DVItoVDU: A TEX Page Previewer

Andrew Trevorrow
University of Adelaide

DVItoVDU is an interactive program that allows the user to view pages from a TEX82 DVI file on a variety of commonly available visual display units (VDUs). It runs under VAX/VMS and is written in Modula-2 from the University of Hamburg.

The software is in the public domain and available on the VAX/VMS distribution tape. Most of this article is based on material from the *DVItoVDU User Guide* and the *DVItoVDU System Guide*. The TEX source files for these two documents are also on the distribution tape.

The version described below is numbered 1.5 (October 1985).

Conception

TEX usage began at Adelaide University early in 1984 and the clamour for a previewing program started soon after. Although there are excellent, rational reasons for such a tool in a TEX system, our users had somewhat more pragmatic concerns: people had to come to the Computing Centre to collect their laser printer output; there was a charge of 10 cents per page! (This has since been dropped.)

Another concern was the high level of paper wastage, particularly in those early days when most users, including myself, were learning about TEX and all its intricacies. While the need for a previewer was obvious, I could see a number of difficulties in writing such a program.

Around this time I happened to stumble upon Hamburg's Modula-2 system. After a few weeks of pleasant experimentation with this new language, I eventually realized that Modula-2's procedure variables provided an elegant, high-level solution to one of the key design problems for the particular previewer I had in mind; that is, the need to efficiently drive a variety of terminals from the one program. Up until then I'd been reluctant to seriously consider starting the project. The thought of having to resort to VAX Pascal wizardry or MACRO magic was just too depressing.

Why not `WEB`? Quite apart from the above design problem, it was never really a serious contender. At the risk of being sacrilegious I must confess to having some reservations about the `WEB` system. I believe its benefits are outweighed by the disadvantages of using a cryptic language in which `WEB`, TEX *and* Pascal errors are all possible.

Modula-2 in fact matches `WEB` in its facilities for creating highly modular programs. In addition, system generation is much faster because Modula-2 allows separate compilation. It is usually possible to make a change to an implementation module, compile and link, and have a new EXE file in seconds. Compare this with the many minutes normally needed to change a `WEB` module, run `TANGLE`, then Pascal and the linker (not to mention `WEAVE`, TEX, etc., if you want your documentation up-to-date).

A modern language such as Modula-2 and a good screen editor are sufficient tools, I believe, to create well-structured software with good-quality, internal commentary. Having access to TEX or some other typesetting system to create accompanying documentation is an added luxury.

I was also keen to write a fairly large program in Modula-2 to see how it compared with Pascal. I must say I was pleasantly surprised by the utility of Modula-2 and the reliability of Hamburg's compiler.

Design Considerations

The main design goal was to have just the one preview program able to work efficiently on various types of terminals used throughout the campus. Since many TEX users do not have access to a high-resolution graphic VDU, the program also had to produce useful displays on a simple ANSI

terminal, such as a VT100. A number of other capabilities were considered essential:

- Absolute page selection using either the natural DVI page order or the TEX page counters.
- Relative page selection by requesting the next page in either direction.
- "Pan and zoom" (the ability to view any region of a selected page, and at any desired scale);
- Error detection in the form of explicit warnings about such problems as a page off the paper or the use of a non-existent font size.

A preliminary version of DVItoVDU that met most of these goals was released in September 1984. A number of substantial changes have been made since then, mainly to improve efficiency and to provide a more flexible user interface.

Running DVItoVDU

The information in this section is a condensation of the *DVItoVDU User Guide*.

We'll assume you've just run `foo.tex` through TEX to create `foo.dvi`. To look at the pages in this DVI file you simply type '`dvitovdu foo`'. Some command options may be necessary if DVItoVDU is to work properly. In particular, the `/vdu` qualifier must correctly describe the type of terminal you are using.

The DVItoVDU command can be followed by a number of qualifiers, where each is assigned a value:

`/vdu`	type of terminal
`/resolution`	pixels per inch
`/xsize`	paper width
`/ysize`	paper height
`/magnification`	override magnification
`/font_directory`	master font directories
`/dummy_font`	used if font not found
`/help_file`	used by `?` command

The last three are really for system wizards; their default values should be set up so that most users need never worry about changing them. (The DVItoVDU command is installed in the system DCL tables using a command language definition file supplied with the software. This CLD file can be modified to specify default qualifier values suitable for your site.) Let's look at all the qualifiers in more detail:

VDU Type The `/vdu=`*string* qualifier is used to tell DVItoVDU what type of VDU you are using. *string* is a string of characters terminated by a space or the start of the next qualifier. Most sites might set up '`/vdu=ANSI`' as the default. If `ANSI` does not describe your VDU, you need to override the default value. For example, if you're using a VISUAL 550 terminal, type '`dvitovdu/vdu=vis550 foo`'.

