes precede all the glue nodes, which is the canonical arrangement.

```
\def\replacestuff#1#2{\edef\@tempa{\noexpand\do@output@MVL{\noexpand\@replacestuff{#1}{#2}}}\@
\def\@replacestuff#1#2{%
 \skip@\lastskip\unskip
 \count@\lastpenalty\unpenalty
 \@if@empty{#1}{}{%
 \@ifnum{\count@>\@M}{}{%
   \@ifnum{\count@=\z@}{\count@=#1\relax}{%
    \@ifnum{\count@<#1\relax}{}{%
     \count@=#1\relax
    }%
   }%
 }%
 }%
 \@ifnum{\count@=\z@}{}{\penalty\count@}%
 \@if@empty{#2}{}{%
  \@tempskipa#2\relax
  \@ifdim{\z@>\@tempskipa}{%
   \advance\skip@-\@tempskipa
  }{%
   \@ifdim{\skip@>\@tempskipa}{}{%
    \skip@\@tempskipa
   }%
  }%
 }%
 \vskip\skip@
}%
```

`\move@insertions` `\hold@insertions` `\move@insert@sw` In order to avoid bolluxing up `\insert` registers by our one-off output routines, we set `\holdinginserts` to zero by default and only clear it (briefly) while we handle cases where we want inserts to show up.

```
\def\move@insertions{\global\holdinginserts\z@}%
\def\hold@insertions{\global\holdinginserts\@ne}%
\hold@insertions
\def\move@insert@sw{\@ifnum{\holdinginserts=\z@}}%
\def\toggle@insert#1#2{%
 \@ifnum{\holdinginserts=\z@}{\hold@insertions#2}{\move@insertions#1}%
}%
```

8.13 Messages to alter the page grid

Here is the implementation of the grid-switching procedures. We perform two checks when changing the page grid; first to ensure that the target page grid is

known (defensive programming), second to ensure that the switch is a non-trivial one. The latter check must be performed within the safety of the output routine, so requires using an output message. Thus, a grid change requires two messages, for a total of six visits to the output routine.

`\do@columngrid` Utility procedure `\do@columngrid` changes the page grid. Note that this command forces an end to the current paragraph. This is necessary, because a page grid change makes no sense unless we can alter the `\hsize` before commencing to typeset the following paragraph. So the command should never be executed in horizontal mode anyway.

```
\def\do@columngrid#1#2{%
 \par
 \expandafter\let\expandafter\@tempa\csname open@column@#1\endcsname
 \@ifx{\relax\@tempa}{%
  \ltxgrid@warn{Unknown page grid #1. No action taken}%
 }{%
  \do@output@MVL{\start@column{#1}{#2}}%
 }%
}%
```

`\start@column` Procedure `\start@column` lays down the interrupts to switch the page grid. If the change to the page grid would have been trivial, it bails out. It seems a reasonable tradeoff of processing versus security: once we commit to changing the page grid, we clear `\holdinginserts`, so there is no turning back.

Note that the second argument to the macro allows us to pass an argument to the page grid that is starting up. This can be handy, because a single procedure can handle multiple page grids, differing only by the value of a parameter.

FIXME: this means that you cannot switch between mlt page grids in a single step. But do we want to do this, at all, at all?

```
\def\start@column#1#2{%
 \def\@tempa{#1}\@ifx{\@tempa\thepagegrid}{%
  \ltxgrid@info{Already in page grid \thepagegrid. No action taken}%
 }{%
  \expandafter\execute@message@insert
  \expandafter{%
                \csname shut@column@\thepagegrid\expandafter\endcsname
                \csname open@column@#1\endcsname{#2}%
                \set@vsize
              }%
 }%
}%
```

`\thepagegrid` The macro `\thepagegrid` tracks what kind of page grid we are in.

Note: Access `\thepagegrid` only within the safety of the output routine.

Warning: The page grid should be changed only within the safety of the output routine. People who write multicol page grid mechanisms appear not to understand the matter, so they should particularly heed this warning. Think about it: obviously Lamport did so, which is why his `\twocolumn` command forced a pagebreak, which is limiting, but safe.

```
\def\thepagegrid{one}%
```

8.14 Application Note: implementing a page grid

If you want to create a new page grid for LaTeX, you must define five procedures with specific names: `\open@column@`name, `\shut@column@`name, `\end@column@`name, `\output@column@`name, and `\@addmarginpar@`name, where "name" is the name of your page grid.

The procedure `\open@column@`name starts the new page grid. It should define `\thepagegrid`, deal with `\box\pagesofar` and `\box\footbox` (perhaps by leaving them alone), and it should set the values of LaTeX's page layout parameters for the column size and height.

The procedure `\shut@column@`name should expect to be called with `\holdinginserts` cleared (it can assume that `\holdinginserts` will automatically be restored). It should properly deal with insertions (like footnotes); calling `\@makecol` is a good way to do this. It should know that the page grid is being terminated in the middle of a page, so it should make arrangements to carry the footnotes down to the bottom of the column or page, and it should possibly salt away the material for later incorporation into the page. The box registers `\footbox` and `\pagesofar` are customarily used for this purpose.

The procedure `\end@column@`name should kick out a possibly short page containing all the floats committed to the page. It will be invoked during `\clearpage` processing. After that, it should `\unvbox\@cclv`.

The procedure `\output@column@`name should ship out or commit the current `\@outputbox`. In a one-column layout, you ship out; in a multicolumn layout, you commit the box as the contents of a particular column, and if that column is the last, you ship out.

The procedure `\@addmarginpar@`name should return a boolean (either `\true@sw` or `\false@sw` or an equivalent) to tell the marginpar mechanism to place the marginal material to the right or left, respectively.

You can use the existing page grids "one" and "mlt" as a point of departure for creating others. The former can be the basis for, say, a single-column page grid with a side column.

`\pagesofar`
`\footbox`

```
1249 \newbox\pagesofar
1250 \newbox\footbox
```

8.14.1 One-column page grid

`\onecolumngrid`
`\open@column@one`
`\shut@column@one`
`\float@column@one`
`\end@column@one`
`\output@column@one`
`\@addmarginpar@one`

Here are all the procedures necessary for the standard page grid named "one": a single column layout. It is, of course, LaTeX's familiar `\onecolumn` layout. We begin with the procedure exposed to the style writer. This is, however, not a LaTeX command; users should not change the page grid.

```
1251 \newcommand\onecolumngrid{\do@columngrid{one}{\@ne}}%
```

Note that a document class that issues the command `\onecolumn` will break. This includes LaTeX's standard classes.dtx-based classes: if your class descends from one of these, you must expunge it of all such commands.

```
1252 \let\onecolumn\@undefined
```

The procedure `\open@column@one` takes advantage of the special nature of the one-column page grid to deal with `\box\pagesofar`, therefore it must also reset `\@colroom`.

```
\def\open@column@one#1{%
 \unvbox\pagesofar
 \gdef\thepagegrid{one}%
 \global\pagegrid@col#1%
 \global\pagegrid@cur\@ne
 \set@colht
%\set@colroom
 \set@column@hsize\pagegrid@col
}%
```

The procedure `\shut@column@one` saves away the one-column material into the box register `\pagesofar`. Because it is called from a message handler, we are assured that marks are properly taken care of.

```
\def\shut@column@one{%
 \@makecol
 \global\setbox\pagesofar\vbox{\unvbox\@outputbox\recover@footins}%
 \set@colht
%\set@colroom
}%
```

The procedure `\float@column@one` takes care of a float column that has been built by `\@tryfcolumn`, in the single-column page grid.

```
\def\float@column@one{%
 \@makecol
 \@outputpage
}%
```

The procedure `\end@column@one` is executed at the end of `\clearpage` processing, if we were in a one-column page grid, once all permissive float pages have been shipped out. At this point, one could perhaps assume that nothing more need be done, but let us anyway test for committed floats and force a shipout.

