The grfpaste package*

LaTeX support for TeXplorators / M. Spivak `dvipaste` program

David Carlisle

1997/07/18

Contents

`Dvipaste` comes with a plain TeX based set of macros in a file `dvipaste.tex` that provide an interface to the `dvipaste` program. That file works perfectly well in LaTeX with only minor modifications (specifically using `\@@line` instead of `\line`, and `\c@page` instead of `\pageno`). A `dvipaste.sty` constructed by just such modifications may be found distributed with Jean-Pierre Drucbert's 'export' package on CTAN. However I thought that it would be more in the 'LaTeX spirit' to provide an interface via the existing LaTeX graphics package.

The files produced by this grfpaste package should be fully compatible with `dvipaste.tex` or `dvipaste.sty`. That is, if DVI fragments are created using the plain TeX interface they may be included using `\includegraphics` via this package, and conversely if this package is used with the `[write]` or `[writepages]` options to produce a file of DVI fragments for including in another file, then the DVI file (and it's accompanying '`.dat`'

*This manual corresponds to grfpaste v0.2, dated 1997/07/18.

file of size information) may be included either with this package, or via the original `dvipaste.tex` interface.

Currently the LaTeX interface uses the graphicx package, it would be possible to also provide an interface to the more basic graphics package, but that is not done here.

The package is used in one of two forms.

1 Producing DVI Fragments

1 a Writing out boxes.

```
\documentclass{article}

\usepackage[write]{grfpaste}

\begin{document}

\sendout{\parbox[b]{2cm}{one \sf two three \tt four \it five}}

\sendout[abc]{hello}

\sendout{\parbox[t]{2cm}{one \sf two three \tt four \it five}}

\end{document}
```

If the package is used with the `[write]` option then the document should just consist of `\sendout` commands, the argument of each will be marked as a 'fragment' that may be pasted into another DVI file by the `dvipaste` program. An auxiliary file with extension `.dat` will be created which contains the size of each of these fragments. Normally the fragments will be accessed by their number, starting from 1, but you may optionally supply a label, as in '`abc`' above which allows fragments to be referenced more easily. The above example forms the example document `grfp1.tex` in this distribution.

1 b Writing out whole pages.

```
\documentclass{article}

\usepackage[writepages]{grfpaste}

\begin{document}
...
\end{document}
```

If the package is used with the `[writepages]` option then each indivdual page will be marked as a 'fragment' for later processing by `dvipaste`. The fragments will be numbered consecutively, starting from 1, whatever the value of the page counter. See the example document `grfp2.tex`.

2 Using DVI Fragments

```
\documentclass{article}

\usepackage{grfpaste}

\begin{document}

aaa\fbox{\includegraphics
       [num=1,natheight=19,natdepth=2pt,natwidth=57]{grfp1.dvi}}bbb

xxx\fbox{\includegraphics[num=2]{grfp1.dvi}}yyy

xxx\fbox{\includegraphics[ref=abc]{grfp1.dvi}}yyy

Page 3 of grfp2.dvi:
aaa\fbox{\includegraphics[num=3]{grfp2}}bbb

\end{document}
```

To use the fragments then load the package with no option (or equivalently with the `[include]` option). You may also use any options understood by the graphicx package, these will be passed on to graphicx which is loaded by this package.

To load the first fragment of a given DVI file you just need `\includegraphics{`⟨*filename*⟩`}` or `\includegraphics{`⟨*filename*⟩`.dvi}` (`.dvi` is added as one of the default extensions by this package)

To access later fragments then you use the new `num=`⟨*??*⟩ key to specify the fragment number, as demonstrated above. If there are a lot of fragments maintaining the correct number may be inconvenient so you may instead use the `ref=` key to refer to a label previously supplied by the `\sendout` command. Note this label information is added to the `.dat` file in the form of a comment (after `%`) and so the `.dat` file is still compatible with `dvipaste.tex`, but if that is used, then the numeric form must be used. Thus the two examples above with `num=2` and `ref=abc` include the same fragment from the file `grfp1.dvi`.

Normally the size of the graphic is read from the '`.dat`' file which is the 'read file' for the DVI graphic type, in the terminoligy of the graphics package documentation. If you have lost the `.dat` file, or wish to override it with altered sizes, you may specify the natural size using the `natheight`, `natwidth` and `natdepth` keys. As usual these take a dimension but the units may be omitted in which case `bp` are assumed. `natheight` and `natwidth` are standard graphicx keys, but `natdepth` is new, as these fragments are TEX boxes, so have height and depth, unlike most other graphic formats that may be included, which always have zero depth.

