TeX2page

Version 4p4k3
Dorai Sitaram

TeX2page is a Scheme[1] program that makes Web pages from TeX documents.[2] It reads an input document that is marked up in a TeX format (viz, plain TeX, LaTeX, Texinfo), 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.

There are several advantages to keeping the document source in TeX: (1) There is no need to write and maintain two separate documents, one for paper and the other for the screen. (2) Creating TeX documents requires no special-purpose software; any text editor will do. (3) TeX is *stable*. It has a mature tradition and widespread intellectual resources behind it [`ctan`]. It is here to stay for at least a couple of centuries. (4) Powerful and reliable tools such as BibTeX [`bibtex`], MakeIndex [`makeindex`], and MetaPost [`metapost`] have developed around TeX, and their benefits can be enjoyed by TeX2page too. (5) 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] TeX2page also runs on Common Lisp.

[2] There are many other TeX-to-HTML converters available. The most popular is LaTeX2HTML [`latex2html`], which converts LaTeX only. TtH [`tth`], which converts both plain TeX and LaTeX, uses a special symbol font to render mathematics as quick-loading text rather than slow graphics. TeX4ht [`tex4ht`] is a highly configurable authoring system that operates on TeX's DVI output rather than the source text.

1 Running TeX2page

TeX2page is invoked in much the same way as plain TeX or LaTeX.[3] 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

`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 4p4k3 (MzScheme 200, unix)
(story.tex)
Added missing \end
[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 (sec 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`.

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

```
    This is TeX, Version 3.14159 (Web2C 7.3.1) (format=tex 2001.7.9)  21
JUN 2002 14:39
    **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.[4] 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.

As with TeX, the filenames encountered by TeX2page in the document are first looked for relative to the working directory. If not found there, they are looked for in the directories listed in the environment variable `TIIPINPUTS`. This is analogous to TeX looking for files in `TEXINPUTS`.[5]

Error recovery in TeX2page is also exactly analogous to TeX, but we will postpone that discussion to sec 10.

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 (eg, `--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

If, for any reason, it is not possible to call `tex2page` from your operating-system command line, you may load the file `tex2page` into your Scheme and call the *procedure* `tex2page` with the name of the TeX file as argument:

```
(load "tex2page") ;use appropriate pathname

(tex2page filename)
```

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

---

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

[5] The `TEXINPUTS` value can use a quite complicated syntax, which is why TeX2page uses its own environment variable instead of simply reusing TeX's `TEXINPUTS`. However, to make a virtue out of a necessity, it is quite unnecessary for TeX2page to attempt parsing most of the TeX files that are accessible via `TEXINPUTS`. Furthermore, you can have files in `TIIPINPUTS` usefully shadow files from `TEXINPUTS`.

1.3 Specifying a target directory

By default, TeX2page generates the output HTML files and other auxiliary files (sec 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 [`tex`] and LaTeX [`latex`] formats.[6] With the aid of a macro file `texinfo.t2p` (sec 2.2), TeX2page can also process Texinfo documents.

Here is a representative but incomplete list of the TeX commands that TeX2page recognizes:

Plain TeX commands recognized include:

- the font switches (`\it`, `\bf`, and the like);
- `%` as the comment character;

[6] 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 (eg, `\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.

- grouping (`{...}`), `\bgroup` and `\egroup`, `\begingroup` and `\endgroup`, `\aftergroup`;
- commands for special characters (`\%`, `\#`, etc);
- accents and special letters (`\"a` for 'ä', `\ae` for 'æ', etc);
- `\beginsection`, `\footnote`, `\input`, `\narrower`, and `\obeylines`;
- the table generator `\halign`;
- `\break`, `\endgraf` and `\par`, the vertical skips (`\smallskip`, `\smallbreak`, etc);
- the tie '`~`' and the explicit space '`\ `';
- `\end`, `\bye`, `\endinput`;
- mathematical notation (text between matching `$` or `$$`), as either image or (if not too complicated) plain HTML;
- a fair fraction of the uses of `\let`, `\def`, `\edef`, `\gdef`, `\xdef`, `\global`, with local definitions and argument delimitation; active characters can also be given definitions after setting their `\catcode` to `\active` (13);
- count register manipulation: `\newcount`, `\advance`, `\the`;
- conditionals `\if`, `\ifx`, `\ifnum`, `\iftrue`, `\iffalse`, with `\else` and `\fi`; and
- debugging aids `\errmessage`, `\message`, `tracingall`, `\tracingcommands`, `\tracingmacros`.

LaTeX commands recognized include:

- the sectioning commands `\chapter`, `\section`, `\subsection`, etc, and their `*`'d counterparts;
- the listing environments `enumerate`, `itemize`, and `description`;
- the cross-referencing aids, viz, `\label` and `\ref`, `\index`, `\cite`, `\nocite`, `\bibliography`, `thebibliography`, and `\bibitem`; and
- the command definers `\newcommand` and `\renewcommand`, `\newenvironment` and `\renewenvironment`, `\newtheorem`.

TeX2page silently ignores TeX commands that it does not understand, and often this is precisely the right treatment. Eg, it is acceptable to ignore commands 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 (sec 8). Eg, 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* (sec 8) 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:

