## The WEB to CWEB conversion of TEX

Martin Ruckert

**Abstract**

This paper describes several aspects of the conversion of TEX's source code from WEB, based on Pascal, to CWEB based on C with web2w. It emphasizes those aspects that are relevant for obtaining a translation that can truly be regarded as source code and lends itself to modifications.

### 1 The advantages of CWEB

In the realm of programming environments, the C language enjoys very good support. Since the CWEB system of structured documentation [1] generates #line directives, this support extends also to programs written in CWEB. Figure 1 shows a debug session with the CWEB version of TEX where a breakpoint was set on the line_break function. After the run command, TEX asks for input and then stops in the CWEB file at the start of the line_break function. After a right click on "final_widow_penalty" in the source window, the debugger displays a menu that offers to display the variable in the data window. The traditional web2c translator expands WEB code to Pascal before translating to C. The debugger then shows fully expanded code in the source window as shown in figure 2.

Another advantage of the CWEB version of TEX is the simple tool chain. From a ctex.w source file, ctangle ctex.w produces ctex.c, and gcc -o ctex ctex.c produces the ctex executable; no additional conversion commands are needed. For experimenting with your own TEX, ctex.w can be modified — preferably with a change file — and the debugger has no difficulties switching the source window between ctex.w and the change file as needed.

### 2 From web2w version 0.4 to 1.0

In the last three years, the development of the HINT project would not have been possible without the CWEB version of TEX's source code produced with web2w version 0.4 [3]. During development, some shortcomings have become apparent, which I address in the new version 1.0 as described below.

#### 2.1 Creating a header file

Combining TEX's source code with C code residing in separate source files frequently requires access to TEX's functions and variables through "extern" declarations. Further, because of TEX's extensive use of macros, any substantial reuse of TEX's code also requires reusing TEX's macros. For this purpose, the new converter can optionally create a header

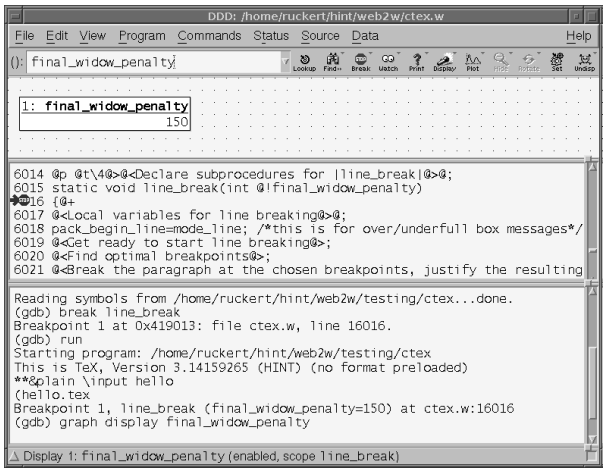


**Figure 1**: Debugging CWEB code with GNU ddd: true source code.

```
1030 void
1031 zlinebreak ( integer finalwidowpenalty )
32 {
1033   /* 30 31 32 33 34 35 22 */ linebreak_regmem
1034   boolean autobreaking  ;
1035   halfword prevp  ;
1036   halfword q, r, s, prevs  ;
1037   internalfontnumber f  ;
1038   smallnumber j  ;
1039   unsigned char c  ;
1040   packbeginline = curlist .mlfield ;
1041   mem [memtop - 3 ].hh .v.RH = mem [curlist .headfield ].hh .v.RH ;
1042   if ( ( curlist .tailfield >= himemmin ) )
1043   {
1044     mem [curlist .tailfield ].hh .v.RH = newpenalty ( 10000 ) ;
1045     curlist .tailfield = mem [curlist .tailfield ].hh .v.RH ;
1046   }
1047   else if ( mem [curlist .tailfield ].hh.b0 != 10 )
1048   {
1049     mem [curlist .tailfield ].hh .v.RH = newpenalty ( 10000 ) ;
1050     curlist .tailfield = mem [curlist .tailfield ].hh .v.RH ;
1051   }
```

**Figure 2**: Debugging web2c code with GNU ddd: fully-expanded code, difficult to read.

file containing TEX's macros, TEX's ⟨Constants in the outer block⟩, its ⟨Types in the outer block⟩, and finally a selection of "extern" declarations. The names of the selected extern variables and functions are taken from a text file given on the command line. As a side effect, all the remaining functions and global variables in TEX's sources are then declared "static".

