TEX2page

Dorai Sitaram

Making books is a skilled trade,
like making clocks.
— Jean de la Bruyère (†1696)

TeX2page makes Web pages from TeX [14] manuscripts. It reads an input document that is marked up in plain TeX or LaTeX [16] and produces an output document with the functionally equivalent HTML markup. TeX2page uses the same input file syntax, calling conventions, and error-recovery mechanisms as TeX. Thus, TeX2page demands no additional expertise of a user already familiar with TeX. TeX2page runs on modern Schemes and Common Lisp.

There are several advantages to keeping the document source in TeX and leaving the task of converting to HTML to TeX2page: There is no need to write and maintain two separate documents, one for paper and the other for the screen. Indeed, there is no need to learn a *new* input format, as TeX2page reuses a format already in wide and stable use for printed documents [6, 24, 4]. Creating TeX source requires no special-purpose software; any text editor will do. Furthermore, powerful and reliable tools such as BibTeX [20], MakeIndex [15], and MetaPost [10] have developed around TeX, and their benefits can be enjoyed by TeX2page too.

Finally, TeX, unlike HTML, is a *programming* language, which lets the composer of the document exercise a fine control over its structure and presentation. A converter such as TeX2page that can convert TeX macro definitions in addition to basic TeX markup enables the format converted to to also benefit from TeX's extensibility. For the cases where TeX2page's implementation of the TeX macro system is not manipulable enough, the document writer can use the TeX2page *extension language*, which is full Scheme augmented with all the TeX2page procedure definitions.

The rest of this manual is organized as follows:

I Running TeX2page

TeX2page is invoked in much the same way as plain TeX or LaTeX.* For instance, given a plain-TeX document file with the relative or full pathname *wherever-it-is/jobname*.*ext*, where *jobname* is the *basename* of the file and .*ext* is its extension, you type

* Hereafter, we will use *TeX* to mean any format of TeX, and *plain TeX* when we specifically mean the "plain" format.

`tex` *wherever-it-is/jobname.ext*

at the operating-system command line.† You do not need to mention the extension *.ext* if it is `.tex`. This creates the output DVI file, *jobname*`.dvi`, in the working directory.‡

TeX2page is called analogously. To create the HTML version of the same file *wherever-it-is/jobname.ext*, type

`tex2page` *wherever-it-is/jobname.ext*

Again, the *.ext* is optional if it is `.tex`. This creates *jobname*`.html` in the working directory.

To try this out, copy into your working directory the example file `story.tex` provided in all TeX distributions. Call TeX2page on it:

```
tex2page story
```

TeX2page will get cracking on `story.tex`, providing the following commentary, or *log*, on your console:

```
This is TeX2page, Version 20070609 (MzScheme 372, unix)
(story.tex)
! Missing \end inserted.
[0]
Output written on story.html (1 page).
```

TeX2page is now done, and the result of its labors is the HTML file `story.html`.

The *log file* `story.hlog` contains a copy of the above log, and is useful if you didn't or couldn't keep track of the console (perhaps because the log was too long). The log says that `story.tex` lacked a document-ending command such as `\end` (or `\bye`) and that TeX2page assumed one anyway. Also, only *one* HTML page was created, and its name is `story.html`. TeX2page could in some cases produce auxiliary HTML pages in addition to the main HTML page *jobname*`.html` (especially for larger documents). The auxiliary HTML pages are reachable from *jobname*`.html` by navigation links (p. 8). As each auxiliary HTML page is completed, the log will show the bracketed numbers `[1]`, `[2]`, etc. The `[0]` in this log refers to the only HTML file created, viz., `story.html`.

All this is of course almost exactly analogous to the way you type `tex story` to get `story.dvi` from `story.tex`, with the log going into `story.log`.

```
This is TeX, Version 3.14159 (Web2C 7.3.1) (format=tex 2002.10.21)  16 NOV
2002 18:29
**story
(story.tex [1])
*\end
Output written on story.dvi (1 page, 668 bytes).
```

The only real difference is that TeX will not add the missing `\end` on its own, but instead waits for the user to supply it explicitly from the console.§ Note that the bracketed numbers now refer to the pages as numbered in the printed document.

---

† The executable `tex` expects its input file to be marked up in plain TeX. For a LaTeX document, the executable to use is `latex`.

‡ Nowadays a PDF (rather than DVI) output is often preferred, for which the executables are `pdftex` (for plain TeX) and `pdflatex` (for LaTeX). There are two further executables, `pdfetex` and `pdfelatex`, which also produce PDF but which support a slightly larger core markup. All these executables are included in a typical modern TeX distribution. In the rest of this manual, whenever we refer to the output DVI file, the reader using the PDF versions of TeX should read PDF for DVI.

§ The file `story.tex` lacks an `\end` only to demonstrate some interactive capabilities of TeX, which are not relevant for TeX2page.

Thus, from one TeX source file, you can get both a printable `.dvi` and a browsable `.html` document, using the same calling conventions.

When TeX encounters a filename *f*, it searches for it in a standard list of directories, which can be modified by the user via the environment variable `TEXINPUTS`. The filename *f*`.tex` is tried before *f* itself is tried. In most modern TeXs, the search is performed using the `kpathsea` library.

By default, TeX2page will look for files using the same `kpathsea` mechanism as TeX. However, it is possible to supply a different list of search directories via the environment variable `TIIPINPUTS`. It may be useful to have files in `TIIPINPUTS` shadow files from `TEXINPUTS`, because the latter are not really HTML-specific, and can thus be unsuitable for HTML-minded parsing by TeX2page.

In TeXs without the `kpathsea` library, `TIIPINPUTS` is the only way to get TeX2page to automatically access files outside the working directory. Note that `TIIPINPUTS` should be a simple list of directory names, colon-separated in Unix and semicolon-separated in Windows. It cannot use the enhanced syntax (viz., `*` and `//`) that is typically permitted for `TEXINPUTS`.

Error recovery in TeX2page is also exactly analogous to TeX, but we will postpone that discussion to p. 27.

Non-file arguments

Like most recent versions of TeX, TeX2page also supports the standard self-identification arguments `--help` and `--version`. These arguments elicit help only if there isn't an input file (e.g., `--help.tex`) that could match them.

TeX2page called without an argument displays a help message and exits. Unlike TeX, TeX2page will not try to conjure up an input document based purely on console chitchat with an increasingly befuddled user.

In all these cases, the help displayed on the console is also saved in the specially named log file `texput.hlog`.

Calling TeX2page from Scheme

You may load the library `tex2page.ss` into Scheme and call the *procedure* `tex2page` with the name of the TeX file as argument:

```
(require (lib "tex2page.ss" "tex2page"))

(tex2page filename)
```

You can call the procedure `tex2page` several times from the same Scheme session, on the same file or on different files.

Specifying a target directory

By default, TeX2page generates the output HTML files and other auxiliary files (p. 28) in the current working directory. You can tell TeX2page to place its output and auxiliary files in a different directory and thus avoid cluttering up your working directory.

The files used for specifying the target directory are: *jobname*`.hdir` in the working directory, `.tex2page.hdir` in the working directory, and `.tex2page.hdir` in the user's `HOME` directory. The first line of the first of these files that exists is taken to be the name of the target directory. If none of the files exists, the current working directory is the target directory.

For example, if `story.hdir` contains the filename `story` as its first line, the HTML and aux files are created in a subdirectory `story` of the current directory.

The filename may contain the TeX control sequence `\jobname`, which expands to the basename of the TeX document. To always use an auxiliary subdirectory with the same

name as the basename of the TeX document, have `~/.tex2page.hdir` contain the line "`\jobname`" (without quotes).

II TeX commands

A TeX document is a text file. Most of the text represents the content of the document, but a few characters are used specially to embed *markup commands* within the text. When the program TeX is called on a TeX document, it uses the markup commands in the document to create an appropriately typeset version of the document in a *DVI* file, which can then be printed. The TeX program, which recognizes a list of primitive commands, is invariably called with a *format*, which is essentially a preloaded set of definitions of some additional commands. The two most popular formats are plain TeX [14] and LaTeX [16].

TeX2page understands many of the commands of plain TeX and LaTeX.* It uses this understanding to convert a TeX document to its HTML version, much the same way that TeX converts the same document into its DVI version. TeX2page also recognizes some commonly used commands that are not part of the formats but are loaded from LaTeX packages (e.g., `color.sty`, `epsfig.sty`, `graphicx.sty`, `path.sty`, `verbatim.sty`) or from external macro files (e.g., `btxmac.tex`, `eplain.tex` [1], `epsf.tex`).

While TeX2page recognizes many commonly used commands, there are plenty of commands in both the format proper and the vast arena of macro files and packages that TeX2page does not know. If in math mode (assuming math is translated as text and not image (p. 14)), TeX2page simply types the command's name without the leading escape character. This is sometimes a good choice, as in the following text (where `\sin` and `\cos` are not explicitly recognized by TeX2page):

```
$$\sin^2 x + \cos^2 x = 1$$
```

This becomes

$$\sin^2 x + \cos^2 x = 1$$

which is clear enough.

If in non-math mode, TeX2page simply ignores commands it doesn't understand. This is also usually a good thing, as commands like `\leavevmode`, `\/`, and `\-` are best translated into HTML as nothing at all.