FIXME: this procedure does the same as `\end@column@mlt`(except for the test of `\@ifx@empty\@dbltoplist`): the two could almost be the same procedure.

I have changed this procedure to avoid the testing it once did: it simply puts down interrupts, upon which it relies to correctly do what `\clearpage` requires.

```
\def\end@column@one{%
 \unvbox\@cclv\setbox\z@\lastbox
 \protect@penalty\do@newpage@pen
}%
```

The procedure `\output@column@one` is dispatched from the output routine when we have completed a page (that is, a column in a one-column page grid). It ships out the page using the `\@outputpage` of standard LaTeX, which has been retained (it is needed also in `\output@column@mlt`, and in any case should remain as the sole procedure in LaTeX where `\shipout` is performed). It will be followed up with an output routine message to prepare a new column.

```
\def\output@column@one{%
 \@outputpage
}%
```

The following procedure determines which side of the page a marginpar will appear. It reproduces the behavior of standard LaTeX.

```
\def\@addmarginpar@one{%
 \@if@sw\if@mparswitch\fi{%
  \@ifodd\c@page{\false@sw}{\true@sw}%
```

```
 }{\false@sw}{%
  \@if@sw\if@reversemargin\fi{\false@sw}{\true@sw}%
 }{%
  \@if@sw\if@reversemargin\fi{\true@sw}{\false@sw}%
 }%
}%
```

The following procedure yields a Boolean value; it determines whether a float in the deferred queue is appropriate for placing. In the one-column grid, all floats are so.

```
\def\@floatselect@sw@one#1{\true@sw}%
```

```
\def\onecolumngrid@push{%
 \do@output@MVL{%
  \@ifnum{\pagegrid@col=\@ne}{%
   \global\let\restorecolumngrid\@empty
  }{%
   \xdef\restorecolumngrid{%
    \noexpand\start@column{\thepagegrid}{\the\pagegrid@col}%
   }%
   \start@column{one}{\@ne}%
  }%
 }%
}%
\def\onecolumngrid@pop{%
 \do@output@MVL{\restorecolumngrid}%
}%
```

8.14.2 Two-column page grid

`\twocolumngrid` `\open@column@mlt` `\shut@column@mlt` `\end@column@mlt` `\output@column@mlt` `\@addmarginpar@mlt`

Here are all the procedures necessary for the standard page grid named "mlt": the multi-column page grid. With an argument of "2", it is, of course, LaTeX's familiar `\twocolumn` layout.

We start with the procedure to switch to the two-column page grid.

```
\newcommand\twocolumngrid{\do@columngrid{mlt}{\tw@}}%
```

The corresponding command of LaTeX is obsolete.

```
\let\twocolumn\@undefined
```

Of course, `\@topnewpage` is also obsolete. Just do

`\clearpage\onecolumngrid`¡vertical mode material¿`\twocolumngrid`.

```
\let\@topnewpage\@undefined
```

If your document class descends from one of LaTeX's standard classes.dtx-derived classes, it will break. You must expunge from it all such commands.

```
\def\open@column@mlt#1{%
 \gdef\thepagegrid{mlt}%
 \global\pagegrid@col#1%
 \global\pagegrid@cur\@ne
 \set@column@hsize\pagegrid@col
 \set@colht
%\set@colroom
}%
```

The procedure `\shut@column@mlt` ends the current column, balances the columns, and salts away all in `\pagesofar`. Because it is called in a message handler, we are assured that marks are handled properly. Attention: because this procedure balances columns, all footnotes are held aside in `\footbox` for placement at the bottom of the page.

Bug note: the last macro executed by this procedure is `\set@colht`, but had been erroneously `\set@colroom`. I now believe that the latter should be changed pretty much everywhere to the former.

```
\def\shut@column@mlt{%
 \@cclv@nontrivial@sw{%
\@makecol
\@ifnum{\pagegrid@cur<\pagegrid@col}{%
\expandafter\global\expandafter\setbox\csname col@\the\pagegrid@cur\endcsname\box\@outputbox
\global\advance\pagegrid@cur\@ne
}{}%
 }{%
  {\setbox\z@\box\@cclv}%
 }%
\@ifnum{\pagegrid@cur>\@ne}{%
\csname balance@\the\pagegrid@col\endcsname
\grid@column{}%
\@combinepage
\@combinedblfloats
\global\setbox\pagesofar\box\@outputbox
 }{}%
 \set@colht
}%
```

The procedure `\float@column@mlt` takes care of a float page that has been built by `\@tryfcolumn`, in the multi-column page grid.

```
\def\float@column@mlt{%
  \@combinepage
  \@combinedblfloats
  \@outputpage
  \global\pagegrid@cur\@ne
  \protect@penalty\do@startpage@pen
}%
```

The procedure `\end@column@mlt` is executed at the end of `\clearpage` processing, if we were in a multi-column page grid, once all permissive float pages have been shipped out. If no floats are committed and if no columns are yet filled, we have nothing to do. Otherwise, we kick out a column and try again.

Note that in our code to kick out a column, we must deal properly with the case where the column is trivial: it will have nothing but `\topskip` glue plus a protection box. We substitute an ordinary `\null` for the protection box.

```
\def\end@column@mlt{%
 \@ifx@empty\@toplist{%
  \@ifx@empty\@botlist{%
   \@ifx@empty\@dbltoplist{%
    \@ifx@empty\@deferlist{%
     \@ifnum{\pagegrid@cur=\@ne}{%
      \false@sw
     }{%
```

```
      \true@sw
     }%
    }{%
     \true@sw
    }%
   }{%
    \true@sw
   }%
  }{%
   \true@sw
  }%
 }{%
  \true@sw
 }%
 % true = kick out a column and try again
 {%
  \@cclv@nontrivial@sw{%
   \unvbox\@cclv\setbox\z@\lastbox
  }{%
   \unvbox\@cclv\setbox\z@\lastbox\unskip\null
  }%
  \protect@penalty\do@newpage@pen
  \protect@penalty\do@endpage@pen
 }{%
  \unvbox\@cclv\setbox\z@\lastbox
 }%
}%
```

The procedure `\output@column@mlt`(cf. `\output@column@one`) is dispatched from the output routine when we have completed a column in a multi-column page grid). (It replaces the `\@outputdblcol` of standard LaTeX.) If a complete set of columns is at hand, it ships out the page and lays down an interrupt for `\do@startpage@pen`, which will commit the full-page-width floats to the next page. Like `\output@column@mlt`, this is followed by an output routine message to prepare a new column.

```
\def\output@column@mlt{%
 \@ifnum{\pagegrid@cur<\pagegrid@col}{%
  \expandafter\global\expandafter\setbox\csname col@\the\pagegrid@cur\endcsname\box\@outputbox
  \global\advance\pagegrid@cur\@ne
 }{%
  \set@adj@colht\dimen@
% \advance\dimen@-\topskip
  \grid@column{}%{\dimen@}%
  \@combinepage
  \@combinedblfloats
  \@outputpage
  \global\pagegrid@cur\@ne
  \protect@penalty\do@startpage@pen
 }%
}%
```

The procedure `\output@column@mlt` obsoletes LaTeX's `\@outputdblcol`

```
\let\@outputdblcol\@undefined
```

The following procedure yields a Boolean value; it determines whether a float

in the deferred queue is appropriate for placement in the column. In the multi-column grid, only those narrower than \textwidth are so.

```
\def\@floatselect@sw@mlt#1{\@if@notdblfloat{#1}}%
```

The following procedure determines which side of the page a marginpar will appear. It reproduces the behavior of standard LaTeX.

```
\def\@addmarginpar@mlt{% emits a boolean
 \@ifnum{\pagegrid@cur=\@ne}%
}%
```

