requires some changes in origin placement but is generally fairly simple.

Future printing systems

Although high-speed, high-volume printing systems do not yet offer color printing capabilities, future systems likely will. Through TEX macros and `\special` commands, the user should be able to control the color of the text and/or graphics on a page.

Binding is another feature which may be offered by newer printing systems. If a printer has options for user-controlled binding, the DVI driver should be able to support them.

Any standards or guidelines which are developed concerning `\special`s and DVI drivers need to be open-ended so that capabilities of future printing systems can be incorporated into them.

Conclusion

Demand printing applications made possible by high-speed, high-volume printing systems make additional demands of the document preparation software. TEX represents a sound system for handling the typesetting aspect of the document preparation. However, DVI drivers need to take on a more prominent role in the document preparation cycle by providing the non-typesetting capabilities needed for support of these printing systems.

Since the significance of the responsibilities of the DVI driver are raised to the level of those of TEX, more attention to the DVI driver is needed. While TEX represents a standard for typesetting, no such standard currently exists for DVI drivers — each driver author is left to his or her own imagination. Efforts are presently underway to propose a standard for `\special`s; similar efforts are needed to formulate standards and/or guidelines for DVI drivers.

`\special` issues

Glenn L. Vanderburg and Thomas J. Reid
Texas A&M University

ABSTRACT: As more and more DVI translation programs appear, it is important to have a standard set of `\special` commands, and a standard format for those commands, so that attention can be focused on the documents being prepared rather than on the printer being used. This article contains a discussion of the various problems associated with specials and describes a format and set of specials which could serve as the foundation for a standard.

One of the most farsighted and useful features of the TEX language is the `\special` command. In acknowledging the incompleteness of TEX and of the DVI format, Professor Knuth gave us the mechanism by which we could do much to extend the language and adapt it to changing needs and capabilities.

In *The TEXbook* ([16] page 229), Knuth has this to say:

> Whenever you use `\special`, you are taking a chance that your output file will not be printable on all output devices, because all `\special` functions are extensions to TEX. However, the author anticipates that certain standards for common graphic operations will emerge in the TEX user community, after careful experiments have been made by different groups of people; then there will be a chance for some uniformity in the use of `\special` extensions.

The time has come to define the standards which Knuth proposed. The TEX community is maturing rapidly. DVI drivers exist for many printers, but most such programs are still being developed. Powerful page-description languages now exist, and TEX users are eager to exploit their capabilities. Many drivers have implemented some specials, and some lessons have been learned. Questions and articles in *TUGboat* and *TEXhax* have demonstrated the kind of functionality which is required. A standard for the format (and function) of special commands is important, and it will be most useful now, while so many drivers are being developed. The authors have been considering this problem for more than a year, in connection with the development of a DVI driver for the Xerox 9700 family of printers, and would like to present their ideas and suggestions. Many of the ideas in this article are due (at least in part) to Dr. Bart Childs, who has spent a great deal of time considering these issues and sharing his ideas. This is not meant to imply that he endorses the recommendations here, as his original ideas have been modified considerably. His input was important, however.

We also were influenced by Robert McGaffey [18], Don Hosek [12], and others too numerous to name; we've seen many drivers and talked to many users about their needs. *TUGboat* and *TEXhax* have been invaluable aids, with articles and discussion about specials and the problems that TEX users face.

We hope that the TEX community, and specifically the TEX Users Group (TUG), can adopt a standard which will detail the recommended behavior for all DVI translation programs. It will not be necessary for every driver to implement each special command in the standard, but each driver should support a minimal subset of the standard specials, so that users will know what to expect with regard to certain very basic operations.

The remainder of this article considers problems of function, form, and use of specials, along with the problem of device-dependence.

Guidelines and Goals

There are certain guidelines (or design principles) which should govern the standard, and certain goals to drive it.

- Any DVI file should work with any driver as well as possible. Unrecognized specials should not cause problems, and a given special should produce equivalent effects (where possible) on all printers. In particular, this means that specials should not be tagged with driver or printer names.
- No device-specific information should be included in special commands, except where absolutely necessary.
- Keep specials general. When there are frequently used special cases, implement them as aliases for a more general special (although ideally those special cases should be accessed by a TEX macro to generate the special).
- Text should be handled by the same specials that are used for other graphical objects. This will make the generation of special effects with specials much easier. PostScript is a good example of how successful this approach can be [4].
- The DVI philosophy of compactness should be followed ([15], section 584).
- Keep the format of specials simple and easy to parse. Don't make the driver do extra work that could be done more easily by TEX. This is not to say that "if it can be done in TEX, you don't need a special for it." However, the work should be done where it's easiest.
- It is desirable for TEX users to be able to produce output in its final form (camera-ready, for instance) directly from the DVI driver. Where this is not possible, it should be because of inherent printer limitations or because the standard has not been fully implemented, rather than because of limitations in the standard itself.
- Design for a very capable printer and driver. This will encourage the development of such printers and drivers, by accenting the deficiency of those which cannot (or do not) support the whole standard.
- All applicable command-line options *must* be accessible via specials. Frequently, TEX users must print new copies of a document which was written years ago. If that document required special command-line options to print correctly (`landscape`, for instance), it might take some experimentation to rediscover those options. It would be extremely convenient to have those options encoded in the DVI file. Options to specify printing of multiple copies or to select pages from the document would not be good candidates for such specials, however.
- There should be a convention for global specials.
- A conforming driver should not be tied to any one format (or set of formats) for graphics files.
- The DVI driver should be a good DVI driver and little else; it should not attempt to be a graphics interpreter or translator.
- Don't underestimate the number, variety, or complexity of users' requirements. Make provisions for extension of the standard.

