Adventures in Programming: Frame•It

It was the kiss of death. On a particular Saturday morning I got that programming feeling and just had to do it. The idea was old, but the enthusiasm was fresh.

The concept was simple—to make a program which would take a screen dump (created with command-shift-3) and draw a border around it, thereby saving all the time and trouble usually spent doing the task in FullPaint. Not only was the concept simple, so was the task. After consulting various scriptures from Inside Macintosh, consulting various Apple Tech Note prophets and coming up with a general concept, the task was accomplished within two hours. And it worked.

But the next problem was the user interface. As any Macintosh programmer would swear, the user interface can be 90% of the work. It never appears this way in the design concept phase, but it does work out this way. Especially all the tweaking which takes place in ResEdit.

So, what's the concept? A quick, yet effective, title graphic was generated with the phenomenal PosterMaker Plus and the creative thoughts began to flow. How about a flashing dot on null events? How about a circular progress gauge which was part of the title graphic? Or how about the StuffIt form of file selection? YES!

For those unfamiliar with StuffIt, or more particularly its means of entering multiple files to be stuffed (I love that terminology!), StuffIt presents a reasonably standard ‘Open’ dialog box and allows the user to select files, folders and various options. All this is conceptually simple and has been seen in other software previously.

The part which hasn't been seen before, however, is the way in which Raymond Lau managed to accept files and folders without the dialog box disappearing between selections. If you don't comprehend, go and try it yourself. And if you don't have StuffIt, then you don't know what you are missing.

The magic dialog box in StuffIt adds files to a list in a window behind the dialog box. And because the dialog box does not continually open and close, the scroll position is always retained. This is immensely useful for selecting multiple files from similar positions. This is the effect I sought for my program.

OK. Simple, right? Just create a dialog hook to pass to the Macintosh SFGetFile routine which will do the processing without the need to close the dialog. In effect, this would make the program just one big call to a specialised SFGetFile with hook attached. I proceeded to do so and incorporated the title graphic, complete with flashing dot and the potential progress gauge, into the standard ‘Open’ dialog box (resource DLOG #-4000).

It looked good and everything. I could detect when the user clicked on the ‘Open’ button (which was renamed to ‘Frame’) and even when additional, informational buttons were selected. The only problem was in knowing which file the user was selecting.

To understand this problem, let's take a step back. The SFGetFile call format is as follows:

```
SFGetFile (where: Point; prompt: Str255; fileFilter: ProcPtr; numTypes: Integer; typeList: SFTypeList; dlgHook:
ProcPtr; VAR reply: SFReply);
```

This format, it should be noted, was obtained from that wonderful desk accessory, *Inside Macintosh*, by Arne Schirmacher from somewhere in Germany (the information box is all in German!).

Quickly going through the parameters to the call, we have the point where the dialog should appear on the screen, a prompt which is only included for so-called ‘historical purposes’, a pointer to a file filter which, in this case, was instructed to search for paint files of type PNTG, a number and list of types for automatic searching, the famed dialog hook to intercept events in the dialog (including null events for things like flashing dots) and the reply field which supplies information on the file selected.

Do you see the problem? The information on the file selected is returned to the calling program, and is not available to the dialog hook routine. (If you're still puzzled, the dialog hook routine operates while the dialog is displayed on screen, enabling additional things to happen which the guys at Apple had never thought about, yet had made possible.) The problem was how to get this information to the dialog hook routine without exiting the dialog box.

The reply information basically contains the name of the selected file and a volume reference number to indicate on which disk the file can be found. With the advent of HFS, this number was changed to also allow a working directory to be returned, thereby indicated the disk AND folder in which the file was located.

To overcome the difficulty of passing this information to the dialog hook routine, some means of information passing was necessary. While the calling program also contained the dialog hook routine, there was no standard means of passing information between the two. What was needed was some ‘global’ means of passing at least the address of where the information could be found.

