Software

Inside Type & Set

Graham Asher

Abstract

Type & Set is a typesetting system consisting of TEX, several TEX macro packages, and a suite of C programs including a style sheet editor, an automatic page make-up system which replaces TEX's output mechanism, and a family of drivers. It solves many of the problems which make plain TEX difficult to use for commercial journal and book publishing. This article explains in detail how Type & Set works.

History of the project

Type & Set has been under development at Informat Computer Communications since February 1987. Informat is the software development and typesetting division of Current Science (formerly Gower Academic Journals), a publishing house, and because the two companies share the same premises we have had constant access to users and their suggestions and criticism. Some ideas in Type & Set are taken from an earlier (non-TEX) package of the same name which it has superseded. For these ideas (principally the style sheet hierarchy, the mark-up system and the input format for the table generator) I am indebted to Mr. A. Harris, a former programmer at Informat. I take full responsibility for the present form of the system. The first version of Type & Set was installed in June 1988, but since then nearly every part has been rewritten.

What problems does Type & Set solve?

Using Type & Set rather than TEX incurs costs in running time and disk space. However, Type & Set solves or ameliorates the following problems, many of which are discussed in detail by Mittelbach [1]. The severity of these problems amply justifies the increased use of resources.

Page breaking is taken away from TEX completely and given to a program called PAGE which analyses the DVI file and writes a new DVI file, optimally paginated, with balanced columns, figure spaces, running material, headers and footers. PAGE takes its formatting information from a *style sheet* created using Type & Set's style sheet editor.

Varying numbers of columns. Type & Set can switch freely, as many times as you like (and as many times as you like *on the same page*), between text in one, two, three and four columns.

Baseline-to-baseline spacing occurs as a result of using PAGE rather than TEX to make up pages. All vertical dimensions in the Type & Set system are measured from the baseline of one line of text to the baseline of another. In particular, baselines at the bottom of pairs of columns align with each other, as do those of the last lines of text on facing pages. This also allows style sheets to specify a grid of lines on to which all baselines should fall if possible: that is, the y coordinate of a baseline should be an exact multiple of the grid interval.

Composite fonts (I prefer this term, suggested by Beebe [2], to the less descriptive 'virtual fonts') are used where necessary in the drivers. Readable data files called FD or 'FontData' files provide all the information a driver needs to convert a TEX character code into a device character, using transformations and superimposition if necessary. A utility, MAKETFM, is used to create TFM files for various output devices, given appropriate FD files and width tables.

Tables are created using a quasi-*wysiwyg* format in a text editor and converted into TEX by a program called TABLE. Horizontal spans, vertical and horizontal rules, and centring around any character (such as a decimal point), are all supported. Tables are very easily created and modified using this system.

Graphics is absent from TEX, and should not be added. The prevailing standard for graphics is PostScript, and so the Type & Set PostScript driver will pick up a named Encapsulated PostScript file, translate and scale it, and embed it in a figure space. The driver is told to do this by a `\special` written by a macro placed in the text and passed through by PAGE. A more general feature of PAGE, that can be used with any driver, is its ability to load, scale and embed a DVI file in exactly the same way.

Ease of use. Once a style sheet has been created for each kind of document to be typeset the rest is very easy. Staff at all levels of the publishing process, including those with no specialist computer knowledge (that is, nothing beyond basic abilities such as the use of the file system and rudimentary text editing) can be trained to use Type & Set in a day or two.

The rest of this article describes in detail how the problems were solved and how Type & Set works. What is described is a working system which was designed and implemented at a publishing house over a period of four years, and is now in daily use.

Data flow and general operation

Input. The user types a document using his or her favourite word processor. This may, for example, be Wordstar or WordPerfect, or (as I prefer, being in part an unreconstructed TEX hacker) an ordinary ASCII text editor. The document contains little or no TEX apart from markup codes known as *mode names* which are determined by the style sheet to be used. Mode names look like ordinary TEX control sequences, mainly because that is what they are. The preferred Type & Set style places mode names on separate lines. The mode determines all the stylistic and structural parameters of the text: its font, justification, indents, paragraph spacing, and whether it is part of the body text, a figure caption, or, say, a running header—and many other details. *Tag* is the term preferred in the world of desktop publishing, but we stay with *mode* for historic reasons.

Preprocessing. The first part of Type & Set to be run is the appropriate preprocessor for the text editor or word processor that has been used. In the case of Wordstar this is WS2TEX, which strips the high bits that Wordstar uses to mark the ends of words, converts Wordstar codes for italic, bold face, etc., into `\it`, `\bf`, etc., and emits standard ASCII text of the type TEX reads.