It may seem that some of these goals are contradictory. It is likely that in some cases it will prove impossible to satisfy all of them completely. They are all desirable goals, however, and it should be possible to meet them with a little work, and the standard will be a better one for the effort.

Types of specials and their functions

It seems appropriate to begin by discussing the functions that will be required, since that will have a significant effect on other aspects of the standard.

Global specials. Much has been said and written about the implicit restriction on specials affecting pages other than the one on which they occur. While the random ordering of pages in the DVI file does imply that this cannot occur, the authors see no need for it to be a hard and fast rule. Some convention for global specials should be developed, and then they should be used with care. The DVI format permits specials to occur before the beginning of the first page and between pages ([15] section 588). Unfortunately, TEX82 does not put any specials there. There are some reasonable possibilities, however. Specials which occur at the beginning of the first page in the DVI file and which have a `global` prefix could have global effects. Certain other specials could have effects on pages which succeed them in the final sort order; they would merely have to be used with care. A policy such as that adopted by Adobe Systems, Inc., toward PostScript is advisable [3]: they advocate keeping pages self-contained, but they do not enforce this. The ability to make something on one page affect other pages is there, but Adobe advises caution when using it. If the sort order is changed drastically or if a single page is extracted from the file, chaos will ensue. These are unusual cases, however, and the usefulness of global specials makes the danger worthwhile, in the opinion of the authors. An alternative approach is to have an optional companion file for the DVI file which would contain specials which could have a global effect.

Text and graphics manipulation. These specials will affect a region of text or graphic objects, some of which may be produced by other specials. Note that "region" refers to a region of the DVI file, rather than any particular area of the physical page. We will use "area" to specify a portion of the physical page. We should also take this opportunity to point out two other important points: the order in which transformations are applied is important, and some decision needs to be made on whether delimited specials should be allowed to nest. To permit special effects, the standard should provide for all of the common two-dimensional transformations, especially since they are easily accessible in modern page-description languages.

Rotation: A special for rotation should be capable of rotating graphics or text, up to a whole page. Of course, when graphics or text are stored in bitmap form, it is only feasible to rotate in 90° increments (and sometimes that isn't possible either, depending on the output device). But where it is possible to rotate, the portion of the page to be rotated could range from a single character to the entire page. As an example of how complex the design of a special can be, we will attempt to specify a general rotation special in its entirety.

First, the driver needs to know what to rotate. We could limit the special to rotating a single box (an hbox or a vbox), and use a single special which would rotate the next box in the DVI file (box structure can be inferred from the levels of the *push* and *pop* commands in the file [15][17]). This has two disadvantages, however. It makes extra work for the driver, keeping track of pushes and pops, and it also limits the scope of the command. A rotation special defined in this way would not be able to reliably rotate an entire page. In Plain TEX, a page is placed in a box by the output routine, and then shipped out. The special would go inside that box, and could not be placed outside of it. Of course, we could define the special as rotating the immediately enclosing box, but this is hard to control and it runs contrary to a user's intuitive understanding of the special. So we choose to have a delimited special, begun by, say, `rotate` and delimited by, say, `etator`. In this way, we can specify an arbitrary region to be rotated in a very general way. Whole pages can be rotated easily, and we can handle this in a global fashion by modifying the output routine to use `\everybop` and `\everyeop` token-list registers, containing commands which would be placed at the beginning and ending of every page. In such a case, the `etator` would not be strictly necessary; the region of a delimited special would be bounded by the end of a page (except possibly if it had been declared `global` — see *Global specials*, above).

Next, the driver needs to know the pivot point (the point about which to rotate). It is easy to just have all text rotated around the current coördinates, but that is needlessly restrictive. Such a restriction would result in a lot of clumsy TEX hackery to achieve other effects. Therefore a user should be able to specify an offset, using any of the standard TEX dimensions (note that "true" dimensions must be supported also). Macros can be constructed to expand to the offset of the upper-right-hand corner of a box, or the center, etc., so that this scheme would not be too difficult to use. One should also be allowed to specify exact coördinates, or to use a predefined point (see *Plotting*, below).

