

# TeX to HTML Translation via Tagged DVI Files

Michael D. Sofka

Computing Information Services  
Rensselaer Polytechnic Institute

`sofkam@rpi.edu`

**Slide 1**

`http://www.rpi.edu/~sofkam/`

TeXNortheast TUG Conference  
March 22–24, 1998, New York City

**Notes for slide 1:**

Introduce yourself, affiliation, etc.

**Problem:** HTML output from T<sub>E</sub>X.

**Solution:** T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X to HTML translator.

**Problem:** Translators cannot understanding T<sub>E</sub>X's macro language; easy to fool.

**Solution:** Restrict the input to L<sup>A</sup>T<sub>E</sub>X.

**Alternative:** Use T<sub>E</sub>X to parse T<sub>E</sub>X, guaranteed to work.

**Slide 2** **Problem:** T<sub>E</sub>X's output is a low-level DVI file in which much of the document structure is lost.

**Solution:** Use `\special` and macros to tag DVI file with high-level information.

**Notes for slide 2:**

How many people have converted a T<sub>E</sub>X document into HTML?

- HTML output is a common request.
- T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X translators have been written.
- But, most translators cannot understanding T<sub>E</sub>X's macro language, and are easily fooled. T<sub>E</sub>X is not easy to parse: category codes, no keywords, macros redefine the syntax.
- So, most translators accept a restricted input, usually L<sup>A</sup>T<sub>E</sub>X.
- An alternative is to use T<sub>E</sub>X to parse T<sub>E</sub>X, guaranteed to work.
- But, T<sub>E</sub>X's output is a low-level DVI file in which most of the high-level document structure is lost.
- This talk is about using `\specials` and some macro to tag DVI files with higher-order information.

- What is a DVI file?
- How are DVI files tagged?
- Beyond tagging: Smart(er) DVI Viewers
- Two-View Editing

A DVI File:

- Binary, 1-byte op-codes
- Slide 3**
- Fonts, characters, rules and movement
  - `\special's`

**Notes for slide 3:**

This talk has three parts.

- A brief description of the DVI-file—just enough to get started, there is more detail in the paper and references.
- Explain how DVI files can be tagged, concentrating on specials and macros to make tagging transparent to the author.
- Discuss how tagging can allow Smart(er) DVI viewers and drivers, including drivers that can page documents, do layout and even adjust paragraph parameters.
- The inspiration and long-term goal is Two-View editing.

This gives me about 4 minutes per section.

What are DVI files:

- A DVI file is a binary file, difficult to view with an editor.
- op-codes and their parameters.
- The op-codes fonts, characters, rules and movement,
- and `\specials`

	<i>Category</i>	<i>op-codes</i>
	Print Character	<i>set_char0...set_char127</i> , <i>set1, set2, set3, set4</i> , <i>put1, put2, put3, put4</i>
	Select Font	<i>fnt_num0...fnt_num63</i> , <i>fnt1, fnt2, fnt3, fnt4</i>
	Define Font	<i>fnt_def1...fnt_def4</i>
	Print rule	<i>set_rule, put_rule</i>
<b>Slide 4</b>	Horizontal Movement	<i>right1...right4, w0</i> , <i>w1...w4, x0, x1...x4</i>
	Vertical Movement	<i>down1...down4, y0</i> , <i>y1...y4, z0, z1...z4</i>
	Header	<i>pre, post, post-post</i>
	Page	<i>bop, eop</i>
	Stack	<i>push, pop</i>
	Special	<i>xxx1, xxx4</i>
	Undefined/nop	<i>nop, 250-255</i>

**Notes for slide 4:**

The complete list of opcodes is:

1. 136 of the 250 DVI op-codes are for printing characters (128 for the characters 0...127 on a font chart), and
2. 68 are used to select a font.  
This is to optimize the output, most characters and font changes require a single op-code.
3. There are 14 horizontal and 14 vertical movement commands, including the *w* and *x* horizontal register commands, and the *y* and *z* vertical register commands.
4. Many commands have four versions, depending on if the parameter is 1, 2, 3, or 4 bytes long— $\text{\TeX}$  tends to optimize the selection.
5. *push/pop* are very important: they saves and restores position and register values, and correspond to boxes in the  $\text{\TeX}$  file.
6. *xxx1-4* are the macro expanded contents of  $\backslash\text{special}$  control sequence.