The current version of DVItoVDU will accept the following `/vdu` values:

`AED483`	AED with 512 by 483 screen
`AED512`	AED with 512 by 512 screen
`ANSI`	any ANSI compatible VDU
`REGIS`	any ReGIS compatible VDU
`VIS500`	VISUAL 500
`VIS550`	VISUAL 550
`VT100132`	VT100 in 132 column mode
`VT640`	VT100 with Retro-Graphics

`VT100` and `VT220` are synonyms for `ANSI`. `GIGI`, `VK100`, `VT125` and `VT240` are synonyms for `REGIS`.

Printer Resolution The `/resolution=`*i* command tells DVItoVDU the resolution of the device that will be used to print your document. DVItoVDU treats the imaginary sheet of paper on which a DVI page will appear as a two-dimensional array of tiny dots known as "paper pixels." *i* is a positive integer that defines the number of paper pixels per inch, horizontally *and* vertically. We have an Imagen IMPRINT-10 laser printer, so our default `/resolution` value is 240.

Printer Page Size The `/xsize=`*dimen* and `/ysize=`*dimen* qualifiers qualifiers define the dimensions of the paper upon which your document will be printed. `/xsize` defines the width and `/ysize` the height. Every time you select a page, DVItoVDU will use these paper dimensions to check that the page edges fall within the paper edges. *dimen* is a positive integer or real number followed by a two-letter unit: `in`, `cm`, `mm`, `pc`, `pt` or `px`. Most of these should be familiar from TEX. DVItoVDU provides an additional unit, `px`, for paper pixels. (These two-letter sequences are the same as the commands used to change the units of dimensions; more about all the commands on page 28.) Our laser printer uses A4 paper by default; i.e., `/xsize=8.3in` and `/ysize=11.7in`.

```
Total pages=n   DVI page=0   TeX page=[0]   Next=>   Terse
Window at (h,v) wwd by wht   Page at (minh,minv) pwd by pht   IN
status
Command:
```

Explanation of entries:

`Total pages=`*n*	The total number of pages in the DVI file.
`DVI page=0 TeX page=[0]`	The current page number and its corresponding TEX page counters.
`Next=>`	The direction the `N` command will advance through pages (initial setting is forward).
`Terse`	The current display mode — one of `Terse`, `Box`, and `Full` (initially terse).
`Window at (`*h*`,`*v*`)` *wwd* `by` *wht*	The current location of the window's upper left corner and the size of the window in paper coordinates.
`Page at (`*minh*`,`*minv*`)` *pwd* `by` *pht*	The location of the page rectangle's upper left corner and the size of the rectangle. (The page rectangle is the smallest rectangle enclosing all rules and characters on the current page).
`IN`	The current unit of measure — one of `IN`, `CM`, `MM`, `PT`, `PC`, and `PX` (initially inches).
status	Line for status and error message display (initially blank).
`Command:`	Line for command entry.

Figure 1.
The Initial Dialogue Region.

DVI File Magnification The `/magnification=`*i* qualifier allows you to replace the magnification used in the DVI file with some other value; *i* is a positive integer 1000 × the desired magnification. The given value should be chosen carefully so that the new font sizes still correspond to existing PXL files. You should only supply a replacement magnification if you intend to print the DVI file with the same override.

Font Directory The `/font_directory=`*list* qualifier selects a master directory containing subdirectories of font PXL files. DVItoVDU gets all its font information from PXL files. *list* is a list of values separated by commas and enclosed in parentheses. A typical default list might be `(d1:[local.fonts],d1:[tex.fonts])`.

Dummy Font The `/dummy_font=`*string* qualifier specifies a font to be substituted when a font cannot be found at the size required by your document. DVItoVDU will warn you if your document uses a font at a non-existent size. Rather than abort, it will load the PXL file specified by `/dummy_font` and continue so you can look for more errors. Paragraphs using this dummy information are likely to have ragged right margins. A typical default dummy font might be `d1:[tex.fonts.1200]amr10.pxl`.

Help File The `/help_file=`*string* qualifier specifies a file containing the text to be displayed by the `?` command. Our default file is `d1:[tex]dvitovdu.hlp`.

VDU Dialogue Region

If your command line is correct and if the `/vdu` value matches the type of terminal you're actually using, then DVItoVDU will clear the screen and display something similar to Figure 1. These four lines represent the "dialogue region." The rest of the screen is called the "window region" and should be blank at this stage.

The top two lines show status information. Until a page is selected, most of the status values are meaningless and set to zero.

The third line is initially blank. DVItoVDU displays messages of various kinds in this line. Some of these messages appear only briefly but may convey helpful information. Others are more important and indicate some sort of problem, such as an invalid command or a page that won't fit on the paper; in these cases DVItoVDU will prompt you to hit the `RETURN` key before continuing.

The last line in the dialogue region is for entering commands. The first thing you normally want to do is choose a particular page for display. For example, typing '1' will select the first page in the DVI file. Many commands can be entered in the one command line. Hit `RETURN` to execute the command(s).