Finally, the driver needs to know the angle to rotate through. In the simplest case, it could be a number which would represent degrees. Ordinarily, it is not important to know from what direction an angle is measured; the default case is 0°, and angles are measured counterclockwise from that. It would help, however, to settle on a "text direction" concept to simplify things for users and for the authors of drivers. Angles should be measured counterclockwise from a horizontal line. This is intuitive, and it also makes the transformation easier for the driver, because it simplifies the mathematics; the transformation matrix to be used becomes

$$\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

for the angle θ. If the angle is in units of degrees, it may be necessary for the driver to convert it to radians before computing the sine and cosine.

When the angle is given in its component parts (i.e., two numbers), the two numbers should be assumed to be related to the sine and cosine of the angle. Thus, the angle θ can be determined from the components x and y by one of the following relations:

$$\begin{array}{ll} \theta \leftarrow \arctan y/x; & \text{for } x > 0 \\ \theta \leftarrow \arctan y/x + 180; & \text{for } x < 0 \\ \theta \leftarrow 90; & \text{for } x = 0 \text{ and } y > 0 \\ \theta \leftarrow -90; & \text{for } x = 0 \text{ and } y < 0 \\ \text{error}; & \text{for } x = 0 \text{ and } y = 0. \end{array}$$

This gives the angle θ in the range $[-90, 270)$. If it is desired to have the angles in the range $[0, 360)$, one can say:

$$\begin{array}{ll} \theta \leftarrow \theta; & \text{for } \theta \geq 0 \\ \theta \leftarrow \theta + 360; & \text{for } \theta < 0. \end{array}$$

In working with rotations, the angle itself is not generally useful; the sine and cosine are the important values, as shown in the transformation matrix above. If an angle is given in component form, it is possible to calculate the sine and cosine more efficiently using the following relations:

```
$d \leftarrow \sqrt{x^2 + y^2}$
**if** $d = 0$ **then** error
$\cos\theta \leftarrow x/d$
$\sin\theta \leftarrow y/d$.
```

So in order to include all of the necessary information, the `rotate` special must have the following form (or something similar):

`rotate` *hoffset voffset angle* ⟨*stuff*⟩ `etator`.

It should be obvious that a well-designed special is the result of a lot of work. Many factors need to be considered: desired capabilities, the user, TEX, the DVI format, implementation details, and printer capabilities.

There is one further note about the direction of text. Here, we have decided that angles should be measured counterclockwise from the positive h axis. Readers of right-to-left languages [17] might not choose that, however; they might choose clockwise from the negative h axis. At least some consideration should be given to making the "direction of text" equal to the direction in which the current point is displaced when printing it. This would complicate the implementation, but it might make the mechanism more versatile.

Scaling: If the printer supports it, there should be a special for scaling of text and other graphical objects (whenever they are not stored as bitmaps, of course). Possibly an offset would be needed here, also; do we want all of the expansion to be away from the current reference point? The scaling special should also be delimited, and it should support scaling by factors and also to a certain explicit size. Independent horizontal and vertical scaling should be possible. This would distort pictures in an undesirable fashion, but it would leave the data content of many graphs intact, while fitting them into a space of arbitrary size.

Translation: Translation would be helpful in placing graphic objects on the page. For example, suppose one wanted to use graphics inclusion to print a form which would overlay the entire page. Horizontal and vertical offsets should be the arguments here. Whether or not it should be delimited is a matter for debate; it can be strongly argued that any objects actually created by TEX (i.e., not by the driver itself) should be positioned on the page using TEX.

Reflection: Reflection requires knowledge of a region and an axis. The axis could be specified by two points or by one point and an angle. Note that this special is not strictly necessary, as reflection can be accomplished by a combination of other specials.

Clipping: The driver should be able to do simple clipping. This would be very useful when including graphics such as the METAFONT proofs produced by `GFtoDVI`. That program puts a title on the top of the page, and there is no option to suppress the title. Therefore it would be nice to be able to tell the driver to print a certain area of the page, and print it here. The standard could only permit rectangular clipping paths, although a more general mechanism of path specification might be necessary for a `fill` special (see *Plotting*, below). Note that the user should never assume that characters will be partially clipped, even though page-description languages such as PostScript permit it. The standard should specify the handling of characters which fall partially within the clipping area, and it should also permit specification of whether to clip inside or outside of the clipping area. A generalization of an "off-the-page" check would make this relatively easy to implement.

Color: Printers are beginning to incorporate color. The PostScript and Interpress [21] page-description languages have facilities for controlling color, and Adobe's Illustrator program [2] (which generates PostScript) can generate color documents automatically. The standard for specials needs to provide a way to access color facilities.