TEX. Any version of TEX can be used, with the proviso that if the document contains large tables a version with the biggest possible memory is desirable. TEX loads a customised format file, very similar to `plain.fmt` (indeed, *almost* upwardly compatible) called `tsplain.fmt`. The first command TEX finds in its input file is something like `\input mystyle.sty`, which loads the style sheet, which defines all the mode names and other markup codes used in the document. TEX then runs normally and writes a `DVI` file, using the minimal output routine from `tsplain.fmt`. This output file is effectively a *galley* in traditional typesetting terms, in that the text has been set in the desired fonts and *counted* or broken into lines, but has not yet been made up into pages. The `DVI` file contains numerous `\specials`, mostly for use by PAGE.

Page make-up. PAGE reads the `DVI` file and analyses it into lines, determining the mode of each line from a `\special`. A packet of information is built up for each line giving quick access to its mode, leading, position in the `DVI` file, and so on. At this point any *automatic material* is added. This consists mainly of the spacing and rules which the style sheet specifies for insertion before or after certain modes, or between paragraphs, or around blocks of text. PAGE then uses the line information to write a new file, given the extension `DVP`, but precisely conforming to the `DVI` format, which contains the made-up pages. If TEX fonts were used the work of Type & Set proper could end here, and the `DVP` file could be typeset using a third-party driver.

Previewing. Both `DVI` and `DVP` files for any printing device can be previewed on the screen using the Type & Set previewer, DVISCR, which draws characters using a device-independent vector font. This evades the problem of screen bitmap fonts not being available for, say, Optima on the Linotronic 100.

Proofing. Proofing is generally done on a laser printer, using the PostScript or Hewlett Packard LaserJet driver as appropriate. All drivers have the ability to emulate a font not found on the output device by using a similar one that *is* present; and the PostScript driver is especially optimised for emulation.

Printing. The driver family includes drivers for PostScript, Hewlett Packard LaserJet, Linotype devices using CORA V, Chelgraph devices using ACE, and Agfa Compugraphic devices. All drivers share common code which reads the `FD` (FontData) file, interprets the `DVI` file, and implements the composite font system.

Style sheets

The style sheet system both endows Type & Set with much of its power and limits it in various ways. Style sheets embody a generalisation about the possible forms of a document: a model which necessarily excludes some possible documents. The Type & Set style sheet model is designed to handle most types of journal and book design, but not magazines or newspapers, which in any case are laid out manually, page by page, rather than being intended for automatic page make-up.

Type & Set documents organise their text into two major divisions:

- body text, and
- running matter.

The body text is a single continuous sequence laid out over as many pages as necessary within a certain rectangle known as the *text area*, which may be positioned differently on left and right pages. Footnotes and figures are included within the broad heading of body text: these are positioned within the ordinary text under the control of *callouts* or references to them.

The running matter comprises running headers and footers, and folios (page numbers). These items are placed in the margin outside the text area at fixed positions on each page. Style sheets allow you to specify different positions and different text for left, right, start and end pages.

Style sheets have three levels, *page*, *block* and *mode*. Each level has its own dialog within the interactive style sheet editor, STYLE. To create a style sheet a designer runs STYLE and fills in the boxes in the dialogs. The following paragraphs explain the meaning of each level.

Page level. This is where you specify the size and position of the page and of the text area within it. In the present version of Type & Set each document can have only a single page style: but multiple page styles are an obvious and not impossibly difficult extension which may be considered in the future.

Block level. You must create a block for each structurally different type of text in the document. Blocks are named objects belonging to one of the following *categories*, each of which has a two-letter symbolic name:

• `te`	text
• `hs`	running header, start page
• `he`	running header, end page
• `hl`	running header, left pages
• `hr`	running header, right pages
• `fs`	running footer, start page
• `fe`	running footer, end page
• `fl`	running footer, left pages
• `fr`	running footer, right pages
• `fn`	footnote
• `ps`	folio, start page
• `pe`	folio, end page
• `pl`	folio, left pages
• `pr`	folio, right pages
• `fi`	figure

It is often necessary to have more than one text block. Blocks may be set in one, two, three or four columns: if you need to switch between text in different numbers of columns, as, for instance, in the case of a document with full-width single-column headings and two-column text, then the single-column text must have one block and the double-column text another.

The other main motivation for multiple text blocks is the need to position the blocks differently: each block may be offset from the left margin of the text area by a different amount, and this may be specified separately for left and right pages. This facility enables you to design a document (as in the case of a medical textbook published using Type & Set) where the headings project beyond the text, inward toward the margin.

