MWEB User Manual

Mark Potse

May 22, 2000

1 Introduction

MWEB is a literate-programming system like `WEB` [1, 2, 3], made for the MATLAB programming language [4, 5]. It is created with the Spider system, and therefore it closely resembles Levy's CWEB [6] and many spidery webs (There are spidery webs for ada, awk, C, postscript, turing, and several others). This article describes how to learn using MWEB. If you want to learn what Literate Programming is, the best thing to do is read the article "*Literate Programming*" by D.E. Knuth [3], or the book by the same name [2].

2 How to learn to use mweb

Writing a comprehensive user manual takes a lot of time, more than the author of MWEB can afford to spend on it. Fortunately, there is the "*Spidery `WEB` User Manual*" [7], the common user manual for all `WEB`s of the Spider class. Most of the material in there applies to MWEB. To learn using MWEB, read that manual, and afterwards read section 3 (and perhaps the appendices) of this document.

3 Notes on using mweb

The following are features of MWEB that are not present in any Spidery web. These include language-specific features, bells and whistles that are only useful in MATLAB code, and fixes to the private version of Spider that are not present in the official version. The latter include bug fixes as well as extra features.

- It is very difficult for `mweave` and `mtangle` to see if a quote is a transpose operator or a string delimiter. In test programs it all goes quite well now, but there may be errors still. If you have problems with this, i.e., if MWEB thinks a transpose operator is a string delimiter or v.v., you can:

  – put the string and its delimiters in parentheses: "`('some␣silly␣string')`"

  – put the argument of the transpose operator in parentheses: "`(a)'`"

  This should always work.

  If you get the error message "`!unknown left context for qoute`", `mweave` could not decide on the question. Please inform the author about this, particularly about the context involved. In such cases, `mweave` will assume it is a transpose-operator as a default, and in most cases you can solve the problem with parentheses as stated above.

- MWEB can be used with LaTeX instead of TeX, by means of the `webfiles` package[1] [8].

- Strings opening with a left quote (`` ` ``) are typeset as normal code by `mweave`; `mtangle` translates the left quote to a normal quote, and does macro expansion and module expansion inside the string. This is useful if you write strings containing MATLAB code, as is done a great deal in user interface stuff: You get prettyprinted code, identifiers are indexed, and you can use modules and macros in

[1] With `webfiles`, you can import any number of `WEB` documents, from `mweave`, `cweave`, and several others, in a single LaTeX document and, of course, use LaTeX as the documentation language in the `WEB` documents themselves

the string. You can even use these “formatted strings” in macro replacement texts and have macro arguments expanded inside!

To create a formatted string inside another formatted string, use another single left quote, and close it with a single right quote. To create a normal string or a transpose operator inside a formatted string, use a double-quote character (‘`"`’). These rules apply also in modules that are used in the string, and to macro arguments that are used in a formatted string in the macro’s replacement text! Line continuations (‘`...`’) may be used in the formatted string to enable `mweave` to parse the text correctly; they are ignored by `mtangle`.

The left and right quotes in a formatted string must be balanced. Opening and closing quotes are typeset using the `\FQL` and `\FQR` macros. These are defined in `mweb.tex`. They expand into n left or right quotes, where n is the string’s nesting level. There are two versions of the `\FQL` and `\FQR` macros in `mweb.tex`; the author prefers the version that uses guillemets instead of quotes, but this requires a nonstandard font. Of course you can redefine them to whatever you like.

Note that, although a formatted string can contain newlines, MATLAB strings cannot. The newline characters are removed by `mtangle`. This means that you must sometimes add a comma or semicolon, as appropriate, to keep the interpreter happy, in places where a newline would be enough in normal code. Also, line continuation tokens are not necessary to make the text correct for MATLAB, but they may be necessary to `mweave`. `mtangle` translates a formatted string into a concatenation of strings, one for each line in the `.web` file, in order not to overflow MATLAB’s input buffer. The final result—that is, what gets evaluated by MATLAB—is a single string in MATLAB’s memory.