The `\sendout` command defined by the `[write]` option matches that used in the plain TEX support, but the plain TEX version does not use `\includegraphics` to include the fragment. It defines a command `\paste` to achieve this. In order to help move

documents between the two versions, a `[defpaste]` option is provided for this package which specifies that a compatible `\paste` command should be defined. It is defined to be the equivalent call to `\includegraphics`, `\includegraphics[num=#2]{#1}`.

See the example file `grfp3.tex` to see examples of including DVI fragments.

3 Putting it all together

a) Produce one or more files of fragments using LaTeX and this package with the `[write]` or `[writepages]` options. (or plain TeX using `dvipaste.tex`).

b) LaTeX your master document, including this package and using `\includegraphics` to include the fragments at the appropriate points.

   At this stage the file will have blank spaces (but if the correct size) at the points where the DVI fragments should appear

c) Run `dvipaste` on the master DVI file. It will incorporate the fragments and re-write the file.

d) Preview or print the modified DVI file in your usual manner.

4 Running the examples

- `grfp1.tex` is an example of using the `[write]` option
- `grfp2.tex` is an example of using the `[writepages]` option
- `grfp3.tex` is an example of using the `[include]` option

To process the final document (`grfp3.dvi`) the following steps need be taken.

```
1 latex grfp1
2 latex grfp2
3 latex grfp2
4 latex grfp3
5 dvipaste grfp3
```

Note that `grfp2` needs to be processed twice to generate a full table of contents. After `grfp3.tex` is processed by LaTeX the DVI file will show blank spaces at the points that the fragments are to be included. These will be filled in by running `dvipaste`, after which the `grfp3.dvi` may be processed by your driver in the normal way.

5 Compiling `dvipaste.c`

```
/* config.h for LINUX machines for dvipaste.c (DPC) */


#define ANSI
#undef MICROSOFTC
#define LITTLENDIAN

/* I guess this is what was meant. */
/* (dvipaste.c defines strcmp to itself */
/* leaving stricmp undefined) .*/
#ifndef MICROSOFTC
#define strnicmp strncmp
#define stricmp strcmp
#endif

#ifdef ANSI
/* Other things in ANSI C are good, but this is a crock */
#define READ_BINARY "rb"
#define WRITE_BINARY "wb"
#else
/* The good old simple and logical K&R I/O */
#define READ_BINARY "r"
#define WRITE_BINARY "w"
#endif
```

When I tried compiling `dvipaste.c` I got error due to the function `stricmp` being undefined. The file has a circular definition of `strcmp` to itself which looks like a typo, so I corrected that by making `stricmp` an alias for `strcmp` by altering `config.h` as above (which also sets `LITTLENDIAN` for Linux. It seems to work for me, but no promises.

The above suggested `config.h` leaves `stricmp` meaning the same thing as `strcmp` (string comparison). Jean-Pierre Drucbert points out that `stricmp` is (on systems that it is defined) a case-insensitive comparison function, and that rather than the crude substitution above one may define it as follows.

> `str{n}icmp` is not available on all systems. So I got the following code in `ci.c` :

```
#include <ctype.h>
#include "config.h"
/*
* str{n}icmp - case-insensitive flavors of strcmp(), strncmp()
*/
#ifdef PROTOS
int stricmp(register char *s1,
register char *s2)
#else
int stricmp(s1, s2)
register char *s1, *s2;
```

```
#endif
{
  register char c1, c2;

  for ( ; (c1 = TOLOWER(*s1)) == (c2 = TOLOWER(*s2)); s1++, s2++)
  if (c1 == '\0')
  return 0;

  return c1 - c2;
}


#ifdef PROTOS
int strnicmp(register char *s1,
register char *s2,
int n)
#else
int strnicmp(s1, s2, n)
register char *s1, *s2;
int n;
#endif
{
  register char c1, c2;

  for ( ; --n >= 0 &&
  (c1 = TOLOWER(*s1)) == (c2 = TOLOWER(*s2)); s1++, s2++)
  if (c1 == '\0')
  return 0;

  return n < 0 ? 0 : c1 - c2;
}
```

By default `dvipaste.c` has

```
#define STRINGSIZE (5124) /* maximum string size */
```

Jean-Pierre Drucbert commented:

> it is necessary to increase `STRINGSIZE` if you paste a lot of DVI pieces (I increased it from 40592 to 2597888). In practice, do not export more than ~100 pieces from one file. You can import a lot more into one file. So it is preferable to import from several files if you have more than 100 pieces to import.