Usually no more than one block will belong to each of the running header and running footer categories.

To sum up, the following information is specified at block level:

- name of the block
- category: see previous table
- offset from left margin
- absolute coordinates, unless category = text
- width
- number of columns
- gutter between columns
- weight of gutter rule, if any
- weight of box rule, if any
- margins inside box rule
- grid spacing, if any
- automatic spaces and rules
- explicit spaces, rules and indents

A typical simple style sheet will have six or seven blocks: a complex one will have twenty or thirty.

Mode level. This is the lowest level of description, corresponding to the *tags* used in desktop publishing packages such as Ventura Publisher. Here all the information about fonts and point sizes is stored, along with the justification and indents. The mode may be indented within its column, so the *measure* or, in TeX terms, the `\hsize` of a paragraph of text is determined by width of a column (set at block level) minus any left or right indents applied at mode level.

The name you give a mode is the actual markup code used in the input text, and may be any alphabetic sequence up to ten letters long. In the text all that is needed is a prefixed backslash: to invoke mode 'ref', the command `\ref` is used, on a line of its own.

Every mode belongs to a block, and normally several modes belong to the same block. In an extremely simple document consisting only of text and headings, there might be two modes, `\text`, in a fully justified roman font, and `\head`, in a left justified bold font. Automatic spacing can be used to insert space between the heading and the text.

Not one but four fonts are specified in every mode. These are roman, bold, italic, and bold italic, and will nearly always come from the same face or family unless special effects are intended. For example, a text mode might have Garamond Light, Garamond Book, Garamond Light Italic and Garamond Book Italic; within this mode the control sequences `\rm`, `\bf`, `\it` and `\bi` respectively would be used to select each of the four fonts. Where the mode is inherently bold, as in a heading, the fonts are usually chosen so that roman and bold are identical, as are italic and bold italic.

Although Type & Set gives you control, via PAGE, over the degree of tolerance extended to widows and orphans, sometimes absolute prohibition of unwanted page and column breaks is preferred. This is done at mode level. For example, if you want the first two lines of each paragraph to be locked together and never split in any circumstances you can give *paragraph start lock* the value 2. Headings must never be separated from the text that follows, and this is done by specifying that the heading mode is to be locked to the next mode, as well as having all its lines locked together. Of course, this does not mean that all the heading lines in the document are locked into one huge block: the lock applies only to continuous sequences of lines belonging to the same mode.

To sum up, the following information is specified at mode level:

- name of the mode
- the block it belongs to
- the four fonts
- pointsize
- leading
- justification
- hyphenation tolerance
- looseness of word spacing
- left and right indents
- paragraph indent
- glue between paragraphs (`\parskip`)
- indent the first paragraph?
- lines to lock at start of mode
- lock all lines of mode together?
- lock this mode to following text?
- lines to lock at start of paragraph
- lines to lock at end of paragraph
- automatic spaces and rules
- explicit spaces, rules and indents

Table modes. I shall not deal with table modes in detail. They are at the same level of description as ordinary modes, and contain much of the same information, with the addition of some things needed specifically for tables, such as the amount of space to leave between the table and its caption, if any. Tables are explained below.

Fonts and font families

All text in a Type & Set document belongs to one of the modes of the style sheet in use. When STYLE creates a style sheet one of the files it writes is a large TeX macro package: each mode is a macro. This is how the fonts are selected. When a mode macro is interpreted by TeX, ten control sequences (among others) acquire new meanings:

• `\rm`	roman text
• `\it`	italics
• `\bf`	bold face
• `\bi`	bold italics
• `\sp`	superscript
• `\sb`	subscript
• `\mi`	math italic font
• `\sy`	symbol font
• `\mx`	math extension font
• `\xx`	Type & Set extension font

These invoke lower-level macros to select the appropriate fonts and pointsizes. The first six need not be used if WordStar or WordPerfect is used to input the text, because Type & Set can convert the control characters used by the word processors if necessary. The last four are also rarely seen in Type & Set text, but are invoked automatically in mathematical text and when special characters from those fonts are used.

The `xx` or Type & Set extension font is a ragbag of characters which seem to be required in journal and book publishing, and are generally provided on typesetting equipment, but are absent from the standard TeX layouts. These include solid triangles, solid circles, copyright symbols as single characters rather than composites, guillemets, etc.

Plain TeX preloads the sixteen most popular Computer Modern fonts and sets up a math font system at 10pt. The Type & Set format, `tsplain.fmt`, preloads no fonts at all: Type & Set is designed to be used on a wide variety of different devices, many with differing TFM files for fonts with the same names; so to preload fonts would cause confusion and errors.