VDU Window Region

A paper pixel can be either black (corresponding to a tiny blob of ink) or white (no ink). A typical DVI page contains characters from one or more fonts, and perhaps a few rules. A rule is simply a rectangular region of black pixels, usually in the shape of a thin horizontal or vertical line. A character is usually a more complicated pattern of black and white pixels. Every character and rule has a paper position—or reference point—defined by a pair of pixel values (h,v) where h is the horizontal coordinate and v is the vertical coordinate. DVItoVDU uses a paper coordinate scheme in which the position (0,0) is a pixel one inch in from the top and left edges of the paper. Vertical coordinates increase down the paper, horizontal coordinates increase to the right. Confused? Figure 2 may help clear things up.

The window region is used to view the current DVI page. DVItoVDU treats this region of the VDU screen as a two-dimensional array of dots, but we refer to these dots as "screen pixels" to distinguish them from the paper pixels described above. A screen pixel is usually the smallest possible area on a VDU screen that can be drawn or erased; the greater the number of screen pixels in a given area, the higher the resolution of the VDU. (DVItoVDU's definition of a screen pixel is more precisely known as an "addressable location.")

The initial window sizes (widths by heights in pixels) for the VDUs currently implemented are:

`AED483`	512 by 442
`AED512`	512 by 471
`ANSI`	80 by 20
`REGIS`	768 by 400
`VIS500, VIS550`	1024 by 688
`VT100132`	132 by 20
`VT640`	1024 by 650

The most accurate representation of a page will occur at these unscaled values, since one paper pixel equals one screen pixel. You can increase or decrease the area currently visible by using the `H` and `V` commands to change the size of the window region. The higher the resolution of the VDU, the greater the accuracy of such scaled displays.

Note the very low resolution of the `ANSI` and `VT100132` VDUs. These are not really graphic terminals; DVItoVDU has to define a screen pixel to be an entire character position, since the individual dots making up characters cannot be turned on and off. An `ANSI` screen typically consists of 24 lines of 80 columns, therefore the initial window region is 80 pixels wide and 20 pixels high (the top 4 lines are used for the dialogue region). At more useful window sizes the resulting displays will be extremely crude. Nevertheless, a variety of formatting errors can still be detected on such terminals; just don't try proofreading your document!

The size and location of the window region are automatically set every time a page is selected. DVItoVDU tries to show as much of the paper (and presumably the page) as possible, and without too much distortion. After comparing the shape of the paper with the shape of your VDU's *initial* window region, and depending on the location of the page, DVItoVDU may show the entire paper, or the top or bottom half, or the left or right half. If any part of the page is off the paper then the entire paper *and* the entire page will be shown. Since most paper sizes have portrait dimensions (width < height), and most VDU screens have landscape dimensions (width > height), DVItoVDU will normally show the top half of a sheet of paper containing the selected page.

Every time the window region needs to be updated, the entire screen is first erased.

DVItoVDU Commands

In response to the '`Command:`' prompt you can enter one or more of the following commands in upper- or lowercase. Multiple commands are processed in the order given but the window region is only updated, if necessary, at the end. For example, `NFD` gets the Next page, switchs to Full display mode, moves the window Down, and only then displays the page. If an invalid command is detected, any further commands are ignored. Most commands consist of only one or two characters; some can be followed by parameters. Spaces before and after commands and parameters are optional.

The DVItoVDU commands are summarized in Figure 3, and they are described in detail in the sections that follow.

Miscellaneous Commands

Help The `?` command displays help on the available commands.

