Kent Lindquist White Paper

Number 1998-001

Creating a Framemaker Macro from a Perl Script

Kent Lindquist

Sept. 9, 1998

Creating a FrameMaker macro from a PERL script

Kent Lindquist, 1998

Framemaker 5.5 and Perl 5.4.3

FrameMaker macros can be saved in a file. That file under normal conditions is called *fmMacros*. FrameMaker can find this file in a number of places, including the current working directory. Here is an example of an *fmMacros* file. There are two macros in this file. The first macro is triggered by a Cntl-1 pressed in an open copy of a particular FrameMaker template document of mine. This macro is fairly complex. It takes the template, fills in some text about a recent earthquake, and puts in a picture of the earthquake's location. The second macro in this file, triggered by a Cntl-3, merely prints the current document and exits FrameMaker.

```
 1: Macros saved on Tue Sep  8 18:28:19 AKDT 1998
 2:
 3: aeic release macro:
 4:       <Macro Macro2
 5:      <Label Macro2>
 6:       <Trigger ^1>
 7:       <TriggerLabel ^1>
 8:      <Definition \!fa/START_DIALOG ^/Tab /End
 9:      ^u/home/kent/work/response/980825172246/myrelease.fm/Return /END_DIALOG
10:       \!fif/START_DIALOG ^/Tab /End
11:       ^u/home/kent/work/response/980825172246/aeic_release.makertext/Tab /Tab /Tab
 \s/Tab
12:       /Return /END_DIALOG /START_DIALOG /Tab ^/Tab /Tab /Return /END_DIALOG
13:       /START_DIALOG ^/Tab /Tab \s/Tab /Tab /Tab /Return /END_DIALOG
14:       \!fif/START_DIALOG ^/Tab /End ^u/home/kent/work/response/980825172246/db_rel
ease.epsi
15:       /Tab /Tab /Tab \s/Tab /Return /END_DIALOG
16:       \!go/START_DIALOG ^/Tab /Tab /Tab /End ^u3.375/Tab /End ^u4.694
17:       /Tab /Tab /Tab /Tab /Tab /Tab /Tab /Return /END_DIALOG \!fs
18:       >
19:       <Mode NonMath>>
20:
21: aeic closeout macro:
22:       <Macro Macro3
23:       <Label Macro3>
24:       <Trigger ^3>
25:       <TriggerLabel ^3>
26:       <Definition \!fp/START_DIALOG /Return /END_DIALOG
27:       \!fc
28:       >
29:       <Mode NonMath>>
```

Now we're going to pick this apart a bit. The line numbers are not in the original file. I added them for discussion.

Lines one through three contain boring stuff. Actually lets look at the second macro. It is entirely contained in the angle brackets on lines 22 and 29. Inside is a a label, contained in angle brackets on line 23. Boring. Line 24 says that hitting control-3 in a framemaker document will make the macro do it's work. Line 25 seems redundant but who are we to argue. The definition of the macro, i.e. the statement of what work it will do, is contained entirely within angle brackets on lines 26 through 28. On line 26, you'll see that the first phrase is **<Definition \!fp**. If you're familiar with the keyboard shortcuts that you can use in FrameMaker, you know that Escape-f-p brings up the dialog box for printing. That's what we just said in the macro: **\!** means the escape key, and **f** and **p** mean "file" menu and "print" option. The rest of what we're going to do in the dialog box is bracketed by the **/START_DIALOG** and **/END_DIALOG** keywords. Note that the forward-slash character, /, is a special one in framemaker macros. The default action for this dialog box is to print out the current document, and hitting a return key in the print dialog box prints the document out. Therefore the only thing we have in our macro for this dialog box is to hit the return key, indicated by **/Return**. This macro ends by closing the current document. The keyboard shortcut for that is escape-c-b, so the macro, on line 27, reads **\!fc**.

That's the basic pattern for a macro as described in the *fmMacros* file. Now lets look at a few things in the more complicated macro. This macro starts out by launching the *file->save as* dialog box. This happens on line 8 with **\!fa/START_DIALOG**. Tab keys are used to navigate amongst the fields in a dialog box. In this case, we use Control-Tab, indicated in the macro by **^/Tab**, in order to go to the first field of the dialog box. We might be there already but you have to be sure. This first field of the save-as dialog box is where you put in the filename you want to save your document under. It might not be empty, though. Therefore, at the end of line 8, we put **/End** to go to the end of what's in the text-entry box. At the beginning of line 9, we put a Cntl-u, indicated with **^u**, to flush out whatever's in the text-entry field. Then we just enter the pathname we want, followed by a return character and the **/END_DIALOG** delimiter. You'll notice this same pattern in the rest of the macro: go to the beginning of the dialog box, tab through to set the fields you want. For each field where we're entering text, first empty out the text-entry field by going to the end of it and typing Cntl-u. By the way, the reason the first step of the macro is saving the document is that I want to register the correct filename rather than having someone accidentally obliterate my template. That's basically all of the macro. On line 15 there's a **\s** from where I changed the state of a little radio-button click box. Line 16 enters some numbers, which are the measurements in inches needed to position my imported picture correctly on the page.