**Example DVI file:**

```
push
  right3(786432)font_num0
  set_char71 set_char101 set_char110 right2(-78643)
  set_char116 set_char108 set_char101 w3(334233)
  set_char114 set_char101 set_char97 set_char100
  set_char101 set_char114 set_char115 w0 ...
```

```
pop
y3(786432)
```

**Slide 5**

```
push
  set_char105 set_char109 set_char116 ...
```

```
pop
y0
push...
```

**Notes for slide 5:**

ASCII translated, the section of a DVI file for typesetting the word “Gentle” would look like this:

- All distances in a DVI file are in scaled-points, which are 1/65536th of a point. This is the smallest unit  $\text{\TeX}$  can move.
- Each line is nested in a *push/pop* pair.
- Within this pair the *w* register is used for inter-word spacing,
- A *right* (or the *x*) register is used for kerning.
- Each line is separated by a *y* register command.
- Paragraphs are usually separated by a *z* register command if  $\backslash\text{parskip}$  is non-zero.

That’s a DVI file, simple.

The question is, how can we discover that, for example, this paragraph is a quotation, or part of an enumerated list, or maybe a one-head?

```

\spaceskip=1sp
\xspaceskip=1sp
\hsize=\maxdimen

\baselineskip=1sp
\lineskip=0pt
\lineskiplimit=-16383pt

\parskip=0pt

```

**Slide 6** *push*

```

fnt_num0
set_char71 set_char101 set_char110 right2<78643>
set_char116 set_char108 set_char101 w1<1>
set_char114 set_char101 set_char97 set_char100
set_char101 set_char114 set_char115 w0
...
pop
y1<1>
push set_char105 set_char109 set_char116... pop

```

**Notes for slide 6:**

There are at least three ways:

1. We set T<sub>E</sub>X's parameters so that specific known movement amounts, usually in scaled points, correspond to particular elements.

For example: By setting the parameters as show in slide 6, all paragraphs take exactly one line, and are separated by vertical 1 scaled point, and all words are separated by one horizontal scaled point.

There are many variations on this which can, for example, use `\leftskip` tag lines in paragraphs, quotations, enumerated lists, etc.

2. We can identify font face, or face at a specific size as belonging to some elements.

For example: all of the three heads may be Optima at 10pt. If we needed to distinguish between one heads and, for example, table heads which also used Optima at 10pt, the table heads can be set at 14pt and 1sp. This small difference between Optima at 655360 sp and Optima at 655361 sp will make no difference in the typeset output, but will be a different font in the DVI file.

3. Or, we can `\specials` to insert markup tags in the DVI file.

Scale-point movements, and Fonts are useful, and I've had good results using them to convert DVI into Quark Express tagged files. But, `\specials` are the real powerhouse of T<sub>E</sub>X extensions.

Specials:

- Macro expanded contents of `\special`
- `TEX` knows nothing about the contents
- Can occur out of order
- The `xxx1 . . . xxx4` op-codes.

### Slide 7

#### Notes for slide 7:

What are `\specials`?

- Specials are the macro expanded contents of `\special` primitive.
- `TEX` knows nothing about the contents of the `\special`—it is up to the macro writer/user to see to it that the `\specials` are semantically correct.
- Because of `TEX`'s wonderful asynchronous output routine, `\specials` can occur out of order in the DVI file. It is up to the translator writer and macro writer to fix this.
- The `xxx1 . . . xxx4` op-codes. `TEX` uses `xxx1` for specials under 256 characters in length, `xxx4` otherwise.

Delimited Tags:

```
\catcode'\@=11
```

```
\let\t@gsection=\section
```

```
\def\section#1{%
```

```
  \special{::tag begin(section)}%
```

```
  \t@gsection{#1}%
```

```
  \special{::tag end(section)}}
```

```
\catcode'\@=12
```

**Slide 8**

```
xxx1\langle 20\rangle::tag begin(section)
```

```
[contents of \section]
```

```
xxx1\langle 18\rangle::tag end(section)
```

**Notes for slide 8:**

Here's an example of tagging a `\section` head using `\special`'s.

- The old definition of `\section` is saved using `\let`.
- The new definition is the old `\section` macro, with tags inserted around it.
- This is called a delimited tag, because:
  1. One special starts the tagging, and
  2. A second special ends the tagging.