The family mechanism used in plain TeX's math setting has to be retained, since it is hard-coded into TeX; but the other families are slightly different. Type & Set has:

family	*description*
0	text
1	math italic
2	symbol
3	math extension
4	italic
5	bold face
6	bold italic
7	Type & Set extension

No fonts are assigned to members of these families in `tsplain.fmt`. This is all done when the style sheet is loaded. For each mode, a font is assigned for all eight families at three different sizes, making a possible total of twenty-four fonts per mode. Very large style sheets may exhaust TeX's font memory, but in practice that does not happen very often, because many of the fonts belonging to one mode will be exactly the same as those of another; and STYLE is optimised to make use of any coincidences when writing the style sheet macros.

Superscripts and subscripts are implemented in a different way from plain TeX's method, except within math mode, where everything as far as possible is identical to plain TeX. Outside math mode the `\sp` and `\sb` macros provide more consistent text-mode superscripting and subscripting than `_` and `^`. They use TeX's font family system to determine the appropriate `\scriptfont` or `\scriptscriptfont` to use.

Type & Set's consistent approach allows everything to work in the same way whatever the current point size. In particular, mathematical setting is the same at any size, while remaining compatible with plain TeX.

Page make-up

`DVI` files written using Type & Set style sheets are completely standard and can be translated using any driver. All the extra information needed by PAGE, Type & Set's page make-up program, is to be found in *xxx* commands written by TeX's `\special` primitive.

PAGE is line-based: it analyses the `DVI` file into separate lines and moves these around, but goes no deeper than that except in the case of page numbers or *folios*, which must be often inserted into the middle of lines. Finding out where a line starts and ends in a `DVI` file originally seemed difficult, and a complex algorithm for finding minimal push-pop pairs enclosing pieces of text was used in an early version of PAGE, but eventually it was realised that with a little care one can ensure that every line is a first-level push-pop group.

PAGE can if needed optimise its layout over a whole document, by building a directed acyclic graph in which nodes are page breaks and arcs are possible pages labelled with their cost or 'badness', which is assessed using TeX-like criteria; and finding the lowest-cost traversal of the graph. In practice, however, users of Type & Set accept PAGE's first attempt at a solution, which is produced by successively taking the lowest cost for the current page and then moving on to consider the next. This willingness to accept compromise is caused by the slowness of PAGE when in whole-document optimisation mode, and obviously this is a shortcoming of the system. Nevertheless, people still find whole-document optimisation useful for improving documents which are badly laid out at the first attempt, usually because of problems with figure placement.

Blocks and column balancing. Pages are made up block by block. Each group of contiguous lines belonging to a single text block is collected together, with any figures called out somewhere among these lines, and any figures held over from previous pages. Lines are grouped into *shims*—bundles which cannot be split because they are locked together, or because they comprise a figure. (I have borrowed the term shim from Michael Plass [3, p. 36], who uses it in a slightly different way.) The list of shims for a block are then split into columns, and any mode-level space appearing at the top or bottom of a column is discarded. Block-level space is retained except when it appears at the top or bottom of a page.

Columns are balanced using a method that is similar, but not identical, to the method used by TeX for breaking paragraphs into lines. The method must be different, for the problem is different: paragraphs are split into an unknown number of lines, each of a known length, while in column balancing a known number of columns must be produced, each of an unknown length. The badness of a group of balanced columns is calculated in the same way that TeX uses for lines, using a function proportional to the cube of the glue ratio. Glue is set in such a way that the baselines of the bottom line of text in each column align together as well as those of the top lines. Since the height of the block is measured from its top baseline to its bottom baseline, this ensures that pages also align properly.

Figures. A figure in a Type & Set document is any section of the document starting with `\figure`, ending with `\endfigure`, and containing a sequence of figure spaces and captions. Figure spaces are inserted by writing `\figurespace <dimen>`, where `<dimen>` is the height of the space; and captions are chunks of arbitrary text in any mode belonging to a block of category `fi`.

PAGE extracts the figures and assigns each one a callout number which determines where it goes in the text. In fact, every contiguous sequence of lines of a certain category is given a callout number. This means that in a document consisting of some ordinary text, followed by a figure, followed by some more ordinary text, the callout numbers 0, 1, and 2 would be assigned to the three sections. This would cause the figure, callout number 1, to be placed somewhere after the first section of text. A figure's ideal position for Type & Set is where it appears in the original source text: the further away it ends up, the greater the penalty levied.