If color is to be accessed via a special, a complete range of colors must be supported. Gray scales should be available. Most importantly, a common color model should be used. Color models are difficult to understand; the user shouldn't have to learn a new one. The common color models are described in [9] on pages 611–623. From page 611:

> Three hardware-oriented models are the RGB (red, green, blue), used with color TV monitors; YIQ, which is the broadcast TV color system; and CMY (cyan, magenta, yellow) for color printing devices. Unfortunately, none of these models are particularly easy for a programmer or application user to control, because they do not directly relate to our intuitive color notions of hue, saturation, and brightness. Therefore, another class of models has been developed with ease of use as a goal. Several such models exist; we shall discuss only two: the HSV (hue, saturation, value) and HLS (hue, lightness, saturation) models.

RGB and YIQ were designed for luminous devices, CMY for hardcopy devices, and HSV and HLS for users. One of these models will almost certainly be understood (in some form) by the output device, and the driver will have to map whatever is specified in the special to the printer's model. Fortunately, there are standard mappings between all of these models; in fact, Foley and Van Dam give the code in their book. None of the algorithms are particularly complex. It seems reasonable, therefore, for the standard to support all five models. Individual drivers may not support all of the models, but they should at least support the printer's native model, and it would not be difficult to support all of them.

Drivers for black-and-white printers could render color in grayscales by mapping to the YIQ model. YIQ was designed for downward compatibility with black-and-white TV; The Y component can be used alone to represent grayscale.

It is important to also permit specification of background color. This would permit setting white text on a black background. This special should specify an area in the same manner as the `clip` and `fill` specials.

Graphics inclusion. This is really one of the stickiest areas of a special standard, because it contains most of the device-dependent details. We should resist the temptation to incorporate scaling, translation, and clipping parameters into these specials; it would be better to use the general specials described above for such things.

DVI inclusion: This one keeps free of device dependency, but it's certainly not easy to implement. Fonts in a DVI file are numbered [15], so font numbers and coördinates need to be kept separately for different DVI files [20]. It would be really nice, however, to be able to include pages from other DVI files into a document, or even parts of pages. It also gives access to the magnification feature for scaling a page to be included. This could be very useful: *The METAFONTbook* could have been produced with no manual pasteup of any kind. The proofs from METAFONT could have been included in this way. In our opinion, this is the only kind of file inclusion for which the DVI driver should act as an interpreter in any way.

Standard graphic formats: Other programs should be used to translate the files to printer-specific format. This keeps the driver from being too complicated, and it allows easy extension to other formats. In addition, the other programs could create a companion TEX file for the graphic with TEX commands detailing the size of the graphic, etc. It could even place those commands at the beginning of the graphic file itself; an `\endinput` command would cause TEX to exit from the file before it hit the graphic information, and the driver could strip those commands out when it read the file. The preferred method, in the opinion of the authors, is to have a companion file.

Printer-specific formats: This would seem to be the easiest task. We should just include the file and go, trusting that the file itself is error free. If the file is properly encoded, this should work on almost all printers. A special conversion program to properly "encapsulate" the file would be a fairly simple task. There is, however, at least one complication, which is the subject of the next section.

Orientation-specific files: On some printers (e.g., the Xerox 9700 family), graphics files are orientation-dependent. A graphic created to be printed in landscape will not work correctly in inverse landscape. This can become a problem, especially when imposing pages. Suppose we are imposing pages to produce a booklet which can be folded and stapled straight from the printer. Pages on the front of a sheet are in landscape, and pages on the back of a sheet are in inverse landscape. When the document is being formatted we do not know which side of the sheet the graphic will finally fall on; this may even be dynamic, and we may print the document in simplex for proofing to speed turnaround. Admittedly, this is a printer limitation, but we should provide for it. It would probably be wise to provide an `if else` special. If so, it should be generalized to test not only orientation, but other conditions and user-defined values, as well.

Direct printing features. Some of these specials will also be device-dependent, and site-dependent as well. Defining such site-dependencies should be done in a configuration file, separate from the other source files if possible.

Paper types: Many computer centers offer multiple paper types. Usually they are referenced by a special code. This special should take multiple arguments, and the driver should map to a paper type code for the particular site. At a center where the users are familiar with the codes, the argument would be the code itself. At other places, users might specify `green cardstock`, and the driver would substitute the proper code (if, that is, the driver is told to go ahead and print the file). Note that there should be a way to specify which type of paper should be put into which input tray, for printers with multiple trays.

Tray selection: Many printers now have multiple paper input trays. The Xerox 9700 has two, the DEC PrintServer 40 has three, and even the Apple LaserWriter has two (if you count the manual feed). It is extremely convenient to be able to specify that, for instance, the cover and chapter dividers should be pulled from tray 2, so that no hand collating is necessary. This is obviously printer-dependent.