8.14.3 Page grid utility procedures

\pagegrid@cur
\pagegrid@col
\col@
\pagegrid@init

We take over LaTeX's \col@number and \@leftcolumn, which are obsolete. We create two counters to hold the columns in the page grid and the current column within. We also create the first of a set of box registers to hold the committted columns.

```
\let\pagegrid@cur\col@number
\let\col@number\@undefined
\newcount\pagegrid@col
\pagegrid@cur\@ne
\expandafter\let\csname col@\the\pagegrid@cur\endcsname\@leftcolumn
\let\@leftcolumn\@undefined
```

The default is for maximum two columns. If your class will require more columns, assign that number to \pagegrid@col before \begin{document} time.

```
\pagegrid@col\tw@
```

The procedure \pagegrid@init exercises \newbox sufficiently to create the boxes for holding the columns in the page grid.

```
\def\pagegrid@init{%
 \advance\pagegrid@cur\@ne
 \@ifnum{\pagegrid@cur<\pagegrid@col}{%
  \csname newbox\expandafter\endcsname\csname col@\the\pagegrid@cur\endcsname
  \pagegrid@init
 }{%
 }%
}%
\appdef\class@documenthook{%
 \pagegrid@init
}%
```

\grid@column

The procedure \grid@column knows how to lay up the columns in a multi-column page grid. It uses utility procedures \append@column and \box@column.

```
\def\grid@column#1{%
 \global\setbox\@outputbox\vbox{%
  \hb@xt@\textwidth{%
   \vrule\@height\z@\@width\z@\@if@empty{#1}{}{\@depth#1}%
   \pagegrid@cur\@ne
   \append@column
   \box@column\@outputbox
  }%
  \vskip\z@skip % FIXME: page depth!
 }%
}%
```

\append@column
\box@column
\marry@baselines

The procedure \append@column appends columns for \grid@column, \box@column builds the columns for \append@column, and \marry@baselines pastes vertical things back together.

Note that \box@column makes an attempt to prevent excessive \topskip or \baselineskip glue from being applied by TEX when \@outputbox is contributed to the MVL. If this is not done, it is possible to get into an infinite loop in the corner case, wherein the page grid is changed to one column and the balanced-up columns are already sufficient to fill the page.

Note (AO 0920): I have changed the dimension involved with \box@column from \vsize to \textheight, because the former is certainly not the correct value to use: it will change if floats have been placed in the last column of the page. I believe \textheight is the correct parameter to use here.

A REVTeX4 beta user, Sergey Strelkov (strelkov@maik.rssi.ru), wants the option of ragged-bottom columns. Implementing this feature properly means reboxing the columns to their natural height only if \raggedcolumn@sw is true. Otherwise, they get reboxed to their common height (\@colht?).

Note that the default has hereby changed from ragged to flush. It's not clear that anyone but Sergey will notice.

The macro \marry@skip addresses (in a limited way) the fact that neither the value of \baselineskip nor that of \topskip can be relied upon for the purpose of marrying the baselines of two split columns. (Because there might have been a local change to their values at the point where the output routine got triggered.)

For best results, your document class should call for grid changes only when in basal text settings. The \marry@baselines procedure will use the values appropriate to that point when attempting to put the columns back together.

In any case, we are not attempting to solve the more general problem of how to marry baselines where the leading can change arbitrarily within the galley or where glue could have been trimmed at a page top.

```
\def\append@column{%
 \@ifnum{\pagegrid@cur<\pagegrid@col}{%
  \expandafter\box@column\csname col@\the\pagegrid@cur\endcsname
  \hfil
  \vrule \@width\columnseprule
  \hfil
  \advance\pagegrid@cur\@ne
  \append@column
 }{%
 }%
}%
\def\box@column#1{%
 \raise\topskip
 \hb@xt@\columnwidth{%
  \dimen@\ht#1\@ifdim{\dimen@>\@colht}{\dimen@\@colht}{}%
% \advance\dimen@-\topskip
  \count@\vbadness\vbadness\@M
  \dimen@ii\vfuzz\vfuzz\maxdimen
  \outputdebug@sw{\saythe\@colht\saythe\dimen@}{}%
  \vtop to\dimen@
% \@ifdim{\ht#1>\textheight}{to\textheight}{}%
  {\hrule\@height\z@
   \unvbox#1%
```

```
    \raggedcolumn@skip
   }%
   \vfuzz\dimen@ii
   \vbadness\count@
   \hss
  }%
}%
\def\marry@baselines{%
%{\tracingall\scrollmode\showlists}%
%\skip@\baselineskip\advance\skip@-\topskip %FIXME: cannot assume \baselineskip nor \topskip
 \vskip\marry@skip\relax
}%
\gdef\marry@skip{\z@skip}%
\def\set@marry@skip{%
\begingroup
  \skip@\baselineskip\advance\skip@-\topskip
  \@ifdim{\skip@>\z@}{%
   \xdef\marry@skip{\the\skip@}%
  }{}%
 \endgroup
}%
\AtBeginDocument{%
 \@ifxundefined\raggedcolumn@sw{\@booleanfalse\raggedcolumn@sw}{}%
}%
\def\raggedcolumn@skip{%
 \vskip\z@\raggedcolumn@sw{\@plus.0001fil\@minus.0001fil}{}\relax
}%
```

`\@combinepage` The procedure `\@combinepage` prepends the stored page to `\@outputbox`.

```
\def\@combinepage{%
 \@ifvoid\pagesofar{}{%
  \setbox\@outputbox\vbox{%
   \unvbox\pagesofar
   \marry@baselines
   \unvbox\@outputbox
  }%
 }%
 \@ifvoid\footbox{}{%
  \setbox\@outputbox\vbox{%
   \unvbox\@outputbox
   \marry@baselines
   \unvbox\footbox
  }%
 }%
}%
```

`\@combinedblfloats` We modify LaTeX's `\@combinedblfloats` to be more appropriate for incremental page building: we `\unvbox` the `\@outputbox`.

```
\def\@combinedblfloats{%
 \@ifx@empty\@dbltoplist{}{%
  \setbox\@tempboxa\vbox{}%
  \let\@elt\@comdblflelt\@dbltoplist
  \let\@elt\relax\xdef\@freelist{\@freelist\@dbltoplist}%
  \global\let\@dbltoplist\@empty
```

```
  \setbox\@outputbox\vbox{%
   %\boxmaxdepth\maxdepth   %% probably not needed, CAR
   \unvbox\@tempboxa\unskip
   \@ifnum{\@dbltopnum>\m@ne}{\dblfigrule}{}%FIXME: how is \@dbltopnum maintained?
   \vskip\dbltextfloatsep
   \unvbox\@outputbox
  }%
 }%
}%
```

`\set@column@hsize` The procedure `\set@column@hsize` takes care of setting up the horizontal dimensions for the current page grid. The present routine will certainly not be adequate for more complex page layouts (e.g., with a side column), but works for the common ones.

```
\def\set@column@hsize#1{%
 \pagegrid@col#1%
 \global\columnwidth\textwidth
 \global\advance\columnwidth\columnsep
 \global\divide\columnwidth\pagegrid@col
 \global\advance\columnwidth-\columnsep
 \global\hsize\columnwidth
 \global\linewidth\columnwidth
 \skip@\baselineskip\advance\skip@-\topskip
 \@ifnum{\pagegrid@col>\@ne}{\set@marry@skip}{}%
}%
```

`\set@colht` `\set@colroom` `\set@vsize` `\set@adj@colht` The story of `\textheight`, `\@colht`, `\@colroom`, and `\vsize`.