Internally figures are split into two groups: narrow and wide. Narrow figures span a single column in multi-column text, while wide figures are those which span all the columns of the block. (Type & Set cannot yet handle figures spanning more than one column but not all columns, such as two-column figures in a three-column block.) Wide figures are easy to place: they are inserted as soon as enough room is found, at or after their ideal position. Narrow figures are treated in a special way by the column-balancing system, in that they are allowed to float forwards from their ideal positions if the columns cannot be balanced otherwise. The algorithm which does this has recently been improved and now will very rarely fail to produce an acceptable page; but if there are just too many figures and not enough text, figures will inevitably appear one or more pages after their callouts.

PostScript and `DVI` embedding. Type & Set makes it very easy to embed PostScript pictures or existing `DVI` files in a document. The simplest way to do this is to place the command `\picture <filename>` on a line of its own in the source text. The `\picture` macro will write a `\special` to be read by PAGE, which then finds the file and determines its bounding box, deciding whether it is an encapsulated PostScript (EPS) or a `DVI` file from the first two bytes: `%!` for the former and bytes with the decimal values 247 and 2 for the latter. EPS files divulge their bounding boxes via a comment of the form `%%BoundingBox <lower-left-x> <lower-left-y> <upper-right-x> <upper-right-y>`, while for `DVI` files PAGE uses the values l and u from the postamble, unless it finds a `\special` of the form `page: bounds <n>,<n>,<n>,<n>`, which it interprets in the same way as the PostScript `BoundingBox` comment.

Having found out how big the figure is, PAGE scales it to fit the column width of the current mode. Unless the user has requested otherwise (via variants of `\picture` allowing greater control) the scaling is isomorphic: the width of the figure is scaled to be the same as the width of the column, then the height is adjusted to preserve the aspect ratio.

If the figure is a `DVI` file it is directly embedded in PAGE's output `DVI` file. To do this PAGE has to assign new numbers to the fonts in the embedded file so that they do not conflict with any in the main file; and all dimensions and fonts must be scaled by the appropriate amount as the file is read in. Only a single page, the first page of the file, is embedded: PAGE copies and scales everything between the BOP (beginning of page) and EOP, inserting a PUSH and a move to the correct x coordinate before the embedded code and a POP after it. The y coordinate need not be set explicitly: by the time PAGE has arrived at this point it will already be at the correct vertical position, since every line above the figure will have moved the current y coordinate down by its leading.

The procedure is different for PostScript files. PAGE calculates the coordinates and scaling factors needed and places them, with the filename, in a new `\special`, or rather, since it is not written by TeX, *xxx* command. DVIPS, the Type & Set PostScript driver, reads this information and loads the file in, creating a transformation matrix to scale and translate the embedded graphics.

Using this method one of Current Science's associated companies, Current Patents, publishes a journal giving details of the latest pharmacological patents. Diagrams of the chemical structures are created using commercial software which writes EPS files that are loaded with no modification into the PostScript output of Type & Set.

The following variants of `\picture` exist:

`\picture <filename>`:	scale to fit column
`\apicture <filename>`:	set at actual size
`\spicture <filename> <scale>`:	set at given scale
`\wpicture <filename> <width>`:	or given width
`\hpicture <filename> <height>`:	or given height
`\xpicture <filename> <indent>`:	or indented

A further variant, `\gpicture` or 'general picture,' gives you control over all the variables at once. In fact, all the above variants are expressed internally using `\gpicture`. Some examples:

```
\gpicture{mypic.ps}{2pc}{1in}{3cm}{0}{0}
  {0}
```

will be indented 2pc and forced to be 1in wide and 3cm high;

```
\gpicture{mypic.ps}{4pc}{20pc}{0pt}{0}
  {0}{0}
```

will be indented 4pc and forced to 20pc wide, and its height will be calculated from that: `0pt` means 'don't force'; and

```
\gpicture{mypic.ps}{0pt}{0pt}{0pt}{500}
  {750}{3}
```

will be scaled to half-size horizontally and three-quarters vertically and centred.

The `\gpicture` format is: `\gpicture {<file>} {<indent>} {<width>} {<height>} {<xscale>} {<yscale>} {<alignment>}`. If `<xscale>` is 0, `<width>` is used; and if `<yscale>` is 0, `<height>` is used. If `<alignment>` is non-zero and `<indent>` is zero the picture is aligned according to 1 = left, 2 = right, 3 = centre; otherwise it is aligned in the same way as the current mode. If `<width>` is zero it is calculated from height and vice versa. If `<width>` and `<height>` are zero both are calculated from the current column width.

Drivers and font layouts

The Type & Set drivers (with the exception of the screen previewer, which, being interactive, has to work in a different way) are all linked to two library packages, one to perform the basic DVI file interpretation, and the other to create data structures representing lines, phrases, words and characters, and to read the translation tables specifying the way TEX characters are rendered by device characters.

