T_EX2page

Dorai Sitaram

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

TeX2page makes Web pages from TeX [11] manuscripts. It reads an input document that is marked up in a TeX format (viz., plain TeX, LaTeX [13], Texinfo [6]), 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, and thus 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 [5, 21, 3]. Creating TeX source requires no special-purpose software; any text editor will do. Furthermore, powerful and reliable tools such as BibTeX [18], MakeIndex [12], and MetaPost [9] 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:

1 Running TeX2page

TeX2page is invoked in much the same way as plain TeX or LaTeX.[1] Recall how these programs are called: Given a TeX source 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 either

---

[1] 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*

or

> `latex` *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 20050501 (MzScheme 300, 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 (section 3.1). 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.[2] Note that the bracketed numbers now refer to physical pages.

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

---

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

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 section 11.

1.1 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`.

1.2 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.

1.3 Specifying a target directory

By default, TeX2page generates the output HTML files and other auxiliary files (section A) 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).

2 TeX and TeX2page 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. The
TeX program, which recognizes a list of primitive commands, along with a *format* file that
defines some additional commands, reads the text file, and uses the markup commands in
the text to create an appropriately typeset version of the document in a *DVI file*, which can
then be printed.

TeX2page understands many of the commands of TeX. 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 can process documents written in both the
plain TeX [11] and LaTeX [13] formats.[3] TeX2page also recognizes some commonly used
macros that are loaded from external macro files or LaTeX packages. With the aid of a
macro file `texi2p.tex` (section 2.2), TeX2page can also process Texinfo documents [6].

TeX2page silently ignores non-mathematical TeX commands that it does not under-
stand, and often this is precisely the right treatment. E.g., it is acceptable to ignore com-
mands such as `\leavevmode`, `\noindent`, `\/`, and `\-` when creating an HTML document
from TeX source.

While TeX2page will attempt gamely to process any TeX definitions that you use in
your document (perhaps by `\input`ting external TeX macro files), it is usually a good
idea to have them explicitly ignored (section 9). E.g., you can use macros 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.

2.1 `tex2page.tex` and `tex2page.sty`

TeX2page also processes some TeX-like commands that are not present by default in the
TeX formats. These include commands that are specific to HTML and its hyperlinks;
commands for verbatim text, with special emphasis on *syntax highlighting* for computer-
language fragments; and some rarely used (indeed discouraged) but sometimes unavoidable
directives (section 9) that allow TeX2page and TeX to produce *differing content*. If you use
these commands in your document, and you want your document to still be processable by
TeX, you need to supply some workable TeX definitions for them — even if they do not
quite produce the same effect in the DVI output as they do in the HTML output. Such
definitions are provided in the macro file `tex2page.tex`. It may be included in your TeX
document as

> `\input tex2page`

LaTeX users may alternatively access the macros of `tex2page.tex` via the file `tex2page.sty`,
which has a name that fits better with LaTeX's `\usepackage` command:

---

[3] 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 different 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.

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

This ensures that your document can be processed by both TeX2page and TeX.

As we have seen above, the language recognized by TeX2page is a combination of plain TeX and LaTeX. Most plain-TeX commands are available to the LaTeX user; however the reverse is certainly not the case. This means that a plain-TeX user of TeX2page can use quite a few LaTeX commands in his source that are processable by TeX2page but not by plain TeX. In the interest of generality, the file `tex2page.tex` includes some plain-TeX definitions for these LaTeX commands. You can either choose not to use these commands or override their definitions in `tex2page.tex` 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`.

In this manual, we will use the term "TeX2page command" to refer to commands for which a definition is provided in `tex2page.tex`. It is not necessary that a TeX2page command not be provided by a standard TeX format. For example, the term "LaTeX and TeX2page command" refers to a command that is defined in the LaTeX format, but that is also defined in `tex2page.tex`, and as such can be used in a non-LaTeX format (typically plain TeX) by loading `tex2page.tex`.

2.2 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. `.t2p` files can also be used to adapt TeX2page to other formats of TeX besides plain TeX and LaTeX. For example, the file `texi2p.tex` (provided in the distribution) helps TeX2page process Texinfo documents.

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.