As an extreme example, assume that you want to reuse TEX's x_over_n function: you can create a text file with a single line just containing the name x_over_n and use it to convert tex.web to ctex.w. Then run ctangle on ctex.w to get ctex.c and ctex.h. The file ctex.h will then end with the line "extern scaled x_over_n(scaled x,int n);".

Because all the other functions and variables of TEX are declared static, any decent C compiler can figure out that almost all of these are dead variables and dead functions. Running "gcc -O3 -c ctex.c -o ctex.o" will therefore produce a surprisingly small object file of just 1520 bytes. Analyzing the object file with GNU nm reveals:



doi.org/10.47397/tb/42-2/tb131ruckert-cweb

| Size | Type | Name |
|---|---|---|
| 0000000000000001 | b | `arith_error` |
| 0000000000000004 | b | `rem` |
| 0000000000000089 | T | `x_over_n` |

The object file contains two variables, one byte for the boolean `arith_error`, four bytes for the integer `rem`, and 89 bytes for the function `x_over_n`. All the rest has been removed by the compiler. You can link `ctex.o` to your `main` program without incurring any unnecessary overhead. If you want access to `arith_error` and `rem`, just add two lines with these names to your text file.

### 2.2 Eliminating the static string pool

Another spot that needed further attention is TEX's string pool. Strings enclosed in C-like double quotes receive special treatment by `tangle`: the strings are collected in a string pool file and replaced by string numbers in the Pascal source. In `ctangle` no such mechanism is available.

The previous converter replaced literal strings by section names, which were then expanded to an index in the `str_start` array. So

```
primitive("par",par_end,256);
```

became

```
primitive(<"par">, par_end, 256);
```

together with "`<"par">=444`" following in the appendix. With a suitable static initialization of the `str_start` and `str_pool` array, this had the desired effect and it was quite readable. For a limited subset of the literal strings — those used exclusively for printing — the converter took extra action to keep them in the code as literal C strings. This had the advantage that these strings were easier to change when modifying the generated `.w` files.

When implementing the HINT viewer, however, it turned out that TEX's entire string pool was still necessary for compilation; the code inherited from TEX still contained a few references to the string pool. This seemed unnecessary because the HINT viewer does not use TEX's control sequences or string handling functions. When implementing version 1.0, I started to further reduce the number of strings entering the string pool until only the names of control sequences remained in the string pool. With these names, the first argument of the `primitive` function remained a string number.

Defining new TEX primitives in change files is, however, common for extensions of TEX, and with `ctex.w` this had proved to be a bit cumbersome. You could not just write

```
primitive(<"newname">, ...);
```

but also needed to define the section name such that it would expand to a string number, which depended on the initialization of the `str_start` array, which in turn depended on the initialization of the `str_pool` array. A cumbersome and error-prone process.

So in the end, I decided to eliminate the static initialization of the string pool entirely. In version 1.0 the string pool is initialized at runtime with the first 256 single character strings and the empty string. All other strings are added during TEX's initialization, for example by calling `primitive("par", par_end, 256)`. The advantage is simplicity and readability; the disadvantage is the overhead in time and space because names of control sequences will now exist twice: the static string that is the argument of "`primitive`" and its copy in the string pool.

Here is some data on the incurred overhead to justify the decision: Version 0.4 already reduced the initial number of strings in the string pool from 1044 to 730 and the initial size of the string pool from 22742 bytes to 4700 bytes. Further reductions in version 1.0 left 611 strings with a total of 3701 bytes in the string pool — still without simplifying the addition of new primitive control sequences. The next logical step was abandoning the static initialization of the string pool altogether. Two alternatives came to my mind: removing the string pool completely or switching to a dynamic initialization of the string pool. I decided on the second alternative for the following reasons:

- Dynamic initialization adds overhead in time and space. The space overhead, however, is small (about 1% of the executable's size) and the time overhead is incurred only in the `INITEX` version of TEX.
- Dynamic initialization makes addition of new string literals as simple as possible.
- Dynamic initialization simplifies the implementation of `web2w`.
- Keeping the string pool avoids changing TEX's data structures and algorithms.