`\textheight`—height of the text column. Not a running parameter, however, each time a page is shipped out, the `\textheight` could in principle be altered. This must be done before

`\@colht`—`\textheight` minus the height of any full-page-width floats. The latter are committed only just after shipping out, and only if we are in a multicolumn page grid. Therefore, `\@colht` should be set after a `\shipout` (by `\@outputpage`) and will be adjusted when full-page-width floats are committed to the fresh page by `\do@startpage`.

`\@colroom`—`\@colht` (adjusted by `\pagesofar`) minus the height of any column-width floats. The latter are committed anywhere on the page, at which point `\@colroom` must be adjusted. Therefore, `\@colroom` should be set (by `\set@colroom`) whenever a column is prepared (by). FIXME: committed (by `\output@column@`) and will be adjusted (by `\@add@float` or `\do@startcolumn`) whenever a float is committted to the column.

`\vsize`—`\@colroom`. Therefore, `\vsize` should be set (by `\set@vsize`) whenever the `\@colroom` is set (by `\set@colroom`) or adjusted (by `\@add@float` or `\do@startcolumn`) FIXME: or when the `\pagesofar` box is changed (after invoking `\open@column@`).

Question: what if there are committed floats? Footnotes? Answer: full-page-width floats are only committed at top, and they are already reckoned with in `\@colht`. Column-width committed floats are incorporated by `\@makecol`; footnotes need help.

Note: FIXME: adjusting for `\pagesofar` is done at not quite the right time. I need to reexamine `\set@colht`, because `\@dbltoplist` and `\pagesofar` really

should be on the same footing. Perhaps `\@colht` and `\@colroom` should both deal with their respective "lists" in the same way?

These concerns will be particularly germane if we ever extend this package to deal with full-page-width floats placed at the bottom of the page, or committed on the same page as called out.

It occurs to me that we should ditch `\set@colroom` and only ever execute `\set@colht`, which sets `\@colroom` as a side effect. If so, we can make `\@colht` take `\pagesofar` into account, as it should. Then `\@colht` will return to its original significance as the value that `\@colroom` is set to after a column is committed.

On the other hand, why not simply forget all this caching and (re-)calculate `\vsize` as late as possible? Paticularly, `\@colht` is an artifact of the old way of doing things, where once it was set, it would never change.

```
\def\set@colht{%
 \set@adj@textheight\@colht
 \global\let\enlarge@colroom\@empty
 \set@colroom
}%
\def\set@adj@textheight#1{%
 #1\textheight
 \def\@elt{\adj@page#1}%
 \@booleantrue\firsttime@sw\@dbltoplist
 \let\@elt\relax
%\@ifvoid\pagesofar{}{%
% \advance#1-\ht\pagesofar\advance#1-\dp\pagesofar
%}%
 \global#1#1\relax
 \outputdebug@sw{\saythe#1}{}%
}%
\def\set@colroom{%
 \set@adj@colht\@colroom
 \@if@empty\enlarge@colroom{}{%
  \global\advance\@colroom\enlarge@colroom\relax
 }%
 \outputdebug@sw{\saythe\@colroom}{}%
 \@ifdim{\@colroom>\topskip}{}{%
  \ltxgrid@info{Not enough room: \string\@colroom=\the\@colroom; increasing to \the\topskip}%
  \@colroom\topskip
 }%
 \global\@colroom\@colroom
 \set@vsize
}%
%
\def\set@vsize{%
 \global\vsize\@colroom
 \outputdebug@sw{\saythe\vsize}{}%
}%
%
\def\set@adj@colht#1{%
 #1\@colht
 \@ifvoid\pagesofar{}{%
  \advance#1-\ht\pagesofar\advance#1-\dp\pagesofar
 }%
 \@ifvoid\footbox{}{%
```

```
  \advance#1-\ht\footbox\advance#1-\dp\footbox
 }%
 \def\@elt{\adj@column#1}%
 \@booleantrue\firsttime@sw\@toplist
 \@booleantrue\firsttime@sw\@botlist
 \let\@elt\relax
 \outputdebug@sw{\saythe#1}{}%
}%
\def\adj@column#1#2{%
 \advance#1-\ht#2%
 \advance#1-\firsttime@sw{\textfloatsep\@booleanfalse\firsttime@sw}{\floatsep}%
}%
\def\adj@page#1#2{%
 \advance#1-\ht#2%
 \advance#1-\firsttime@sw{\dbltextfloatsep\@booleanfalse\firsttime@sw}{\dblfloatsep}%
}%
```

`\@outputpage` At the tail of `\@outputpage`, we set `\@colht` and the float placement parameters (this is the one point where it is appropriate to set `\@colht`). At `\do@startpage` time, we adjust `\@colht`'s value to reflect committed full-page-width floats.

Note: with a correctly written output routine, a call to `\@outputpage` will inevitably be followed by a call to `\do@startpage`, so these procedure calls would be unneeded.

```
\appdef\@outputpage{%
 \set@colht          % FIXME: needed?
 \@floatplacement    % FIXME: needed?
 \@dblfloatplacement % FIXME: needed?
}%
```

`balance@2` We define procedures for balancing columns in a multicolumn layout. For now, we define only one: a procedure for the two-column grid. All others will simply `\relax` out.

```
\@namedef{balance@2}{%
 \expandafter\balance@two\csname col@1\endcsname\@outputbox
 % Avoid a bug by preventing a restore when leaving this group
 \global\setbox\csname col@1\endcsname\box\csname col@1\endcsname
 \@ifvoid\footbox{}{%
  \global\setbox\footbox\vbox{%
   \setbox\z@\box\@tempboxa
   \let\recover@footins\relax
   \balance@two\footbox\@tempboxa
   \hb@xt@\textwidth{\box\footbox\hfil\box\@tempboxa}%
  }%
 }%
}%
```

`\balance@two` The procedure `\balance@two` takes two columns and balances them; in the process it removes any footnotes that may be present to a place of safety, for later placement at the foot of the shipped-out page. The box register `\box\@ne` is the aggregate of all columns. The box register `\box\z@` is the last column. The box register `\box\tw@` is the first column. The `\dimen` register `\dimen@` is the trial value to balance to, initially half the height of `\box\@ne`. The `\dimen` register `\dimen@i` is the increment for the next trial; its initial value is equal to the initial

value of \dimen@. The \dimen register \dimen@ii is the difference of the heights of the two columns.

The procedure uses a binary search for that value of \dimen@ which is stable to within .5\p@ and which makes the last column be shorter than the others.

This procedure can be extended to multiple columns simply by changing it to execute \vsplit multiple times (one less than the total number of columns in the page layout) and to calculating \dimen@ii to be the difference of the heights of last column and the \dimen@. Upon termination of the search, one would excute the \vsplits once again, this time using the actual \col@ box registers to store the balanced columns, thereby clobbering their former contents.

Bug Note: as originally written, this macro had a bug, which is well worth avoiding under similar circumstances anywhere. So, learn from the mistakes of others, as they say. In trying to remove the depth of the boxes created via \vsplit within the \loopwhile control, I originally coded \unvbox\z@ \setbox\z@\lastbox \dimen@\dp\z@ \box\z@ \vskip- \dimen@. The error here is that the shift of the last box in the vertical list will be lost in the process. Simply put, \setbox\z@\lastbox fails to retain the shift of the box node in the vertical list, and when it is put down again via \box\z@, it will no longer have the correct shift.

This bug affected things placed in the MVL with \moveleft, \moveright, \parshape, and \hangindent, as well as things shifted by TEX's primitive mechanisms.

A superior strategy for removing the depth of the last line of the list is more expensive, but safer: make a separate copy of the list, measure the depth of the last box as above, but then discard the list, retaining only the value of the dimension.

Note that this procedure will not work if the material within is excessively chunky. A particular failure mode exists where none of the material is allocated to the last (right) column. We detect this case and revert to unbalanced columns.

Another failure mode is where a large chunk occurs at the beginning of the composite box. In this case, the left column may fill up even when \dimen@ is very small. If this configuration leaves the left column longer than the right, then we are done, but \dimen@ by no means represents the height of either finished box.