3 Sections

The command `\title` may be used to title the entire document.

```
\title{The Odyssey}
```

You can use \\ to insert linebreaks in a multiline \title.

If you wish a different "external" title for the Web document, use the TeX2page command \externaltitle. TeX will ignore \externaltitle.

For Plain TeX documents, TeX2page will set the title where \title is called. In LaTeX, however, \title merely stores the title; the command that actually prints the title is \maketitle. The LaTeX commands \author and \date can be used to pass additional information to \maketitle. If \date isn't specified, TeX2page, like LaTeX, will use \today.

TeX2page recognizes the following sectioning commands: \section, \subsection, \subsubsection, \paragraph, and \subparagraph. Usage:

    \section{The Princess at the River}

The sectioning commands are numbered, and can be cross-referenced using *labels* (section 7). Unnumbered sections can created by affixing '*' to the sectioning command, e.g.,

    \section*{The Princess at the River}

Section heads may be collected into a *table of contents* (section 7.3).

The section number is a dotted number that reflects the section's *depth*. E.g., the second \subsubsection in the fourth \subsection of the third \section is numbered "3.4.2".

TeX2page recognizes subsections deeper than \subparagraph (depth = 5), although it does not provide the \sub...subsection or \sub...paragraph macro at these depths. To specify a section at depth n, use \sectiond{n}. Thus, \subsection is merely a convenient abbreviation for \sectiond{2}.

The command \chapter can also be used, and is useful for book-length documents. Following LaTeX convention, \chapters are considered to be at depth 0. \chapter causes a page break (section 3.1) and typesets the header more prominently than \section. \chapter* produces unnumbered chapter headings.

The command \appendix causes subsequent top-level (i.e., depth = 0 if \chapters are used, depth = 1 otherwise) headings to be identified *alphabetically* rather than *numerically*.

3.1 Producing several HTML pages

Typically, TeX2page produces a single HTML page for the entire document. There are a couple of exceptions: The \chapter command will start a new HTML page. For some documents, you may want to split the document into pages at your own discretion.

As in TeX, use the commands \eject, \supereject, or \dosupereject to force a page break. (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.) LaTeX users can additionally use \pagebreak, \newpage, \clearpage, \cleardoublepage. The TeX2page command \htmlpagebreak can be used to create a page break only for HTML.

Each of the resulting pages has a *navigation bar* at the top and at the bottom that let you travel quickly to the *first*, *previous*, or *next* page. If the document has a table of contents or an index, buttons for these are also embedded in the navigation bar.

4 Style

You can have your HTML output use *style sheets* [24, 15, 19]. The command

    \inputcss basic.css

in your TeX source will cause the HTML output to use the presentation advice contained in the style sheet basic.css. The \inputcss command has no relevance for the DVI output.

In the style sheet, you can have rules for the various HTML elements to change the appearance of your document. E.g.,

```
h1      {color: navy}
```

will cause all top-level headers to be navy-blue. 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 (sections 6.4 and 6.5).

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.,

```
\cssblock
h1      {color: navy}
\endcssblock
```

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

TeX2page generates a default style sheet for the converted document, *jobname*`-Z-S.css`. 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* [25].

4.1 Making a slideshow

The MozPoint [16] library (which relies on Javascript and style sheets) allows a sequence of HTML pages to be used as a presentation or slideshow. 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, press `p`, left or up arrow. To go back to the first (i.e., title) slide, press `t` or `0`. To go the *n*th 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 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.

5 Color

In contrast to style sheets which affect only the HTML output, the commands `\color` and `\definecolor` may be used to specify text color for both HTML and DVI output. These commands are provided by the standard LaTeX package `color.sty`, and are also defined in `tex2page.tex` for use with plain TeX.

> `\color[`*color-model*`]{`*color-spec*`}`

specifies that the rest of the text in the current group should be in the color given by *color-spec* using *color-model*.

color-model is one of `rgb`, `cmyk`, `gray`, `RGB`, or `named`, and may be omitted (with the brackets) if it is `named`. The *color-spec* for the model `gray` is a number between 0 and 1 (inclusive); for `rgb`, a comma-separated list of three numbers, all between 0 and 1; for `cmyk`, a list of four numbers, all between 0 and 1; for `RGB`, a list of three integers, all between 0 and 255 (inclusive); and for `named`, a pre- or user-defined color name.