It may not be perfectly legal or altogether standard, but the method chosen to accomplish this task was to create an invisible window on the screen and store the required information in the window's reference constant variable space. It is possible for any program to search through the chain of windows on the screen (even those which are invisible). This special window was also identified by a unique window ID number to avoid embarrassing situations where other windows were also on the screen.

It worked. The dialog hook routine was able to obtain the address of where the reply information was stored in the calling program. And nicely enough, SFGetFile always updates the filename field whenever a new file is selected. The program worked magnificently. That is, until a file from another disk or folder was selected.

The reason for this is that, while the filename field was continually updated, the volume reference number was only provided when the dialog box is closed—far too late in this case. The solution? Ignore it and go do something else.

This method of problem avoidance always manages to produce a result. Sometimes it even produces a solution. The solution in this case stemmed form a previous programming experience dealing with the ‘Open’ dialog box *(see “Adventures In Programming: PathFinder”)*.

This previous experience brought with it the discovery of a global variable used by the Macintosh. Global variables are bits of memory set aside by the gurus at Apple for some purpose they thought was important at the time. The global variable in question can be found on page IV-309 of Inside Macintosh and is named ‘CurDirStore’. The description explains its function as “Directory ID of directory last opened (long)”. This is how the Macintosh always manages to open the dialog box to the same directory as was last used. Programs such as DiskTop also make use of this global variable.

In the case of Frame•It, all that was necessary was to retrieve this value and use it to tell the FSOpen command where to find the file to be opened. A technique which took considerable time to discover (during the development of *PathFinder*) is show here:

```
type
  LongIntPtr:^LongInteger;
var
  DirPtr:LongIntPtr;

DirPtr := LongIntPtr($398);
```

Then use DirPtr^ to reference the Directory ID.

To use a global variable, a pointer to the global is created. The global variable in this case is a long integer, so the pointer was defined as pointing to a long integer. The location of CurDirStore is $398. Finally, when referencing the long integer value, the pointer is de-referenced.

Fine! Great! Fantastic! I now had both the filename AND the directory ID of the file. Only one problem—there was no indication of the DISK which contained the directory. Disks are capable of having the same directory ID, just like they can have the same filenames. Therefore, to open the required file, a volume reference number was still needed.

This one was a problem. The volume reference number returned by SFGetFile, as previously noted, was not correct. In fact, it just contained random garbage from whatever previously occupied that bit of memory. And calling the routine GetVol, which returns the current default volume number, was of no use either because it was not being changed.

The answer (after unmentionable frustration) came from another global variable, ‘SFSaveDisk’ which contains the “Negative of volume reference number, used by Standard File Package (word)”. The same technique can be applied to this global as above, except it is to be treated as an integer (not a *long* integer) and contains a negative value. Be sure not to negate the value pointer, just the de-referenced value.

Having the volume reference number and the directory ID enables a working directory to be created. The necessary routines can be found on page IV-158 of Inside Macintosh, at the very end of the routine listings.

The PBOpenWD routine will return a volume reference number which is actually a working directory reference number, but may be treated as a volume reference number in all those routines originally created before HFS was envisaged. Compatibility is maintained, but at an overhead cost.

The PBOpenWD routine also requires a program identifier when creating the working directory. This is because working directories should be disposed with PBCloseWD when no longer required, and if any other program happened to be using that working directory, they would get a severe headache (very important when using MultiFinder). To avoid this problem, each program passes a unique ID to indicate which program wants the working directory closed. Theoretically, the value passed should be derived from the program creator used in the Finder, which is theoretically unique throughout the world. But it is doubtful. Instead, I just used the same value with which the invisible ‘global’ window was identified.

After various standard difficulties such as misnamed variables and forgotten negations, the routine finally worked. The result was a totally functioning dialog box which would accept a painting file, frame the picture and then return to the user without destroying the dialog box.

Was it all worth it? Certainly not! The time spent programming was far less than would ever be spent framing pictures by hand. But then again, that's what hacking is all about.

— John Rotenstein

Identification:

Adventures In Programming #1: Frame•It
Date: 3 November 1988

John Rotenstein
PO Box 165
Double Bay NSW 2028
AUSTRALIA.