Other options: The direct printing features demonstrate the need for extensibility. Some printing centers may offer binding. Should there be a special to control binding? Xerox' Interpress page-description language [21] has facilities to specify binding, cutting, etc. The *Adobe Systems Document Structuring Conventions* [1] also define notations which control such things. It should be clear that at this time it is extremely unlikely that all possibilities can be anticipated.

Page selection and ordering. This will be a very difficult set of specials to design. Even simple sorting is a sticky problem. Is page zero odd or even? Positive or negative? Should there be a special to control that? When more sophisticated capabilities are desired, there are even more issues to consider. These options would be most useful on the command line, but they would also be useful as specials.

Sort keys: At the beginning of every page, the values of `\count0` through `\count9` are included in the DVI file. It would be very convenient to be able to specify which of these values would be used as the primary sort key, which would be the secondary sort key, etc. In this way, page order could be controlled explicitly.

Sorting of pages should be allowed using all ten of the `\count` registers as well as at least one special register. One special register is defined as the pages are being scanned in the DVI file. Its value is the sequential number of the page in the DVI file, the first being 1, the second being 2, etc. The backpointers in the DVI file could still be used to skip around in the file if this register was not going to be used for sorting. Some mechanism for handling duplicate page numbers would be necessary; ideally a document would not have any duplicates.

As an example of how different sort fields might be used, consider a document with the page number made up as follows:

`\count0`	is the current chapter number;
`\count1`	is the current section number within the chapter;
`\count2`	is the page number within the current section and chapter;
`\count3`	is used for any inserted pages.

The pages for this document need to be sorted by all four of those registers, with `\count0` as the most significant sort key. It is important to note, however, that the significance of the registers should be variable. Also, we might not always want to sort groups of pages with negative sort keys by the absolute value of the key. It might even be desirable to sort in descending order, especially for printers that stack their output face-up.

Page parity: Page parity controls whether or not a page should be placed on the front or back of a sheet when printing on both sides of a page. (We will refer to such printing as "duplex" printing, and to printing on one side of the page as "simplex" printing.) In the simplest case, odd page numbers will go on the front, and even page numbers will go on the back. Readers of Japanese or Hebrew may feel differently, however.

Determining the "parity" of a page is fairly easy if only one `\count` register is used to give the page number: the parity is simply the parity of the `\count` register (with provisions for the value zero and right-to-left languages). However, when more than one `\count` register is used, the problem is more complex.

Again, consider the sample document in the previous section. Four `\count` registers are used to contain the page number. Chapter and section numbers have little to do with which side of the sheet a page should print on. Therefore, `\count0` and `\count1` should be ignored in determining page parity. But this still leaves `\count2` and `\count3`. The actual parity of the page should be based on the sum of these two registers. This is because page 1.2.2 should fall on the back of a sheet (with page 1.2.1 on the front) while page 1.2.2.1 falls on the front of the next sheet. Similarly, page 1.2.3 goes on the front and page 1.2.3.1 should be printed on the back of the same sheet. A special to control parity should be able to make use of the same registers used for sorting.

Page pairing: Another aspect of page parity is choosing which two pages should be placed on opposite sides of the same sheet. This determination is made by selecting pages from the sorted page list. The algorithm takes one or two pages from the page list and defines the page or pages to be placed on a single sheet. As long as the page numbers remain consecutive, this is not a big problem, but it can get sticky. Should page six fall on the back of page one if it occurs next in the final sort order?

Version numbers: Imagine a user documentation division of a computer corporation. They might have files called `usermanualV1.tex` and `usermanualV1--1.tex`. They contain version 1 and version 1–1 of the usermanual, respectively. The first should take care of itself, but the second might pose some problems. The group uses macros to mark changes that are inserted. Those macros use some plotting specials to insert changebars into the text, but that is not our concern right now. Two different kinds of behavior will be desired from that file. The entire version 1–1 manual will need to be printed for distribution to new customers, but only the changed pages will be needed for distribution to old customers as updates. Those pages should be marked somehow by a special. One possibility is to allocate a count register (say, `\count9`) as a change flag, and have a global special and command-line option which instructs the driver to print the page only if that register has a certain value. The register to use should not be hard-coded into the special. Perhaps a `dontprint` special could be used in conjunction with the aforementioned `if else` pair. Note that for duplex printing the driver will have to also print the page on the other side of the sheet, even if that page was not changed. The nature of this special makes it easier if it occurs immediately after the beginning of the document.

Page manipulation. These specials are for describing the page to the driver, moving it around, etc. These are good candidates for `global` treatment.

Page size: This special would be used to define the size of the sheet of paper to the driver. Many printers are equipped to handle multiple sizes of paper, so that should be settable via a special. A default setting should be defined in a configuration file, so that users in Europe can use A4 settings, for example.