Eg:

```
{\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},
{\color[named]{magenta} magenta}, and
{\color{blue} blue}.
```

produces:

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

Predefined color names are `red`, `blue`, `green`, `cyan`, `magenta`, `yellow`, `black`, and `white`. New color names can be defined by:

\definecolor{*new-color-name*}{*color-model*}{*color-spec*}

Eg:

```
\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. For plain TeX documents, load `dvipsnam.def` *after* loading `tex2page.tex`.

6 Verbatim text

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{The Test of the Bow}} types \verb{The
Test of the Bow} as a section title.
```

is converted to

The command \section{The Test of the Bow} types `The Test of the Bow` 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.

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"

The command `\path` is similar to `\verb`. The only difference is that when the document is TeX'd, the text specified by `\path` can be broken across lines at '.' and '/'. This is useful for long URLs and filenames.

TeX2page also understands LaTeX's `{verbatim}` and `{verbatim*}` environments, which█ set displayed verbatim text with spaces and newlines as is. `{verbatim*}` differs from `{verbatim}` in that spaces are highlighted as something visible.

Note: Unfortunately, you cannot use `\verb` and `\path` in LaTeX section headers. While TeX2page itself has no problem with this sort of construction, LaTeX *will* cause error. Use `\tt` instead, perhaps with some other definitions for special characters. The section macros provided in `tex2page.tex` for use with plain TeX do not have this problem.

6.1 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.

6.1.1 Changing the verbatim escape character

You can use the macro `\verbescapechar` to postulate a character other than '|' as the verbatim escape. E.g.,

```
\verbescapechar\@
```

makes '@' the verbatim escape.

6.2 Inserting files verbatim

You can insert files verbatim with the command \verbatiminput. Usage:

```
\verbatiminput progam.scm      % or
\verbatiminput{program.scm}
```

This displays the contents of the file `program.scm` "as is". Useful for listings.[4]

6.3 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`.[5]

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.

6.4 Verbatim style

The verbatim commands \verb, \path and \verbatiminput 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 (section 4). E.g.,

```
.verbatim       {color: darkgreen}
```

makes all verbatim text dark green.

6.5 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 fact
  "The factorial function"
  (lambda (n)
    (if (= n 0) 1 ;the base case
        (* n (fact (- n 1))))))
}
```

---

[4] For the DVI output, you can use the definition for \verbatiminput in `tex2page.tex` or, in LaTeX, load the package `verbatim.sty`. Note that the latter only accepts a braced filename as argument. `tex2page.tex` will not overwrite the definition from `verbatim.sty`.

[5] 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{*command*}, instead of writing *command* 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` [5] or `--enable-write18` [21].

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

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 (or perhaps other properties) of these classes in a style sheet (section 4). E.g., the style sheet for this document uses:

```
.scheme                  {color: brown} /* background punctuation */
.scheme   .keyword    {color: black; font-weight: bold}
.scheme   .builtin    {color: #990000}
.scheme   .variable   {color: navy}
.scheme   .global     {color: purple}
.scheme   .selfeval   {color: green}
.scheme   .comment    {color: teal}
```

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 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}
```

6.5.1 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, declare `\slatexlikecomments` before first use.

`\slatexlikecomments` is not an unmixed blessing, however, as it restricts your Scheme comments to text that is valid TeX. Use `\noslatexlikecomments` to go back to verbatim comments.

6.6 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` (sec. 6.3).

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

7 Cross-references

The command `\label{`*tag*`}` *anchors* a location in the document that can be referred to elsewhere in the document with a `\ref{`*tag*`}`.[6] The location represented by a `\label` is the header of the smallest sectioning environment enclosing the `\label`. This sectioning environment can be a chapter, (sub)section, or footnote.

A `\label` command is typically placed after the sectioning command, and the corresponding `\ref` will print as that section's number. In HTML, the `\ref` text will additionally be a *link*: *Clicking* that link will cause the browser to display the part of the web page starting the labeled section.