Therefore the last step in the process is to rebox the two columns to a common height determined independently of the balancing process.

The dimension involved is checked against the current \@colroom to guard against the case where excessive material happens to fall in either column.

```
\def\balance@two#1#2{%
\outputdebug@sw{{\tracingall\scrollmode\showbox#1\showbox#2}}{}%
 \setbox\@ne\vbox{%
  \@ifvoid#1{}{%
   \unvcopy#1\recover@footins
   \@ifvoid#2{}{\marry@baselines}%
  }%
  \@ifvoid#2{}{%
   \unvcopy#2\recover@footins
  }%
 }%
 \dimen@\ht\@ne\divide\dimen@\tw@
 \dimen@i\dimen@
 \vbadness\@M
 \vfuzz\maxdimen
```

```
 \loopwhile{%
  \dimen@i=.5\dimen@i
  \outputdebug@sw{\saythe\dimen@\saythe\dimen@i\saythe\dimen@ii}{}%
  \setbox\z@\copy\@ne\setbox\tw@\vsplit\z@ to\dimen@
  \setbox\z@ \vbox{%
   \unvcopy\z@
   \setbox\z@\vbox{\unvbox\z@ \setbox\z@\lastbox\aftergroup\vskip\aftergroup-\expandafter}\the
  }%
  \setbox\tw@\vbox{%
   \unvcopy\tw@
   \setbox\z@\vbox{\unvbox\tw@\setbox\z@\lastbox\aftergroup\vskip\aftergroup-\expandafter}\the
  }%
  \dimen@ii\ht\tw@\advance\dimen@ii-\ht\z@
  \@ifdim{\dimen@i>.5\p@}{%
   \advance\dimen@\@ifdim{\dimen@ii<\z@}{}{-}\dimen@i
   \true@sw
  }{%
   \@ifdim{\dimen@ii<\z@}{%
    \advance\dimen@\tw@\dimen@i
    \true@sw
   }{%
    \false@sw
   }%
  }%
 }%
 \outputdebug@sw{\saythe\dimen@\saythe\dimen@i\saythe\dimen@ii}{}%
\@ifdim{\ht\z@=\z@}{%
\@ifdim{\ht\tw@=\z@}{%
\true@sw
}{%
\false@sw
}%
}{%
\true@sw
}%
{%
}{%
\ltxgrid@info{Unsatifactorily balanced columns: giving up}%
\setbox\tw@\box#1%
\setbox\z@ \box#2%
}%
 \setbox\tw@\vbox{\unvbox\tw@\vskip\z@skip}%
 \setbox\z@ \vbox{\unvbox\z@ \vskip\z@skip}%
 \set@colroom
\dimen@\ht\z@\@ifdim{\dimen@<\ht\tw@}{\dimen@\ht\tw@}{}%
\@ifdim{\dimen@>\@colroom}{\dimen@\@colroom}{}%
 \outputdebug@sw{\saythe{\ht\z@}\saythe{\ht\tw@}\saythe\@colroom\saythe\dimen@}{}%
\setbox#1\vbox to\dimen@{\unvbox\tw@\unskip\raggedcolumn@skip}%
\setbox#2\vbox to\dimen@{\unvbox\z@ \unskip\raggedcolumn@skip}%
\outputdebug@sw{{\tracingall\scrollmode\showbox#1\showbox#2}}{}%
}%
```

`\recover@footins` The procedure `\recover@footins` is the utility procedure for recovering the footnotes from the bottom of a column. It is used when the page grid is changed, so

that footnotes can be set at the bottom of the shipped out page.

```
\def\recover@footins{%
 \skip\z@ \lastskip\unskip
 \skip\@ne\lastskip\unskip
 \setbox\z@\lastbox
 \@ifvbox\z@{%
\setbox\z@\vbox{%
\unvbox\z@
\setbox\z@\lastbox
% \outputdebug@sw{{\tracingall\showbox\lastbox}}{}%
\@ifvoid\z@{}{%
\global\setbox\footbox\vbox{%
 \unvbox\footbox
      \@ifvbox\z@{%
   \unvbox\z@
  }{%
   \box\z@
  }%
}%
}%
}%
 }{}%
 \outputdebug@sw{{\tracingall\scrollmode\showbox\footbox}}{}%
}%
```

`\@begindocumenthook` Initialization: we initialize to the page grid named "one". If the class decides to initially set type in a different grid, it should execute these same commands, but changing the first to the appropriate procedure.

Note that the point where this sequence is executed would be an excellent place to arrange for floats to be committed to the first page of a document. That is, we execute `\do@startpage`, which triggers `\do@startcolumn`.

FIXME: it should be the job of the page grid to determine the procedure to execute at the start of the job. Make this a hook.

```
\prepdef\@begindocumenthook{%
 \open@column@one\@ne
 \set@colht
 \@floatplacement
 \@dblfloatplacement
}%
```

Comment: our technique of balancing columns is severely limited, because it cannot properly work with `longtable`, which places material at the bottom and top of the column break.

The proper way to handle a grid change in the middle of the page is to accumulate all the material for an entire article (or chapter) and then assemble finished pages therefrom. This approach is fundamentally superior for complex layouts: it corresponds to real-world workflows. Such a scheme is an excellent subject for another LaTeX package.

8.15 Patches for the longtable package

LaTeX's "required" package `longtable` (written by David P. Carlilsle), which is part of /latex/required/tools, is incmpatible with both LaTeX's "required" pack-

age `multicol` and with LaTeX's native `\twocolumn` capability. There is no essential reason for this incompatability, aside from implementation details, and the `ltxgrid` package gives us the ability to lift them.

Only four of `longtable`'s procedures require rewriting: `\longtable`, `\endlongtable`, `\LT@start`, and `\LT@end@hd@ft`. The procedure `\switch@longtable` checks against their expected meanings and, if all is as expected, applies the patches. In the process, we simplify things considerably and also make them more secure.

Why does `longtable` need to access the output routine, anyway? What it comes down to, is what happens when a pagebreak falls within a long table. If this happens, we would like to append a row at the bottom of the broken table and add a row at the top of the next page.

These things can be accomodated easily by the `ltxgrid` output routine hooks.

`\longtable`

```
\def\longtable@longtable{%
 \par
 \ifx\multicols\@undefined\else\ifnum\col@number>\@ne\@twocolumntrue\fi\fi
 \if@twocolumn\LT@err{longtable not in 1-column mode}\@ehc\fi
 \begingroup
 \@ifnextchar[\LT@array{\LT@array[x]}%
}%
\def\longtable@new{%
 \par
  \@ifnextchar[\LT@array{\LT@array[x]}%
}%
```

`\endlongtable`

```
\def\endlongtable@longtable{%
  \crcr
  \noalign{%
    \let\LT@entry\LT@entry@chop
    \xdef\LT@save@row{\LT@save@row}}%
  \LT@echunk
  \LT@start
  \unvbox\z@
  \LT@get@widths
  \if@filesw
    {\let\LT@entry\LT@entry@write\immediate\write\@auxout{%
      \gdef\expandafter\noexpand
        \csname LT@\romannumeral\c@LT@tables\endcsname
          {\LT@save@row}}}%
  \fi
  \ifx\LT@save@row\LT@@save@row
  \else
    \LT@warn{Column \@width s have changed\MessageBreak
             in table \thetable}%
    \LT@final@warn
  \fi
  \endgraf\penalty -\LT@end@pen
  \endgroup
  \global\@mparbottom\z@
  \pagegoal\vsize
  \endgraf\penalty\z@\addvspace\LTpost
```

```
  \ifvoid\footins\else\insert\footins{}\fi
}%
\def\endlongtable@new{%
  \crcr
  \noalign{%
   \let\LT@entry\LT@entry@chop
   \xdef\LT@save@row{\LT@save@row}%
  }%
  \LT@echunk
  \LT@start
  \unvbox\z@
  \LT@get@widths
  \@if@sw\if@filesw\fi{%
   {%
    \let\LT@entry\LT@entry@write
    \immediate\write\@auxout{%
     \gdef\expandafter\noexpand\csname LT@\romannumeral\c@LT@tables\endcsname
     {\LT@save@row}%
    }%
   }%
  }{}%
  \@ifx\LT@save@row\LT@@save@row{}{%
   \LT@warn{%
    Column \@width s have changed\MessageBreak in table \thetable
   }\LT@final@warn
  }%
  \endgraf
  \nobreak
  \box\@ifvoid\LT@lastfoot{\LT@foot}{\LT@lastfoot}%
 \global\@mparbottom\z@
 \endgraf
 \LT@post
}%
```