TeX2page ignores calls to include external LaTeX packages: These are files with extension `.sty` and are loaded with `\usepackage` or `\input`. If the commands offered by these packages are not already recognized by TeX2page, they will be ignored too, and often this is not a problem. E.g., you can use a package for generating double columns — while this is a great paper-saver for your printed copy, it is generally not important for the HTML version and so is no loss if ignored by TeX2page.

* TeX2page processes both plain TeX and LaTeX commands, without the need for a format file parameter. It can even process documents written in a mix of plain TeX and LaTeX. This is not an uncommon scenario, with LaTeX users frequently using plain TeX commands, and plain TeX users frequently implementing their own version of sectioning and other commands using the LaTeX names. In the few cases where the same command name (e.g., `\footnote`) is used in both formats but with differing behavior, TeX2page will choose the correct behavior based on which format it thinks the overall document is in. The plain TeX and LaTeX document structures are sufficiently different (as human readers can readily testify by reading just a few opening lines) to allow this disambiguation.

Defining TeX-only and HTML-only text regions

TeX2page will attempt gamely to process any TeX definitions that you use in your document or in external macro files without the extension `.sty`, but it may be a good idea to have them explicitly ignored, if these macros that are print-specific, or if having TeX2page try to parse them will cause error. A way to have TeX2page ignore such fragments in your document is to use TeX conditionals, and indeed to exploit the fact that TeX2page does not know certain TeX commands such as `\shipout`:

```
\ifx\shipout\UnDeFiNeD
  ... for HTML only ...
\else
  ... for DVI only ...
\fi
```

For ordinary text, this is good enough, but if the HTML portion contains calls to raw HTML (see below) or raw Scheme (see p. 25), it is advisable to wrap it inside an environment called `\htmlonly`:

```
\ifx\shipout\UnDeFiNeD
  \htmlonly
    ... for HTML only ...
  \endhtmlonly
\else
  ... for DVI only ...
\fi
```

Failing to use `\htmlonly` does not affect TeX2page's operation, but may cause problems when TeX is called on the same document. Incidentally, the fact that `\htmlonly` and `\endhtmlonly` have no definition in TeX is quite all right, as TeX will only see the else branch of the `\ifx`.

Now, let's say you want to have your document load the macro file `manmac.tex`, but in a way that TeX2page will ignore it. Use:

```
\ifx\shipout\UnDeFiNeD
\else
  \input manmac
\fi
```

However, there will also be many commands that you do not want ignored in the HTML. In such cases, while you may not be able to use the print-specific definition, you should nevertheless furnish a definition that TeX2page can handle. For instance, although it may be acceptable to ignore the print-specific macros of `manmac`, the `\bull` macro defined in `manmac` should be translated by TeX2page. The following is a possible definition for TeX2page:

```
\def\bull{{\bf *}}
```

Of course, we want this definition to be seen only by TeX2page, as we don't want to override the original, more sophisticated `\vrule`-based definition as seen by TeX. We therefore make our definition HTML-only using a conditional:

```
\ifx\shipout\UnDeFiNeD % HTML only
  \def\bull{{\bf *}}
\fi
```

Note that the HTML-only text continues to use TeX syntax. To specify some of this text as *raw* HTML, enclose it in `\rawhtml ... \endrawhtml`. With `\rawhtml`, we can spruce up the HTML-only definition of `\bull`:

```
\ifx\shipout\UnDeFiNeD % HTML only
  \htmlonly
  \def\bull{{\bf
  \rawhtml<span style="color: #990000">&spades;</span>\endrawhtml
  }}
  \endhtmlonly
\fi
```

One can put HTML-only definitions in separate files that are loaded just like regular TeX macro files. Indeed, one such external file, `texi2p.tex`, is used by TeX2page to process Texinfo documents. Texinfo [7] is another TeX format, and files using this format `\input texinfo` as the first thing they do. TeX2page takes that as a cue to load `texi2p.tex`, which provides TeX2page-suitable definitions for the Texinfo commands. `texi2p.tex` is included in the TeX2page distribution.

Paper and screen

One could use HTML-only and DVI-only regions to cordon off any text at all, not just macro definitions. E.g.,

```
The paper book is
\ifx\shipout\UnDeFiNeD % HTML only
an antiquated
\else % DVI only
a time-tested
\fi
technology.
```

Use of these directives regions may seem to miss the point of TeX2page. `\ifx\shipout\UnDeFiNeD` violates the principle of avoiding writing *two* texts, one for HTML, the other for DVI. `\rawhtml` violates the principle of avoiding writing raw HTML at all. `\rawhtml` in particular is dangerous because it voids the guarantee that the output pages will be valid HTML. Nevertheless, these directives are often useful, especially when the text will profit from exploiting the presentational differences between HTML and DVI.

The .t2p file

Before processing a TeX document, TeX2page will automatically load a file with the same basename as the TeX main file but with extension `.t2p`, *if* this file exists. This is a good place to put HTML-specific definitions for the document without making changes in the document itself.

`.t2p` files are especially valuable when HTMLizing legacy or third-party documents without compromising their authenticity, integrity, and timestamp.

Note that the definitions in the `.t2p` file are processed *before* the main file. But it often makes sense to activate these definitions sometime later. E.g., activating the `.t2p` definitions *after* the preamble in a LaTeX document allows you to redefine the preamble macros in a manner that is appropriate for HTML. Here is a technique for accomplishing this:

```
\let\PRIMdocument\document

\def\document{
  ... HTML-specific definitions ...
  \PRIMdocument}
```

This code, which goes in the `.t2p` file, redefines the `\document` command to include a hook that loads some *HTML-specific definitions*. Since the `\document` command is called right

after the preamble, the definitions introduced by the hook will shadow the preamble macros, as intended.

Sample `.t2p` files may be found in the TeX2page distribution.

III TeX2page commands

The previous chapter explained how a document that is acceptable to TeX may contain several commands that TeX2page does not inherently understand and for which suitable HTML-only definitions must be provided. The reverse problem also obtains: A document that is processed successfully by TeX2page may contain commands that plain TeX and LaTeX may not recognize.

As we have seen, TeX2page recognizes a mix of plain-TeX and LaTeX commands, and commands from external macro files besides. Thus, a plain TeX document could very well use LaTeX commands and pass through TeX2page without hitch. When running plain TeX on the document, of course, the LaTeX commands will fail, unless TeX-only definitions for the missing commands are provided. Examples of common LaTeX commands recognized by TeX2page include the commands for title, chapters and sections, footnotes, labels for cross-referencing, table of contents, verbatim text, itemizations, enumerations, index, and bibliography.

It is also possible that the document uses commands from external macro files without actually loading said files. TeX2page will be indulgent, but plain TeX and LaTeX won't. In this case, simply loading the macro files should be enough. Thus, a plain TeX document using the `\cite` command should load `btxmac.tex`, and a LaTeX document that uses the `\color` and `\definecolor` commands should load the package `color.sty`.

A plain TeX document using macros provided by a LaTeX package may also be able to load the package in some cases, with the aid of the `miniltx.tex` macro file. To load `color.sty`, for instance:

```
\expandafter\def\csname Gin@driver\endcsname{pdftex.def}
% The above may not be needed in newer TeX distributions,
% which will load the appropriate driver automatically.
% Replace pdftex.def with dvips.def if output print format
% is PostScript rather than PDF.

\input miniltx
\input color.sty
\resetatcatcode
```

Although the above is only meant for TeX, it is not necessary (but not incorrect either) to wrap it inside `\ifx\shipout\UnDeFiNeD\else ... \fi`, as TeX2page knows enough to skip such loads.

Other useful LaTeX packages that can be loaded into plain TeX using `miniltx` are `graphicx.sty` and `url.sty`. (Unfortunately, one has to say `\expandafter\def\expandafter\+\expandafter{\` before `\input`ting `url.sty`.)

It is possible to `\input` several `.sty` files between the calls to `\input miniltx` and `\resetatcatcode`. However, `miniltx` is a bit of a compromise, and it causes each `.sty` file to re-evaluate the supposedly per-document commands in the driver file (e.g., `pdftex.def`), which can cause infinite loops. This is avoided by preceding the loading of the second and subsequent `.sty` files with `\let\color\@ldc@l@r`. E.g.,

```
\input miniltx
\input color.sty
  \let\color\@ldc@l@r
\input graphicx.sty
```

```
  \let\color\@ldc@l@r
  \expandafter\def\expandafter\+\expandafter{\+}
\input url.sty
\resetatcatcode
```

The LaTeX package `path.sty` can be loaded into plain TeX directly, without `miniltx`.

tex2page.tex and tex2page.sty
A collection of workable TeX definitions for TeX2page commands is provided in the macro file `tex2page.tex` in the TeX2page distribution. These allow your document to be processed by TeX, even if they do not (and sometimes should not) have the same effect in the DVI output as they do in the HTML output. It can be included in your TeX document as

```
\input tex2page
```

The file is also available under the LaTeX-friendlier name `tex2page.sty`, and can be loaded into LaTeX with:

```
\usepackage{tex2page} % if document is in LaTeX
```

The macros in `tex2page.tex` are just sample definitions of the TeX2page commands missing in TeX. You can either choose not to use these commands or to override their definitions with your own, better, definitions.

Note that TeX2page itself does not *need* the file `tex2page.tex`. Rather, plain TeX and LaTeX need the `tex2page.tex` macros in order to process files written using the extra commands supported by TeX2page. If your document does not use these extra commands, then you can do without `tex2page.tex`.