Note that the `\label` merely names pre-selected anchor points in the documents, viz., the openings of chapters, (sub)sections, and footnotes. To insert your own anchors at arbitrary locations in the document, use `\tag{`*atag*`}{`*tagvalue*`}`. (You need to supply a *tagvalue*, because there is no sectioning environment whose number `\tag` can assume.) This causes `\ref{`*atag*`}` to expand to *tagvalue*, and furthermore, in the HTML, to be a hyperlink to the location of the `\tag`. The main use for the `\tag` command is to enrich the HTML version of your document with hyperlinks — unlike `\label`, `\tag` does not add value to the print version.

You may need to rerun TeX2page in order to resolve the cross-references in a document. TeX2page will tell you if this is the case.

7.1 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. 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 a 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 `\tag` 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

---

[6] If you are using the Eplain [1] format, use the name `\tagref` instead of `\ref` to get the behavior described here. Or do `\let\ref\tagref` after `\input`ting `tex2page.tex`. Eplain uses the name `\ref` for a somewhat different operation, and `tex2page.tex` will not by itself overwrite it.

```
    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 (sec. 3.1), 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.

7.1.1 Abbreviations for specifying links

\urlhd takes three arguments. In some cases the second and third arguments may be mere repetitions of a preceding argument. For such cases, TeX2page provides some convenient abbreviations.

\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" 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 `\tag` and `\urlh` is useful for inserting internal cross-references ■ in the HTML document without affecting the print document. For example, the following text

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

... lots of intervening stuff ...

\tag{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*`})}`.

`\mailto` is a single-argument command for specifying email addresses. `\mailto{`*address*`}` ■ abbreviates `\urlh{mailto:`*address*`}{\path{`*address*`}}`.

7.2 Referring to labels in related documents

TeX2page allows several related online documents to refer to labels in one another, much as each would refer to labels within itself. Each document needs to identify the location of the other documents that it wishes to refer to. The commands `\includeexternallabels` and `\inputexternallabels` accomplish two flavors of such mutual cross-referencing.

7.2.1 \includeexternallabels

`\includeexternallabels` *external-jobname*

allows the current document to incorporate labels from *external-jobname*, and refer to them as it would to its own labels, i.e., using `\ref`. It is therefore important that the label names used by the documents should not clash.

The argument *external-jobname* is the appropriate pathname to the external document. Use the name of the main HTML file, without the extension. If using a relative pathname, take care to calculate the pathname from the location of the current document's main HTML (*not* source) file to the location of the external document's main HTML (again, *not* source) file.

For example, if you have two documents `docA.tex` and `docB.tex` in the current directory, and you have used the `.hdir` mechanism to direct the HTML files to go into

subdirectories `docA` and `docB`[7], then `docA.tex` should contain

> `\includeexternallabels ../docB/docB`

and `docB.tex` should contain

> `\includeexternallabels ../docA/docA`

7.2.2 \inputexternallabels

> `\inputexternallabels` *external-jobname*

also allows the current document to refer to labels from *external-jobname*. However, unlike `\includeexternallabels`, the labels are not incorporated into the current documents pool of labels. The call merely identifies that the labels from *external-jobname* are available for use in the current document, but with appropriate qualification. Instead of `\ref`, the command `\htmlrefexternal` is used to refer these labels. The call

> `\htmlrefexternal{`*text*`}{`*external-jobname*`}{`*external-label*`}`

typesets *text* as a hyperlink to the label *external-label* in the HTML document of basename *external-jobname*.

Note that the external document is named explicitly. While the situation is more verbose than that with `\includeexternallabels`, it makes no requirement that the names of the labels in the two documents should not clash.

7.3 Table of contents

The command `\tableofcontents` emits a list of the section names in the document, with their numbers. In HTML, these entries are links to the corresponding sections.

7.4 Footnotes

Footnotes are a more print-oriented form of cross-reference. TeX2page recognizes plain TeX's `\footnote` as well as a new macro called `\numfootnote`. The latter numbers footnotes automatically and sequentially (i.e., the user need not think up footnote marks).

Both `\footnote` and `\numfootnote` produce the expected output. The footnote mark occurs both in the body of the text and at the bottom of the relevant (DVI or HTML) page, with the latter accompanied by the footnote text.

Additionally, in the HTML output, the footnote mark in the text body is a link to the footnote mark in the footnote text, and vice versa. This paragraph ends with a footnote. Click the footnote mark to see the footnote text at the bottom of the HTML page. You can either click the "back" button on your browser or the footnote mark itself to get back to the body of the text.[8]

The footnote above (below?) was generated by