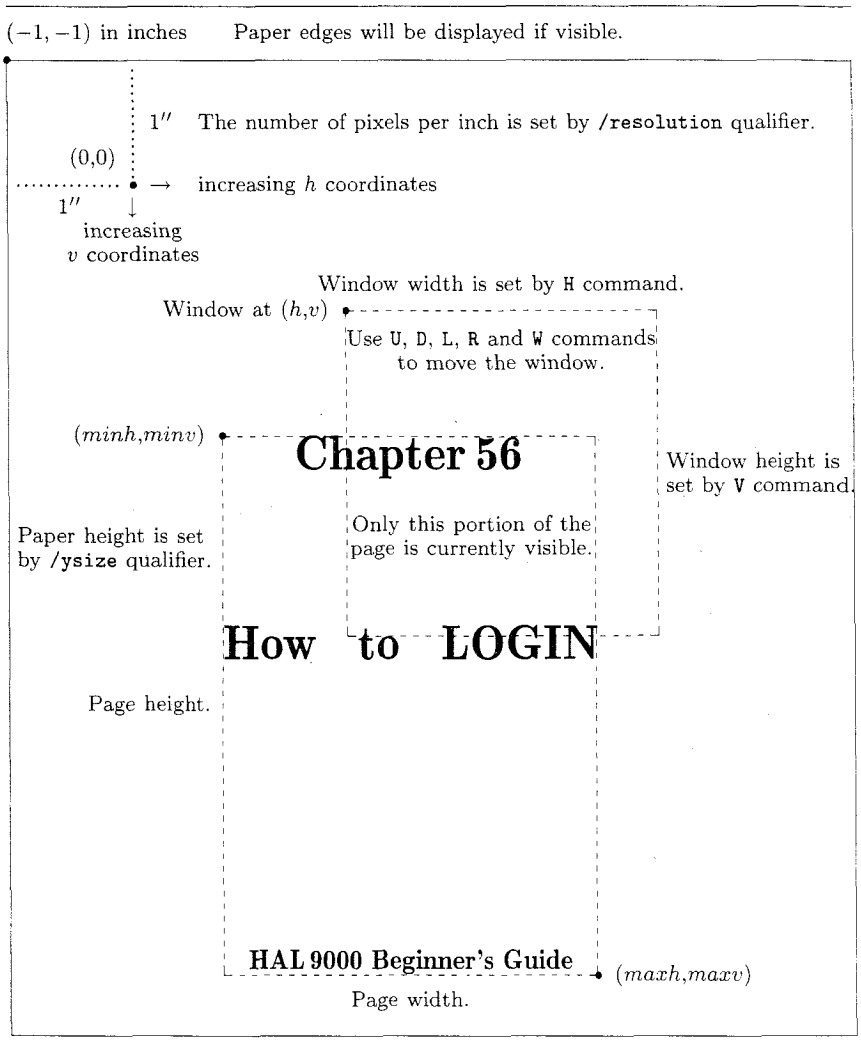


Figure 2.
DVItoVDU's Paper Coordinate Scheme.

Status The `S` command shows various qualifier values and statistics about the number of fonts, characters and rules used on the current page.

Quit The `Q` command quits from DVItoVDU.

Page Selection

Selection by Page Position The i command selects the ith DVI page. i must be a positive integer from 1 to n where n is the total number of pages in the DVI file.

Selection by Page Counters The $[i_0.i_1.\ \cdots\ .i_9]$ command selects the DVI page whose ten TEX page counters match the given specification. i_0 to i_9 are integers separated by periods. Each integer is optional and trailing periods may be omitted. An absent integer will match any value in the corresponding counter. If more than one DVI page matches, the lowest will be chosen. For example, `[]` is equivalent to `[.........]` and will select the first DVI page, even though the request matches every possible page. If your TEX source file doesn't change the value of `\count0` (`\pageno` in plain TEX) then the ith DVI page will match TEX page $[i]$.

Next Page The N command selects the next DVI page, depending on the current DVI page and the current direction (> or <). Before any page has been requested, N will select the first DVI page if the current direction is >, or the last DVI page if the current direction is <.

Set Forward Direction The > command arranges for future N commands to select DVI pages in ascending order. If i is the current DVI page, an N command will get page $i + 1$ unless i is the last DVI page.

Set Backward Direction The < command arranges for future N commands to select DVI pages in descending order. If i is the current DVI page, an N command will get page $i - 1$ unless i is 1.

Changing the Page Display

The way in which the current page is displayed can be varied from a full, accurate representation to a terse, fast display for when fine details are unimportant. The window region is updated in the following manner: Visible paper edges are drawn first followed by visible rules (shown in full no matter what the display mode). Visible characters are finally shown on a font by font basis; those fonts with the least number of characters on the page are drawn first. Every few rules or characters, DVItoVDU will check to see if you've typed something at the keyboard; you can hit the RETURN key to abort the display, or you can change the display mode by hitting the T, B or F keys.

Terse Character Display The T command displays a terse representation of characters. On most VDUs the TEX text fonts should be readable; the characters will be in approximately the right position and may even be about the right size. Note that the text fonts will all look alike; you won't be able to distinguish between roman and bold characters for example. Most VDUs assume all characters come from a TEX text font and map them into similar-looking ASCII characters. Characters from non-text fonts, such as math symbols, will usually appear incorrect.

Outline Box Character Display The B command displays box outlines of the smallest rectangles containing all black pixels in characters. The reference points of most TEX characters are usually located near the bottom left corners of these boxes. Box mode is intermediate in speed between Terse and Full modes.

Full Character Display The F command displays a full representation of all pixels in characters. This display is the most accurate but may take some time; hit RETURN or switch to Terse or Box mode if you get bored. On the higher resolution VDUs, a good compromise between speed and accuracy is to start off in Full mode so that math symbols and any other special characters are displayed correctly, and to switch to Terse mode when the bulk of the text begins.

Changing Units of Dimensions

All the numbers in the second line of the dialogue region are dimensions in terms of the units shown at the end of the line. The parameters following some commands are also dimensions in terms of these units. Unlike the dimensions in TEX, you don't explicitly type the units when you need to specify a dimension to DVItoVDU; simply enter an integer value or real value (which will be truncated to four decimal places if necessary). A given value is rounded up internally to the nearest paper pixel based on the current units and the conversion factors shown below.

Inches The IN command causes dimensions to be shown and entered in terms of inches (/resolution defines the number of paper pixels per inch).

Centimetres The CM command causes dimensions to be shown and entered in terms of centimetres (2.54 cm = 1 in).