Imposition options: This is used to place multiple pages on a single sheet which can later be folded or cut to make a booklet. This special needs to be very general; for example, in magazine production, eight pages are placed on each plate, so that each sheet coming off the press contains sixteen pages (eight on each side). This large sheet is then folded in a weird way, stapled with zero or more similar assemblies, and trimmed to make the final magazine [13]. The imposition special should allow specification of the number of pages per sheet, their placements on the sheet, and their orientations. Note that scaling should not be the responsibility of the driver; it is the user's responsibility to make the pages the correct size and to set the margins. Note also that the sort order might be different depending on whether the sheets are to be folded together or cut, punched, and put into a binder.

Overprinting: It might be very useful to provide for superimposing two pages, one on top of the other. This would be an easy way to produce a portrait page (with portrait headline and footline) containing a landscape table. It should probably not be a separate special, though. If a special to include a page from another DVI file is implemented, it should be general enough to permit including another page from the same DVI file onto the current page. The translation special could be used for positioning.

Full-page orientation: Although the rotation special we mentioned above is general enough to rotate an entire page, a special which is devoted to rotating the entire page might be worthwhile. It would be much easier to use as a `global` special. The same applies for a full-page translation special, for margin adjustment.

Plotting. TEX users really need some sort of simple plotting interface. Using a separate graphics package to create simple pictures or graphs and then including them in the TEX document is often too difficult. A simple plotting interface would ease this difficulty, and allow users to produce figures like those produced so easily with the *pic*, *grap*, and *chem* preprocessors for *troff* [6][19]. Plotting functions would also make TEX more suitable for special applications such as music typesetting [10][11]. (Incidentally, anyone not familiar with *grap* [7][8] should take a little time to study it; it is a fantastic example of a well-designed little package. It is elegant, very easy to use, has reasonable default behavior, and can be persuaded to do some fantastic things.)

Point definition: This idea came from drivers produced by ArborText. We saw it in a driver for the DEC LN03, written by Flavio Rose. The driver maintains an array of point coördinates (initially all zero). A special is used to define one of those points at the current location. That point can later be used as the location of a dot or the end of a line. The great thing about this is that it allows the position of a point on a page to be located by TEX's formatting (which is very difficult to do without some sort of special), and that location can then be used on other pages. Admittedly, no great application for this is immediately obvious, but it does seem useful, and it is so difficult to do with just TEX and the other specials that this special seems worthwhile. It also violates the implicit restriction on specials affecting other pages, but it is a simple fact that sometimes we will need just that (see *Global specials*, above).

Splines: This special would be used to specify an arbitrary curve between two points (which can be explicit coördinates, offsets, or points defined by the `pointdef` special). One should be able to specify a curve from the current point to another, and also between two arbitrary points. The shape of the curve should be determined by two control points. Specials to define line thickness and cap shape should also be available.

Geometric shapes: For ease of implementation, it might be desirable to supply straight lines, circles, and other common shapes just as special cases of the spline-drawing facility, but there are some good reasons not to. First, computation of the shapes will be faster if there is a special facility devoted to them. Second, there are special techniques for digitizing straight lines which do not work as well for curved lines. Offering a separate special for straight lines will permit special handling.

Filling: This should be restricted to filling with a solid color. Any filling which requires a complex pattern should be done as a separate graphic which would be included. Some way of defining a closed path will be necessary, as will a convention for handling what METAFONT calls "strange paths" [14].

Miscellaneous specials. As one might expect, there are some specials which do not fit easily into categories.

Communication with the user: Just as TEX has the `\message` command, it might be desirable for the DVI driver to have a `message` special. In addition, drivers should optionally warn users of unrecognized specials instead of ignoring them. There should be a special to invoke that option. It would be reasonable to make that special the only one which *must* be supported by a conforming driver.

Job logging: Several drivers keep logs in much the same fashion as TEX does. It might be advisable to offer specials to control this. Logging could be enabled or disabled via a special. In addition, one could specify whether the driver would keep a separate log or append to the TEX log file.

Literal commands (`\special{special}`*):* For many printers, passing literal commands straight through makes no sense; the effect would be entirely unpredictable. For some printers, it would be possible, though. The wisdom of providing such a special is debatable (especially if the other specials offer a great deal of functionality), but it should be considered. Note that this is the most device-dependent of all specials.

DVI format: There is already at least one extension to TEX which makes use of some of the unused commands in the DVI format: the `TeX-XeT` processor described in [17]. Perhaps such extensions should use a different DVI identification byte, but it is also conceivable that those variations could be identified by a special. Either way, a single driver could support multiple extensions to TEX. Incidentally, the authors prefer using the DVI id byte for this.

Accessing specials from TEX

Specials should be generated by macros whenever possible. Users should not have to learn to use another user interface; they should just have a few new macros. `\landscape` could be a macro meaning that the current page should be rotated about the origin and then translated by *pageheight* − 2 *inches* in the $-h$ direction. This would simplify the user interface greatly.