Thus most of a driver program is well-tested standard code, leaving only a small (300–400 lines of C code) device-dependent section containing the procedures needed to drive the actual printer or typesetter.

The `main()` function in a driver immediately calls the library function `DVImain()` with arguments giving the name of the default FD file to be read, whether accents are to be associated with the characters they are on or positioned separately, whether the device needs lines of text or separate characters, whether the output language is textual, like PostScript, or binary, like the Compugraphic language; and other information.

`DVImain()` has control for the entire run, calling other library functions to interpret the FD file and the DVI file and callback functions in the device-dependent module to set lines of text or individual characters.

Part of the reason for the simplicity of this approach is the use of a completely standard character layout on all devices. Type & Set is rigorously device-independent, like TEX itself, and apart from the font metrics no device-specific information is known at the time of running TEX or PAGE. Type & Set has a character set consisting of 640 characters, divided into five layouts:

- text
- math italic
- symbol
- math extension
- Type & Set extension

For a given printing device there are always *many* text fonts, but only *one* font in each of the other layouts: at least in logical terms, for the purposes of Type & Set. This reflects the fact that at sites using typesetting machinery such as the Linotron 100 and the Chelgraph IBX many text fonts exist, but only a few *pi* or symbol fonts. Similarly, PostScript provides a large number of text fonts but originally only one Symbol font.

The text layout is the same as that of plain TEX [4, p. 427] except for the following differences:

- character 14 changes from ﬃ to å
- character 15 changes from ﬄ to Å
- character 35 changes from # to £

These changes are all motivated by the need to have different variants of these characters in each text font, rather than have to use, say, the same pound Sterling with all the different fonts, whether bold or light, roman or italic.

The math italic, symbol and math extension layouts are identical to the plain TEX layouts [4, pp. 430–432]. This enables Type & Set to be 100% compatible with TEX mathematical setting, a feature which apart from its evident convenience absolves us from the task of writing a manual. The Type & Set extension layout, as described above, contains characters essential to book and journal publishing but entirely absent from the TEX layouts; and, since this is not the text font but one of the *pi* fonts, of which only one version exists per device, only symbols can go here, not textual characters. At present this font contains some forty characters. New accessions are made with reluctance, and only if the candidate character actually exists or can be emulated on most of the output devices.

As previously mentioned, drivers can elect to be passed complete lines or individual characters. There is flexibility too in the way a line is represented. A line-based driver will define a `DVIline()` callback function that is passed the address of a line structure. When a line arrives it is guaranteed to contain characters lying within a certain vertical distance of a common baseline, with no kerns greater than a certain size, and with no horizontal spaces greater than a predefined maximum. The idea is that drivers such as DVICORA, which generates Cora

V code for Linotron typesetters, can set the entire line at once, taking advantage of Cora's justification system and ability to interpret kerns, small amounts of up and down movement, and font changes within the line.

PostScript's `widthshow` primitive, however, although able to justify to a given measure, takes a string which must consist only of characters and spaces: movements and font changes have to be done separately. DVIPS, the PostScript driver, works at the *phrase* level rather than the *line* level. A line is a list of phrases, and phrases are defined in a much stricter way: each phrase is guaranteed to contain characters on precisely the same baseline, all in the same font, and possibly some spaces, each of which must be the same width within a certain predefined tolerance. This means that DVIPS can set each phrase using `widthshow`.

Each phrase is divided into *words*. These are composed of characters abutting horizontally, sharing a common font and baseline. Each character may have an associated accent. This last feature is used where accents are positioned by the device, and TeX's positioning must be discarded, as on Linotron devices using Cora V. No Type & Set drivers yet work at the word level, but the Compugraphic driver DVICG is an example of a character-level driver. These define a `DVIchar()` callback function that receives every character separately.

Composite fonts

A composite font is a font containing characters from more than one device font, or containing characters rendered by distorting or overlaying one or more device characters. Type & Set's composite font system is defined by FD or *FontData* files, one for each type of output device. An FD file is a readable ASCII file in a format modelled on that of Adobe Font Metric (AFM) files: that is, it is made up of sections starting with `Start<name>` and ending with `End<name>`, possibly nested, and within these sections there are data lines of the form `<key> <data>`. The main sections are `DevFonts`, mapping device font names or numbers to the names of their AFM files; `TSFonts`, mapping Type & Set fonts to device fonts; and several `Layout` sections, mapping standard Type & Set character codes to local character codes.