IV TeX commands with a difference

Most TeX commands, whether from the format or from macro files, are translated into an obvious HTML equivalent, and therefore need no further description than what is available in the TeX and LaTeX manuals. However, some commands are treated specially. We now turn to these commands.

Multiple pages
Unlike TeX, TeX2page does not automatically split the document into pages at regular vertical lengths. TeX2page will start a new HTML page only at `\chapter` commands and at explicit page break commands such as `\eject` or LaTeX's `\clearpage`. (It is advisable to place a `\vfill` before `\eject` so the DVI document doesn't cause the pre-`\eject` text to increase its interparagraph space unsightlily in order to fill the physical page.)

Conditionals may be used to specify page breaks for only the DVI output, or for only the HTML output.

By default, TeX2page will generate a *navigation bar* at the top and at the bottom of each HTML page, with links to the *first*, *previous*, and *next* page. If the document has a table of contents or an index, links to the pages containing these elements are also included in the navigation bar. The nav bar is customizable if you set the `\TZPtexlayout` flag (p. 15).

Table of contents
TeX2page recognizes LaTeX's `\tableofcontents` and Eplain's `\readtocfile`, both of which list the table of contents. Each section name in the ToC links to the page on which the section occurs. In LaTeX, the ToC lists the numbered section names in the document, upto the depth specified by the count `\tocdepth`.

In formats other than LaTeX, the user would have to define section commands that explicitly wrote to the ToC. A helper macro for this is `\writenumberedtocline`, whose three arguments are the section's depth, number, and heading, all of which can be empty (`{}`). Your TeX format may have its own macro for writing lines into the ToC (`\writenumberedtocline` is the Eplain name for this macro). TeX2page also understands LaTeX's `\addcontentsline`. For other formats, if you intend to use their explicit ToC addition macro, you will need to furnish an HTML-only definition for it in terms of `\writenumberedtocline`.

Footnotes

Both unnumbered and numbered footnotes — plain TeX's `\footnote` and `\vfootnote`, LaTeX's `\footnote`, Eplain's `\numberedfootnote` – are recognized. They are translated as in DVI (modulo what constitutes a page), but additionally, the footnote mark in the text body is a link to the footnote mark in the footnote text, and vice versa. Here is an example footnote.*

Since Plain TeX does not provide an automatically numbering footnote macro, users can define their own as follows:

```
\newcount\footnotenumber

\def\numberedfootnote{%
  \global\advance\footnotenumber by 1
  \footnote{$^{\the\footnotenumber}$}}
```

This definition could be made TeX-only, since TeX2page already recognizes `\numberedfootnote`. However, the TeX programming in this definition is recognized by TeX2page, so it does not matter if it overrides TeX2page's internal definition.

Indeed, one could define a more complicated macro such as the following `\sfootnote`, which produces symbolic rather than numeric footnote marks, cycling through a set of nine symbols:

```
\def\fnsymbol#1{%
  % #1 is between 1 and 9 inclusive
  \ifcase#1\or
    *\or\dag\or\ddag\or\S\or\P\or
    $\Vert$\or**\or\dag\dag\or\ddag\ddag
  \fi}

\def\sfootnote{%
  \global\advance\footnotenumber by 1
  \ifnum\footnotenumber>9 \global\footnotenumber=1 \fi
  \footnote{\fnsymbol{\the\footnotenumber}}}
```

TeX2page produces the expected output for all the footnote macros, including the user-defined ones.

Bibliographies

TeX2page can use the external program BibTeX [16, 20] to generate bibliographies from bibliographic database (`.bib`) files. The bibliography commands — `\cite` and the rest — are included in LaTeX, but for Plain TeX must be explicitly loaded via the macro file `btxmac.tex`. In HTML, the citation created by `\cite` is a link to the corresponding entry in the bibliography.

* Footnotes are separated from the body of the page by a horizontal rule.

Bibliographies can also be manually embedded in the document via the `thebibliography` environment, without the need for the BibTeX program. For more details, see the LaTeX manual [16, sec 4.3.2, p 71].

Index generation

TeX2page can use the external program MakeIndex [3, 15] to generate indices. TeX2page's index-generation feature follows the same conventions as traditionally used with TeX and its derived packages [16, sec. 4.5 & appendix A].

The sorted index is inserted in the document with a command such LaTeX's `\printindex` or the more general `\inputindex`. The latter does not include a section header, so you can print your index your own way, e.g., with a different section type and title, and with some introductory prose.

For HTML, the page numbers listed for an index entry are of course the HTML page numbers, and are furthermore links to them. Two things need be noted:

1. The link in the index goes directly to the spot where the corresponding `\index` command was called. This is convenient, especially as the target HTML page could be arbitrarily long, and it may not be as easy to hunt for the occurrence of the indexed item as on a paper page.

2. An index entry could link to the same HTML page several times. In the print index, a page would be mentioned only once per entry, but since an HTML page could be equivalent to many contiguous paper pages, it makes little sense to collapse into one all these references to (different locations in) the same page. The TeX2page index therefore repeats the page number with different links, and adds a roman number to the second and subsequent occurrences to distinguish them visually.

Internal cross-references

LaTeX's `\label` and `\ref` produce internal cross-references in the HTML. The `\label` *anchors* a location in the document, and a `\ref` anywhere else in the document links to it. The link text used by `\ref` is the number of closest section (or footnote or other document fragment) that surrounds the `\label`. The LaTeX `\pageref` takes a label and produces the number of the page where the label was defined. In HTML, this is the HTML page number, and it is a link.

For formats that do not assume the presence of numbered sections as intensely as LaTeX, TeX2page also recognizes the `\xrtag` command as a generator of anchors. Where `\label` uses the nearest section number, `\xrtag` requires an explicit link text as its second argument.

`\xrtag{`*alabel*`}{`*alabelvalue*`}`

A reference to this tag, i.e., `\ref{`*alabel*`}`, will typeset as *alabelvalue*, which will link to the location identified by the `\xrtag`.†

Color

TeX2page recognizes the macros `\color` and `\definecolor` [2]. These are provided by the LaTeX package `color.sty`, which can also be loaded into plain TeX using `miniltx`. E.g.,

```
{\color[gray]{.17} light gray},
{\color[rgb]{.69, .19, .38} maroon},
{\color[cmyk]{0, .89, .94, .28} brick red},
{\color[RGB]{220, 20, 60} crimson},
```

† Eplain offers a version of `\xrtag` which it calls `\definexref`. TeX2page recognizes Eplain's `\refn` and `\xrefn` as synonyms for the `\ref` described above. Eplain also has a `\ref` macro, but it behaves differently than a LaTeX-like `\ref`.

```
and {\color{blue} blue}.
```

produces:

light gray, maroon, brick red, crimson, and blue.

`\definecolor` defines new color names, e.g.,

```
\definecolor{BrickRed}{cmyk}{0, .89, .94, .28}
```

Most color-capable browsers support the very large list of named colors in the X11 file `rgb.txt`. In order for your printed document to have access to these same color names, definitions for them are provided in the TeX macro file `x11rgb.tex`, included in the TeX2page distribution.

`cmyk` definitions for the 68 standard DVIPS color names are available in the standard LaTeX macro file `dvipsnam.def`. These are *not* predefined by browsers, so you will need to load `dvipsnam.def` explicitly if your HTML document is to benefit from them.

V Referring to external documents

The command

`\urlhd{`*URL*`}{`*HTML text*`}{`*DVI text*`}`

lets you link to arbitrary URLs, not just to labels within your document (although it can do that too). In the HTML output, a hyperlink to '*URL*' is created, with the link text being '*HTML text*'. In the DVI output, the part '*DVI text*' is output. Example:

```
For more details, consult
\urlhd{http://www.ithaka.org/odyssey.html}{the Odyssey}{the
{\it Odyssey\/} document in the Ithaka repository}.
```

In the DVI output, this becomes

For more details, consult the *Odyssey* document in the Ithaka repository.

In the HTML output, it would be

For more details, consult *the Odyssey*.

where "*the Odyssey*" is an HTML link to the site `http://www.ithaka.org/odyssey.html`.

`\urlhd` is named to be mnemonic for its argument sequence, viz., the *URL*, followed by the *H*tml text, followed by the *D*vi text.

Note that you can use `\urlhd` for cross-referencing within the document also. The URL in such cases will be a label as specified by a `\label` or a `\xrtag` command, but should add a '`#`' prefix. E.g.,

```
See \urlhd{#hairy}{below}{section~\ref{hairy}}
for further details.
```

where the further details are described in a section annotated with `\label{hairy}`. Assume this section is numbered 21. Then, the reference typesets as "See section 21 ..." in the DVI output and "See *below* ..." in the HTML output (with *below* being a link). In contrast, if we had written

```
See section~\ref{hairy} for further details.
```

we would have had "See section 21 ..." in both DVI and HTML. `\urlhd` is thus more flexible than `\ref`.

Because of page breaks in the HTML output (p. 8), it is possible that a label's definition and the references to it do not ultimately sit on the same *physical* HTML page. Nevertheless, your TeX source can use `#tag`-style URLs to refer to anchors anywhere within it. TeX2page will automatically convert a `#tag`-style URL to its correct fully qualified equivalent.