Block Scoped Tags:

`$$X\over Y$$`

```
$$\special{::tag begin(numerator)}  
  X  
  \special{::tag end(denominator)}  
\over  
  \special{::tag begin(denominator)}  
  Y  
  \special{::tag end(denominator)}$$
```

Slide 9

**Notes for slide 9:**

This will not work in all cases. For example, consider the simple equation `$$X\over Y. $$`

It is too much to ask authors to type in the `\special` around the numerator and denominator—even with suitable macros. And, this is a simple equation.

However, consider the following macro (put up 10).

```

\def\tag#1{\special{::tag block(#1)}}

\catcode'\@=11
\let\t@gover=\over
\def\over{\tag{num}\t@gover\tag{den}}
\catcode'\@=12


$$X\over Y$$


```

**Slide 10**

```

push
  set_char88
  xxx1\langle 16\rangle::tag block(num)
pop
right4\langle n\rangle down3\langle m\rangle
putrule\langle a\rangle\langle b\rangle down3\langle m\rangle
push
  xxx1\langle 16\rangle::tag block(den)
  set_char89
pop

```

**Notes for slide 10:**

This macro:

- Redefines `\over` to place a tag:
  1. At the end of the numerator, and
  2. at the beginning of the denominator.
- This is called a “block” special, because it affects the *push/pop* block in which it appears.

The output of this macro in the DVI file is:

- *push*, set the character “X”, the numerator tag, and *pop*.
- Some more DVI commands to move and set a rule.
- *push*, the denominator tag, the character “Y” and *pop*.

The article in the proceedings shows a similar method for tagging the contents of aligned equations, and `\halign`.

It also discusses some of the issues of resolving scope when delimited and block specials interact, and when the output routine is invoked in the middle of a specials scope.

### Smart DVI Viewers:

```
\nopagenumbers
\def\s#1{\special{before #1}}
\def\s#1{\special{after #1}}

\gdef\numberlines{\special{%
    line: \jobname:\number\inputlineno}}

Slide 11
{\catcode'\^M=\active%
 \gdef\startnumbering{\catcode'\^M\active%
 \let^M=\numberlines}%
 \global\let^M=\numberlines} %

\startnumbering
```

### Notes for slide 11:

What else can we do with tagged DVI files?

How about this macro which (imperfectly) inserts the  $\TeX$  input file and line number into the DVI file. With this information, it is possible for a DVI viewer to identify the  $\TeX$  code which produced the DVI.

This information could be used to do simple layout editing—cut and past the DVI file—while inserting the necessary page and column breaks back in the original  $\TeX$  code. (This requires some macro cooperation, but not much).

**Slide 12**

- `dvitag` (sofka, 1993) and Elsevier’s SGML converter (Rahtz, 1995) tagging to guide DVI conversion.
- `TEX4ht` (Gurari, 1997a, 1997b) tagging, *push/pop* for Hypertext.
- `VorTEX` (Harrison, 1989): Incremental compilation of `TEX`.
- `Lightning TEXtures` (Hampson & Smith, 1992): Fast `TEX` interpreter—reprocesses input file with each keystroke.
- `Type & Set` (Asher, 1992): Uses `\special` and DVI processor for optimal and automatic pagination.
- `Lilac` (Brooks, 1988, 1991): Two-view editor using `box&glue` language, similar to but not `TEX`.

**Notes for slide 12:**

How far can this approach be applied?

- `dvitag` and Elsevier’s SGML converter both use tagging to guide conversion.
- `TEX4ht` uses tagging and *push/pop* structure to convert `TEX` into hypertext.
- `Vortex`: Incrementally compiled `TEX` code. Not very efficient, used page-by-page decomposition of the `TEX` input and DVI files.
- `Lightning TEXtures`: Processes the `TEX` document from the beginning. More recent version does some optimization.
- `Type & Set`: A package that should have gotten more attention. It uses specials to mark page-able elements—identifies the line breaks in the DVI file using *push/pop* structure—and a DVI processor to optimally and automatically page complex documents.
- `Lilac`: Brooks two-view editor—changes in the source are reflected in the viewer and visa-versa.

Books didn’t use `TEX` because of `TEX`’s dynamic scoping. That is, it is very difficult in the middle of a `TEX` file to know what effect a change may have. It would be nice if the `LATEX3` team addressed this. It would be helpful if all layout information were available from the environment structure—some macros should be inaccessible to the author.