```
... to get back to the body of the
text.\numfootnote{Footnotes are separated ...
by a horizontal rule.}
```

---

[7] This could be done, for instance, by having both `docA.hdir` and `docB.hdir` contain `\jobname`.

[8] Footnotes are separated from the body of the page by a horizontal rule.

7.5 Bibliographies

TeX2page can use the external program BibTeX [13, 18] to generate bibliographies from bibliographic database files. A bibliographic database file is a `.bib` file containing bibliographic entries of the form

```
@book{tex,
  author = "D E Knuth",
  title = "{The TeXbook}",
  publisher = "Addison-Wesley",
  year = 1993
}
```

TeX source can *cite* a bibliographic entry using the command `\cite`. E.g.,

```
Here's an example diagram from {\it The
TeXbook\/}~\cite[p.~389]{tex}.
```

`\cite` and the other bibliography-related commands described below are included in LaTeX. For plain TeX, you will need to explicitly load the macro file `btxmac.tex`, present in all TeX distributions.

The command `\bibliographstyle` specifies the style of the citations: `plain` numbers the bibliography items, whereas `alpha` gives them mnemonic alphanumeric keys. The command `\bibliography` specifies one or more `.bib` files to search for the citations, and generates a *bibliography*, i.e., a sorted list of all the cited entries. E.g.,