Common hyperlink abbreviations

In some cases, `\urlhd`'s second and third arguments may be mere repetitions of a preceding argument. For such cases, TeX2page recognizes some convenient abbreviations, viz., `\urlh`, `\urlp`, and `\url`. (A version of `\urlh` is provided under the name `\href` by the LaTeX package `hyperref.sty`, and `\url` is provided by the LaTeX package `url.sty`. `\url.sty` is loadable in plain TeX with `miniltx`.)

`\urlh` takes *two* arguments. The first argument is the URL, and the second is the descriptive text that is used in *both* the HTML and the DVI outputs. For example:

```
TeX is available at
\urlh{http://www.tug.org}{the TUG website}.
```

produces

TeX is available at the TUG website.

In the HTML output, "the TUG website" is a hyperlink to `http://www.tug.org`.

An optional `\\` may be used inside `\urlh`'s second argument. The text before the `\\` is used in both the HTML and the DVI outputs. The text after the `\\` is used only in the DVI output. This helps you to specify extra information for the DVI output, which may be necessary because the DVI output lacks the information implicit in the hyperlink. For example:

```
TeX is available at
\urlh{http://www.tug.org}{TUG\\ (\path!tug.org!)}.
```

produces, in the DVI output.

TeX is available at TUG (`tug.org`).

The HTML output will not mention the parenthesized domain name, since the word "TUG" already hyperlinks to it.

`\\` is useful for internal cross-references too. For example (assuming the label `callcc` refers to section 2.3):

```
More complicated forms of program control are possible
using \urlh{#callcc}{{\tt callcc}\\ (section~\ref{callcc})}.
```

produces

More complicated forms of program control are possible using `callcc` (section 2.3).

in the DVI output. In the HTML output, the parenthesized section reference will be dropped as redundant, as the word "`callcc`" hyperlinks to the relevant section.

An optional `\1` may be used after `\\` to refer to `\urlh`'s first argument, i.e., the URL. Example:

```
TeX is available at
\urlh{http://www.tug.org}{TUG\\ (\1)}.
```

produces

TeX is available at TUG (`http://www.tug.org`).

in the DVI output. In the HTML output, the parenthesized URL is dropped as redundant, as the word "TUG" hyperlinks to it.

Finally, the combination of `\xrtag` and `\urlh` is useful for inserting internal cross-references in the HTML document without affecting the print document. For example, the following text

```
\xrtag{ex1}{ignore}
\urlh{#ans1}{\bf Exercise 1.} Statement of a problem ...

... lots of intervening stuff ...
```

```
\xrtag{ans1}{ignore}
\urlh{#ex1}{\bf Answer 1.} Answer to exercise 1 ...
```

prints as

Exercise 1. Statement of a problem ...
... lots of intervening stuff ...
Answer 1. Answer to exercise 1 ...

in both the DVI and the HTML. However, in the HTML, the proclamations "**Exercise 1.**" and "**Answer 1.**" are also helpful hyperlinks to each other.

`\url` takes just one argument, the URL. For the descriptive text, both the HTML and the DVI outputs simply use the URL itself. Example:

```
TeX is available at \url{http://www.tug.org}.
```

`\urlp` takes two arguments. In the HTML output, the first argument is the link text and the second is the URL. In the DVI output, the first argument is typeset followed by a space followed by the URL in parentheses. `\urlp{`*text*`}{`*URL*`}` abbreviates `\urlh{`*URL*`}{`*text*`\\ (\path+`*URL*`+)}`.

VI Flags

TeX2page has a handful of flags that you can set to govern the HTML output. These flags are TeX control sequences such as `\TZPcolophontimestamp`, etc., and are set using `\let` or `\def` or even `\gdef`, `\edef`, or `\xdef`. (If using one of the `\def`s, only the first character in the `\def`-body is significant.)

All these TeX2page flags are named `\TZP...`, and as long as you avoid using these names for other purposes, setting these flags should not affect the print output.

External title

By default, the "external" title of the HTML document — i.e., the string that appears on the title bar of the browser window — is either the same as the internal title, as specified with the `\title` command, or, if that is not available, the basename of the document source file. If you wish the external title to be something else, you can set the control sequence `\TZPtitle`, e.g.,

```
\def\TZPtitle{The Odyssey}
```

Note that in LaTeX, `\title` merely specifies the title; the command that actually places the title is `\maketitle`. TeX2page does the same for LaTeX documents. For other formats, TeX2page will assume that `\title` should set its argument as a title immediately. For TeX to do likewise, the document must include supply an appropriate TeX-only definition for `\title`, e.g., the one in `tex2page.tex`.

Colophon flags

These flags govern the placement and content of the colophon. By default, TeX2page prints a two-line colophon at the bottom of the first page, the first line giving the time of last change of the source document, and the second line identifying TeX2page and linking to its website.

To put the colophon on the last rather than the first page,

```
\let\TZPcolophonlastpage=1
```

If this request is to have meaning, it should be made in the document before the text for the second HTML page starts. Otherwise, the default first-page placement of the colophon will have already taken effect.

To avoid mentioning the timestamp of the document in the colophon,

```
\let\TZPcolophontimestamp=0
```

If the underlying Scheme is incapable of determining a file's write date, TeX2page will not mention the timestamp, regardless of the `\TZPcolophontimestamp` setting.

To avoid crediting TeX2page in the colophon,

```
\let\TZPcolophoncredit=0
```

To avoid linking to the TeX2page website in the colophon,

```
\let\TZPcolophonweblink=0
```

If `\TZPcolophoncredit` is 0, the colophon won't link to the TeX2page site, regardless of the `\TZPcolophonweblink` setting.

Math image flag

The `\TZPmathimage` flag specifies whether mathematical fragments in the document should be rendered as image or ascii. By default, TeX2page will generate images for displayed math and for "complicated" in-text math (i.e., math embedded in running text). If the in-text math is simple according its judgment, TeX2page will economize by generating its ascii equivalent.

The assignment

```
\let\TZPmathimage=0
```

forces subsequent math in ascii, until the end of the document or until you reset to the flag to 1.

It is a good idea to set this flag to 0 if the mathematical notation in part or all of your document is simple enough to not require complicated notation.

Image conversion flags

These flags specify the conversion tactics used for generating HTML-suitable images from the user's graphics requests. TeX2page invokes a combination of TeX and Dvips [22] to create a PostScript version of the graphic, and then Ghostscript [8] and either (a) the NetPBM library [19], or (b) the ImageMagick library [11], to convert the PS into the HTML-suitable image.

The defaults are: NetPBM to convert, and GIF to convert to. You may set the flags `\TZPimageconverter` and `\TZPimageformat` to change this, e.g,

```
\def\TZPimageconverter{imagemagick}  % use ImageMagick
\def\TZPimageconverter{netpbm}       % use NetPBM (default)

\def\TZPimageformat{png}  % for PNG images
\def\TZPimageformat{jpeg} % for JPEG images
\def\TZPimageformat{gif}  % for GIF images (default)
```

Flag for Scheme code comments

The flag `\TZPslatexcomments` governs whether Scheme comments should inside verbatim Scheme code should be rendered verbatim or as TeX. By default, they are rendered verbatim. To allow TeX commands inside Scheme comment text, do

```
\let\TZPslatexcomments=1
```

This is the style favored by SLaTeX. Note that `\TZPslatexcomments=1` is not an unmixed blessing, as it restricts your Scheme comments to text that is valid TeX.

Flags for page and paragraph layout
By default, TeX2page produces block paragraphs with about about a baseline's worth of vertical space separating paragraphs, and text width expands to fill the browser width, allowing for some margins.

To produce more TeX-like layout, i.e., with no parskip and with some parindent, with the page width not exceeding `\hsize`, with left and top margins that are an inch greater than `\hoffset` and `\voffset` respectively, and with a navigation bar that uses `\headline` and `\footline`, do

```
\let\TZPtexlayout=1
```

The HTML page will be set according to the values of the lengths `\hsize`, `\hoffset`, `\voffset`, `\parskip`, `\parindent`, and the tokens `\headline` and `\footline` as they are at the end of the document.

The command `\folio` inside the tokens `\headline` and `\footline` — and only inside them — produces in HTML not the current page number but rather twin links to the adjacent pages. `\folio` thus lets you create navigation bars.

Note that the plain TeX default (which TeX2page reuses) is to have `\folio` in the `\footline` and nothing in the `\headline`. If you wish to have a navigation bar in the header, you should set `\headline`.

It is not necessary that these values be identical to what TeX sees for the same document, as you can make HTML-only settings such as

```
\ifx\shipout\UnDeFiNeD
  \hsize=36em
  \headline={\folio, ~~ \urlh{#ToC}{ToC}, ~~ \urlh{#Index}{Index}}
  \footline={\the\headline}  % i.e., footline same as headline
\fi
```

where the tags `ToC` and `Index` are set near the ToC and Index respectively.

Paragraphs preceded by `\noindent` will always have no initial indentation, even if `\TZPtexlayout=1` and `\parskip` is non-zero.

Because it is too drastic a change, `\TZPtexlayout=1` will not cause text to be right-justified. To make that happen,

```
\let\TZPraggedright=0
```

Setting `\TZPraggedright` has no effect unless `\TZPtexlayout` is also set.