```
\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.[7] 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 notation supported by TeX2page. If your document does not use this extra notation, then you can do without `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 `texinfo.t2p` (provided in the distribution) helps TeX2page process Texinfo [`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. Eg, 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
```

[7] Plain TeX users should type `\itemize` and `\enditemize` respectively in place of LaTeX's `\begin{itemize}` and `\end{itemize}`, etc.

```
\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 are provided in the `t2p-example` subdirectory 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 `\externaltitle`. TeX will ignore `\externaltitle`.

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* (sec 5). Unnumbered sections can created by affixing '`*`' to the sectioning command, eg,

```
\section*{The Princess at the River}
```

Section heads may be collected into a *table of contents* (sec 5.2).

The section number is a dotted number that reflects the section's *depth*. Eg, 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, `\chapter`s are considered to be at depth 0. `\chapter` causes a page break (sec 3.1) and typesets the header more prominently than `\section`. `\chapter*` produces unnumbered chapter headings.

The command `\appendix` causes subsequent top-level (ie, depth = 0 if `\chapter`s 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.[8] LaTeX users can additionally use `\pagebreak`, `\newpage`, `\clearpage`, `\cleardoublepage`. 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.

4 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. 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. Eg,

```
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. Eg,

```
\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.

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.

[8] 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.

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.

4.1 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.

4.1.1 Changing the verbatim escape character

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

```
\verbescapechar\@
```

makes '`@`' the verbatim escape.

4.2 Inserting files verbatim

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

```
\verbinput progam.scm
```

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

4.3 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`.

To specify another text file, use `\verbfilename`. Eg,

```
\verbfilename notes-to-myself.txt
```

This will cause subsequent calls to `\verbwrite` upto the next `\verbfilename` or end of document (whichever comes first) to send text into the file `notes-to-myself.txt`.

4.4 Verbatim style

The verbatim commands `\verb`, `\path` and `\verbinput` 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 (sec 7). Eg,

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

makes all verbatim text dark green.

4.5 Syntax-highlighting of program code

The commands `\scm` and `\scminput` are variants of `\verb` and `\verbinput`. They are useful for producing syntax-highlighted Scheme code in the HTML file. Eg,

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

Six categories of code text are distinguished:

1. self-evaluating atoms (numbers, booleans, characters, strings);
2. syntactic keywords;
3. global or special variables, viz, identifiers that begin and end with an asterisk;
4. other variables;
5. comments; and
6. background punctuation.

To distinguish between the categories of Scheme code text, TeX2page uses a style class called `scheme` with five subclasses, viz, `selfeval`, `keyword`, `global`, `variable`, and `comment`. You can set the `color` property (or perhaps other properties) of these classes in a style sheet. Eg, the style sheet for this document uses

```
.scheme             {color: brown} /* background punctuation */
.scheme  .keyword   {color: #cc0000; font-weight: bold}
.scheme  .variable  {color: navy}
.scheme  .global    {color: purple}
.scheme  .selfeval  {color: green}
.scheme  .comment   {color: teal}
```

Users can add their own keywords and constants with `\scmkeyword` and `\scmconstant`. Eg,

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

4.6 Documenting your code

You can use TeX2page to do a form of *literate programming*, ie, 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 a code file. The code file has by default the same basename as the document, but with extension `.scm`.

To specify another code file, use `\scmfilename`. Eg,

```
\scmfilename factorial.lisp
```

This will cause subsequent `\scmdribble`d code upto the next `\scmfilename` or end of document (whichever comes first) to go into the file `factorial.lisp`.

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

5 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*`}`.[9] 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.