Some extra macros would also simplify the `global` problem slightly. `\everybop` and `\everyeop` token-list registers could be defined which would be placed by the output routine at the beginning and ending of every page. Also, a `\globalspecial` register could be used. If this register was defined before the first `\eject`, the output routine would ensure that the contents occurred first on the very first page, before any *set* or *put* commands. This would require only trivial modification to the output routines, and the benefits would be enormous.

Handling device-dependence

As much as we try to avoid it, some specials are going to be device-dependent. This is a hazard of using specials. The trauma could be reduced, however. METAFONT has a nice little feature in its ability to handle "modes" [14]; different modes for different printers. Such a facility could be implemented in TEX with macros. It would rarely be used, but it would help with certain device-dependent specials. Typing `\mode{X9700}` would define certain special-generating macros to fit the device. Modes would be defined in a `local.tex` file, just as METAFONT's modes are usually defined in a `local.mf` file. This would greatly simplify the process of producing proof copy on one device and then moving to another device for the final output. Note that it may be advisable to name the modes after the specific driver rather than the printer.

Miscellaneous recommendations for driver behavior

Robert McGaffey's article, "The Ideal TEX Driver" [18], offers some excellent recommendations for the behavior of DVI drivers. Some of his suggestions (those regarding the handling of missing fonts) are also good candidates for handling with specials. We would like to offer some additional recommendations.

- The driver should be able to read a configuration file on startup, so that the user can include certain default options there. McGaffey mentions this, but the idea is important enough to merit discussion here as well. Possibly a system-wide configuration file should be read first, and then the user's. This might be one solution to the problem of global specials.
- The driver should also be able to look at certain environment variables defined by the user. Where the authors work, default paper type and destination are set in this way.
- Where there is complex logic (such as sorting logic), it is inevitable that a situation will surface where that logic is inappropriate. The driver should provide an option to turn off that complex logic and go back to the brute force method.
- In the configuration file or an environment variable, the user should be able to specify a "path" for the location of TFM and bitmap files, so that personal METAFONT-generated fonts can be used with ease. This mechanism should be able to deal with the various complicated and contradictory conventions for naming and storing such files. The authors are working on a mechanism for a driver driver which apparently will permit such specification in a very general way.
- There is naturally some confusion about how to handle the PostScript fonts and their ability to be scaled. TFM files for those fonts should probably be generated assuming a default size of 10 points, to be consistent with standard TEX assumptions.
- When dealing with PostScript (and in particular PostScript fonts), one should remember that the PostScript "point" (before any transformations have been applied to the user space) is equivalent to the TEX "big point" [4][16]. This distinction is important. When a user requests Palatino-Roman at 10pt, the driver should produce Palatino-Roman at 10 TEX points. Some fonts are special cases; *Sonata* was designed with the convention that the point size would be equal to the distance between the centerlines of the top and bottom lines of the staff character [5]. This convention should be maintained. The important thing is that the size of a "point" within a TEX file should always be consistent.

Accessing specials from TeX

Specials should be generated by macros whenever possible. Users should not have to learn to use another user interface; they should just have a few new macros. `\landscape` could be a macro meaning that the current page should be rotated about the origin and then translated by *pageheight* − 2 *inches* in the $-h$ direction. This would simplify the user interface greatly.

Some extra macros would also simplify the `global` problem slightly. `\everybop` and `\everyeop` token-list registers could be defined which would be placed by the output routine at the beginning and ending of every page. Also, a `\globalspecial` register could be used. If this register was defined before the first `\eject`, the output routine would ensure that the contents occurred first on the very first page, before any *set* or *put* commands. This would require only trivial modification to the output routines, and the benefits would be enormous.

Handling device-dependence

As much as we try to avoid it, some specials are going to be device-dependent. This is a hazard of using specials. The trauma could be reduced, however. METAFONT has a nice little feature in its ability to handle "modes" [14]; different modes for different printers. Such a facility could be implemented in TeX with macros. It would rarely be used, but it would help with certain device-dependent specials. Typing `\mode{X9700}` would define certain special-generating macros to fit the device. Modes would be defined in a `local.tex` file, just as METAFONT's modes are usually defined in a `local.mf` file. This would greatly simplify the process of producing proof copy on one device and then moving to another device for the final output. Note that it may be advisable to name the modes after the specific driver rather than the printer.

Miscellaneous recommendations for driver behavior

Robert McGaffey's article, "The Ideal TeX Driver" [18], offers some excellent recommendations for the behavior of DVI drivers. Some of his suggestions (those regarding the handling of missing fonts) are also good candidates for handling with specials. We would like to offer some additional recommendations.

