ShareWare

This application is shareware covered by the GNU Public License. That means that you are welcome to copy it and distribute it as you see fit as long as you follow the GNU Public License, and do not modify the README.rtf file, the COPYING file, or modify any of the panels to remove the indication that this is ShareWare covered by the GPL.

I think this is a very useful application and I hope that you will too. If you find that you are using this application, I would appreciate a contribution to my efforts. I would suggest a contribution of $10. Consider $10 as the price of buying me a 6-pack of good beer. If you provide your electronic mail address as well, I'd be happy to notify you of future releases of the program.

There has been a little confusion about the concept of mixing shareware with the GPL. My intent is that if you are using my program (or a modified verion of my program) you should pay me the shareware fee. If you make changes to the program you should send them to me, so that I can incorporate them. If you give this program to anyone else you must provide all the sources. If you'd like to use parts of the code for another different program of your own, you're welcome to, but they will be covered by the GPL. I hope the intent is clear even if the legalese is muddy.

Thanks,
Daryll
daryll@harlot.rb.ca.us

Daryll Strauss
2607 Nelson Ave. #G
Redondo Beach, Ca. 90278

The Compiler's Companion -- Version 2.0

The Compiler's Companion is an application to make life easier for anyone to develop applications on the NeXT. It does so by automating the steps in the edit-compile loop.

Aside: The motivation for this program was that I don't believe that one should ever have to repeat awkward sequences of commands to perform a task. I realized, before I wrote the Compiler's Companion, that I was constantly going from Edit, to a terminal window to do a make, back to edit and using "command-l" to find the line of an error that make had reported, and then repeating the process. This was just plain awkward. I had seen a better facility in the compile elisp package within emacs, and decided the same facility should be available for people using Edit on the NeXT. And so, the Compiler's Companion was born.

The Compiler's Companion provides two services (primarily directed at users within Edit) the first is Make. This is tied to the the user typing command-k. When this service is invoked the first time, Compiler's Companion will request a directory and command to execute. When the command completes the results will appear in a window. This removes the first awkward effort, having to switch to the Terminal window to execute makes. After the first invocation, Compiler's Companion will execute the same command in the same directory, unless you use the 'Make¼' menu item. The command you execute may use the variables $dir, $file, and $stripped, which represent the directory, file, and the file without its extension from which the command was last executed.

The second service is labelled next error, and is bound to the keyboard as command-comma. It will scan the output window from the location of the selection until it finds a line that begins with the form: '<filename>:<line>: ' At that point it will send a message to Edit, causing it to open the indicated file and highlight the indicated line. That eliminates the second awkward effort, having to use command-l to move to the error.

You may only have one compile executing at a time.   If you would like to stop a make that is in progress, use the "Stop Make" menu item. The make occurs in the background, so that you can start processing the error messages as soon as they appear. If you click on a line in the make output, and then hit command-comma, the next error after that line is displayed. This lets you go back through your errors if desired. Also realize that you can use Compiler's Companion to process the results of any program that uses the same filename:line format for errors.

CC will attempt to decide whether or not the command being executed was successful and perform a few tasks depending on that result. The way that it determines the success is first by looking at the exit status of the command. If that is anything other than zero the command is deemed to have failed. If that is zero, it looks through the output of the make for lines of the form '*** Exit <num>' If it finds that it deems the make to have failed. If the command is successful, the default action is to play a particular sound and hide CC. You can change the command (you probably do want a different sound at least!) and decide whether or not CC should hide. If the command fails, a difference command is executed. The default is to play a difference (also weird) sound.

The preferences menu item (under the info menu) allows you to configure Compiler's Companion. The first field, labelled directory, is used to indicate what directory Compiler's Companion should default to when it compiles something and doesn't know where it came from. Normally, you invoke the make command by using the services item, and the directory will be set to the directory which contains the source file you were editing. The second field, is the default command to be executed. Normally this will be some form of make, because that's what Compiler's Companions parsing is tailored to, but you can execute any command. The next two fields, are the commands to execute on successfull and unsuccessfull execution of the command. I like to use this to play a sound, so that I can ignore the compile until it is done. There are also three check boxes. The one on the right,
labelled Enable Services, indicates whether Compiler's Companions' services should be made available to other applications. The top one on the left, labelled Hide on success, indicates whether Compilers Companion should hide itself after a successful make. The bottom one on the left, labelled Use Preferences Directory, is whether Compiler

Companion should always use the default directory specified on the first field. All these fields are saved in the defaults database with names "directory", "command", "executeGood", "executeBad", "services", "hideOnSuccess" and "usePrefDir" in that order.

Installation

The application and all the source should be included in the distribution, if it is not please contact whoever you received CC from. You can type 'make install' and the application will install into your home directory's App directory. If you would prefer to install it into another directory type 'make INSTALLDIR=<your-directory> install' instead. (Note that if you install it somewhere else, you probably want to change the default actions for the success or failure, because they rely on the installation in the users home directory. To change this, just edit the Full_Defaults_Vector at the beginning of the Controller.m file and remake. You'll see two string contants that start with sndplay. You can change these to anything you like) The services will be available after you log out and back in again. If you would like to make them available right away, run the program 'make_services' after you install the application. If you then run 'make_services -' you should see two new services in the list that look something like this:

```
Port: <546> Compiler
Send Types: (21) <307> NeXT filename pasteboard type
Menu Items: (40) <105> default, <555> Compiler/Make
Key Equivalents: (43) <105> default, k
isHead: NO
Next: 10
Head: 2

Message #10: <569> nextError
Port: <546> Compiler
Send Types: (21) <307> NeXT filename pasteboard type
Menu Items: (46) <105> default, <579> Compiler/Show Error
Key Equivalents: (49) <105> default, ,
isHead: NO
Next: 0
Head: 2
```

If you find these, then any application launched from this point on should include a Compiler sub-menu in the services menu.

Misfeature

There is one last misfeature that I have not been able to avoid. The problem is that when you invoke "show error" via services, your edit window will become the key window, but will not become the front window. Instead, CC's output window will remain on top. This means you have to click on the Edit window (but not in the text area or you will lose the highlighted line that CC set up for you) to bring it to the front. Thats kind of awkward. The problem is that CC is getting moved to the front *after* my service has completed. I've tried various tricks to get the current window, send my windows to the back, but because my window is being moved after my code completes, none of that has worked. Special thanks to anyone who finds a patch for this misfeature!

Special Thanks/Contributors

Daniel Faken - For providing the solution for having to have something selected before the services would work and, of course, a thank you to everyone who has contributed to my development of CC.