```
\section*{Bibliography}

\bibliographystyle{plain}
\bibliography{tex,scheme,html}
```

Here `tex.bib`, `scheme.bib`, and `html.bib` are the `.bib` files used, presumably containing entries specific to TeX, Scheme, and HTML respectively.

`\nocite{`*citation*`}` will include in the bibliography the entry for *citation*, without needing to cite it in the text. `\nocite{*}` will include *all* the entries from *all* the supplied `.bib` files.

A first run of TeX2page on the document *jobname*`.ext` creates an auxiliary file *jobname*-`Z-B.aux`. A subsequent run of TeX2page calls BibTeX on *jobname*-`Z-B.aux` to produce the corresponding sorted bibliography in the file *jobname*-`Z-B.bbl`, which is slurped into the output document. (You may call BibTeX yourself, as you would have to do when TeXing the document for the DVI output.)

If TeX2page cannot create or find *jobname*-`Z-B.bbl` despite its best efforts, it will inform you that you need to generate it manually. Once created, the file is reused as is in future runs. Delete the file to have it regenerated (perhaps because your document has changed).

BibTeX is convenient for selecting, sorting, and writing out in appropriate format the relevant bibliographic entries for your document, but if for some reason you want to do it all on your own, you can.[9] Use the `thebibliography` environment to enclose your bibliographic entries, and introduce each entry with `\bibitem`. For more details, see the LaTeX manual [13, section 4.3.2, p. 71], or see a sample `.bbl` file generated by BibTeX and imitate.

---

[9] This approach, while tedious and a maintenance millstone, *can* be rational sometimes: E.g., if your bibliographic entries are written in a raconteur's style and include opinions or digressions that are tailormade for the particular document at hand, they are likely inappropriate for inclusion in a quasi-central bibliographic database.

7.6 Index generation

TeX2page can use the external program MakeIndex [2, 12] to generate indices. TeX2page's index-generation feature follows the same conventions as traditionally used with TeX and its derived packages [13, section 4.5, appendix A].

This means that an occurrence of \index{*item*} in the TeX source causes *item* to be entered into an unsorted index file, *jobname*-Z-I.idx. A subsequent run of TeX2page calls MakeIndex on *jobname*-Z-I.idx to produce the sorted index in *jobname*-Z-I.ind, which is included in the output using a command such as \printindex. (You may call MakeIndex yourself, as you would have to do when TeXing the document for the DVI output.)

The auxiliary files have the basename *jobname*-Z-I rather than *jobname* as in TeX, because the HTML index is necessarily different in character from the DVI index: Whereas the DVI index item mentions one or more page numbers in the main text where the indexed item occurs, the HTML index item is a *hyperlink* into the main text.

If an indexed item needs to point to multiple occurrences in the main text, the hyperlink associated with the index entry points to the first occurrence. The hyperlinks for succeeding occurrences are notated by bracketed numbers starting from 2. (The number represents the number of the occurrence.)

TeX2page recognizes two macros for index insertion. First, there is the conventional \printindex which emits the sorted index inside a section called "Index". In addition, there is also \inputindex, which emits just the index without a section header. This is so that you can set the index your own way. E.g., you may want to have a different section header or include some introductory prose.

If TeX2page cannot find the sorted index file (*jobname*-Z-I.ind) despite its best efforts, it will inform you that you need to generate it manually. Once created, the file is reused as is in future runs. Delete the file to have it regenerated (perhaps because your document has changed).

7.7 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. You can control both the placement and the detail of the colophon using the \htmlcolophon command.

\htmlcolophon{last-page} puts the colophon on the last, instead of the first, page. \htmlcolophon{no-timestamp} prevents mention of the last modification time of the document.[10] \htmlcolophon{dont-link-to-tex2page-website} will mention TeX2page, but without hyperlinking to the TeX2page website. To avoid mentioning TeX2page at all, use \htmlcolophon{dont-credit-tex2page}, which also has the convenient shorter form \htmlcolophon{ingrate}.

These arguments to \htmlcolophon can be grouped together, with whitespace separating them. Thus, \htmlcolophon{last-page dont-credit-tex2page} produces on the last page a colophon containing only the timestamp. To produce no colophon at all, do \htmlcolophon{no-timestamp dont-credit-tex2page}.

A call to \htmlcolophon requesting last-page is best placed in the document before text for the second page starts, so as to avoid the default of the colophon appearing at the end of the first page.

---

[10] If the underlying Scheme is incapable of determining a file's write date, no-timestamp is automatically assumed.

8 Images

Some portions of your TeX source may be explicitly images, or text that is particularly resistant to conversion to HTML. Examples are encapsulated PostScript inserts, mathematics, and the LaTeX {picture} environment. In such instances, TeX2page invokes a combination of TeX, Dvips [20], Ghostscript [7] and the NetPBM library [17] to produce *image files*, which are inserted into the HTML output.

By default, the image files employ the GIF format. You may change the format to PNG[11] or JPEG using the \htmlimageformat command, e.g.,

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

8.1 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 $\sqrt{ (\Delta x)^2 + (\Delta y)^2 }$.

---

[11] PNG would have been the default image format of choice, were it not for the fact that browser support for *transparent* PNGs is currently poor. If your HTML background color is pure white, PNG is a good choice as lack of transparency is not a concern.

You can control whether your mathematics should be specified as image or ascii with the command \htmlmathstyle:

\htmlmathstyle{no-in-text-image} sets in-text math (i.e., math embedded in running text) as ascii. \htmlmathstyle{no-display-image} sets displayed math as ascii. \htmlmathstyle{no-image} abbreviates \htmlmathstyle{no-in-text-image no-display-image}█ and sets all math as ascii.

\htmlmathstyle{in-text-image}, \htmlmathstyle{display-image}, and \htmlmathstyle{image}█ respectively set in-text, displayed, and all math as images. By default, \htmlmathstyle{image}█ holds.

If the mathematical notation in your document is simple enough not to need images, it is advisable to set one or both of the no- options of \htmlmathstyle.

If you do all your mathematics in roman numbers, you can avoid math-related images completely. TeX2page recognizes the TeX command \romannumeral, which produces the roman equivalent of the following arabic number (\romannumeral 1986 = mcmlxxxvi). \romannumeral produces lower-case letters — tex2page.tex includes \Romannumeral, whose█ result is all-upper-case (\Romannumeral 1986 = MCMLXXXVI).

8.2 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 Xfig [26] and The Gimp [8], or algebraic ones like MetaPost [9]. 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 [14], whose TeX macros transform a picture specification inside the document into an external META-FONT [10] 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}        % LaTeX only
```