VII Style

You can use the TeX2page command `\inputcss` to have your HTML output use *style sheets* [27, 18, 21]. E.g.,

```
\ifx\shipout\UnDeFiNeD % HTML only
  \inputcss basic.css
\fi
```

in your TeX source will cause the HTML output to use the presentation advice contained in the style sheet `basic.css`.

In the style sheet, you can have rules for the various HTML elements to change the appearance of your document. E.g., if you do not want your HTML page width to increase beyond a certain point (regardless of how the wide the reader makes their browser window), you could put the following in your style sheet:

```
body {
  max-width: 36em;
}
```

You can get finer control on the look of your document by defining rules for some classes that are peculiar to TeX2page. These special classes are discussed in this manual alongside the commands that they govern (p. 19).

You can have as many `\inputcss`'s in your document as you wish. They will be combined in the sequence in which they appear. It is perhaps necessary to add that style sheets are completely optional.

You can also *embed* style sheet information in the TeX source between the control sequences `\cssblock` and `\endcssblock`. E.g.,

```
\ifx\shipout\UnDeFiNeD % HTML only
  \cssblock
  body {
    max-width: 36em;
  }
  \endcssblock
\fi
```

You can have multiple `\cssblock`s in the document; they are all evaluated in sequence.

TeX2page generates a very basic default style that does little beyond setting some margins. You can augment or override the default style by supplying your own style info via `\cssblock` or by loading stylesheets with `\inputcss`. Some general-purpose style sheets are the *W3C Core Styles* [28].

Making a slideshow

To cause your source to be converted into slideshow-ready HTML pages, use the macro file `t2pslides.tex` by embedding `\input t2pslides` anywhere in the document. `t2pslides.tex` is included in the TeX2page distribution, and uses a version of the MozPoint [23] library to produce the appropriate Javascript and style sheets to convert your sequence of HTML pages into a Web presentation. `t2pslides.tex` has no effect on the DVI version of the document.

HTML pages meant for presentation use larger, bolder fonts, and avoid navigation bars. To bring up the next slide, left-click the mouse anywhere on the screen, or press the space bar, `n`, right or down arrow. To go back to the previous slide, right-click the mouse, or press `p`, left or up arrow. To go immediately to the first (i.e., title) slide, press `t` or `0`. To go immediately to the nth slide after the title slide, type n. If n has two or more digits, they should be pressed fairly rapidly so that they are interpreted together. To blank the screen, press `b` (black) or `w` (white); to unblank, press the same key again.

VIII Verbatim text

TeX2page recognizes a slightly enhanced version of LaTeX's `\verb` command. Recall that LaTeX's `\verb`'s argument is enclosed within a pair of identical characters (not whitespace or `*`), and this argument is printed verbatim. This is useful for typesetting things like fragments of computer code.

The TeX2page version of `\verb` can also use matching braces to enclose its argument, provided the latter does not contain unmatched braces. argument in

The command `\verb` is used for text that should be set verbatim, such as fragments of computer code. `\verb`'s argument is enclosed within a pair of identical characters (that aren't whitespace, `{`, or `*`). For example,

```
A \verb|cons|-cell has two components: a \verb+car+ and
a \verb&cdr&.
```

is converted to

A `cons`-cell has two components: a `car` and a `cdr`.

You could also use matching braces to enclose `\verb`'s argument, provided the latter does not contain unmatched braces. E.g.,

```
The command \verb{\section{Imagining a Conscious Robot}}
typesets \verb{Imagining a Conscious Robot} as a section
title.
```

is converted to

The command `\section{Imagining a Conscious Robot}` typesets `Imagining a Conscious Robot` as a section title.

If `\verb`'s argument commences with a newline, it is set as a display. E.g.,

```
\verb{
(define compose
  (lambda (f g)
    (lambda aa
      (f (apply g aa)))))
}
```

produces

```
(define compose
  (lambda (f g)
    (lambda aa
      (f (apply g aa)))))
```

Note that such displays faithfully typeset all the whitespace of the text, including linebreaks and indentation.

As in LaTeX, if a `*` immediately follows `\verb`, any spaces in `\verb`'s argument text are highlighted as something that is visible. This allows you to easily count spaces or tell if there is trailing space on a line.

```
``\verb*{three   spaces}''
```

produces

"three␣␣␣spaces"

TeX2page also understands LaTeX's `{verbatim}` environment, the `\path` macro from `path.sty`, and Eplain's `\verbatim`.

Commands within verbatim

Often you want to use TeX commands in special spots within verbatim text, especially displayed verbatim material. For this reason, the character '`|`' is allowed as an escape character *if the verbatim text is enclosed within braces*.

As an example, let's say you've defined an `\evalsto` macro to use in cases where you want to say a program expression *evaluates to* a result. A possible definition is:

```
\def\evalsto{::==}
```

You could use `\evalsto` inside a verbatim display as follows:

```
\verb{
(cons 1 2) |evalsto (1 . 2)
}
```

This will produce

```
(cons 1 2) ::== (1 . 2)
```

Some standard commands that can be used inside braced verbatim are: `||` to insert the escape character itself; and `|{` and `|}` to insert the occasional non-matching brace.

Changing the verbatim escape character
You can use the Eplain and TeX2page macro `\verbatimescapechar` to postulate a character other than '`|`' as the verbatim escape. E.g.,

```
\verbatimescapechar\@
```

makes '`@`' the verbatim escape.

Inserting files verbatim
You can insert files verbatim with the command `\verbatiminput` (from LaTeX's `verbatim.sty`) or with Eplain's `\listing`. Usage:

```
\verbatiminput{program.scm}
```

Writing to files
The command `\verbwrite`, used like `\verb`, does not typeset its enclosed text but outputs it verbatim into a text file. The text file has by default the same basename as the document, but with extension `.txt`.

To specify another text file, use `\verbwritefile`. E.g.,

```
\verbwritefile notes-to-myself.txt    % or
\verbwritefile{notes-to-myself.txt}
```

This will cause subsequent calls to `\verbwrite` upto the next `\verbwritefile` or end of document (whichever comes first) to send text into the file `notes-to-myself.txt`. `\verbwritefile` deletes any pre-existing contents of its argument file.

TeX2page also recognizes the TeX command `\write`, which takes two arguments: an *output stream number* and a TeX expression to be output. Recall that TeX allows only the numbers 0–15 for output streams that can be associated with files; numbers outside this range are deemed to represent standard output. However: TeX2page follows modern TeX implementation practice in treating the output stream 18 specially. `\write18{`*oscommand*`}`, instead of writing *oscommand* to standard output, will execute it as an operating-system command! This is not standard TeX behavior, but most modern TeXs enable this feature with a command-line option that is either `--shell-escape` [6] or `--enable-write18` [24].

Verbatim style
The verbatim commands (`\verb`, `\path` and `\verbatiminput`, etc.) introduced above use a style class called `verbatim`. You can affect the appearance of your verbatim text by defining a style for `verbatim` in a style sheet (p. 15). E.g.,

```
.verbatim        {color: rgb(0,49,83)}
```

sets `all verbatim text in Prussian blue`.

Syntax-highlighting of program code
The commands `\scm` and `\scminput` are variants of `\verb` and `\verbatiminput`. They are useful for producing syntax-highlighted Scheme code in the HTML file. E.g.,

```
\scm{
(define factorial
  (lambda (n)
    (if *debug?*
        (printf "Calling factorial ~a\n" n))
    (if (= n 0) 1 ;the base case
        (* n (fact (- n 1))))))
}
```

Seven categories of code text are distinguished: (1) background punctuation; (2) self-evaluating atoms (numbers, booleans, characters, strings); (3) syntactic keywords; (4) builtin variables; (5) global or special variables, viz., identifiers that begin and end with an asterisk; (6) other variables; and (7) comments.

To distinguish between the categories of Scheme code text, TeX2page uses a style class called `scheme` with six subclasses, viz., `selfeval`, `keyword`, `builtin`, `global`, `variable`, and `comment`. You can set the `color` property (and other properties like `font-weight` and `font-style`) of these classes in a style sheet (p. 15). The default settings are:

```
.scheme             {color: #993333} /* background punctuation */
.scheme  .selfeval  {color: #006600}
.scheme  .keyword   {color: #660000; font-weight: bold}
.scheme  .builtin   {color: #660000}
.scheme  .global    {color: #660066}
.scheme  .variable  {color: #000066}
.scheme  .comment   {color: #006666; font-style: oblique}
```

TeX2page initially only recognizes some well-known syntactic keywords, global variables, and self-evaluators. It does not recognize builtins as apart from the general run of variables. Users who want builtins distinguished can use `\scmbuiltin`, e.g.,

```
\scmbuiltin{cons car cdr}
```

Users who do not want to distinguish Common Lisp–style global ("special") variables as a separate category from other variables should give the style class `.global` the same properties as `.variable`, e.g.,

```
.scheme .global {color: #000066}
```

Users can add their own keywords with `\scmkeyword`. E.g.,

```
\scmkeyword{define-class unwind-protect}
```

By default, tokens that don't fall in any of the other categories are set as variables. However, `\scmvariable` can be used to explicitly identify as variables those tokens that are currently treated as non-variables (e.g., keywords or self-evaluators). E.g.,

```
\scmvariable{and 42 +i}
```

Using SLaTeX commands

TeX2page also syntax-highlights Scheme code introduced using the SLaTeX commands, chiefly `\scheme` and `{schemedisplay}`. SLaTeX users know that these commands typeset code in the DVI output using fonts (rather than color) for highlighting. For the HTML, TeX2page will use color.