### 2.3 64-bit memory words

The biggest problem with the previous `ctex.w` was the limitation of a 16-bit pointer type, allowing access to at most $2^{16}$ of TEX's memory words. To run a typical LATEX job, loading only a few of the most common packages, this is usually insufficient. And for many people, TEX is just an abbreviation for LATEX. Hence, an implementation of TEX that is restricted to 16-bit pointers is a nice research project but is not suitable for processing practical LATEX workloads.

Allocating bigger arrays is not the problem; the problem is the storage space needed for the array

indices. Indices are stored in a `halfword` which in version 0.4 was a `uint16_t`. Two `halfword`s make up a `memory_word`. These data structures needed redefinition and it was unclear how the packing and unpacking of memory words would be affected by the change. Still, I expected that changing `max_halfword` should be feasible without creating too many complications, because I remember having seen in the 1970s a TEX implementation using 48-bit words. Because TEX tests "`if 2 * max_halfword < mem_top - mem_min then bad:=41;`", the value of `2 * max_halfword` must not produce an overflow. So I changed the value of `max_halfword` to `0x3FFF FFFF`.

After that, `web2w` ran without complaint. My patch file needed a few changes, such as replacing a `uint8_t` by a `uint16_t` at one place and a `uint16_t` by a `uint32_t` at another place, until, to my own surprise, the GNU C compiler would again compile `ctex.c` without further errors or warnings and even the infamous TRIP test had no objections.

### 2.4 Minor changes

#### 2.4.1 C-style macros

Avoiding name conflicts with the long list of TEX's macros — including common names like "x0", "day", "pop", "link", "next", "name" — was a constant concern when working on the `HINT` project. In the C language, there is no separate name space for macros, but it is common practice to avoid conflicts between macros and variables, functions, or other identifiers by using all uppercase names for macros. So version 1.0 implements a command line option that makes macro names use uppercase letters.

Changing all macro names to upper case does, however, impact the visual appearance of the TEX program considerably. Therefore this replacement is optional.

The `WEB` system allows defining parameterized macros using a single `#` sign to mark the insertion point(s) for the parameter text. This does not prevent you from passing multiple parameters because commas are allowed in the parameter text. But at every insertion point, the complete parameter text—with all its commas—is inserted. To overcome this restriction, TEX resorts to "tail calls" in it macro definitions. An example is TEX's definition of `char_info`:

```
@d char_info_end(#)==#].qqqq
@d char_info(#)==
    font_info[char_base[#]+char_info_end
```

which is used as

```
char_info(f)(c)
```

The `char_info` macro ends with `char_info_end`, the tail call, without specifying the parameter for it.

`web2w` is a source code converter. It strives to produce readable source code as its output. While Pascal does not know about macros, they are a common feature of C and there is a well-established way of using them. So translating the above literally to C does not yield code that has the right "look and feel". So, version 1.0 of `web2w` now implements the unrolling of the tail calls and generates true C-style macro definitions:

```
@d char_info(A, B) \
    font_info[char_base[A]+B].qqq
```

which is used as

```
char_info(f, c)
```

As another improvement, `web2w` now counts the number of uses and eliminates macro definitions that are not or — as in the case of `char_info_end` — are no longer used.

#### 2.4.2 Placing the `case` keyword

Respecting the common coding style of C also demanded an improvement in the placement of C's `case` keyword. In the code for processing a ligature or kern command, TEX uses case labels of the form "`qi(1),qi(5): ...`" where the macro `qi(A)` is defined as `A+min_quarterword`. From this, `web2w` version 0.4 produce:

```
qi(case 1):  qi(case 5):  ...
```

which is correct C code but looks odd. Finding the right place for the `case` keyword is not trivial; for example, it would be an error to place the keyword before the "`A`" in the expansion text of macro `qi`. Version 1.0 now produces the better-looking:

```
case qi(1):  case qi(5):  ...
```

#### 2.4.3 Mixed arithmetic with signed and unsigned integers

In Pascal, comparing of two integer expressions will always return a correct value, because Pascal maps both operands onto a common ordinal type, "large enough" for both operands, before comparing them.