Millimetres The MM command causes dimensions to be shown and entered in terms of millimetres (10 mm = 1 cm).

Picas The PC command causes dimensions to be shown and entered in terms of picas (1 pc = 12 pt).

Points The PT command causes dimensions to be shown and entered in terms of points (72.27 pt = 1 in).

Pixels The PX command causes dimensions to be shown and entered in terms of paper pixels.

Moving the Window

The window region can be moved to any position over the current page. The parameters h and v are dimensions ranging from −480 inches to +480 inches (for those of you TEXing billboards). You will be told if the entire window moves outside the page rectangle defined by *minh*, *minv*, *maxh* and *maxv*. If this does happen, the movement is restricted to *just outside* the edges to make it easier to get back over the page using only the U, D, L and R commands. Note that the location of the window is

automatically set every time a page is selected. This position will normally be $(-1, -1)$ in inches; i.e., the top left corner of the paper.

Set Window Position The `W` h,v command moves the window region's top left corner to the given paper position. h is the horizontal coordinate, v is the vertical coordinate. If h and v are absent then the window is moved to (*minh*,*minv*), the top left corner of the page rectangle.

Up The `U` v command moves the window up v units. If v is absent then window moves up by its current height.

Down The `D` v command moves the window down v units. If v is absent then window moves down by its current height.

Left The `L` h command moves the window left h units. If h is absent then window moves left by its current width.

Right The `R` h command moves the window right h units. If h is absent then window moves right by its current width.

A positive integer following a command that can take a parameter is never interpreted as a DVI page selection. For example, '`D5`' will always be interpreted as "move the window down 5 units" and never as "move the window down by its current height and then select page 5." As it turns out, this ambiguity is not a problem because all the commands that can have parameters only affect the *current* page. There is no point in moving the window (or changing its size) and then selecting a page in the same command line since the window location is automatically reset every time a page is selected.

Changing the Window Size

The width and height of the window region can be changed independently. It is up to you to maintain a suitable aspect ratio. The location of the window will not change unless the page becomes invisible. The parameters *wd* and *ht* are dimensions ranging from 1 pixel to 480 inches. Note that the width and height of the window are automatically set every time a page is selected; their values will normally depend on the paper dimensions.

Horizontal Size The `H` *wd* command sets the Horizontal size of the window to the given width. If *wd* is absent then window width is set to its initial, unscaled value.

Miscellaneous

`?` Show help on commands.
`S` Show qualifier values and current page status.
`Q` Quit.

Page Selection

i Select ith DVI page.
$[i_0.i_1.\cdots.i_9]$ Select DVI page according to TEX page counters.
`N` Select next DVI page.
`>` Make `N` command move forward.
`<` Make `N` command move backward.

Page Display

`T` Select terse character display.
`B` Select bounding rectangle character display.
`F` Select full pixel character display.

Units of Dimensions

`IN` Use inches.
`CM` Use centimetres.
`MM` Use millimetres.
`PC` Use picas.
`PT` Use points.
`PX` Use pixels.

Moving the Window

`W` h,v Move top left corner to (h, v).
`U` v Move up v units.
`D` v Move down v units.
`L` h Move left h units.
`R` h Move right h units.

Changing Window Size

`H` *wd* Set horizontal size to *wd*.
`V` *ht* Set vertical size *ht*.

Figure 3. *Summary of Commands.*

Vertical Size The `V` *ht* command sets the Vertical size of the window to the given height. If *ht* is absent then window height is set to its initial, unscaled value.

Just a brief note on the scaling method used by DVItoVDU: Rules and glyphs having the same top, bottom, left or right paper coordinates also have the same scaled coordinates. This ensures baselines will line up but means that rules and glyphs may change shape when the window is moved to a new position. The effect is most noticeable on the low-resolution VDUs.

Command Examples

The spaces between commands in the following examples are for clarity; they are not mandatory. Commands can also be typed in lowercase.

`< N N N`
If these commands are given before any page has been requested, they will select the 3rd last page (assuming there are at least 3 pages). Note that DVItoVDU will only display the 3rd last page. The intervening pages are still processed though, and you'll be warned about any problems with them (such as a page off the paper).

`F W H V`
This command sequence can be very useful for looking carefully at the results of a small TEX experiment. You might, for instance, want to check the appearance of two characters moved closer together by a negative `\kern`. Remember that the unscaled window size chosen by `HV` produces the most accurate display, since each screen pixel corresponds to exactly one paper pixel.

`R9999 L D9999 U`
Sometimes you need to move quickly to the right edge of the page to have a look at line breaks, or you might want to go to the bottom and look at where the page was broken. This particular command sequence will move the window's *bottom right* corner to the bottom right corner of the current page (*maxh*,*maxv*). `R9999` moves the entire window to the right of the page, but only just. `L` then moves the window left by its current width so that the *right* edges of the window and the page coincide. `D9999` moves the entire window below the page, but only just. `U` then moves the window up by its current height so that the *bottom* edges of the window and page also coincide.