\LT@start

```
\def\LT@start@longtable{%
 \let\LT@start\endgraf
 \endgraf
 \penalty\z@
 \vskip\LTpre
 \dimen@\pagetotal
 \advance\dimen@ \ht\ifvoid\LT@firsthead\LT@head\else\LT@firsthead\fi
 \advance\dimen@ \dp\ifvoid\LT@firsthead\LT@head\else\LT@firsthead\fi
 \advance\dimen@ \ht\LT@foot
 \dimen@ii\vfuzz\vfuzz\maxdimen
   \setbox\tw@\copy\z@
   \setbox\tw@\vsplit\tw@ to \ht\@arstrutbox
   \setbox\tw@\vbox{\unvbox\tw@}%
 \vfuzz\dimen@ii
 \advance\dimen@ \ht
       \ifdim\ht\@arstrutbox>\ht\tw@\@arstrutbox\else\tw@\fi
 \advance\dimen@\dp
       \ifdim\dp\@arstrutbox>\dp\tw@\@arstrutbox\else\tw@\fi
 \advance\dimen@ -\pagegoal
```

```
\ifdim \dimen@>\z@\vfil\break\fi
    \global\@colroom\@colht
\ifvoid\LT@foot\else
  \advance\vsize-\ht\LT@foot
  \global\advance\@colroom-\ht\LT@foot
  \dimen@\pagegoal\advance\dimen@-\ht\LT@foot\pagegoal\dimen@
  \maxdepth\z@
\fi
\ifvoid\LT@firsthead\copy\LT@head\else\box\LT@firsthead\fi
\output{\LT@output}%
}%
\def\LT@start@new{%
\let\LT@start\endgraf
\endgraf
\markthr@@{}%
\LT@pre
\@ifvoid\LT@firsthead{\LT@top}{\box\LT@firsthead\nobreak}%
\mark@envir{longtable}%
}%
```

\LT@end

```
\def\LT@end@hd@ft@longtable#1{%
\LT@echunk
\ifx\LT@start\endgraf
 \LT@err{Longtable head or foot not at start of table}{Increase LTchunksize}%
\fi
\setbox#1\box\z@
\LT@get@widths\LT@bchunk
}%
\def\LT@end@hd@ft@new#1{%
\LT@echunk
\@ifx{\LT@start\endgraf}{%
 \LT@err{Longtable head or foot not at start of table}{Increase LTchunksize}%
}%
\global\setbox#1\box\z@
\LT@get@widths
\LT@bchunk
}%
```

\LT@array

```
\def\LT@array@longtable[#1]#2{%
  \refstepcounter{table}\stepcounter{LT@tables}%
  \if l#1%
    \LTleft\z@ \LTright\fill
  \else\if r#1%
    \LTleft\fill \LTright\z@
  \else\if c#1%
    \LTleft\fill \LTright\fill
  \fi\fi\fi
  \let\LT@mcol\multicolumn
  \let\LT@@tabarray\@tabarray
  \let\LT@@hl\hline
  \def\@tabarray{%
    \let\hline\LT@@hl
```

```
    \LT@@tabarray}%
  \let\\\LT@tabularcr\let\tabularnewline\\%
  \def\newpage{\noalign{\break}}%
  \def\pagebreak{\noalign{\ifnum`}=0\fi\@testopt{\LT@no@pgbk-}4}%
  \def\nopagebreak{\noalign{\ifnum`}=0\fi\@testopt\LT@no@pgbk4}%
  \let\hline\LT@hline \let\kill\LT@kill\let\caption\LT@caption
  \@tempdima\ht\strutbox
  \let\@endpbox\LT@endpbox
  \ifx\extrarowheight\@undefined
    \let\@acol\@tabacol
    \let\@classz\@tabclassz \let\@classiv\@tabclassiv
    \def\@startpbox{\vtop\LT@startpbox}%
    \let\@@startpbox\@startpbox
    \let\@@endpbox\@endpbox
    \let\LT@LL@FM@cr\@tabularcr
  \else
    \advance\@tempdima\extrarowheight
    \col@sep\tabcolsep
    \let\@startpbox\LT@startpbox\let\LT@LL@FM@cr\@arraycr
  \fi
  \setbox\@arstrutbox\hbox{\vrule
    \@height \arraystretch \@tempdima
    \@depth \arraystretch \dp \strutbox
    \@width \z@}%
  \let\@sharp##\let\protect\relax
   \begingroup
    \@mkpream{#2}%
    \xdef\LT@bchunk{%
       \global\advance\c@LT@chunks\@ne
       \global\LT@rows\z@\setbox\z@\vbox\bgroup
       \LT@setprevdepth
       \tabskip\LTleft\halign to\hsize\bgroup
      \tabskip\z@ \@arstrut \@preamble \tabskip\LTright \cr}%
  \endgroup
  \expandafter\LT@nofcols\LT@bchunk&\LT@nofcols
  \LT@make@row
  \m@th\let\par\@empty
  \everycr{}\lineskip\z@\baselineskip\z@
  \LT@bchunk
}%
\def\LT@LR@l{\LTleft\z@   \LTright\fill}%
\def\LT@LR@r{\LTleft\fill \LTright\z@  }%
\def\LT@LR@c{\LTleft\fill \LTright\fill}%
\def\LT@array@new[#1]#2{%
 \refstepcounter{table}\stepcounter{LT@tables}%
 \table@hook
 \LTleft\fill \LTright\fill
 \csname LT@LR@#1\endcsname
 \let\LT@mcol\multicolumn
 \let\LT@@hl\hline
 \prepdef\@tabarray{\let\hline\LT@@hl}%
 \let\\\LT@tabularcr
 \let\tabularnewline\\%
 \def\newpage{\noalign{\break}}%
```

```
\def\pagebreak{\noalign{\ifnum`}=0\fi\@testopt{\LT@no@pgbk-}4}%
\def\nopagebreak{\noalign{\ifnum`}=0\fi\@testopt\LT@no@pgbk4}%
\let\hline\LT@hline
\let\kill\LT@kill
\let\caption\LT@caption
\@tempdima\ht\strutbox
\let\@endpbox\LT@endpbox
\@ifxundefined\extrarowheight{%
 \let\@acol\@tabacol
 \let\@classz\@tabclassz
 \let\@classiv\@tabclassiv
 \def\@startpbox{\vtop\LT@startpbox}%
 \let\@@startpbox\@startpbox
 \let\@@endpbox\@endpbox
 \let\LT@LL@FM@cr\@tabularcr
}{%
 \advance\@tempdima\extrarowheight
 \col@sep\tabcolsep
 \let\@startpbox\LT@startpbox
 \let\LT@LL@FM@cr\@arraycr
}%
%
 \let\@acoll\@tabacoll
 \let\@acolr\@tabacolr
 \let\@acol\@tabacol
%
 \setbox\@arstrutbox\hbox{%
  \vrule
  \@height \arraystretch \@tempdima
  \@depth \arraystretch \dp \strutbox
  \@width \z@
 }%
 \let\@sharp##%
 \let\protect\relax
 \begingroup
  \@mkpream{#2}%
  \@mkpream@relax
  \edef\@preamble{\@preamble}%
  \prepdef\@preamble{%
   \global\advance\c@LT@chunks\@ne
   \global\LT@rows\z@
   \setbox\z@\vbox\bgroup
    \LT@setprevdepth
    \tabskip\LTleft
    \halign to\hsize\bgroup
     \tabskip\z@
     \@arstrut
  }%
  \appdef\@preamble{%
     \tabskip\LTright
     \cr
  }%
  \global\let\LT@bchunk\@preamble
 \endgroup
```

```
\expandafter\LT@nofcols\LT@bchunk&\LT@nofcols
\LT@make@row
\m@th
\let\par\@empty
\everycr{}%
\lineskip\z@
\baselineskip\z@
\LT@bchunk
}%
\appdef\table@hook{}%
```

`\switch@longtable` Here is the switch from standard `longtable` to the new, `ltxgrid`-compatible values.