This is different in C, where the rules for implicit type casting of operands are complicated and comparing a signed and an unsigned integer might not produce the expected outcome. Fortunately, the C compiler will, with the right warnings enabled, emit complaints about signed/unsigned comparisons. Similar problems occur when signed and unsigned integers are mixed in other arithmetic operations. Version 0.4 tried to replace a Pascal subrange type with the smallest standard C integer type that is large enough. So

```
beta:1..16;
```
became `uint8_t beta;` and
```
shown_mode:-mmode..mmode;
```
was converted to `int16_t shown_mode;`

The information about the true range of values is unavoidably lost by this translation. Considering the problems caused by the constant mix of different integer types, it seems preferable to forgo the approximation with the smallest possible C integer type and choose a plain `int` wherever this is sufficient. This is the approach taken by version 1.0 which eliminated all (if I can trust the compiler) problems with mixed integer expressions.

## 3 `ctex` and TEX Live

While `ctex.w` is a complete and functional implementation of TEX, it does not offer the functionality and amenities that users expect from a modern TEX engine.

When running `ctex`, probably the first thing you observe is that `ctex` will not search the "usual" directories for TEX's font metric files. As described in *The TEXbook*, only the current directory and the subdirectory `TeXfonts` are searched. The directories `TeXformats` and `TeXinputs` are searched for formats and input files. You are probably also accustomed to specifying input files and various options on the command line, but `ctex` will ignore your command line and present you, after displaying the banner, with a plain "`**`" prompt.

For a modern TEX engine, the `kpathsearch` library has become the standard to find all sorts of TEX-related files, and the TEX Live distribution has promulgated de facto standards on command line functionality.

When you try to run LATEX, sooner or later you will also find out that quite a few LATEX packages will assume that some primitive control sequences are present that are not specified in *The TEXbook* but are part of ε-TEX.

The extensions of ε-TEX are happily straightforward to obtain. Using the program `tie`, you can apply `etex.ch` to `tex.web` and obtain `etex.web`. Then apply `web2w` to get `etex.w`. To provide file searching with the `kpathsearch` library and a usable command line, another change file, `ktex.ch`, is part of the `web2w` distribution. It can be applied to `etex.w` to obtain `ktex.w`.

To fully support LATEX even more primitive control sequences are necessary [2]. At the time of writing, another change file is in preparation to add these control sequences to `ktex.w`. It is planned that `ktex.w` will be part of the next TEX Live distribution.

## 4 Conclusion

With `web2w` I have tried to achieve a source code to source code translation of TEX that strives to provide TEX source code in a form that is as close as possible to Donald Knuth's original source code while at the same time is supported by modern programming environments using the C programming language. While there are still many improvements to be made, `ctex.w` in its present form is well-suited to compile, run, and interactively study TEX. It also provides a good basis for conducting experiments with TEX, trying new extensions of TEX, or using parts of TEX in other software projects.

With `ktex.w` an extended TEX will be available that can cope with serious workloads. I have developed `ktex.w` primarily for using it as a basis for HiTEX, a new specialized TEX engine [4]. At the outset, I had no plans for making `ktex.w` available to the public, but while `ktex` is not intended for the average TEX or LATEX user, it may serve others as a basis for their development of specialized versions of TEX.

## References

[1] Donald E. Knuth and Silvio Levy. *The CWEB System of Structured Documentation*. Addison Wesley, 1994. `https://ctan.org/pkg/cweb`.

[2] LATEX Project Team. LATEX news, issue 31, February 2020. *TUGboat*, 41(1):34–38, 2020. `https://tug.org/TUGboat/tb41-1/tb127ltnews31.pdf`. Section "LATEX requirements on engine primitives". See also: `latex-l` thread of April 20, 2021, "LaTeX required primitives: some questions", at `https://listserv.uni-heidelberg.de/cgi-bin/wa?A0=latex-l`.

[3] Martin Ruckert. Converting TEX from `WEB` to cweb. *TUGboat*, 38(3):353–358, 2017. `https://tug.org/TUGboat/tb38-3/tb120ruckert.pdf`.

[4] Martin Ruckert. HINT: Reflowing TEX output. *TUGboat*, 39(3):217–223, 2018. `https://tug.org/TUGboat/tb39-3/tb123ruckert-hint.pdf`.

◇ Martin Ruckert
Hochschule München
Lothstrasse 64
80336 München
Germany
`martin.ruckert (at) hm dot edu`