`IN W-1,-1 V12 H16 T`
It is often useful to get an overview of the positioning of the page within the entire paper. These commands will do just that on all the VDUs currently implemented (assuming A4 paper dimensions). `IN` sets the current units to inches just in case they were something else and `W-1,-1` moves the window to the top left corner of the paper. `V12` sets the window height to slightly more than A4 paper height and `H16` sets the window width to a value that will ensure the paper shape maintains approximately the right proportions. `T` sets the display mode to Terse since you're probably not concerned with finer details when looking at the entire paper.

`[.56] W IN H2.5 V2.6 R1 U1 F`
This is the likely command sequence that led to the window display shown in Figure 2. The author has been clever enough to define a `\chapter` macro that sets `\count1` to the given chapter number. `[.56]` will thus select the appropriate DVI page (`\count0` is ignored).

Description of System Design

Most of the material in this section is from the *DVItoVDU System Guide*.

As its name would imply, a Modula-2 program is typically built up from a number of separately compiled modules. Each module can import data and procedures from other modules. An imported module is further decomposed into a definition module and an implementation module. The definition part contains declarations for all exported objects and serves as the interface to client modules. The details of how exported objects are actually realized are hidden from clients in the implementation part. Rapid system regeneration is possible because client modules depend only on the definition part; the implementation part may be modified (e.g., optimized) without the need to recompile any client modules. Figure 4 shows the importation dependencies of all the modules making up DVItoVDU.

Not only do modules allow a large program to be broken up into more manageable chunks, they also provide a sensible basis for a description of that program:

DVItoVDU The main module is primarily concerned with the user interface. It handles all the interactive command processing and contains the high-level logic used to update the dialogue and window regions. Data and procedures are imported from the following modules to help the main module carry out its many tasks.

DCLInterface This module provides the interface to the VAX/VMS command language interpreter. (DCL stands for Digital Command Language, just in case you're wondering.) The command line used to invoke DVItoVDU is parsed and the DVI file name extracted. All the qualifiers are also initialized, either to explicit values given in the command line or to site-dependent default values.

VDUInterface DVItoVDU can work efficiently on different types of VDUs by letting Modula-2 procedure variables act as generic VDU routines. These routines, along with the generic VDU parameters, are defined in *VDUInterface*. As little as possible is assumed about the capabilities of a VDU. In particular, the availability of a graphic input device is ignored. DVItoVDU should be able to work on any terminal that can:

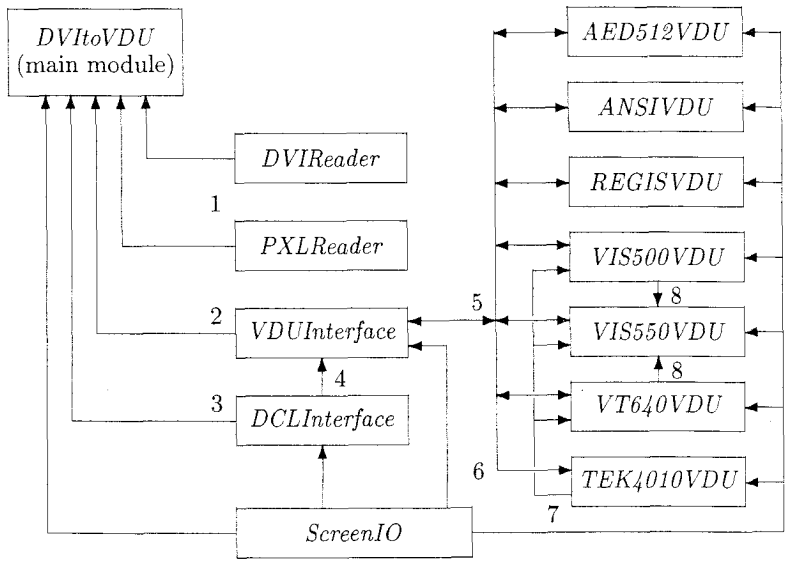


Figure 4.

DVItoVDU's Module Map. An arrow from module A to module B indicates the latter imports data and/or procedures from the former (B is said to be a client *of A). If the definition part of module A changes, then all client modules must be recompiled and DVItoVDU relinked. If the implementation part of module A changes, only the relinking stage is necessary.*

Points of interest:

1. *DVIReader* does not depend on *PXLReader*. It does know that PXL files contain crucial typesetting data, but leaves it up to the main module to specify how and where to get such information.
2. *VDUInterface* exports the generic VDU routines and parameters used in the main module.
3. *DCLInterface* extracts the DVI file name and qualifiers from the VMS command line.
4. *VDUInterface* needs the `/vdu` value to select an appropriate VDU initialization routine.
5. Specific VDU modules import the generic VDU routines and parameters from *VDUInterface* and, in return, export an initialization routine. *VDUInterface* will decide which one is actually executed.
6. The Tektronix 4010 module imports the *TeXtoASCII* array to map TEX characters into corresponding ASCII characters. Note that *VDUInterface* does not know about the Tektronix 4010 module.
7. The VISUAL 500, VISUAL 550 and VT640 terminals all emulate Tektronix 4010 graphics.
8. The VISUAL 550 terminal uses VISUAL 500 graphic routines to update the window region and VT640 routines to update the dialogue region.