5.1 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`'

[9] If you are using the Eplain [`eplain`] 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.

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. Eg,

```
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 (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.

5.1.1 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}\\ (sec~\ref{callcc})}.
```

produces

More complicated forms of program control are possible using `callcc` (sec 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, ie, 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 ...

\ldots\ 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}}`.

5.2 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.

5.3 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 (ie, 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.[10]

The footnote above (below?) was generated by

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

5.4 Bibliographies

TeX2page can use the external program BibTeX [`latex`, `bibtex`] 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",
```

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

```
  title = "{The TeXbook}",
  publisher = "Addison-Wesley",
  year = 1993
)
```

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

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

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*, ie, a sorted list of all the cited entries. Eg,

```
...

\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 creates an auxiliary file *jobname*`--h.aux`. A subsequent run of TeX2page calls BibTeX on *jobname*`--h.aux` to produce the corresponding sorted bibliography in the file *jobname*`--h.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*`--h.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.[11] Use the `thebibliography` environment to enclose your bibliographic entries, and introduce each entry with `\bibitem`. For more details, see the LaTeX manual [`latex`, sec 4.3.2, p 71], or see a sample `.bbl` file generated by BibTeX and imitate.

5.5 Index generation

[11] This approach, while tedious and a maintenance millstone, *can* be rational sometimes: Eg, 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.

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

This means that an occurrence of \index{*item*} in the TeX source causes *item* to be entered into an unsorted index file, *jobname*--h.idx. A subsequent run of TeX2page calls MakeIndex on *jobname*--h.idx to produce the sorted index in *jobname*--h.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*--h 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 more 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. Eg, you may want to have a different section header or include some introductory prose.

If TeX2page cannot find the sorted index file (*jobname*--h.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).

6 Images

Some portions of your TeX source may be particularly resistant to conversion to HTML. Examples are mathematics and the various approaches to typesetting pictures or diagrams. In such instances, TeX2page invokes a combination of TeX, Dvips [dvips], Ghostscript [gs] and the NetPBM library [netpbm, netpbm4redhat, netpbm4windows] 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[12] or JPEG using the \htmlimgformat command, eg,

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

Any TeX fragment enclosed between the control sequences \htmlimg ... \endhtmlimg is converted into an image. Some TeX fragments are automatically converted to images without the need for an explicit \htmlimg. Such fragments are

[12] 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.

mathematics, calls to the LaTeX `picture` environment, and MFpic [mfpic] diagrams.[13]

6.1 Mathematics

Math is typically text between `$...$` (in-text math) and `$$...$$` (displayed math). Here are some samples of mathematics with TeX2page:

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

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

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

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. Eg,

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

If the mathematical notation in your text is simple enough not to need images, it is advisable to use `\dontuseimgforhtmlmath` to force math to be set as text rather than images.

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`

[13] MFpic diagrams are enclosed between `\mfpic ... \endmfpic`. The MFpic macros translate picture specifications into METAFONT [metafont] or MetaPost [metapost] programs. Both METAFONT and MetaPost are included in modern TeX distributions, which makes MFpic a very attractive option for picture-drawing in TeX. Please see the MFpic distribution [mfpic] for more details.

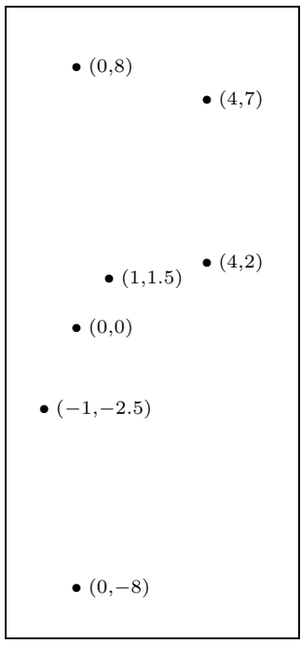
1986 = mcmlxxxvi). `\romannumeral` produces lower-case letters — `tex2page.tex` includes `\Romannumeral`, whose result is all-upper-case (`\Romannumeral 1986` = MCMLXXXVI).