At this point, we extend `longtable` with a `longtable*` form, which signifies that we want to use the full page width for setting the table. You can think this way: `longtable*` is to `longtable` as `table*` is to `table`.

FIXME: the following is no longer true: Note that it is not enough to define the environment itself; we also have to create the corresponding `\output` routine procedures, which provide for continued footers and headers (the very feature of `longtable` requiring support in the output routine).

This same consideration would arise in defining any syntactic extension to `longtable`, because the environment name itself is exposed in the output routine.

```
\def\switch@longtable{%
\@ifpackageloaded{longtable}{%
 \@ifx{\longtable\longtable@longtable}{%
  \@ifx{\endlongtable\endlongtable@longtable}{%
   \@ifx{\LT@start\LT@start@longtable}{%
    \@ifx{\LT@end@hd@ft\LT@end@hd@ft@longtable}{%
     \@ifx{\LT@array\LT@array@longtable}{%
      \true@sw
     }{\false@sw}%
    }{\false@sw}%
   }{\false@sw}%
  }{\false@sw}%
 }{\false@sw}%
 {%
  \class@info{Patching longtable package}%
 }{%
  \class@info{Patching unrecognized longtable package. (Proceeding with fingers crossed)}%
 }%
 \let\longtable\longtable@new
 \let\endlongtable\endlongtable@new
 \let\LT@start\LT@start@new
 \let\LT@end@hd@ft\LT@end@hd@ft@new
 \let\LT@array\LT@array@new
 \newenvironment{longtable*}{%
  \onecolumngrid@push
  \longtable
 }{%
  \endlongtable
  \onecolumngrid@pop
 }%
% \expandafter\let\csname output@init@longtable*\endcsname\output@init@longtable
```

```
% \expandafter\let\csname output@prep@longtable*\endcsname\output@prep@longtable
% \expandafter\let\csname output@post@longtable*\endcsname\output@post@longtable
 }{}%
}%
```

Note that at the end of the longtable environment, we reestablish the `\mark@envir` of the containing environment. We have left `\curr@envir` alone, so this will work.

`\LT@pre` `\LT@bot` `\LT@top` `\LT@post` `\LT@adj`

```
\def\LT@pre{\penalty\z@\vskip\LTpre}%
\def\LT@bot{\nobreak\copy\LT@foot\vfil}%
\def\LT@top{\copy\LT@head\nobreak}%
\def\LT@post{\penalty\z@\addvspace\LTpost\mark@envir{\curr@envir}}%
\def\LT@adj{%
 \setbox\z@\vbox{\null}\dimen@-\ht\z@
 \setbox\z@\vbox{\unvbox\z@\LT@bot}\advance\dimen@\ht\z@
 \global\advance\vsize-\dimen@
}%
```

`output@init` `output@prep` `output@post`

```
\def\output@init@longtable{\LT@adj}%
\def\output@prep@longtable{\setbox\@cclv\vbox{\unvbox\@cclv\LT@bot}}%
\def\output@post@longtable{\LT@top}%
```

8.16 Patches for index processing

Another feature that uses the output routine hooks occurs within an index, where one wishes to apply a "continue head" when a column breaks within a primary index entry. Some book designs call for the continue head to only be applied at a turnpage break.

In any case, it is easy enough for `\output@post@theindex` to do this in conjunction with component marks. Only the bare outlines are shown here.

`\output@init` `\output@prep` `\output@post`

```
\let\output@init@theindex\@empty
\let\output@prep@theindex\@empty
\def\output@post@theindex{%
 \@ifodd\c@page{}{%
  \@ifnum{\pagegrid@cur=\@ne}{% we have the leftmost column of a verso page
   % insert the current top-level continued head
  }%
 }%
}%
```

8.17 Checking the auxiliary file

We relegate the checking of the auxiliary file to the output routine. This task must wait until the last page is shipped out, because otherwise the stream might get closed before the last page is shipped out. Obviously, we must use `\do@output@MVL` for the job.

`\check@aux`

```
\def\check@aux{\do@output@MVL{\do@check@aux}}%
```

8.18 Dealing with stuck floats and stalled float dequeueing

LaTeX's float placement mechanism is fundamentally flawed, as evidenced by its warning message "too many unprocessed floats", which users understandably find frustrating. The `ltxgrid` package provides tools for ameliorating the situation somewhat.

Two cases require detection and rectification:

1. A float is "stuck" in the `\@deferlist`: for whatever reason, the float fails to be committed, even at the start of a fresh page. Once this condition prevails, following floats can never be committed, subsequently all of LaTeX's float registers are used up.

   If this condition is detected, we reconsider float dequeueing under permissive (`\clearpage`-style) processing.

2. The `\@freelist` is exhausted: a large concentration of floats, say, uses up all of LaTeX's float registers all at once. This condition commonly occurs when the user collects floats at the end of the document, for some reason.

   When a float is encountered, LaTeX uses a float register (allocated from a pool of free registers) to contain it until it can be placed. However, no further action is taken until the pagebuilder is visited, so floats can accumulate. Also, even after the pagebuilder is visited, deferred floats can accumulate, and these are not committed until a column (or page) of text is completed.

   Once the last free float register is used, action should be taken that will commit some of the deferred floats, even if this might require ending the page right where we are (resulting in a short page).

   Perhaps, committed floats should be stored using some mechanism other than a list, as is currently done. A feasible alternative storage method would be to use a `\box` register in place of `\@toplist`, `\@botlist`, and `\@dbltoplist`. This is probably just fine, since such committed floats are not reconsidered (I think).

The emergency processing implemented here immediately ends the current page and begins to output float pages under (`\clearpage`-style) rules. It proceeds until all deferred floats have been flushed.

Users should expect non-optimal page makeup under these circumstances.

Note that there is a weakness in our approach that we have not attempted to repair: if floats are being added as part of a paragraph, we will not be able to take these remedial steps until the paragraph ends. This means that the approach implemented here cannot fix all LaTeX documents. Users can still construct documents that exhaust LaTeX's pool of float registers!

`\check@deferlist@stuck`
`\@outputpage`

We detect the case where, at the start of a fresh page, there are deferred floats, but none are committed. We memorize the `\@deferlist` at `\shipout` time, then examine it at the point where our efforts to commit floats to the new page are complete. If it has not changed, the first float must be stuck, and we attempt to fix things via `\force@deferlist@stuck`.

This simple approach is comp[letely effective in for typical documents.