- Mix text and graphics on the screen (some VDUs make no distinction).
- Erase all of the screen, or individual text lines.
- Move the cursor to any given screen pixel.
- Display a rectangular region of screen pixels (possibly just one).

The generic VDU routines are:

StartText	switch to "text mode"
ClearTextLine	erase given line
MoveToTextLine	move to start of given line
ClearScreen	erase the entire screen
StartGraphics	switch to "graphics mode"
LoadFont	for later *ShowChar* calls
ShowChar	show given Terse character
ShowRectangle	show given rectangle
ResetVDU	may need to reset VDU

Most are quite trivial to implement for a specific VDU. The main module looks after all the tricky graphic operations such as the clipping of characters and rules outside the current window region, and the way in which visible paper pixels are scaled to screen pixels.

From an efficiency point of view, the two most critical routines are *ShowChar* and *ShowRectangle*. *ShowChar* is used by the main module to display a character in Terse mode. The only information given is the TEX character (currently restricted to `\char0..\char127`) and its screen position. Since characters are displayed one font at a time, some VDUs can use scaling information sent by the most recent *LoadFont* routine to select an appropriate hardware font.

ShowRectangle is used for all other window graphics. It is used to draw the paper edges, to draw all rules (regardless of display mode), to draw all glyph outlines in a Box display, and to draw the horizontal lines making up all glyphs in a Full display. The majority of rectangles are in fact horizontal or vertical lines just one screen pixel thick. The generic VDU parameters are:

DVIstatusl	DVI status line, usually 1
windowstatusl	window status line, usually 2
messagel	message line, usually 3
commandl	command line, usually 4
bottoml	bottom text line in screen
windowh	window's top left h coord
windowv	window's top left v coord
windowwd	unscaled window width
windowht	unscaled window height

These parameters are actually integer variables. The main module treats them as constants.

There are two screen coordinate systems used by DVItoVDU:

- When updating the screen in text mode (i.e., when updating the dialogue region or during a `?` or `S` command), DVItoVDU assumes text lines start at 1 and increase downwards. The bottom text line on the screen is given by the parameter *bottoml*.
- When updating the screen in graphics mode, DVItoVDU assumes the top left screen pixel is at (0,0). Horizontal coordinates increase to the right and vertical coordinates increase down the screen. The top left pixel in the window region is at (*windowh*,*windowv*). Specific VDU modules may have to do a translation to the actual coordinate scheme used by the VDU. The size of the window region in screen pixels is given by *windowwd* and *windowht*.

AED512VDU, ANSIVDU, REGISVDU, ... Each specific VDU module exports an initialization routine that will assign appropriate procedures to the generic VDU routines and specific integers to the generic VDU parameters. *VDUInterface* imports all these initialization routines and uses the `/vdu` value set in *DCLInterface* to execute one of them.

DVIReader This module exports the routines and data structures needed to move about randomly in a DVI file and interpret selected pages. Although the main module is currently the only client, it is anticipated that *DVIReader* could just as well form the basis of a more conventional DVI translator such as a non-interactive device driver.

Font, character and rule information is stored in dynamically allocated lists to avoid imposing any limit on their numbers. The length of the font list is determined soon after opening the DVI file by reading all the font definitions in the postamble. Each font node is a record made up of many fields; one of these fields is the head of a character list. The nodes in each character list will store the positions and TEX codes of all characters on a page. Besides the font list, there is also a rule list. The nodes in the rule list will store the positions and dimensions of all rules on a page. (Character and rule positions are stored as pairs of horizontal and vertical paper pixel coordinates. The manner in which *DVIReader* calculates such positions is based firmly on Donald Knuth's `DVItype`.)

Just before interpreting a selected DVI page, the rule list and all the character lists are deallocated if necessary. During interpretation, *DVIReader* adds a new rule or character node to the *tail* of an appropriate list. When the main module processes such lists, rules and characters will be displayed in somewhat the same sequence as seen in the DVI page; i.e., top-to-bottom and left-to-right. (Since there is a separate rule list, as well as a character list for each font, the precise sequence is not remembered.) After interpretation, the nodes in the font list are sorted so that fonts with the least number of characters (> 0) will be processed first.

The various lists contain most of the information needed to display the page; they are traversed by the main module whenever the window region is updated. If the page isn't empty then *DVIReader* will also determine the edges of the page rectangle. This is the smallest rectangle containing all black pixels in glyphs and rules, as well as all character reference points (needed for Terse displays). The main module uses the page rectangle to decide if

the page is off the paper, and to restrict window movement.