TeX2page converts such calls into images. Here is an example: 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[12]

---

[12] The file lambda.mp was actually written out from this document's source using \verbwrite█ (sec. 6.3), 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.
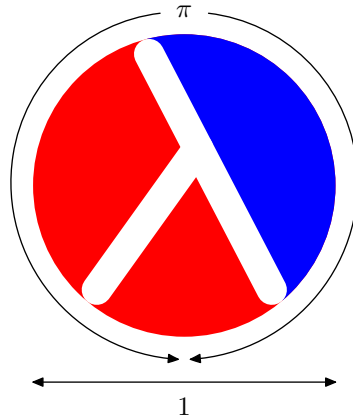
For `\epsfbox`, you can specify the desired image width and height by assigning to the dimen registers `\epsfxsize` and `\epsfysize` (specifying only one of them will cause the other to change as well, maintaining the image's aspect ratio). TeX2page will respect such sizes, equating one browser pixel to one point ($= 1/72.27$ inch). Thus,

```
\epsfxsize=1.5in
```

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

Note that `\epsfbox` and `\includegraphics` are defined external to plain TeX and 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 available only to LaTeX, and is defined in the package `graphicx.sty`.

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 [23] 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. PDF LaTeX uses `\includegraphics` as before, whereas in PDF plain TeX, a command like

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

is used. TeX2page recognizes `\pdfximage` including the scaling information, with one browser pixel equated to one point.

8.3 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* [11, 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:
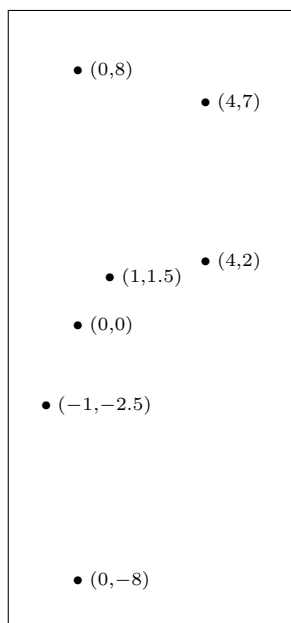
\makehtmlimage's argument is a group containing no unmatched braces.

8.4 Image preamble

When converting math, EPS, and other implicit or explicit \htmlimages 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. 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"

```
\imgpreamble
  \input some-pic-macs
  \let\gO\Omega
  \def\I#1#2{\int_{#1}^#2}
\endimgpreamble
```

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.

8.5 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.,

```
\imgpreamble
  \magnification\magstep1
  ...
\endimgpreamble
```

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* [11].

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:

```
\imgpreamble
  \let\LaTeXdocument\document
  \def\document{\LaTeXdocument\Large}
\endimgpreamble
```

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

8.6 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, change the `\def` to an `\imgdef`:

```
\imgdef\ohm{$\Omega$}
```

8.7 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.

9 Paper and screen

Some text you want to go only to TeX. Other text you want to go only to HTML. For such purposes, use the directives

```
\texonly
   ... for TeX only ...
\endtexonly
```

and

```
\htmlonly
   ... for HTML only ...
\endhtmlonly
```

The `\texonly` and `\htmlonly` directives are defined in the macro file `tex2page.tex`. Thus, you *must* have your document `\input tex2page.tex` before you can use these directives. If not, running the document through TeX will produce errors of undefinition.

If you *don't* want to load the macros of `tex2page.tex` and would yet like to distinguish between TeX-only and HTML-only text in your document, you can exploit the fact that certain TeX control sequences such as `\shipout` have no definition in TeX2page. Thus, the conditional

```
\ifx\shipout\undefined
   ... for HTML only ...
\else
   ... for TeX only ...
\fi
```

produces differing text in the DVI and HTML output, without the need for `tex2page.tex`.

`\htmlonly` and `\texonly` are more robust than using conditionals such as `\ifx\shipout`. If you must use the latter approach, make sure that the "then" branch constitutes the HTML-only portion. This is because any verbatim occurrence of `\else`, `\fi`, or `\if...` commands in the TeX-only portion (to be sure, not a common situation) may cause TeX2page to misread where the "then" branch ends and the "else" branch begins. (In general, you need to be careful with `\if...` nesting in TeX too [11, ch. 20, p. 211], not just for TeX2page.)