Note also that this feature allows you to be very unreasonable to your computer, with great ease: If you type ‘‘’ after typing ‘‘‘‘‘‘‘’ (i.e. in a string at level 6), it prints as “““ (7 left quotes), and is tangled into

```
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

(2^6 quotes). `mtangle` can handle nested formatted strings upto at least level 15, which yields 16384 quotes in the output. MATLAB seems to have an input buffer of 1024 characters; a single line of code may not be longer than that.

Examples of the use of formatted strings are given in appendix D. The complete source of this appendix is in the file `example.web`.

- You can give string arguments to macros, which get inserted in a normal string (i.e. not a formatted string as discussed above) in the expansion text. The argument called ‘*foo*’ is referred to as ‘`${foo}`’ in the string; for example:

```
@d a_macro(str) = disp('the string is: ${str}')
...
a_macro('hello');
```

  is tangled into

```
disp('the string is: hello');
```

  This can circumvent the use of the *sprintf* function when a constant string argument is given to a macro, for example to save time in tight loops.

- Comments, introduced with a percent sign, are ignored by `mtangle`. If you want a comment to be printed in the m-file, for example to make a comment block for MATLAB’s *help* function, you can use ‘`@%`’ instead of ‘`%`’. This creates a comment that is copied verbatim to the m-file, and is printed in typewriter type by `mweave`.

- The special identifier *TeX* can be used in format definitions, like in CWEB. If one defines "**format** *foo TeX*," then the identifier *foo* will be printed as the TeX control sequence `\foo` by `mweave`. A definition for this control sequence may be provided in the limbo section, or may be standard in plain TeX or LaTeX, e.g. for the identifier *phi*, which becomes `\phi` which prints as ϕ. Note that the control sequence will be used in math mode. By this mechanism, an identifier can be made to look like anything.

- Some Spider bugs were fixed: Prevent multiple occurrences of sections in cross-reference lists, TeX's special characters are protected in output files, and full support for other languages than English.

- The `@s` control sequence from CWEB will be implemented.

4 An example

As an example, I present here an MWEB document that is short enough to read, but long enough to get a feel of how it works. The following sections contain:

- an excerpt from the `.web` file, the source of all,

- an excerpt from the `.m` file produced by `mtangle`,

- an excerpt from the `.tex` file produced by `mweave`, and

- the resulting documentation.

A An excerpt from `example.web`

Here is the source of it all (starting at module 10). Note that the indentation used by the author, which is used to enhance readability of the `.web` file itself, is ignored by `mweave`.

```
@ Here, a macro argument is used in a formatted string that is part of
  the replacement text of the macro. (This isn't useful, it's just an
  example.)
@d print_arg(win,fun,arg) = set(win,"DeleteFcn", `fun(arg);')
@<unused code@>=
  print_arg(gcf, disp, "aargh!");
  print_arg(gca, printf, "foo!");

@ Array elements may be separated by commas or spaces or both.
@<unused code@>=
  x = [0 0 -2 pi,2*2, 2*pi,5, -4.5 20 9243857 inf];
  a = 1/4;
  aa = 1.25e4 / 3.54E10;
  b = 2e20i,  c = pi/4;
```

B An excerpt from `example.m`

Here is a small part of the `lines.m` file that `mtangle` produces, corresponding to the `.web` text above.

```
  set(gcf,'DeleteFcn', ['disp(''aargh!'');']);
  set(gca,'DeleteFcn', ['printf(''foo!'');']);

  x  =  [0 0 -2 pi,2 * 2, 2 * pi,5, -4.5 20 9243857 inf];
  a  =  1/4;
  aa  =  1.25e4 / 3.54E10;
  b  =  2e20i,  c  =  pi/4;
```

C An excerpt from `example.tex`

The `.tex` file that is output by `mweave` is not meant for human readers (although it can perfectly well be handled by TeX).

```
\M10. Here, a macro argument is used in a formatted string that is part of
the replacement text of the macro. (This isn't useful, it's just an
example.)
\Y\P\4\D$\\{print\_arg}(\\{win},\\{fun},\\{arg})\S$\5
$\\{set}(\1\\{win},\;$\39$\.{'DeleteFcn'},\;$\39$\FQL$\1$\\{fun}(\1\\{arg})\2;%
\;\FQR$\2$)\2$ \par
\Y\P\4\X9:unused code\X${}\mathrel+\S{}$\6
$\\{print\_arg}(\1\\{gcf},\;$\39$\\{disp},\;$\39$\.{'aargh!'})\2;\;$\Y\par
$\\{print\_arg}(\1\\{gca},\;$\39$\\{printf},\;$\39$\.{'foo!'})\2;\;$\Y\par
\fi

\M11. Array elements may be separated by commas or spaces or both.
\Y\P\4\X9:unused code\X${}\mathrel+\S{}$\6
$\|{x}=[\1\O{0}\ \O{0}-\O{2}\ \pi,\;$\39$\O{2}{*}\O{2},\;$\39$\O{2}{*}\pi,\;$%
\39$\O{5},\;$\39$-\O{4.5}\ \O{20}\ \O{9243857}\ \infty]\2;\;$\6
$\|{a}=\O{1}/\O{4};\;$\6
$\\{aa}=\O{1.25\_4}/\O{3.54\_10};\;$\6
$\|{b}=\O{2\_20}\ \|{i},\;$\6
$\|{c}=\pi/\O{4};\;$\Y\par
\fi
```

D The documentation of `example`

This is the final document. It was typeset using the `webfiles` package [8].

Example

1. Example. This file serves to test `MWEB` (Matlab Web) and to illustrate its features.

2. This code is written in a file called `lines.m`. This function creates a figure window where lines can be drawn interactively by pushing the left mouse button at the desired begin point, dragging the mouse, and releasing the button at the desired end point.