Rather than look at an exhaustive definition of the FD syntax and semantics, it will be more illuminating to follow two examples all the way from the DVI file to their representation in a typical output language, PostScript.

First, an ordinary character. Opcode number 12 is read from the DVI file and interpreted as 'set character 12 and move right by its escapement'. The current font is font 0, which has already been mapped to a TFM file called `time`. To convert the character into PostScript these two pieces of information are sufficient.

The TFM name `time` is is used as a key into a section in the FD file bracketed by `StartTSFonts` and `EndTSFonts`. This section associates Type & Set logical fonts with font layouts and names of AFM files, and contains the line `time`, meaning that, no layout having been specified, this font uses the default layout, given at the start of the FD file by the line `DefaultLayout text`. This tells the driver to look at the section starting with `StartLayout text` and ending with `EndLayout`. The character code, 12, is used as a key into a section within the layout called the `CharDefs` section and the line `12 174` is found, meaning that on this device Type & Set text character 12 (which, incidentally, is the fi ligature) is to be translated into character 174, the PostScript code for fi.

The PostScript font is determined by reference to a section starting `StartDevFonts`, which contains lines mapping the names of AFM or Adobe Font Metric files to a string of characters identifying the font on the device: in this case, the name `Times-Roman`. AFM files are used as the standard readable format for font metrics on all devices—not just for PostScript.

Now a composite character. This time the character code is 11, or the `ff` ligature, which does not exist in the PostScript text layout. If TFM files prepared specifically for PostScript fonts are used this character will of course never appear in the DVI file; but in this example we assume that DVIPS is being used for proofing, and must do its best to produce an emulation of something which will eventually appear, say, on the Chelgraph IBX typesetter, which *does* possess an ff ligature.

Everything happens in the same way until the layout line is reached, which is `11 102 # # 102 # [1 0 0 1 .25 0]`. Here `11` is the TeX character code, and the rest consists of two triplets, `102 # #` and `102 # [1 0 0 1 .25 0]`. Each triplet represents a device character, and is of the form `<code> <font> <transform>`, with `#` indicating that the default is to be used. The `<transform>` is a transformation matrix in the PostScript format [5, p. 65]. To render a composite character the output device takes each triplet in turn, setting the specified character from the specified device font, transforming it in the specified way. Here, if the second triplet had been identical to the first only a single f would have appeared, since they would have been superimposed; but the transform on the second f moves it right by a quarter of an em, giving a tolerable rendition of an ff ligature.

You can see that the ordinary character in the first example uses the same syntax as the composite character once you know that any trailing `#` tokens can be dropped: the layout line for character 11 can be expressed more pedantically as `11 174 # #`.

At present there are two noticeable defects in the composite font system. The first is that a character cannot contain rules, and the second is that the transformation cannot make use of character metrics. It would be very useful to be able to define a transformation to move a character right by half the width of another character, but at the moment the x and y translation elements in a transform can be expressed only in ems, or, more accurately speaking, in fractions of the current pointsize.

`FD` files, as well as being read by the drivers, are used to generate `TFM` files for the various devices. For non-PostScript devices a utility called MAKEAFM creates `AFM` files from the width tables and other metrics supplied by manufacturers, while PostScript `AFM`s are downloaded via the Adobe PostScript Archive File Server. The `AFM`s are then used by another utility, MAKETFM, in conjunction with the `FD` files, to write the `TFM`s. This has to be done after any revision to the `FD` or `AFM` files that could affect character metrics.

Tables

Medical journals are full of tables, most of which contain spans, brace rules and numbers centred on a decimal point or a ± sign. The difficulty of creating tables in TeX makes an automatic table generator not only desirable but essential. The way Type & Set solves this problem is for the user to type the table in a sort of *wysiwyg* format, preferably, but not necessarily, using a special editor giving access to symbols representing column delimiters and rules. All the user has to do is ensure that the table is topologically equivalent to the desired result. Specifically, he or she must align the column breaks and arrange for the right things to span.

Each cell of the table can contain ordinary text, with or without TeX control sequences, plus special characters to tell the table processor how to align the cell. The absence of an alignment character causes the text in the cell to be centred. Macros are also available to insert multi-line paragraphs into cells if necessary. These are justified and hyphenated according to parameters defined in the table mode.

Here is an example: a modified and shortened version of a table in *The TeXbook* [4, p. 246]. The user creates a file called, say, `mytable.tab`, containing the following:

```
\narrow
|──────────|──────────────────|
| Year     | World Population |
|──────────|──────────────────|
| 8000 B.C.| 5,000,000        |
| 50 A.D.  | 200,000,000      |
|──────────|──────────────────|
```

The vertical and horizontal rules here represent symbols taken from the standard IBM PC extended character set; but any symbols can be used, since the table system itself reads its special characters from a table. The control sequence `\narrow` is the table mode to be used.