A minor point is that SLaTeX's commands allow TeX commands inside Scheme comments. This is useful if you want to highlight mentions of Scheme code inside Scheme comments. To get the same effect with TeX2page, use the `\TZPslatexcomments` flag (p. 15).

Documenting your code
You can use TeX2page to do a form of *literate programming*, i.e., combining your documentation with your code. The command `\scmdribble`, which is used like `\scm`, will not only display the enclosed code, but also send it to the external file named by the most recent `\verbwritefile`.

To specify code that should go into the external file but should not be displayed, simply use `\verbwrite` instead of `\scmdribble`.

IX Images

Some portions of your TeX source may be explicitly images, or text that is particularly resistant to conversion to HTML. Examples are JPEGs from digital cameras, encapsulated PostScript inserts, mathematics, and the LaTeX `{picture}` environment. TeX2page embeds these as images in the HTML output.

Mathematics
Math is typically text between `$...$` (in-text math) and `$$...$$` (displayed math). Here are some samples of mathematics with TeX:

```
$$ F = G {m_1 m_2 \over r^2 } $$

$$ \int_0^\infty { t - ib \over t^2 + b^2} e^{iat}\,dt =
e^{ab} E_1(ab), \qquad a, b > 0 $$

$$ A =
\left(
\matrix{ x - \lambda & 1           & 0           \cr
         0           & x - \lambda & 1           \cr
         0           & 0           & x - \lambda \cr}
\right) $$
```

These produce, respectively:

$$F = G\frac{m_1 m_2}{r^2}$$

$$\int_0^\infty \frac{t - ib}{t^2 + b^2} e^{iat}\,dt = e^{ab} E_1(ab), \qquad a, b > 0$$

$$A = \begin{pmatrix} x - \lambda & 1 & 0 \\ 0 & x - \lambda & 1 \\ 0 & 0 & x - \lambda \end{pmatrix}$$

In-text mathematics is also available. E.g.,

```
The Euclidean distance between two points is given by
$\sqrt{ (\Delta x)^2 + (\Delta y)^2 }$.
```

produces

The Euclidean distance between two points is given by $\sqrt{(\Delta x)^2 + (\Delta y)^2}$.

You can control whether your mathematics should be specified as image or ascii with the flag `\TZPmathimage` (p. 14).

Graphics inclusions

Encapsulated PostScript files (EPS) are a convenient and popular way to insert pictures (graphics) into TeX documents. Users create EPS files with their favorite external programs, which can be GUI tools such as Inkscape [12], The Gimp [9], and Xfig [29], or algebraic ones like MetaPost [10]. It is also possible to write a picture's specification in the document, while still relying on an external program to make sense of it. An example is MFpic [17], whose TeX macros transform a picture specification inside the document into an external METAFONT [13] or MetaPost file.

However it is created, an EPS file is typically inserted as a TeX box into a TeX document with calls like

```
\epsfbox{eps-file}
\includegraphics{eps-file}
```

These are commands defined external to plain TeX or LaTeX. Plain TeX documents using `\epsfbox` must load the standard macro file called `epsf.tex`. LaTeX documents using `\epsfbox` can do the same, or they can load the `epsfig.sty` package.

`\includegraphics` is a generic graphics includer that tackles more than EPS files, based on file extension. Thus, if your file has a nonstandard extension, you will have to inform `\includegraphics` of this using directives like the following:

```
\DeclareGraphicsRule{.1}{mps}{*}{}
```

This states that files with extension `.1` are to be treated as EPS files generated by MetaPost.

`\includegraphics` is defined in the LaTeX package `graphicx.sty`, which can also be loaded by plain-TeX documents with the help of `miniltx.tex`, as we saw with `color.sty` on p. 10.*

```
\input miniltx
\input graphicx.sty
\resetatcatcode
```

Here is an example of an EPS insert: The MetaPost file `lambda.mp` is processed by MetaPost generating the PS file `lambda.1`, which we load with

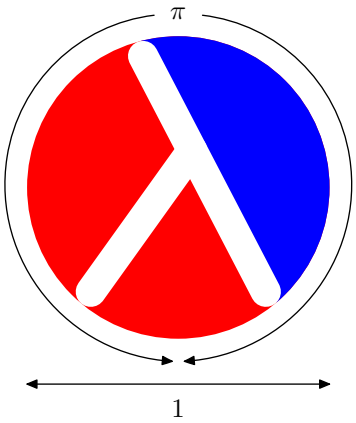
```
\centerline{\epsfbox{lambda.1}}
```

to produce†

---

* It is possible to `\input` several `.sty` files between the calls to `\input miniltx` and `\resetatcatcode`. However, `miniltx` is a bit of a compromise, and it causes each `.sty` file to re-evaluate the supposedly per-document commands in the driver file (e.g., `pdftex.def`), which can cause infinite loops. This is avoided by preceding the loading of the second and subsequent `.sty` files with `\let\color\@ldc@l@r`.

† The file `lambda.mp` was actually written out from this document's source using `\verbwrite` (p. 18), so the file `lambda.1` isn't immediately available. Nevertheless, TeX2page will take care to call MetaPost on the generated `lambda.mp` file, ensuring that the EPS file is available for conversion into an HTML image. In contrast, when getting the DVI version of the document via TeX, it is the user's responsibility to call MetaPost on generated files, and call TeX again. Unfortunately, commands like `\epsfbox` and `\includegraphics`, when they do not find their argument file, will signal error and cause TeX to go into a debug loop, even though the MetaPost file needed for their creation can only be created if TeX successfully finishes processing the source document! To force TeX to finish processing the source file regardless of missing EPS files, you need to run it in `\scrollmode`, or its even more reckless cousins `\nonstopmode` and `\batchmode`. One way to get into these modes is to type `s`, `r`, or `q`, respectively, at the TeX debug prompt. By default, TeX runs in `\errorstopmode`, which is why it stops on the missing-file error.



For `\epsfbox`, you can specify the desired image width and height by assigning to the dimen registers `\epsfxsize` and `\epsfysize` (specifying only one of them will cause the other to change as well, maintaining the image's aspect ratio). TeX2page will respect such sizes, equating one browser pixel to one point (= 1/72.27 inch). Thus,

```
\epsfxsize=1.5in
```

sets the width of an immediately following `\epsfbox`ed image to $1.5 \times 72.27 \approx 108$ pixels. `\epsfxsize` and `\epsfysize` are cleared after each `\epsfbox`.

If you use the PDF versions of TeX (which produce PDF instead of DVI output), you can insert MetaPost-created EPS files with the `\convertMPtoPDF` command:

`\convertMPtoPDF{`*eps-file*`}{1}{1}`

`\convertMPtoPDF` is defined in the macro file `supp-pdf.tex` of the ConTeXt [26] package, which is included in most modern distributions of TeX. Caveat: `\convertMPtoPDF` doesn't work for EPS files that weren't made using MetaPost!

PDF versions of TeX can import common graphics formats such PNG and JPEG: Either use `\includegraphics`, or a primitive call such as

```
\pdfximage height 1.5in {pic.png}\pdfrefximage\pdflastxmimage
```

TeX2page recognizes the scaling information supplied with `\pdfximage` and `\includegraphics`, with one browser pixel equated to one point. Unlike EPS files, PNG and JPEG images are directly supported by HTML, so TeX2page does not need to convert them.

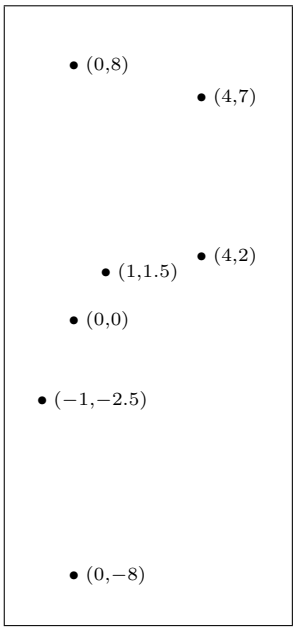
Other image inserts

You may explicitly request any part at all of your TeX document — not just its math or EPS inserts — to be converted into images for your HTML output. The fragment of the document to be converted to image is given as an `\makehtmlimage` argument. Here's an example TeX-based diagram from *The TeXbook* [14, p 389]:

```
\makehtmlimage{
\newdimen\unit
\def\point#1 #2 {\rlap{\kern#1\unit
    \raise#2\unit\hbox{$
      \scriptstyle\bullet\;(#1,#2)$}}}
\unit=\baselineskip
\centerline{\vtop{\hrule
  \hbox{\vrule height10\unit depth9.4\unit \kern2\unit
    \hbox{%
      \point 0 0 % Alioth (Epsilon Ursae Majoris), mag 1.79
      \point 0 8 % Dubhe (Alpha Ursae Majoris),    mag 1.81
      \point 0 -8 % Alkaid (Eta Ursae Majoris),    mag 1.87
```

```
      \point -1 -2.5 % Mizar (Zeta Ursae Majoris), mag 2.26
      \point 4 7 % Merak (Beta Ursae Majoris),     mag 2.37
      \point 4 2 % Phekda (Gamma Ursae Majoris),   mag 2.44
      \point 1 1.5 % Megrez (Delta Ursae Majoris), mag 3.30
      }%    Src: Atlas of the Universe; Astronomy Data Book
    \kern7\unit \vrule}\hrule}}
}
```

This produces the image:



`\makehtmlimage`'s argument is a group containing no unmatched braces.