```
⟨lines.m 2⟩ ≡
  figure;
  axes(...
    'Units', 'normal', ...
    'Position', [0, 0, 1, 1], ...
    'Visible', 'off', ...
    'XLim', [0, 1], ...
    'YLim', [0, 1], ...
    'XLimMode', 'manual', 'YLimMode', 'manual');
  ⟨set the callbacks 3⟩
```

3. Formatted strings. Particularly when creating a user interface, Matlab programmers may want to write nested strings containing code up to three or four levels. This example program defines a WindowButtonDownFcn, which in turn defines a WindowButtonMotionFcn and a WindowButtonUpFcn, which in turn undefines the WindowButtonMotionFcn. Some parts of the code are put in refinements, to keep this section comprehensible.

```
⟨set the callbacks 3⟩ ≡
  set(gcf, 'WindowButtonDownFcn', ‹
      ⟨create the line 5⟩
      set(gcf, 'WindowButtonMotionFcn', «⟨reset the line 6⟩»);
      set(gcf, 'WindowButtonUpFcn', «
          fix_line;
          set(gcf, 'WindowButtonMotionFcn', ⋘⋙); »);
      ›);
```

This code is used in section 2.

4. By the way, `mtangle` expands the above code into:

```
set(gcf,'WindowButtonDownFcn',['',...
'lb = get(gca,''CurrentPoint'');',...
'le = lb;',...
'L = line([lb(1,1),le(1,1)],[lb(1,2),le(1,2)], ',...
'''EraseMode'',''xor'');',...
'set(gcf,''WindowButtonMotionFcn'',''le = get(gca,''''CurrentPoint'''');',...
'set(L,''''XData'''',[lb(1,1),le(1,1)],',...
'''''YData'''',[lb(1,2),le(1,2)]);',...
''');',...
'set(gcf,''WindowButtonUpFcn'',''',...
'set(L,''''Color'''',''''r'''');',...
'set(gcf,''''WindowButtonMotionFcn'''','''''''');'');',...
'']);
```

5. The WindowButtonDownFcn determines the mouse position and creates a line, whose begin- and endpoints are the same.

⟨ create the line 5 ⟩ ≡
  $lb = get(gca,$ `'CurrentPoint'`$);$
  $le = lb;$
  $L = line([lb(1,1), le(1,1)], [lb(1,2), le(1,2)],$ `'EraseMode'`, `'xor'`$);$

This code is used in sections 3 and 8.

6. Then, the WindowButtonMotionFcn makes the endpoint of the line move with the mouse pointer.

⟨ reset the line 6 ⟩ ≡
  $le = get(gca,$ `'CurrentPoint'`$);$
  $set(L,$ `'XData'`, $[lb(1,1), le(1,1)],$ `'YData'`, $[lb(1,2), le(1,2)]);$

This code is used in section 3.

7. And at last, the WindowButtonUpFcn changes the colour of the line and, by undefining the WindowButtonMotionFcn, fixes it.

define *fix_line* ≡ $set(L,$ `'Color'`, `'r'`$);$

8. More examples. The following code is just an example of several Matlab constructs; it does nothing useful.

  ⟨ unused code 9 ⟩
  ⟨ create the line 5 ⟩
  *a_variable_*`@&` *with_*`@&` *a_com*`@&` *pound_name* $= 8;$

9. Complex constants in Matlab are made by appending an 'i' character. In exponential notation, it follows the exponent.

⟨ unused code 9 ⟩ ≡
  $a = 2i;$
  $z = 2 \cdot 10^4 + 3 \cdot 10^4 i;$
  $flarp = rand(3,8) \;./\; linspace(3,8);$
  ⟨ some refinement 13 ⟩

See also sections 10, 11, 12, 16, and 17.
This code is used in section 8.

10. Here, a macro argument is used in a formatted string that is part of the replacement text of the macro. (This isn't useful, it's just an example.)

define $print\_arg(win, fun, arg) \equiv set(win,$ `'DeleteFcn'`, $\langle fun(arg);\ \rangle)$

$\langle$ unused code 9 $\rangle$ $+\equiv$

$print\_arg(gcf, disp,$ `'aargh!'`$);$
$print\_arg(gca, fprintf,$ `'foo!'`$);$

11. Array elements may be separated by commas or spaces or both.

$\langle$ unused code 9 $\rangle$ $+\equiv$

$x = [0\ 0 - 2\ \pi,\ 2{*}2,\ 2{*}\pi,\ 5,\ -4.5\ 20\ 9243857\ \infty];$
$a = 1/4;$
$aa = 1.25 \cdot 10^4/3.54 \cdot 10^{10};$
$b = 2 \cdot 10^{20} i,$
$c = \pi/4;$

12. if-elseif-else-end constructs:

$\langle$ unused code 9 $\rangle$ $+\equiv$

if $(a \neq b), a = b + 1;$ **end**
if $(a \neq b)\ \ a = b + 1;$ **end**
if $a \equiv b$
  $a = b;$
end
if $a \equiv 3\ \ a = b;$ **elseif** $a \equiv 4,\ a = b/2;$ **else**$a = a + 1;$ **end**
if $(a \neq b) \wedge a < c \vee \ldots$
    $(c \geq 30) xor\, a < 3$
  $\langle$ some refinement 13 $\rangle$
elseif $a \equiv 25$
    $c = 4.67/2 + 8{*}6\hat{\ }3;$
  elseif $a < b$
    $c = c + 3;$
  else
    $disp($`'foo'`$);$
  end
end

13. A terrible **for** statement:

$\langle$ some refinement 13 $\rangle$ $\equiv$

for $i = 1:10$
  for $j = 1,\ 5,\ 6$
    for $k = 1,\ 3,\ 4,\ 23,\ 24,\ 83,\ 34,\ 34,\ 23,\ 25,\ 25,\ 27,\ 56,\ 25,\ 45,\ 3,\ 5,\ \ldots$
        $9,\ 3,\ 6,\ 234,\ 56,\ 32324,\ \pi,\ 432,\ 234,\ 453,\ 345,\ 345,\ 93,\ 93,\ \ldots$
        $83,\ i,\ j,\ 8,\ 9,\ 10,\ 54,\ 843,\ 845,\ 8342,\ 7234,\ 7834$
      $matrix = zeros(21, 20);$
      $vect(i) = matrix(2{*}i + 1, j);$
    end
    if $a \equiv b$
      $a = a + 2;$
    end
  end
end

This code is used in sections 9 and 12.

14. Custom-formatting of identifiers. In the limbo section of this file, the following definition was made:

