Q&AIX

by Jim Fox

The Preprocessing Payoff

Jim Fox works as a systems programmer for the University of Washington. He writes and maintains distributed applications that run on a variety of UNIX systems–and some non-UNIX ones. He is also the deputy manager for the Interoperability Project for SHARE's Open Systems Group. Email: `fox@cac.washington.edu`.

wizard's apprentice

super user

wizard

Q: Your October article, "Virtual Windows" Page 75, mentioned two ways to customize the `rc` file used by the `fvwm2` window manager when it starts. One was `cpp` and the other was M4. I thought `cpp` might be easier because I know how to program C, but I haven't been able to figure it out. Whenever I try to use the `cpp` preprocessor option, I get all sorts of errors. What am I doing wrong? Would I be better off going right to M4?

Beverly Rhodes
City University

A: Preprocessing is a common way to add flexibility to a program's configuration data. It also lets you set up a common look and feel to your configuration files—even for disparate programs. Often, a program will invoke a preprocessor for you; `fvwm2`, the C compiler and `xrdb` do this. Other times, you might have to run the preprocessor as a completely separate step.

Learning to use a preprocessor will be well worth the effort. We'll consider `cpp` this month and M4 next month. First, you've got to get into the right frame of mind for `cpp`–you're programming a different language now. Those old `fvwm2` comments just won't do anymore.

Remember, `cpp` is expecting C code. Enclose your comments in `/*` and `*/`, just as you would in a C program. Then, use the normal # style directives that you would also use in a C program. The most common and useful are the following:

• `#include` *`filename`*

Includes the named file, interpreting any commands it finds in the file. This allows you to break up a large `.fvwm2rc` file into smaller, more manageable pieces, and allows you to easily share parts of your file with other users.

• `#define` *`name value`*

Defines a new macro, *`name`*, giving it the value indicated.

```
• #if expression
     true code
  #elif expression
     else true code
  #else
     false code
  #endif
```

Figure 1. Useful Definitions Passed from fvwm2 to the cpp Preprocessor

`WIDTH`	Width of the screen in pixels
`HEIGHT`	Height of the screen in pixels
`BITS_PER_RGB`	Measures the number of colors available
`COLOR`	"Yes" or "No"
`USER`	Username

The `if` clause is self-explanatory. If the expression is true, the `true code` is copied to output; otherwise, the `else if` code or `false code` are copied. *`Expression`* can be any normal C-style expression, mixing numbers and defined symbols. In addition, you can use the phrase `defined(`*`name`*`)`, which is true if the name is defined. There can be zero or many `#elif` clauses and zero or one `#else` clause.

- `#ifdef` *`name`*

Same as `#if defined(`*`name`*`)`.

- `#ifndef` *`name`*

Same as `#if !defined(`*`name`*`)`.

For more information, consult almost any C programming reference. My personal favorite is *The C Programming Language*, 2nd Edition, Brian Kernighan and Dennis Ritchie, Prentice Hall, ISBN 0-13-110362-8. Make sure you get the second edition: There were many additions to the C language, and the `cpp` preprocessor, after the first edition was published.

Using cpp with fvwm

Now we're ready to write a custom, `cpp`-style `fvwm2rc` file. Remember that comments in the original file, `.fvwm2rc`, are enclosed with `/*` and `*/` instead of being prefaced with `#`. The comments won't be passed on to `fvwm2`, but that's OK.

When `fvwm2` runs the `cpp` preprocessor, it defines several names which you can use in your file. The most useful of these are shown in Figure 1. See the `FvwmCpp` man page for the rest.

Here is a common use for the `cpp` preprocessor. Suppose you work at various locations where there are different-size X terminals; maybe a large-screen terminal at your office and a smaller one at home. You might want to use different fonts, depending on the screen

size. You could make some definitions related to screen size at the start of your `rc` file. Notice that these examples follow the usual convention that all capitals represent defined names:

```
/* Define screen sizes */
#define BIG_SCREEN WIDTH>=1500
#define MID_SCREEN WIDTH>=1100
#define SML_SCREEN WIDTH<=1100
```

Later in the file you can make use of those definitions:

```
/* Choose larger window manager fonts
    for larger screens. */

#if BIG_SCREEN       /* large screen */
WindowFont 7x13
IconFont 7x13
#else if MID_SCREEN  /* medium screen */
WindowFont 6x10
IconFont 6x10
#else                /* small screen */
WindowFont 5x8
IconFont 5x8
#endif
```

Then, start `fvwm2` with the `cpp` option:

```
fvwm2 -cmd "FvwmCpp rc_file"
```

Some documentation tells you to use the `-f` option for this command, but that won't work–you have to use `-cmd`, enclose the argument in quotes and specify the `rc` file.

The `cpp` preprocessor is a surprisingly useful tool. You might find other uses. It'll work on any text file; just pipe the file through it like this:

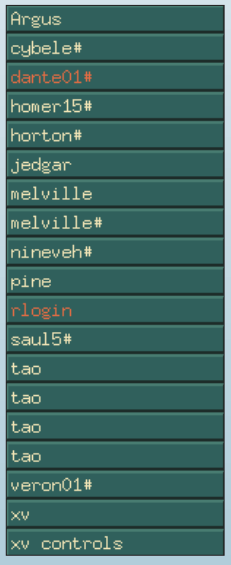
```
cat source | cpp > output
```

Despite the obvious utility, don't go converting all your files to `cpp` code just yet. Next month, we'll look at the other `fvwm2` preprocessor, M4, a more powerful macro processor that just might make you decide never to use `cpp` again.

Q: One of the most convenient features of `fvwm` is the window list module. It's similar to `twm`'s icon manager and allows for a lot of "icons" in a small area of the screen. The problem is, it's unsorted. The window names just appear in random order. I have 20 to 30 windows in that list. I need to have them sorted.

Jim Fox
University of Washington

Figure 2. The Sorted fvwm Icon List



A: Well, I don't expect software authors to think of everything, and it's especially awkward to complain about programs I get for free. Still, it's hard to forgive actual regression from one generation to the next. `twm` sorts that icon list; `fvwm` ought to also–it's easy enough.

Let's take the attitude that if we want something done right we have to do it ourselves. And we can do it ourselves because `fvwm2` is free software for which we get source code, and how hard can it be to sort a little list anyway? Actually, no sorting will be necessary. Because windows are added to the window list one at at time, all we need to do is insert them at the proper point–instead of at the end. Look at the patches (see the Web address at the end of the column) to see how this was done.

And, as long as we're hacking away, let's fix something else. In the window list, iconified windows show up in parentheses. They also show up in a different color. Let's make the parentheses optional, so we can just use color and have a nice tidy list. We'll define a new `.fvwm2rc` file command to specify no parentheses. It might look like this (suppose we're using that `cpp` preprocessor from the last question):

```
#if defined(COLOR)
*FvwmWinListNoIconParens
#endif
```

Our resulting list is shown in Figure 2. It looks good enough. And just in time to make a holiday present. Get your copy from `http://weber.u.washington.edu/~fox/fvwm2/`. There, you'll find patches to the source, along with precompiled binaries for AIX and Linux. Happy holidays.