Image preamble

When converting math, EPS, and other implicit or explicit `\htmlimage`s into images for HTML, TeX2page extracts the small fragment of the TeX document containing the would-be image into a separate, smaller TeX file. The content of this auxiliary TeX file is then cajoled by a bevy of external programs into an image file suitable for HTML (p. 14). This demands that all the TeX code within the auxiliary TeX file be self-sufficient. However, it is quite possible that such TeX fragments contain references to macros defined elsewhere in the larger document. TeX2page therefore provides the `\imgpreamble ... \endimgpreamble` environment, into which are placed all definitions that are necessary for the HTML images. For example, the "image preamble"

```
\ifx\shipout\UnDeFiNeD % HTML only
  \imgpreamble
    \input some-pic-macs
    \let\gO\Omega
    \def\I#1#2{\int_{#1}^#2}
  \endimgpreamble
\fi
```

allows the use of the control sequences `\gO`, `\I`, and those in `some-pic-macs.tex` in the TeX fragments destined for imagehood.

The commands inside `\imgpreamble` are visible only to TeX2page, so a form of them should also be specified outside the `\imgpreamble` for use by TeX when it processes the entire document for DVI.

Note that if you use encapsulated PostScript inserts, it is not necessary (though it doesn't hurt) to specify an image preamble for loading the `epsf.tex` macro file or `graphicx.sty` package. TeX2page will automatically load them when processing the EPS files. You still need to load these files outside the image preamble for your document to be processable by TeX though.

Image magnification

In general, the magnification of the image inserts, whether math or picture, may not match that of the rest of the text in the HTML output. The DVI output has no such problem, because the math and the picture-macros use the same magnification as the surrounding text. In the HTML output, however, the regular text is rendered at the default magnification of your browser, while the images have come via TeX, and the twain may not meet. Typically, the image is too small.

The solution is to adjust the magnification of just the image inserts. In plain TeX, this can be done by a call to the `\magnification` command *inside* the image preamble. E.g.,

```
\ifx\shipout\UnDeFiNeD
  \imgpreamble
    \magnification\magstep1
    ...
  \endimgpreamble
\fi
```

The above will magnify the HTML math and pictures. Note that it will *not* affect the magnification of these same items in the DVI output. Indeed, you can specify an alternate `\magnification` outside `\imgpreamble`, and that will affect overall size of the entire DVI output, inclusive of math and pictures, as advertised in *The TeXbook* [14].

In sum: `\magnification`, when called *outside* the `\imgpreamble`, magnifies the entire DVI document. When called *inside* the `\imgpreamble`, it will magnify just the images in the HTML document. These two uses of `\magnification` will not interfere.

LaTeX users can use the following:

```
\ifx\shipout\UnDeFiNeD
  \imgpreamble
    \let\LaTeXdocument\document
    \def\document{\LaTeXdocument\Large}
  \endimgpreamble
\fi
```

This tacks a hook on to the `\document` command. (This modified `\document` will only operate on the image.)

Reusing image files

`\def`initions that use math (such as the following one for `\ohm`) work as expected in the HTML output.

```
\def\ohm{$\Omega$}

The circuit uses two 10-\ohm\ resistors, three 50-\ohm\
resistors and one 1-k\ohm\ resistor.
```

produces

The circuit uses two 10-Ω resistors, three 50-Ω resistors and one 1-kΩ resistor.

However, this is very inefficient: Every occurrence of `\ohm` in the document will generate a brand new image file. To advise TeX2page to *reuse* the same image for these multiple occurrences, use `\imgdef` for the HTML:

```
\ifx\shipout\UnDeFiNeD % HTML only
  \imgdef\ohm{$\Omega$}
\else
  \def\ohm{$\Omega$}
\fi
```

Recycling image files
The conversion of TeX fragments into images can consume a lot of time. TeX2page will therefore *recycle* existing image files from a previous run, instead of generating them anew. To *force* generation of new image files, delete the old image files.

X Scheme as TeX's extension language

The command `\eval` allows you to use arbitrary Scheme expressions, as opposed to just TeX macros, to guide the course of the typesetter. The text written to standard output by the Scheme code is substituted for the `\eval` statement. E.g., consider the following complete document, `root2.tex`:

```
\input tex2page

The square root of 2 is
\eval{

(display (sqrt 2))

}.

\bye
```

Running TeX2page on `root2.tex` produces the following HTML output:

The square root of 2 is 1.4142135623730951.

In effect, TeX2page processes the `\eval` call using Scheme, producing some output in an auxiliary TeX file, which is then re-inserted into the document at the location of the `\eval`.

A definition for `\eval` that TeX can use is provided in the macro file `eval4tex.tex`, which is available in the eval4tex [25] distribution. `tex2page.tex` will automatically load `eval4tex.tex` if it finds it in `TEXINPUTS`. Thus, running TeX on `root2.tex` produces a DVI file whose content matches the HTML version.

It is clear that Scheme code via `\eval` can serve as a very powerful *second extension language* for TeX, and that its benefits are available to both the DVI and the HTML outputs. As we have seen, TeX2page implements a subset of the TeX macro language, and for those cases where this macro language isn't enough, Scheme can be used to fill the breach. More generally, Scheme may be preferable to the TeX macro language even for just DVI, where no HTML version of the document is contemplated. We'll explore both of these aspects of `\eval`.

Let us first look at a simple example where `\eval` lets you define an HTML version of an already existing TeX macro that is either impossible or at least prohibitively difficult to process using TeX2page's mimicry of TeX. Consider a hypothetical `\proto` macro, used to introduce the description of a Scheme operator by presenting a *prototypical* use of it. Typical calls to `\proto` are:

```
\proto{cons}{a d}{procedure}
\proto{car}{c}{procedure}
\proto{cdr}{c}{procedure}
```

which typeset as follows:

(cons *a d*) ;procedure
(car *c*) ;procedure
(cdr *c*) ;procedure

The macro `\proto` takes three arguments: the operator name; the metavariables for its operands; and the operator kind. In particular, it typesets the operator and the operands in different fonts, surrounding the call in parens. Note the intervening space between operator and operands.

In the case where there are no operands, the intervening space should not. Thus,

```
\proto{gentemp}{}{procedure}
```

should not produce

(gentemp) ;procedure

but rather

(gentemp) ;procedure

(I.e., no space between `gentemp` and the closing paren.)

The `\proto` macro can be written in TeX as follows:

```
\def\proto#1#2#3{\noindent
  \hbox{{\tt(#1}\spaceifnotempty{#2}{\it#2}{\tt)}%
    \qquad ;#3}\par}
```

where, `\spaceifnotempty` is a helper macro that expands to a space only if its argument is not empty. TeX2page can expand this definition for `\proto`, provided it knows how to deal with the `\spaceifnotempty`.

One way to write `\spaceifnotempty` in TeX is:

```
\newdimen\templen
\newbox\tempbox

\def\spaceifnotempty#1{%
  \setbox\tempbox\hbox{#1}%
  \templen\wd\tempbox
  \ifdim\templen>0pt{\ }\fi}
```

This piece of box-measuring contortion is too much for TeX2page's mimicry of the TeX macro system. However, it's easy enough to achieve the same effect using the string-processing capabilities of Scheme:

```
\ifx\shipout\UnDeFiNeD
\htmlonly

\eval{

(define all-blanks?
  (lambda (s)
    (andmap char-whitespace?
      (string->list s))))

}

\def\spaceifnotempty{\eval{
```

```
(let ((x (ungroup (get-token))))
  (if (not (all-blanks? x))
      (display "\\space")))
}}

\endhtmlonly
\fi
```

`\eval`'s argument is a balanced-brace expression that is sent verbatim to Scheme, except that the pipe character ('`|`') functions as the TeX escape. Use `||` to represent a single pipe in the Scheme code. If you need to include an unmatched brace, simply put a bogus matching brace inside a Scheme comment.

Later `\eval`s can use definitions introduced in previous `\eval`s, as with `all-blanks?` in our example.

If being processed by TeX2page only (as in our example), the code inside `\eval` is allowed to use not just general Scheme but also procedures like `ungroup` and `get-token`, which are defined by TeX2page.

eval without regard to HTML

The key thing to remember is that an `\eval`-call is replaced by whatever text the Scheme code in that `\eval`-call writes to its standard output. This approach will work whether the document is being processed by TeX2page to produce HTML or by TeX to produce DVI.

For those TeX documents that are not intended for HTML conversion, but nevertheless use `\eval`, this macro is available in the macro file `eval4tex.tex`. Run TeX (or LaTeX) on such a document, say *jobname*`.tex`, and then evaluate the resultant *jobname*`.eval4tex` in Scheme, to create the necessary aux TeX files. Running TeX on the master document a *second* time will insert these aux TeX files at the location of the corresponding `\eval` calls. This is quite analogous to how TeX2page would have processed the `\eval`s, except that TeX requires you to explicitly call Scheme to create the aux files which it can use on its second run, whereas TeX2page, being written in Scheme, creates and loads the aux files immediately.

For complete details on using `\eval` with TeX, please consult the companion manual, *An* `\eval` *for TeX* [25].

