Don also showed a simple way to find out what TEX is doing:

```
RUN TeX
\relax
\showhyphens{...}
```

As many words as desired can be entered at once; all hyphenation points will be shown for each word. Actually, TEX finds two hyphenation points are found in `Di-jk-stra`; one of them is right.

* * * * * * * * * * *

TEX VS. INITEX

David Fuchs

This note deals with TEX vs. INITEX, `\dump`, `FMT` files, and how to make a 'pre-loaded' TEX. Those who have already seen this information on the TEXhax computer bulletin board may be interested to read the last paragraph, which is new.

First, a review. In the best of all possible worlds, there would only have to be a single version of TEX. Every TEX file would start with a line that `\input` a big package of lots of macros and fonts. In the real world, reading many macros each time you run TEX can become fairly tedious because it is not instantaneous. Reading in fonts is even slower, since most systems take their time in opening files, and each font mentioned requires TEX to access a separate `TFM` file. To help speed things up, we introduced the `FMT` file. INITEX is able to read in macros and fonts (in fact it can do everything TEX can) and then it can dump its entire internal state out into a single file when it is given the `\dump` command. Such a file is called a `FMT`, or 'format' file. Thus, if we:

```
RUN INITEX
\input plain
\dump
```

INITEX will exit and leave us the file `PLAIN.FMT`. We can then run TEX (or INITEX, for that matter) and say:

```
RUN TEX
&plain
```

and we'll be in exactly the same state we were in before we did the `\dump` above. Thus, we can continue by saying `\input myfile`, and TEX will behave as if we had `\input plain` and then `\input myfile`. We have already won, because it is faster to read in `PLAIN.FMT` than it is for TEX to process `PLAIN.TeX` from scratch.

In fact, it was not necessary actually to say `&plain` to regular TEX, because it automatically reads in `PLAIN.FMT`, unless you override it by saying `&fancy` or something. (Of course, INITEX does not automatically read in `PLAIN.FMT` if we don't ask for another `FMT` file; otherwise there would be no way to create `PLAIN.FMT` in the first place!) If you have another set of macros, say `THESIS.TEX`, you could:

```
RUN INITEX
\input thesis
\dump
```

to make `THESIS.FMT`, and then users of that macro package would be able to get their jobs going quickly by saying:

```
RUN TEX
&thesis mythesis
```

(`\input` is assumed even after the `&thesis`.)

On some systems, this is as far as we can go: instead of reading lots of files and doing lots of processing to get TEX set up to read the user's input, TEX need only read one `FMT` file each run, and do a little processing. But `FMT` files are pretty big; maybe there is an even better way to go. Well, consider a system on which the user is able to halt a program in mid-execution, and save away the entire thing to be continued at a later date. If TEX with no `FMT` file built-in is called 'VIRTEX', then we can build a TEX with `PLAIN.FMT` built-in by:

```
RUN VIRTEX
&plain
(Wait for TeX's * prompt, and then HALT it)
SAVE TEX
```

If a TEX wizard does this and puts the results on a common system area, then when users give the `TeX` command they will actually get a copy of TEX with `PLAIN` already set up. If it only were that simple everywhere! There are a few things that need to be considered before you attempt to do this sort of thing on your system. We have to delve deeper into the implementation ...

When the user runs this saved TEX, is it started up from the beginning or is it continued from where it left off? In all the systems I've seen that have this sort of capability, the program is actually restarted from the beginning, but the global data areas are not re-initialized. Well, that's ideal for TEX, because of TEX's use of a magic global variable called *ready_already*. When TEX starts up, it takes a look at *ready_already* (without first assigning a value to it). If the value is not 314159, then TEX figures that there is not a `FMT` file built-in, so it goes ahead and initializes itself, and reads in `PLAIN.FMT` (unless the user asks for a different `FMT` file). Then it sets *ready_already* to 314159, in case the user decides to halt execution and save the core image. If the core image were then to be restarted, *ready_already* would still be 314159, and TEX would realize that

it already had a FMT file loaded in, so it would skip over the whole initialization process.