Note that we try to avoid an infinite loop by examining the value of `\clearpage@sw`: if we come here with that boolean true, we are in a loop.

```
\def\check@deferlist@stuck#1{%
 \@ifx{\@deferlist@postshipout\@empty}{}{%
  \@ifx{\@deferlist@postshipout\@deferlist}{%
   \@fltstk
   \clearpage@sw{%
    \ltxgrid@warn{Deferred float stuck during \string\clearpage\space processing}%
   }{%
    \force@deferlist@stuck#1%
   }%
  }{%
   %Successfully committed float(s)
  }%
  \global\let\@deferlist@postshipout\@empty
 }%
}%
\def\@fltstk{%
 \@latex@warning{A float is stuck (cannot be placed without \string\clearpage)}%
}%
\appdef\@outputpage{%
 \global\let\@deferlist@postshipout\@deferlist
}%
```

`\@next` `\@xnext` We rewrite the LaTeX kernel macros that dequeue float registers from, e.g., `\@deferlist`, providing a test for the condition where the pool of free registers is about to underflow.

In this case, we attempt to fix things via `\force@deferlist@empty`.

```
\def\@next#1#2{%
 \@ifx{#2\@empty}{\false@sw}{%
  \expandafter\@xnext#2\@@#1#2%
  \true@sw
 }%
}%
\def\@xnext\@elt#1#2\@@#3#4{%
 \def#3{#1}%
 \gdef#4{#2}%
 \def\@tempa{#4}\def\@tempb{\@freelist}%
 \@ifx{\@tempa\@tempb}{%
  \@ifx{#4\@empty}{%
   \force@deferlist@empty%{Float register pool exhausted}%
  }{}%
 }{}%
}%
```

`\force@deferlist@stuck` `\force@deferlist@empty` `\force@deferlist@sw` `\do@forcecolumn@pen` `\do@forcecolumn` The procedure `\force@deferlist@empty` is an attempt to rectify a situation where LaTeX's float placement mechanism may fail ("too many unprocessed floats").

We put down interrupts that call for the float placement to be redone, but under permissive conditions, just the same as if `\clearpage` had been invoked.

Note that the attempt to rectify the error is contingent on the setting of `\force@deferlist@sw`, default false. A document class using this package that wishes to enable this error recovery mechanism should set this boolean to true.

The interrupt `\do@forcecolumn@pen`, which invokes the procedure `\do@forcecolumn`, does the same as `\do@startcolumn`, except under permissive conditions: we are

trying to empty out the float registers completely.

In order to properly with the case where there is material in \box\@cclv, \@toplist, \@botlist, \@dbltoplist, etc, we do what amounts to \newpage to get things rolling.

In \force@deferlist@stuck, we take advantage of already being in the output routine: simply reinvoke \do@startcolumn under permissive conditions.

```
\def\force@deferlist@stuck#1{%
\force@deferlist@sw{%
  \@booleantrue\clearpage@sw
  \@booleantrue\forcefloats@sw
  #1%
}{%
}%
}%
\def\force@deferlist@empty{%
 \force@deferlist@sw{%
% \ltxgrid@info{#1, attempting rectification}%
  \penalty-\pagebreak@pen
  \protect@penalty\do@forcecolumn@pen
 }{%
% \ltxgrid@info{#1}%
 }%
}%
\@booleanfalse\force@deferlist@sw
\mathchardef\do@forcecolumn@pen=10009
\@namedef{output@-\the\do@forcecolumn@pen}{\do@forcecolumn}%
\def\do@forcecolumn{%
 \@booleantrue\clearpage@sw
 \@booleantrue\forcefloats@sw
%\unvbox\@cclv
%\vfil
%\penalty-\pagebreak@pen
 \do@startcolumn
}%
```

A more thorough revision of LaTeX's float placement mechanism would involve substituting a single \box register for the \@deferlist. This way, LaTeX's ability to have latent floats would be limited by box memory alone.

Because only the \box and \count components of the float box register are actually used by LaTeX, our scheme can be accomplished if we can find a way to encode the information held in the \count component.

A first-in, first-out mechanism exists, wherein a box-penalty pair is dequeued by \lastbox\lastpenalty\unpenalty and enqueued by \setbox\foo=\hbox\bgroup\penalty\floatp

Note that this scheme is made possible by our change to LaTeX's float placement mechanism, wherein we consolidated the two \@deferlists into one.

9 Support for legacy LaTeX commands

We provide support for the \enlargethispage command.

Note: using a command of this sort is questionable. Instead, people should enlarge the entire spread.

Timing Note: In a multicolumn page grid, the user should issue the `\enlargethispage` command while the first column of the page is being typeset. We provide a helpful message if the timing is wrong.

This code can serve as a model for introducing commands that need to execute within the safety of the output routine. We ensure that the arguments are fully expanded, then execute `\do@output@MVL` to cause an output procedure, `\@@enlargethispage`, to execute. When it does execute, the MVL will be exposed.

The `\@@enlargethispage` procedure simply adjusts the vertical dimensions of the page. The adjustment will persist until the column is committed, at which point the page dimension will revert to its standard value.

```
\def\enlargethispage{%
\@ifstar{%
\@enlargethispage{}%
}{%
\@enlargethispage{}%
}%
}%
\def\@enlargethispage#1#2{%
 \begingroup
  \dimen@#2\relax
  \edef\@tempa{#1}%
  \edef\@tempa{\noexpand\@@enlargethispage{\@tempa}{\the\dimen@}}%
  \expandafter\do@output@MVL\expandafter{\@tempa}%
 \endgroup
}%
\def\@@enlargethispage#1#2{%
 \def\@tempa{one}%
 \@ifx{\thepagegrid\@tempa}{%
  \true@sw
 }{%
  \def\@tempa{mlt}%
  \@ifx{\thepagegrid\@tempa}{%
   \@ifnum{\pagegrid@cur=\@ne}{% OK to adjust this page
    \gdef\enlarge@colroom{#2}%
    \true@sw
   }{% Can only adjust this column; give up
    \ltxgrid@warn{Too late to enlarge this page; move the command to the first column.}%
    \false@sw
   }%
  }{% Unknown page grid
   \ltxgrid@warn{Unable to enlarge a page of this kind.}%
   \false@sw
  }%
 }%
 {%
  \class@info{Enlarging page \thepage\space by #2}%
  \global\advance\@colroom#2\relax
  \set@vsize
 }{%
  % Could not adjust this page
 }%
}%
```

```
\let\enlarge@colroom\@empty
```

The `\@kludgeins` insert register is now unneeded. Ensure that packages using this mechanism break (preferrable to subtle bugs).

```
\let\@kludgeins\@undefined
```

9.0.1 Building the page for shipout

`\@outputpage` The procedures that build `\@outputbox` just before a page is shipped out by `\@outputpage` are: `\@makecol`, `\@combinepage`, and `\@combinedblfloats`. We headpatch `\@outputpage` to make the `\@outputbox` be of fixed height.

```
\@booleantrue\textheight@sw
\prepdef\@outputpage{%
 \textheight@sw{%
  \count@\vbadness\vbadness\@M
  \dimen@\vfuzz\vfuzz\maxdimen
  \setbox\@outputbox\vbox to\textheight{\unvbox\@outputbox}%
  \vfuzz\dimen@
  \vbadness\count@
 }{}%
}%
```

9.0.2 Warning message

`\ltxgrid@info` `\ltxgrid@warn` Something has happened that the user might be interested in. Print a message to the log, but only if the user selected the verbose option.

```
\def\ltxgrid@info{%
 \ltxgrid@info@sw{\class@info}{\@gobble}%
}%
\@booleanfalse\ltxgrid@info@sw
\def\ltxgrid@warn{%
 \ltxgrid@warn@sw{\class@warn}{\@gobble}%
}%
\@booleantrue\ltxgrid@warn@sw
```

10 End of the `ltxgrid` DOCSTRIP module

Here ends the module.

```
%</ltxgrid-krn>
```

Here ends the programmer's documentation.

Index

Symbols

A

D

E

F

G

H

I

M

N

O

P

R

S

T