6.2 Other image inserts

You may explicitly request any part at all of your TeX document — not just its math — to be converted into images for your HTML output. The fragment of the document to be converted to image is enclosed between `\htmlimg` and `\endhtmlimg`. This is particularly useful for rendering pictures in HTML. Here's an example TeX-based diagram from *The TeXbook* [tex, p 389]:

```
\htmlimg
\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}}
\endhtmlimg
```

This produces the image:



6.3 Image preamble

For any but the most garden-variety mathematics and pictures, you probably have a suite of your own TeX control-sequence definitions (or TeX macro files). You can inform TeX2page of their presence by announcing them as your "image preamble". Eg,

```
\imgpreamble
  \input some-pic-macs
  \let\g0\Omega
  \def\I#1#2{\int_{#1}^#2}
\endimgpreamble
```

You can afterward use the control sequences `\g0`, `\I`, and those in `some-pic-macs.tex` in your mathematical or pictorial text.

It is common to use a drawing tool such as XFig to produce TeX subfiles that typeset as pictures. Let the macros used by these TeX picture files be defined in `pepic.tex`. Our TeX source can then be:

```
\input pepic

\imgpreamble
  \input pepic
\endimgpreamble
...
\htmlimg
  \centerline{\input picture-file-1 }
\endhtmlimg
...
```

```
\htmlimg
  \centerline{\input picture-file-2 }
\endhtmlimg
...
```

where `picture-file-1.tex`, `picture-file-2.tex`, ..., are the picture files. The first `\input pepic` is for TeX when it processes the entire document. The second `\input pepic`, the one inside the image preamble, is used by TeX2page when it processes just the image fragments of the source file through TeX.

6.4 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. Eg,

```
\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* [`tex`].

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, but there must be a better way:

```
\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.)

6.5 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 resitors, 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{$\Omega$}
```

6.6 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.

7 Style sheets

You can have your HTML output use *style sheets* [`css`]. 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. Eg,

```
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.

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`. Eg,

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

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

The TeX2page distribution includes the file `t2p-example/css.t2p`, which contains a sample `\cssblock` that focuses on style rules for the TeX2page-specific

classes. It may be modified, or combined with other more general styles such as the *W3C Core Styles* [w3ccorestyles] (which can be included with \inputcss).

8 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 [tex, 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 HTML output pages will be valid. Nevertheless, these directives are often useful, as the following examples show.

8.1 Example uses of TeX2page directives

Many TeX macros, while crucial for printed copy, are irrelevant to HTML, eg, the statement `\parindent=10pt`. Such macros are best enclosed in a `\texonly`, to avoid erroneous or fruitless translation by TeX2page.[14]

The `\evalsto` command we saw above (sec 4.1) was rather shabby. A better alternative would be to exploit TeX's math symbols, ie,

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

or, better still (sec 6.5),

```
\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", eg,

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

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

[14] 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`.

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

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.

9 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. Note that by typesetter we mean both TeX (which produces DVI output) and TeX2page (which produces HTML output).

Since there are two outputs to distinguish, we cannot get by with just one standard output port in the Scheme code introduced by `\eval`. So, we will use the regular standard output port for `\eval`'s TeX output (which eventually becomes DVI), and the port `*html*` for `\eval`'s TeX2page output (which eventually becomes part of the final HTML). We can use this feature to play to a medium-selected gallery.

```
\eval{
(display "Paper beats screen.")
(newline)

(display "Screen beats paper." *html*)
(newline *html*)
}
```

`\eval` allows the document writer access to full Scheme, including the Scheme definitions inside the TeX2page implementation. This serves as a very powerful *second extension language*. The *first* extension language is the TeX macro language, or rather, TeX2page's implementation of a subset of the TeX macro language.

Note that the extensions possible through `\eval` and Scheme are *not* restricted to the HTML output only. The print output is equally maneuverable via Scheme. Indeed, we can use `\eval` to introduce new functionality that is used consistently in both the browsable and printable outputs.

But first we will 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 Lisp operator by giving a *prototypical* call. 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

(Ie, 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 TeX2page's extension language, 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 " " *html*)))
}}

\endhtmlonly
```

`\eval`'s argument is a balanced-brace expression that can contain any character at all, including `%`. (If you need to include an unmatched brace in `\eval`'s argument, simply put a bogus matching brace inside a Scheme comment.)

10 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`.