Note that the text inside the HTML-only portion is in TeX format. To specify some of this text as *raw* HTML, enclose it in `\rawhtml ... \endrawhtml`.

```
\htmlonly
   ... \rawhtml raw HTML text \endrawhtml ...
\endhtmlonly
```

The `\rawhtml` environment can occur only within an HTML-only text (whether described by `\htmlonly` or via `\ifx\shipout`.)

The directives are used as follows: In cases where the print and web *content* must differ, use `\texonly` and `\htmlonly`. In cases where the web content must use raw HTML features, use `\rawhtml`.

Use of these directives may seem to miss the point of TeX2page. `\texonly` and `\htmlonly` violate the principle of avoiding writing *two* documents, one for HTML, the other for TeX. `\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, as the following examples show.

9.1 Example uses of TeX2page directives

Many TeX macros, while crucial for printed copy, are irrelevant to HTML, e.g., the statement `\parindent=10pt`. Such macros are best enclosed in a `\texonly`, to avoid erroneous or fruitless translation by TeX2page.[13]

The `\evalsto` command we saw above (sec 6.1) was rather shabby. A better alternative would be to exploit TeX's math symbols, i.e.,

```
\def\evalsto{\leavevmode\hbox{$\Rightarrow$}}
```

or, better still (section 8.6),

```
\imgdef\evalsto{\leavevmode\hbox{$\Rightarrow$}}
```

which avoids generating a separate image file for each use of `\evalsto`.

While this will work, using an image for this token is probably overkill for your HTML output. So let's make this definition TeX-only:

```
\texonly
  \def\evalsto{\leavevmode\hbox{$\Rightarrow$}}
\endtexonly
```

Now, the DVI version typesets as expected, but the HTML verbatim has no access to any definition for `\evalsto`. In such situations, TeX2page will use the *name* of the command. Thus, given

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

the HTML output will be

```
(cons 1 2) \evalsto (1 . 2)
```

Obviously, this is no improvement over our previous `\evalsto` definition. While we do not want the full-fledged TeX definition, we do want some sort of "poor man's equivalent", e.g.,

```
\htmlonly
  \def\evalsto{\p{=>}}
\endhtmlonly
```

With these definitions in place, the verbatim will now translate, in HTML, to

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

---

[13] Actually, `\parindent` and other common print-specific statements are automatically recognized as irrelevant for the Web by TeX2page without the need for an explicit `\texonly`.

While we may have relinquished TeX's niceties for our HTML version of \evalsto, we can certainly seek to compensate by using HTML's own niceties. \rawhtml comes in handy for this:

```
\htmlonly
  \def\evalsto{\rawhtml<font
      color=blue><b>=&gt;</b></font>\endrawhtml}
\endhtmlonly
```

The \evalsto token now has a higher contrast against the surrounding code.

10 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 eval4tex [22] 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. (See sec. 10.1.)

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:

```
\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
```

26

`\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 `\evals` can use definitions introduced in previous `\evals`, 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.

10.1 `\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 `\evals`, 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* [22].

11 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 counter `\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.[14]

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, the *tracing* directives `\tracingcommands` and `\tracingmacros` produce 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`.

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`.

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 the external programs TeX, Dvips, Ghostscript and NetPBM to convert them to the corresponding image files `story-Z-G-1.gif`, etc. This assumes you are using the GIF format for images. Change the extension `.gif` to `.png` or `.jpeg` if your images are in PNG or JPEG.

---

[14] 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.

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 `\evals` that will be processed by TeX (i.e., those that are not in `\htmlonly`) produce such numbered auxiliary files, since the numbering allows successive runs of TeX to access the correct file. Such `\evals` and their files can also be shared by TeX2page and TeX, without the `\evals` that occur in the `\htmlonly` portions throwing the numbering off. `\evals` in `\htmlonly` 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 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:

- *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`.
- `.tex2page.hdir` in the working directory.
- `.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 et al. Eplain (`http://www.tug.org/eplain`).

[2] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

[16] MozPoint (`http://mozpoint.mozdev.org`).

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

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

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

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

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

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

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

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

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

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

C Concept index