Great. But there are problems. On some systems, the runtimes are completely oblivious to being interrupted and restarted. On these systems, the scheme described above works fine. Other systems, however, require the program to have come to a graceful exit before being saved; otherwise when it is restarted, everything will run amok. (Typically, the runtime might become confused by the fact that the log file was not closed when the core image was saved.) On such systems, you'd think you could build a TEX with PLAIN.FMT built-in by:

```
RUN VIRTEX
&plain
\end
SAVE TEX
```

because the \end will cause TEX to complete its execution by 'falling out the bottom.' The only problem is that, just before it does this, TEX sets *ready_already* to 0. The idea behind this is that if you are running on a system in which global variables are initially random, and you get put in the same spot in real memory as an old TEX job, then your TEX might get fooled into thinking that it has a FMT built-in (since the prior TEX job left *ready_already* at 314159), when all it really has is the left-over internal state of someone else's TEX job. If the last TEX job cleared *ready_already* to zero, then you have less chance of being fooled, so this is what we have TEX do by default. On the other hand, the previous job might have gotten interrupted in the middle, and you'd be fooled anyway, so on this kind of system you're best off using another technique for checking built-in-ness. Anyway, the point is that you should remove the final *ready_already* := 0 if you have the kind of system that requires you to say \end to get all the files closed and other things cleaned up before saving a TEX + (FMT file).

The astute reader may be asking "That still doesn't explain why there needs to be INITEX as well as VIRTEX. Why aren't things set up so that we can just

```
RUN VIRTEX
\input plain
HALT
SAVE TEX  ?"
```

The answer is that VIRTEX is missing one or two capabilities that INITEX has. If you take a look at the WEB-language source for TEX, you'll see that there are two macros, named INIT and TINI, that delimit various sections of code. The idea is that if you Tangle TeX.WEB with INIT and TINI defined as null, then you'll get an INITEX when you compile the resulting Pascal program. If INIT is defined to evaluate to '{', and TINI '}', then when you Tangle you'll get almost the same Pascal program, but with portions commented out. Compiling this program gives VIRTEX.

Well, what are the sections that are commented out, and why? The code in question consists of the modules that initialize TEX's hash tables, read in the POOL file, and process hyphenation \patterns. We can get away with this because the hash table, string pool, and hyphenation trie all get dumped to the FMT file, so non-INITEXs need not be able to recreate them from scratch. This is strictly an efficiency move, since there is no reason to bog regular TEX down with the extra code and data space necessary to process these data structures. Only the missing \patterns primitive can be detected by the ordinary user, but only foreign language hyphenation hackers should be messing with that.

The other code not in VIRTEX is the stuff that handles \dump itself. Of course, if you have an infinitely fast machine with no address-space limitations, and if extra memory is free, then a simple change to TEX would let it load \patterns and do \dumps, and INITEX could be thrown away. Actually, so could FMT files, and the \dump feature.

Early tests on IBM VM/CMS systems seem to indicate that if the FMT files are blocked up into very large records, the system is able to process them quite quickly, so it is not necessary to do pre-loading. It turns out to be convenient to have seperate INITEX and VIRTEX executable programs on this system, however. The advantage is not only that VIRTEX is somewhat smaller due to the missing INIT code, but we also turn on the DEBUG-GUBED and STAT-TATS switches in INITEX, and compile it with the optimizer turned off and the full run-time checks turned on. Thus, INITEX is a relatively large, slow TEX, but it is good not only for creating new FMT files, but also to help check out any suspected bugs in TEX, as well as help in heavy-duty macro debugging.

* * * * * * * * * * *

TEXHAX SUMMARY

David Fuchs

There were a few typos in the TEXhax summary in the last issue of TUGboat. Page 6, second column, last paragraph should say **0**, not **0/**. Similarly, page 7, line 7 should say **0** instead of just **0**. Finally, the program example in the right column should have 8em removed in two places. This

might be a good place to mention that there is one more new feature in WEB: due to popular demand, the format of change files has changed. The new system lets you change just the middle of a module, and also warns you if the code you're changing has changed itself. Here's how it works: The change file has any number of changes of the form

⟨Comment lines⟩
@x
⟨Old lines⟩
@y
⟨New lines⟩
@z

When Tangle or Weave sees a line that matches the first "⟨Old line⟩", it checks that all the "⟨Old lines⟩" match the main web file. If not, an error message is given. In either case, the "⟨Old lines⟩" are discarded, and the "⟨New lines⟩" take their place. Comments can be given on the **@x**, **@y**, and **@z** lines, and between changes. All **@x**, **@y**, and **@z**'s must be at the beginning of a line.

To help you bootstrap the new Tangle and Weave, it may help to know that the code which has changed was in non-system-dependent modules, so your old change files should (almost) work with the new web files when run on the old Tangle. So you should be able to create a working new Tangle easily, using your previous change file; and you can test it out by making a new-format change file for Tangle and seeing if the new Tangle can recreate itself.