This diagnostic information 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 command

```
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--h.aux` and receives information from BibTeX in the file `story--h.bbl`.

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

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` create the auxiliary Scheme files `story-Z-E-1.scm`, etc, which 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. The `\eval`'s that occur inside the `\htmlonly` portions of `story.tex` have slightly different names and are numbered separately: `story-Z-E-H-1.scm`, etc, with their corresponding `story-Z-E-H-1.tex`, etc. This is so that the `\eval`'s in the *non*-`\htmlonly` portions can be shared by TeX2page and TeX, without the `\eval`'s from the `\htmlonly` portions throwing the numbering off.

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, ie, 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

C Concept index

D Downloading and installing TeX2page

Go to the TeX2page website (`http://www.ccs.neu.edu/~dorai/tex2page/tex2page-doc.html`) to download the TeX2page distribution. (The download link immediately follows the title.)

TeX2page is distributed as a `gzip`ped `tar` file, `tex2page.tar.gz`. Unpacking it produces a directory called `tex2page`, which contains, among other files, the Scheme file `tex2page`, the plain TeX file `tex2page.tex`, and the LaTeX style file `tex2page.sty`.

Put copies of (or links to) the files `tex2page.tex` and `tex2page.sty` in a directory that is mentioned in your `TEXINPUTS` environment variable.

D.1 TeX2page out of the box

For your convenience, some pre-configured versions of TeX2page are included in the distribution.

If you run MzScheme on Unix, setup is minimal. Simply put a copy of (or link to) the Scheme file `tex2page` in a directory in your `PATH` environment variable.

If you run MzScheme on Windows, copy the supplied batchfile `tex2page.bat` to your `PATH`, and edit its contents so it contains the correct pathnames to your MzScheme executable and `tex2page` file.

If you run Common Lisp on Unix, copy the CL file `tex2page.lsp` to your `PATH`.

If you run Common Lisp on Windows, copy the batch file `tex2page-lsp.bat` (you can rename it to `tex2page.bat`) to your `PATH`, and edit its contents.

D.2 Using `scmxlate` to configure TeX2page

In general (including the setups listed above), you can use the `scmxlate` program to configure TeX2page for your system. TeX2page is known to run on the Scheme dialects MzScheme [`mzscheme`], Bigloo [`bigloo`], Gambit [`gambit`], Guile [`guile`], MIT Scheme [`mitscheme`], Petite Chez Scheme [`petite`], Pocket Scheme [`pocketscheme`], Scheme 48 [`scheme48`], SCM [`scm`], Scsh [`scsh`], STk [`stk`] and SXM [`sxm`]; on Common Lisp (CLISP [`clisp`]); on Unix and Windows (including Cygwin [`cygwin`]).

Make sure you have the scmxlate (`http://www.ccs.neu.edu/~dorai/scmxlate/scmxlate.html`) program installed on your system.

Edit the file `scmxlate-tex2page`. You can leave it empty. Optional insertions are:

```
(scmxlate-compile? #t)
```

if you'd like a compiled version of `tex2page`.

```
(define *ghostscript* "pathname-of-your-ghostscript-program")
```

TeX2page will guess a pathname for the Ghostscript executable, but it's possible it guesses wrong on Windows. You can explicitly supply the correct pathname here.

Start your Scheme (or Common Lisp) in the `tex2page` directory. Load the file `scmxlate.scm` from the `scmxlate` distribution, using the correct relative or full pathname of `scmxlate.scm`. For example,

```
(load "/home/dorai/share/scmxlate/scmxlate.scm")
```

(assuming you unpacked `scmxlate` in `/home/dorai/share`). You will be asked a couple of questions about your setup. A choice of answers will be provided, so you don't need to be too creative. When `scmxlate` finishes, you will be left with the a version of `tex2page` tailormade for you. If you're using a Scheme, this file is called `my-tex2page` (so as not to clash with the original `tex2page` supplied in the distribution). If Common Lisp, it is called `tex2page.lsp`.

In Unix, put a copy of (or link to) `my-tex2page` (or `tex2page.lsp`) in a directory in your `PATH`. You may wish to rename it to `tex2page`.

In Windows, a batch file called `tex2page.bat` is also created. Move it to a directory in your `PATH`. Edit the contents of `tex2page.bat` so that the pathnames it refers to are correct.

D.3 Can't create or don't want a `tex2page` script?

If the configuration process cannot create an appropriate script file for you to put in your `PATH`, or if you prefer working within Scheme or Common Lisp anyway instead of at the OS command-line, you can still use TeX2page. Simply load your configured `tex2page` Scheme/CL file (ie, `my-tex2page` or `tex2page.lsp`) directly into your Scheme (or CL), and then call the Scheme (CL) procedure `tex2page` on your source document. Eg,

```
(load "my-tex2page")
(tex2page "tex2page-doc.tex")
```