```
\def\phin{\phi_{\rm in}}
```

Which prints as "$\phi_{\rm in}$." Then, the variable that is typed as "`phin`" in the program prints as $\phi_{\rm in}$ too, if the following format statement is used:

format *phin* *TeX*

15. We also want the identifier *beta* to print as β. Because this definition is already provided by plain TEX and LATEX, we need not give it here, we just have to use the following format statement:

format *beta* *TeX*

16. This is the result. The identifier *alpha* is not treated specially, so it is printed in the usual way.

$\langle$ unused code 9 $\rangle$ $+\equiv$

$\phi_{\rm in} = alpha/\beta;$

17. Switch statements are new in Matlab version 5. Here is a pathological case.

⟨ unused code 9 ⟩ +≡

switch $a\_variable + a\_variable\_with\_a\_long\_name - 23239847 + 1.24 \cdot 10^6/34 - \ldots$
$1 + a + b\hat{}2 + 4c$
case 0
$c = 3;$
$b = a + (c\hat{}2 - 24)/\pi;$
case 1, $a = 2;$
case $\{2, 3, 4\}$, $a = 3;$
otherwise
$a = 4 + 5;$
end

Index of `example`

List of Refinements in `example`

References

[1] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(1):97–111, 1984.

[2] Donald E. Knuth. *"Literate Programming"*. CSLI, 1992. CSLI Lecture notes no. 27.

[3] Donald E. Knuth. *The WEB system of structured documentation*. Stanford University, 1983. Computer Science Report CS980.

[4] The Math Works, Inc., Cochituate Place; 24 Prime Park Way; Natick, Mass. 01780; USA. *Matlab User's Guide*, August 1992.

[5] The Math Works, Inc., Cochituate Place; 24 Prime Park Way; Natick, Mass. 01780; USA. *Matlab Reference Guide*, August 1992.

[6] Sylvio Levy. *The CWEB System of Structured Documentation*.

[7] Norman Ramsey. *The Spidery WEB system of Structured Documentation*.

[8] Mark Potse. *The* `webfiles` *Style Option*, september 1994.