The 'almost' is because you may have to change `\count0` to `\pageno`, to conform to the new TEX. You may also have to change **@**'s in TEX files, and lots of **@@**'s in WEB files, to ~'s, since tilde is now the tie character. Also, take a look at the note in `TeX82.DIF` about the big `\set` and `\the` change introduced in version 0.98 to see if that affects you. Don't let all this scare you; there haven't been any problems reported on the systems that have already been done.

A few people have gotten into trouble by trying to alter Tangle to output all reserved words and identifiers in lowercase. Note that Tangle looks back in its output buffer for things like MOD and DIV to help it decide when it's OK to do constant folding. If you aren't careful, you'll end up with a Tangle that outputs incorrect programs. The red WEB manual has a Tangle listing with an index entry 'uppercase' that points to all the relevant modules.

Most of our WEB programs now include a 'history' feature that keeps track of how the run went. Generally, the results are either 'good', 'warning issued', 'error encountered', or 'fatal error'. The history variable is used to print a final status message as each program ends, and it can also be used to send a status code back to the operating system on those systems that support such things.

There are a few things to watch out for when installing the new TEX. The module "⟨Globals in the outer block⟩" has been renamed "⟨Global variables⟩", so you should check that your change file always refers to " ⟨Global ... ⟩" or " ⟨Glob ... ⟩". Also, all references to single-letter identifiers that used to look like "**x**" are now in the form "$|x|$". This may affect your change files slightly. Similarly, `$\ldots$` has been changed to `\dots` in many comments; a few files that used to use `\ifodd` now use `\ifeven` (which takes a number rather than a counter as a parameter); `\ifabsent` is now called `\ifvoid`. All instances of "**debug**", "**init**", and "**stat**" have **@!** in front of them, so that they will be indexed. Discretionary hyphens (`\-`) have been removed from words that TEX82 can hyphenate.

The module that defines the macros "*qi*" and "*qo*" now also defines "*hi*" and "*ho*", so if this module is in your change file, you'll have to alter it in the obvious way. Similarly, the module that defines "*set_glue_ratio_zero*" now includes two new macros, "*float*" and "*unfloat*", to aid in porting TEX to systems where "*glue_ratio*" can not be "*real*". The macro "*float_const*" has been added to point out the few places that use a floating point constant in the code.

All of our WEB programs that use the procedure "*input_ln*" now remove spaces at the end of lines. This means that all input files and macro packages will be impervious to being moved to and from fixed-line-length systems. Since most change files include an altered version of "*input_ln*", you'll probably have to change a little code here.

Some of the modules in TEX that changed in interesting ways (interesting for the TEX installer, anyway) can be found by searching for the strings "*wlog*", "*wterm*", "*name* <= 16", "*name* > 16" and "*read_open*" in `TeX.WEB`. Also note a change of variable name ("*n*" vs. "*m*") in "⟨Input the first line ... ⟩". The "*input_ln*" procedure now has a second parameter that tells whether it should do an initial "*get*"; this simplifies the reading of the first line of `\input` and `\read` files. It also should aid installation on systems with either 'lazy-lookahead' or 'interactive' terminal files.

A new macro called "*wake_up_terminal*" is now called each time TEX is about to issue an error message. On systems where the user might flush the output to the terminal (with a control-O, for instance), this macro marks all the good times to turn output back on. In order to help TEX do this, standard

error messages now say "*print_err*(""message..." instead of "*print_nl*(""! message..." The "*print_err*" takes care of printing the exclamation point and space, after it calls "*wake_up_terminal*". Note that "*wake_up_terminal*" is called from a few other places as well.

Other items from TEXhax: On Tops-20, the 'log file' spoken of in the TEXbook will have an extension of .LST, and on VAX/VMS, it will be .LIS, so as to avoid conflicting with .LOG files produced by batch jobs. The new extensions were chosen to match the systems' conventions about compiler listing file names (TEX is a compiler, after all).

Watch out for an unfortunate 'feature' of \everypar: If a paragraph begins in a group, as in ...\vskip 5pt {\it Horizontal mode... then the effects of the \everypar are local to that group (except for \globals, of course) even if the group ends before the paragraph does.

The VM/CMS version of TEX that is now available through our standard distribution channels represents the combined efforts of many people, who I won't list here for fear of leaving someone out. If you have an IBM system that you'd like to be able to re-compile TEX on, you should be aware of a bug in the Pascal/VS optimizer found by Craig Platt.

```
program foo;
var i: integer;

procedure changeit; forward;

procedure doit;
begin
i := 1;
changeit;
writeln('Should not be one: ',i:1);
end;

procedure changeit;
begin
i := 2;
writeln('Should be two: ',i:1);
end;

begin
doit;
end.
```

The Pascal/VS developers have a fix for this, but the procedure for obtaining it is totally obscure. You're on your own, as far as I can tell.