- The driver should be able to read a configuration file on startup, so that the user can include certain default options there. McGaffey mentions this, but the idea is important enough to merit discussion here as well. Possibly a system-wide configuration file should be read first, and then the user's. This might be one solution to the problem of global specials.
- The driver should also be able to look at certain environment variables defined by the user. Where the authors work, default paper type and destination are set in this way.
- Where there is complex logic (such as sorting logic), it is inevitable that a situation will surface where that logic is inappropriate. The driver should provide an option to turn off that complex logic and go back to the brute force method.
- In the configuration file or an environment variable, the user should be able to specify a "path" for the location of TFM and bitmap files, so that personal METAFONT-generated fonts can be used with ease. This mechanism should be able to deal with the various complicated and contradictory conventions for naming and storing such files. The authors are working on a mechanism for a driver driver which apparently will permit such specification in a very general way.
- There is naturally some confusion about how to handle the PostScript fonts and their ability to be scaled. TFM files for those fonts should probably be generated assuming a default size of 10 points, to be consistent with standard TeX assumptions.
- When dealing with PostScript (and in particular PostScript fonts), one should remember that the PostScript "point" (before any transformations have been applied to the user space) is equivalent to the TeX "big point" [4][16]. This distinction is important. When a user requests Palatino-Roman at 10pt, the driver should produce Palatino-Roman at 10 TeX points. Some fonts are special cases; *Sonata* was designed with the convention that the point size would be equal to the distance between the centerlines of the top and bottom lines of the staff character [5]. This convention should be maintained. The important thing is that the size of a "point" within a TeX file should always be consistent.

Conclusion

While the usefulness of special commands is beyond doubt, the subject is extemely complex. If a driver attempts to provide a versatile command set, there are many difficult problems which must be resolved. With careful thought and planning, however, it seems certain that a powerful, useful standard set of specials can be devised, and hopefully with TUG's sanction it will be used in most available DVI drivers.

Bibliography

[1] Adobe Systems Incorporated. "Adobe Systems Document Structuring Conventions, Version 2.0." Palo Alto, California, January 31, 1987. Manuscript.

[2] Adobe Systems Incorporated. *Colophon4*. May 1987.

[3] Adobe Systems Incorporated. "Encapsulated PostScript File Format, Version 2.0." Palo Alto, California, March 12, 1987. Manuscript.

[4] Adobe Systems Incorporated. *PostScript Language Reference Manual.* Reading, Massachusetts: Addison-Wesley, 1985.

[5] Adobe Systems Incorporated. "*Sonata* Technical Design Specification." Palo Alto, California, January 30, 1987. Manuscript.

[6] Batchelder, N., and T. Darrell. "Bringing troff up to Speed." *Unix Review* **5** (July 1987): 45–51.

[7] Bentley, Jon L. "Little Languages." *Communications of the ACM* **29** (August 1986): 711–721.

[8] Bentley, Jon L., and Brian W. Kernighan. "GRAP—A Language for Typesetting Graphs." *Communications of the ACM* **29** (August 1986): 782–792.

[9] Foley, J. D., and A. Van Dam. *Fundamentals of Interactive Computer Graphics.* Reading, Massachusetts: Addison-Wesley, 1982.

[10] Gourlay, John S. "A Language for Music Printing." *Communications of the ACM* **29** (May 1986): 388–401.

[11] Gourlay, John S., Allen Parrish, Dean K. Roush, F. Javier Sola, and Yiling Tien. *Computer Formatting of Music.* Technical Report OSU-CISRC-2/87-TR3, Ohio State University, 1986.

[12] Hosek, Don. "Proposed `DVI \special` command standard." Harvey Mudd College: Claremont, California, August 6, 1987. Manuscript.

[13] Kim, Scott. "The Three-Fold Way." *Publish* **2** (June 1987): 124.

[14] Knuth, Donald E. *The METAFONTbook.* Reading, Massachusetts: Addison-Wesley, 1986.

[15] Knuth, Donald E. *TEX: The Program.* Reading, Massachusetts: Addison-Wesley, 1986.

[16] Knuth, Donald E. *The TEXbook.* Reading, Massachusetts: Addison-Wesley, 1984.

[17] Knuth, Donald E., and Pierre MacKay. "Mixing right-to-left texts with left-to-right texts." *TUGboat* **8** (April 1987): 14–25.

[18] McGaffey, Robert W. "The Ideal TEX Driver." *TUGboat* **8** (July 1987): 161–163.

[19] McGilton, Henry, and Bill Tuthill. "Progress Through Accretion." *Unix Review* **5** (July 1987): 36–41.

[20] Reid, Thomas J. "TEXROX Installation and Theory of Operation," College Station, Texas. In preparation.

[21] Sproull, Robert F., and Brian K. Reid. *Introduction to Interpress.* El Segundo, California: Xerox Corporation, 1984.