Now we need to create this macro from Perl. Here we go, once again introducing line numbers:

```
 1: sub write_fm_macro {
 2:
 3:      $makertextcode = "$release_dir/$event_timestr/$makertextfile";
 4:      $makertextcode =~ s@/@\\/@g;
 5:
 6:      $fmrelease_filecode = "$release_dir/$event_timestr/$fmrelease_file";
 7:      $fmrelease_filecode =~ s@/@\\/@g;
 8:
 9:      $map_epsi_filecode = "$release_dir/$event_timestr/$map_epsi_file";
10:      $map_epsi_filecode =~ s@/@\\/@g;
```

```
11:
12:     $date = `date`;
13:
14:     $fmmacro = <<“      EOMACRO”;
15:     Macros saved on $date
16:     aeic release macro:
17:     <Macro Macro2
18:     <Label Macro2>
19:     <Trigger ^1>
20:     <TriggerLabel ^1>
21:     <Definition \\!fa/START_DIALOG ^/Tab /End
22:     ^u$fmrelease_filecode/Return /END_DIALOG
23:     \\!fif/START_DIALOG ^/Tab /End
24:     ^u$makertextcode/Tab /Tab /Tab \\s/Tab
25:     /Return /END_DIALOG /START_DIALOG /Tab ^/Tab /Tab /Return /END_DIALOG
26:     /START_DIALOG ^/Tab /Tab \\s/Tab /Tab /Tab /Return /END_DIALOG
27:     \\!fif/START_DIALOG ^/Tab /End ^u$map_epsi_filecode
28:     /Tab /Tab /Tab \\s/Tab /Return /END_DIALOG
29:     \\!go/START_DIALOG ^/Tab /Tab /Tab /End ^u3.375/Tab /End ^u4.694
30:     /Tab /Tab /Tab /Tab /Tab /Tab /Tab /Return /END_DIALOG \\!fs
31:     >
32:     <Mode NonMath>>
33:
34:     aeic closeout macro:
35:     <Macro Macro3
36:     <Label Macro3>
37:     <Trigger ^3>
38:     <TriggerLabel ^3>
39:     <Definition \\!fp/START_DIALOG /Return /END_DIALOG
40:     \\!fc
41:     >
42:     <Mode NonMath>>
43:
44:     EOMACRO
45:
46:     $fmmacro =~ s/^\t//;
47:     $fmmacro =~ s/\n\t/\n/g;
48:
49:     open( M, “>fmMacros” );
50:
51:     print M $fmmacro;
52:
53:     close( M );
54: }
```

The strategy here is to construct the whole macro in the variable **$fmmacro**, then write that macro out in one big blast to the appropriate file. Writing the variable out to the file *fmMacros* is actually just straightforward Perl, on lines 49 through 53. The procedure to fill in **$fmmacro** is simply to make a multi-line quote. Normally in perl you can do this with the construct

```
$moo = “some stuff”;

$myquote=<<ENDOFQUOTE;
some interesting
text, possibly with the value of some variable called moo
interpolated into the text right here: $moo
ENDOFQUOTE
```

This is basically what we do. However, this requires that all the lines are justified to the left side of the page. If we’re indenting the subroutine so the Perl code looks halfway readable, we need to get a little bit fancy. We add a tab character in front of each line; in front of the “ENDOFQUOTE” designator; and then remove those tab characters later. This means that our multi-line quote example above now says

```
    $moo = “some stuff”;

    $myquote=<<“      ENDOFQUOTE”;
    some interesting
    text, possibly with the value of some variable called moo
    interpolated into the text right here: $moo
    ENDOFQUOTE
```

We remove these extra tab characters with lines 46 and 47 of the perl script above. This is a small concession to readability in the script, which is bad enough to begin with.

There are a few differences between the macro as it appears in the perl script and the macro as it appears in the final output file. First of all, the backslash character \ in the macro will mean something in perl--something it’s not supposed to mean, in fact. Therefore, every backslash character in the macro must appear as \\ in the perl script so the Perl interpreter doesn’t gobble it up.The first easy example of this is on line 21 of the perl script, corresponding to line 8 of the fm macro. Another example is on line 28 of the perl script, corresponding to line 15 of the fm macro.

Next, the forward slash / means something to FrameMaker. Therefore if we want to use pathnames in the macro that contain forward slashes, we have to put a backslash in front of them like this \/ to keep from confusing framemaker. Again, though, now perl will want to interpret that backslash, so we have to put another in front of it in the perl script, so it looks like \\/. Now actually in the perl script all of our file names are in variables, so this substitution happens with a regular-expression substitution on each filename-variable, ie three times in lines 3 through 10.

At this point the hardest part of making a perl script that generates an *fmMacros* descriptor file will probably be finding the “Start recording macro now” button in FrameMaker, so you can build yourself an initial template macro and then save it.The macro commands are under *file->utilities*.