XI Recovery from errors

If TeX2page reports an error on your document, you may be able to deduce the cause from the diagnostic information that TeX2page displays on standard output. If you failed to look at this information as it was being displayed, you can always retrieve it from the *log file jobname*`.hlog`. This is exactly analogous to TeX generating diagnostic information on standard output and keeping a copy thereof in the file *jobname*`.log`.

The error message typically displays an *error context*, viz., a few consecutive lines from the source document that contain the likely cause of the error. The number of context lines so displayed is governed by the count register `\errorcontextlines`, which has a default value of 5. Thus, setting `\errorcontextlines=7` will display seven lines. Note that error contexts are often only approximate — be prepared to look a little above or below the reported context.

Like TeX, TeX2page also gives you the option of immediately editing the file containing the error, *at* the location of the error. It does so with the following prompt:

```
Type e to edit file at point of error; x to quit.
?
```

When you type `e` at this prompt, a text editor is fired up. What the editor is depends on the environment variables `TEXEDIT` (which is also used by TeX) and `EDITOR`.

If `TEXEDIT` is set, its string value (e.g., "`vim +%d %s`") is chosen as the entire editor call, with `%s` replaced by the offending file's name, and `%d` replaced by the number of the line containing the error. If `TEXEDIT` is not set, or if it is mis-set, i.e., without `%s` or `%d`, then the editor specified in the environment variable `EDITOR` is chosen. If `EDITOR` is also not set, then the editor name is assumed to be `vi`. When using `EDITOR` or `vi`, the file and line number are tacked on as arguments to the editor, with a `+` preceding the line number. This argument style works for all `vi` and `emacs` clones.*

Tracing more information

Sometimes, the diagnostic information in an error message may not be enough to track down the error. TeX provides various commands for generating more diagnostics — TeX2page recognizes the same syntax to provide its own diagnostics. For instance, setting the count registers `\tracingcommands` and `\tracingmacros` to a positive integer causes more log information. Setting `\tracingcommands=1` tells TeX2page to log all calls to atomic commands. Setting `\tracingmacros=1` tells TeX2page to log all macro expansions. You may turn on these traces at any point in your document. You may subsequently turn them off by setting `\tracingcommands=0` and `\tracingmacros=0` respectively.

The command `\tracingall` turns *on* both `\tracingcommands` and `\tracingmacros`.

The TeX command `\errmessage` can be used to generate meaningful error messages. TeX2page, like TeX, ceases processing the document on encountering `\errmessage`.

The TeX command `\message` can be used to print helpful information at selected break points in the document. LaTeX users may prefer `\typeout`, which does the same thing.

All of these commands display their information on both standard output and in the log file. Judicious use of these commands should pinpoint any error.

A Auxiliary files

Given an input TeX document whose main file is `story.tex`, the call

```
(tex2page "story")
```

typically produces at least one output HTML file `story.html`, and possibly some additional HTML files, which are named `story-Z-H-1.html`, `story-Z-H-2.html`, and so on. Additional HTML files are created whenever the input document has commands requesting page breaks in the HTML output.

This is about all you need to know. However, TeX2page does manipulate many other little auxiliary files in order to communicate information both to external programs and across successive runs of itself. The following briefly describes the functions of these auxiliary files, should you ever need to look at them more closely, either out of curiosity or for debugging your document.

TeX2page displays on standard output the log of its progress with `story.tex`. A copy of this log is kept in the log file `story.hlog`.

TeX2page generates a style sheet in `story-Z-S.css`. This contains some default style information that TeX2page generates for every document, plus any style info supplied by the user via `\cssblock` statements in the document.

If `story.tex` uses the external program BibTeX for its bibliography, TeX2page sends information to BibTeX in the file `story-Z-B.aux` and receives information from BibTeX in the file `story-Z-B.bbl`.

* TeX itself uses just `TEXEDIT`. It does not fall back to `EDITOR` or `vi` if `TEXEDIT` is not set. But most Unix programs that reach for an editor do tend to use `EDITOR`, and failing that, `vi`, so TeX2page does the same.

If `story.tex` contains `\index` commands, TeX2page will dump the unsorted index into `story-Z-I.idx` and get from MakeIndex the sorted index `story-Z-I.ind`.

TeX2page uses the auxiliary files `story-Z-L.scm` and `story-Z-A.scm` to keep track of labels and other internal cross-references. Each run of TeX2page loads the `story-Z-L.scm` and `story-Z-A.scm` created by the previous run. If `story.tex` contains *forward* cross-references, TeX2page must be rerun at least once.

For the image portions of `story.tex`, TeX2page creates the auxiliary TeX files `story-Z-G-1.tex`, etc, and uses external programs (as described on p. 14) to convert them to the corresponding image files `story-Z-G-1.gif`, etc. (This assumes you are using the GIF format for images. If you had requested the PNG or JPEG format for images, the extensions of these aux files would be correspondingly different.)

The above are "single-use" images. `story.tex` may reuse some image files within itself. Such image files have slightly different names and are numbered separately: `story-Z-G-D-1.gif`, etc.

Occurrences of `\eval` in `story.tex` typically create the auxiliary Scheme files `story-Z-E-1.scm`, etc. These are converted (by Scheme) into the corresponding auxiliary TeX files `story-Z-E-1.tex`, etc, which are loaded back into `story.tex` on a subsequent run. Only the `\eval`s that will be processed by TeX (i.e., those that are not in HTML-only regions, p. 5) produce such numbered auxiliary files, since the numbering allows successive runs of TeX to access the correct file. Such `\eval`s and their files can also be shared by TeX2page and TeX, without the `\eval`s that occur in the HTML-only portions throwing the numbering off. `\eval`s in HTML-only regions of the document are processed without any memorable aux files, because TeX won't use them, and TeX2page (which, unlike TeX, can call Scheme immediately in the current run) doesn't need them.

By default, all these files are created in the working directory. To avoid cluttering up your working directory, you can specify a different target directory using one of the following three files:

1. *jobname*`.hdir` in the working directory, i.e., a file with the same basename as the input document but with extension `.hdir`. For `story.tex`, this would be `story.hdir`.
2. `.tex2page.hdir` in the working directory.
3. `.tex2page.hdir` in the user's `HOME` directory.

The first line of the first of these files that exists is taken to be the name of the target directory. If none of these files exist, the current working directory is the target directory.

The `.hdir` file may contain the TeX control sequence `\jobname`, which expands to the basename of the input TeX document.

B Bibliography

[1] Karl Berry, Oleg Katsitadze, et al. Eplain (`http://www.tug.org/eplain`).

[2] D. P. Carlisle. Packages in the 'graphics' bundle.

[3] Pehong Chen and Michael A. Harrison. Index Preparation and Processing. *Software — Practice and Experience*, 19(9):897–915, September 1988. Included with MakeIndex in TeX distributions.

[4] Comprehensive TeX Archive Network (CTAN) (`http://www.tug.org/ctan.html`).

[5] Nikos Drakos. LaTeX2HTML: Bringing high-quality documents to the Web (`http://www.latex2html.org`).

[6] Thomas Esser. teTeX (`http://www.tug.org/teTeX`).

[7] FSF. Texinfo: The GNU Documentation System (`http://www.texinfo.org`).

[8] Ghostscript, Ghostview and GSview (`http://www.cs.wisc.edu/~ghost`).

[9] The Gimp (`http://www.gimp.org`).

[10] John D. Hobby. MetaPost (`http://www.tug.org/metapost.html`).

[11] ImageMagick: Convert, Edit, and Compose Images (`http://www.imagemagick.org`).

[12] Inkscape. Draw Freely (`http://www.inkscape.org`).

[13] Donald E. Knuth. *The METAFONTbook*. Addison-Wesley, 1986.

[14] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1993.

[15] Leslie Lamport. MakeIndex: An Index Processor for LaTeX. Included with MakeIndex in TeX distributions.

[16] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, 2nd edition, 1994.

[17] Thomas E. Leathrum, Geoffrey Tobin, and Daniel H. Luecking. MFpic (`http://comp.uark.edu/~luecking/tex/mfpic.html`).

[18] Håkon Wium Lie and Bert Bos. *Cascading Style Sheets*. Addison Wesley Longman, 1999.

[19] Netpbm home page (`http://netpbm.sourceforge.net`).

[20] Oren Patashnik. BibTeXing, 8 February 1988. Included with BibTeX in TeX distributions.

[21] John Pozadzides and Liam Quinn. Cascading Style Sheets (`http://www.htmlhelp.com/reference/css/`).

[22] Tom Rokicki. Dvips (`http://www.radicaleye.com/dvips.html`).

[23] Gagan Saxena. MozPoint (`http://mozpoint.mozdev.org`).

[24] Christian Schenk. MiKTeX (`http://www.miktex.org`).

[25] Dorai Sitaram. An `\eval` for TeX (`http://www.ccs.neu.edu/~dorai/eval4tex/eval4tex-doc.html`).

[26] PRAGMA Advanced Document Engineering (`http://www.pragma-ade.com`). ConTeXt.

[27] W3C. Cascading Style Sheets (`http://www.w3.org/Style/CSS`).

[28] W3C. W3C Core Styles (`http://www.w3.org/StyleSheets/Core`).

[29] Xfig (`http://www.xfig.org`).

C Concept index