PXLReader DVItoVDU gets all its character information from standard PXL files. *PXLReader* exports routines for moving about in such files and grabbing various bytes and words. A PXL file contains the crucial TFM widths needed by *DVIReader* to correctly position characters when interpreting a DVI page. These widths, along with other details about each glyph, are kept in a PXL file's font directory. A directory is loaded just once for each font (the very first time the font is seen) and DVItoVDU will display '`Loading font data from ...`' in the message line.

A PXL file also contains glyph shape information (in the form of bitmaps) for all characters in a TEX font. The main module uses these bitmaps during a Full display to draw characters a font at a time. Each time a PXL file is opened, the message '`Drawing characters from ...`' will appear.

Because *DVIReader* creates a separate character list for each font used on a page, there never needs to be more than one PXL file open at any given time.

ScreenIO All low-level terminal i/o is handled by the routines defined in this module.

Performance

A number of methods are used to make DVItoVDU an efficient program:

- Output buffering reduces the number of calls to the standard VMS terminal output routine.
- The DVI file and associated PXL files are mapped into virtual memory for fast random access.
- When a glyph is vertically scaled down during a Full display, overlapping rows in its bitmap are first ORed together to reduce the number of *ShowRectangle* calls needed to build up the glyph.
- Since the reference points of most characters in a line will have the same vertical coordinate, some *ShowChar* implementations reduce the number of output bytes needed to update the screen position from one character to the next by remembering the last vertical coordinate. (The sequence of *ShowChar* calls for each font is determined by the way *DVIReader* builds a character list.)
- Specific *ShowChar* routines map a given TEX character into a similar-looking ASCII character using *TeXtoASCII*, a look-up table imported from *VDUInterface*. (Since all TEX characters are assumed to come from text fonts, those from non-text fonts will appear incorrect.)

The following table shows the times taken to display an entire DVI page on different VDUs:

	Terse Display		Full Display	
	compute	elapsed	compute	elapsed
VT220	0.9	10	7.3	56
VT640	1.2	15	13.9	145
AED512	?	23	?	159
GIGI	1.5	34	14.6	2488
VIS500	1.6	52	18.2	193

all times are in seconds

The figures for the VISUAL 500 were obtained while running the program on a VAX-11/780 with 20 interactive users. All the other VDUs were on a VAX-11/785 with 15 to 20 users. Each terminal was operating at 9600 baud.

The page used in the above benchmark contained some 2,500 characters from 16 fonts. The window size was chosen by typing '`H16V12`' in each session. Substantial improvement in performance is unlikely for any of the current batch of VDUs. The large discrepancies between computation and elapsed times, particularly in a Full display, are mostly due to the huge number of bytes that must be sent to a VDU to update the window region.

The times shown in the above table do not include page interpretation. Most pages can be translated in about a second of elapsed time, depending on how busy your VAX is and how many fonts are seen for the very first time. For example, the first 50 pages in the preliminary LATEX manual (the December 1983 version) were interpreted by typing '`NNN...`'. This took just over a minute of elapsed time on a VAX-11/785 with 17 users logged in (compute time was 22.3 seconds). Some 35 fonts were loaded. When the same 50 pages were processed again (by typing '`1NN...`'), the elapsed time dropped to 30 seconds and the compute time to 15.9 seconds.

Conclusion

The current version of DVItoVDU should remain quite stable, particularly from the user's viewpoint. (I am open to complaints or suggestions though!) Any changes are more likely to occur behind the screens, so to speak. One change already in sight is the reading of font data from GF files instead of PXL files.

Other VAX/VMS sites with Modula-2 may be sufficiently motivated to get the program running on new types of VDUs. An implementation on a state-of-the-art, bit-mapped graphics terminal would be something to behold. Note that a new VDU can be added without having to make any changes to the

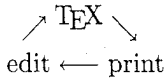
main module. The *DVItoVDU System Guide* lists the necessary steps.

It should be pointed out that DVItoVDU is currently targeted towards relatively primitive terminals. Significant changes to both the display logic and page data structures would probably be needed to take full advantage of the latest graphic workstations. For example, a large amount of bit-mapped memory could store an entire rasterized page and allow much more sophisticated updating of the window region. Pan and zoom operations would also be much easier using a mouse or thumbwheel cursor controls.

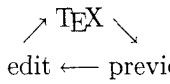
A few hardy souls may even like to translate DVItoVDU into another language or transport it to another operating system. Note, however, that Modula-2 does not specify any i/o facilities as part of its language definition. While such facilities are usually provided as library functions, there is currently no widely accepted standard library. Until this happens, Modula-2 programs will not be very portable.

To help make any conversion easier, all system-dependent code can be quickly located by searching for the string "`SYSDEP`" in the various source files. Terminal i/o is isolated in the *ScreenIO* module and most file operations are confined to the *DVIReader* and *PXLReader* modules. *DCLInterface* is also highly VAX/VMS-dependent. I can only encourage such attempts by stressing the advantages of a TEX page previewer:

- Paper usage is reduced.
- Document preparation time is reduced. The typical proofing cycle of



can be replaced by



- Experimentation is encouraged. The results of changing a global formatting parameter or altering a macro definition can be quickly seen and evaluated.