Here are the current contents of the "Red Alert" file mentioned above:

A few bugs were discovered in TEX 0.999. Details of the changes may be found at the end of the TeX82.BUG file. The TRIP files had to be modified slightly to accommodate these changes. There was also a change to the TEXbook to make it consistent with the fact that \input files no longer have a blank line appended automatically.

A bug in WEBHDR caused index entries to have double @'s where there were supposed to be single @'s. This error shows up in all the red manuals.

A bug in Weave caused any unchanged module followed by a module whose first line had changed to be incorrectly marked as changed. This is fixed in Weave 2.2 with a small change in the logic of "*get_line*".

A minor change to \finsm@sh in PLAIN.TEX.

And here's what TeX82.BUG has to say about changes made after the Version 0.999 red listing of TEX82 (these changes are shown in @x, @y, @z format for clarity; they are not meant to be used in an actual change file):

248. Module 1215, allow space in \read *n* to \cs (by FY, July 25, 1983)

```
@x patch in get_r_token routine
begin restart: get_token;
@y
begin restart: repeat get_token;
until cur_tok <> space_token;
@z
```

. . . . . .

249. Module 498, we must stack the current if type (FY, July 27)

```
@x patch in conditional routine
begin @<Push the condition stack@>;
        @+save_cond_ptr := cond_ptr;@/
@y
@!this_if: small_number; {type of this conditional}
begin @<Push the condition stack@>;
        @+save_cond_ptr := cond_ptr;this_if := cur_chr;@/
@z
```

Also replace *cur_if* by *this_if* in modules 501, 503, 506. The following patches do only what is necessary to make things work:

```
@x
    print_cmd_chr(if_test, cur_if);
@y
    print_cmd_chr(if_test, this_if);
@z

@x
if cur_if = if_int_code then scan_int
        @+else scan_normal_dimen;
@y
if this_if = if_int_code then scan_int
        @+else scan_normal_dimen;
@z

@x
if cur_if = if_char_code then b := (n = cur_chr)
        @+else b := (m = cur_cmd);
@y
if this_if = if_char_code then b := (n = cur_chr)
        @+else b := (m = cur_cmd);
@z
```

250. Module 507, \ifx need not put a control sequence in hash table (July 29)

```
@x
get_token; n := cs_ptr; p := cur_cmd; q := cur_chr;
get_token; if cur_cmd <> p then b := false
@y
get_next; n := cs_ptr; p := cur_cmd; q := cur_chr;
get_next; if cur_cmd <> p then b := false
@z
```

. . . . . .

251. Module 86, message is lost (noticed by HWT, July 31)

```
@x
print("..."); print_ln; return;
@y
print("..."); print_ln; update_terminal; return;
@z
```

. . . . . .

252. Don't put empty at end of \input file! (Aug 1) [This simplifies the rules and the program, and also gets around a bug that occurred at the end of files with \endlinechar < 0.]

```
@x Module 362:
@ An empty line is inserted at the end of the file, if the
last line wasn't already empty, because |input_ln|
sets |last := first| when it discovers an |eof|.
@^empty line at end of file@>

@<Read next line of file into |buffer|, or
    |goto restart| if the file has ended@>=
begin incr(line); first := start;
if not force_eof then
    begin if input_ln(cur_file, true) then {not end of file}
        firm_up_the_line {this sets |limit|}
    else if limit <> start then firm_up_the_line
        {if |pausing|, the user can add more lines}
    else force_eof := true;
@y
@ @<Read next line of file into |buffer|, or
    |goto restart| if the file has ended@>=
begin incr(line); first := start;
if not force_eof then
    begin if input_ln(cur_file, true) then {not end of file}
        firm_up_the_line {this sets |limit|}
    else force_eof := true;
@z
```

*** The changes above went into Version 0.9999, which was widely distributed.

. . . . . .

253. Ridiculous blunder made in change 146 (found by FY, August 16)

```
@x Correction to module 497
else loop@+begin q := cond_ptr;
    if link(q) = p then
        begin type(p) := l; return;
        end;
    if q = null then confusion("if");
@:this can't happen if}{\quad if@>
    q := link(q);
@y
else begin q := cond_ptr;
    loop@+ begin if q = null then confusion("if");
@:this can't happen if}{\quad if@>
        if link(q) = p then
            begin type(p) := l; return;
            end;
        q := link(q);
        end;
@z
```

. . . . . .