When TABLE, the Type & Set table generator, is run, it translates the above into TeX code:

```
\vfil\eject
\narrow
\setbox0\hbox{a}
\boxtable{\offinterlineskip\tabskip=0pt
\halign to \hsize{\vrule # \tabskip=0pt%
&# \tabskip=\tg& # & #\tabskip=0pt%
&\vrule #&# \tabskip=\tg& # & #%
\tabskip=0pt&\vrule #\cr\vrzero height%
\hrzeroht&\multispan{3}\hrf{0}%
&\vrzero height\hrzeroht&\multispan{3}%
\hrf{0}&\vrzero height\hrzeroht\cr
\vrzero\global\tablepos=1&&\tsp Year%
\strut \tsp&&\vrzero%
&&\tsp World Population\tsp&&\vrzero\cr
\vrzero&\multispan{3}\hrf{0}&\vrzero&%
\multispan{3}\hrf{0}&\vrzero\cr
\vrzero&&\tsp 8000 B.C.\strut \tsp&&%
\vrzero&&\tsp 5,000,000\tsp&&\vrzero\cr
\vrzero&&\tsp 50 A.D.\strut \tsp&&\vrzero%
&&\tsp 200,000,000\tsp&&\vrzero\cr
\vrzero depth0pt\global\tablepos=2%
&\multispan{3}\hrf{0}&\vrzero depth0pt%
&\multispan{3}\hrf{0}&\vrzero depth0pt\cr
}}
\tablewrapup
\vfil\eject
\endinput
```

This contains many control sequences referring to parameters defined in the style sheet, such as the width of the table and the weight of the rules. A table is thus 'soft', in the sense that a given `TAB` file can be typeset in varying ways depending on the style sheet; or can even be used in two different documents, looking different in each, because of differences in the definitions of the table mode. It is common practice to define a `\narrow` and a `\wide` table mode, and if a table exceeds the allowed width when typeset `\narrow`, it can be set using `\wide`. The table is included in the source text using the command `\input mytable`, and when PAGE analyses the `DVI` file it recognises the table as such and treats it as a figure, floating it to a convenient position. The code above produces the following result:

Year	World Population
8000 B.C.	5,000,000
50 A.D.	200,000,000

Conclusion

TeX is readily available, standard, stable, reliable and well documented. It is also complicated and hard to program in; and unsuitable for multi-column setting and baseline-to-baseline measurement. These facts have led us to use TeX's incomparable facilities for galley setting as the inner engine of our typesetting system, but to replace TeX's page make-up system with our own post-processor program, written not in TeX but in C. The TeX programming difficulties have been obviated by arranging for TeX macro packages to be written automatically by a style sheet editing program; and tables are coded not in TeX but on the screen in a *wysiwyg* format.

The other big TeX problem concerns fonts. Journals and books must be typeset using standard commercial faces such as Garamond, Optima, Helvetica, Univers, and Gill. TeX, while not in theory connected with any particular typeface or character set, is in practice closely bound to Knuth's Computer Modern family and its character set. The main achievement of Type & Set, apart from the page make-up system, is to retain almost complete compatibility with the plain TeX font layouts while typesetting on standard equipment—such as the Chelgraph IBX—using the standard typefaces available on the equipment.

These things make up Type & Set. This system has enabled us at Informat and Current Science to typeset about thirty academic journals and many other publications automatically using conventional typesetting equipment and standard fonts.

Afterword

Since this article was written two names have changed. The name of the software is to change from Type & Set to PageTeX, which, although similar to several other names of systems involving TeX, highlights the most important feature: automatic page make-up.

Informat Computer Communications is now subsumed into Life Science Communications, the holding organisation for Current Science and other companies, and it is Life Science Communications which will market PageTeX. Only the names have changed: the people and the software are the same.

References

1. Mittelbach, Frank. "E-TeX: Guidelines for Future TeX Extensions." *TUGboat*, 11(3): 337–345, September 1990.

2. Beebe, Nelson. Personal communication, September 1990.

3. Plass, Michael F. *Optimal pagination techniques for automatic typesetting systems*. Department of Computer Science, Stanford University, California, 1981.

4. Knuth, Donald E. *The TeXbook*. Addison-Wesley, May 1989.

5. Adobe Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, 1985.

◇ Graham Asher
Life Science Communications Ltd
34–42 Cleveland Street
London W1P 5FB
England
Telephone +44 81 348 1043