254. Minor amendment to stat(s) printing (cf. change 129) (August 16)

```
@x in module 1334
    wlog_ln(' ', str_ptr - init_str_ptr : 1,
        ' strings out of ',
        max_strings - init_str_ptr : 1);@/
@y
    wlog(' ', str_ptr - init_str_ptr : 1,' string');
    if str_ptr <> init_str_ptr + 1 then wlog('s');
    wlog_ln(' out of ',
@z
```

. . . . . .

255. Bug in \xleader computations (found by FY, August 18)

```
@x in module 592
@! lq,@! lr,@! lx : integer;
        {quantities used in calculations for leaders}
@y
@! lq,@! lr : integer;
        {quantities used in calculations for leaders}
@z

@x in module 626
    begin edge := cur_h + rule_wd;
    @<Let |cur_h| be the position of the first box, and
            set |leader_wd| to the spacing between
            corresponding parts of boxes@>;
    while cur_h + leader_wd <= edge do
        @<Output a leader box at |cur_h|,
            then advance |cur_h| by |leader_wd|@>;
@y
    begin edge := cur_h + rule_wd; lx := 0;
    @<Let |cur_h| be the position of the first box, and
            set |leader_wd + lx| to the spacing between
            corresponding parts of boxes@>;
    while cur_h + leader_wd <= edge do
        @<Output a leader box at |cur_h|, then
            advance |cur_h| by |leader_wd + lx|@>;
@z

@x in module 627
        leader_wd := leader_wd + lx;
@y
@z

@x in module 628
cur_h := save_h + leader_wd;
@y
cur_h := save_h + leader_wd + lx;
@z
```

```
@x in module 635
    begin edge := cur_v + rule_ht:
    @(Let |cur_v| be the position of the first box, and
            set |leader_ht| to the spacing between
            corresponding parts of boxes@);
    while cur_v + leader_ht <= edge do
        @(Output a leader box at |cur_v|, then
            advance |cur_v| by |leader_ht|@);
@y
    begin edge := cur_v + rule_ht; lx := 0;
    @(Let |cur_v| be the position of the first box, and
            set |leader_ht + lx| to the spacing between
            corresponding parts of boxes@);
    while cur_v + leader_ht <= edge do
        @(Output a leader box at |cur_v|, then
            advance |cur_v| by |leader_ht + lx|@);
@z
@x in module 636
        leader_ht := leader_ht + lx;
@y
@z
@x in module 637
cur_v := save_v-height(leader_box) + leader_ht;
@y
cur_v := save_v-height(leader_box) + leader_ht + lx;
@z
```

Also insert the following in modules 619 and 629:

```
@!lx: scaled; {extra space between leader boxes}
```

. . . . . .

256. \/ should apply to ligatures! (August 20)

```
@x in module 1113
var f: internal_font_number;
                {the font in the |char_node|}
begin if is_char_node(tail) and (tail <> head) then
    begin f := font(tail);
    tail_append(new_kern(char_italic(f)
                (char_info(f)(character(tail)))));
@y
label exit;
var p: pointer;
                {|char_node| at the tail of the current list}
@!f: internal_font_number; {the font in the |char_node|}
begin if tail <> head then
    begin if is_char_node(tail) then p := tail
    else if type(tail) = ligature_node then
                p := lig_char(tail)
    else return;
    f =: font(p);
    tail_append(new_kern(char_italic(f)
                (char_info(f)(character(p)))));
@z
@x later in that same module
end;
@y
exit: end;
@z
```

. . . . . .

258. Redundant code eliminated (August 27)

Module 531 needn't set and reset *name_in_progress* [but it's harmless].

. . . . . .

259. Bug: \input shouldn't occur during font size spec (Spivak; fixed August 27)

```
@x module 1258
@ @(Scan the font size specification@)=
@y
@ @(Scan the font size specification@)=
name_in_progress := true;
        {this keeps |cur_name| from being changed}
@z
@x module 1258
else s := -1000
@y
else s := -1000;
name_in_progress := false
@z
```

. . . . . .

261. Serious data structure error (found by Todd Allen, August 29)

```
@x module 478 (an error introduced in change 231)
    q := the_toks; link(p) := link(temp_head); p := q;
@y
    q := the_toks;
    if link(temp_head) <> null then
        begin link(p) := link(temp_head); p := q;
        end;
@z
```

. . . . . .

262. Minor patch for efficiency (August 29)

```
@x module 466
        begin store_new_token(info(r)); r := link(r);
@y
        begin fast_store_new_token(info(r)); r := link(r);
@z
```