Macintosh Technical Notes

Developer Technical Support

#295: Feeder Fodder

Written by: Zz Zimmerman April 1991

This Technical Note discusses the new Feeder button available in the 6.1, and 7.0 versions of the LaserWriter driver. This Feeder button mechanism allows developers to insert code into the LaserWriter driver to support a sheet feeder connected to a LaserWriter. This Note provides a description of the button, as well as information required to implement one.

Introduction

The LaserWriter driver now implements a standard method for handling sheet feeders. Most LaserWriter sheet feeders need a way to present a user with a dialog as well as a way to download the PostScript® code necessary to control the feeder. In the past, most manufacturers resorted to modifying the LaserWriter driver's code resources; however, this functionality is now possible without the need to patch existing resources in the driver—by adding three new resources.

When the LaserWriter driver notices these three special resources in its resource fork, it displays a Feeder button in the lower right corner of the Print dialog box. It is important to note that this feature is not provided for general application use, but rather only for developers of sheet feeders and other LaserWriter add-on devices. The button is always labeled Feeder, and there can be only one set of Feeder resources in the LaserWriter driver. Because of this restriction, you should not attempt to use this feature to implement anything other than a sheet feeder.

The first special resource contains code to implement the user interface of the feeder, and the other two contain the PostScript code required to drive the feeder. When an application calls the LaserWriter driver to display the Print dialog box, the driver looks for three resources of type `'feed'` and displays the Feeder button in the lower right corner of the dialog box if they are found. If no `'feed'` resources are available, it does not display the Feeder button.

When a user selects Print, the driver displays the standard Print dialog box with the Feeder button. If a user clicks on the Feeder button, the driver displays a dialog box in front of the Print dialog box, which allows the user to configure the feeder, then returns to the Print dialog box once the user confirms or cancels the feeder configuration. This feeder dialog box should **not** contain an option to print, as this could override choices made in the standard Print dialog box.

Implementation

To handle interaction with the user, you must install a resource of type `'feed'` (`ID = -8192`) into the LaserWriter driver with the code required to manage the dialog box. Like all Printing Manager code resources, this resource begins with a jump table, followed by the actual code. The code is implemented as a procedure that is passed a single parameter. This parameter is a

rectangle defining the page size selected by the user. This page size is equivalent to the `rPaper` rectangle in the print record, meaning it defines the actual page size, not just the printable area. The rectangle is expressed in 72 dpi coordinates and has a negative origin.

Go Ahead and Jump

The jump table consists of a 68000 `JMP` instruction that jumps to the proper offset in the resource. In this case, there is only one routine, so the code starts immediately following the jump table. To make this step automatic, the jump table is created using a small assembly language header:

```
        IMPORT Feeder       ; Feeder is NOT defined here...
Start   MAIN   EXPORT        ; This is the main entry point for the linker.
        JMP    Feeder        ; The one jump table entry in this table.
        END
```

This example first imports the `Feeder` procedure, which can be defined externally in the language of your choice. Next is `Start`, the main entry point to the jump table. By passing this label to the link command, the jump table is located at the beginning of the resource. The next line is the actual jump table entry, and the `END` is required to end the assembly-language header. That's all there is to it. The only thing one should have to change in this code fragment would be the name of the routine to import.

The Real MacCode

Now that the jump table is complete, it needs some place to jump. Although MPW C and Pascal examples are provided in this Note, the code can be written in any language. As mentioned before, the code is implemented as a procedure that takes one parameter.

C Definition

In C, this looks like:

```
#include <Types.h>
#include <Quickdraw.h>
void FEEDER(Rect *r)
{
    <Code to present and handle dialog...>
}
```

Since the assembler converts all labels to uppercase, the name of the procedure `FEEDER` must be capitalized to match the case of the label in the jump table. If you are using MPW, you can use the assembler's `CASE` directive to prevent the assembler from capitalizing the labels. Since the rectangle is passed using the C calling convention (i.e., the caller strips the parameter), there is no need to declare the procedure as type `Pascal`. However, this convention does make things a little more interesting for the Pascal version:

Pascal Definition

If you are using MPW, you can use the Pascal compiler's `C` directive to define the Feeder procedure as using the C calling convention. This makes the definition look like this:

```
UNIT FeederSample;
    INTERFACE
        USES Types, Quickdraw;

    PROCEDURE Feeder(r: Rect); C;

    IMPLEMENTATION

    PROCEDURE Feeder(r: Rect);
    BEGIN
        <Code to present and handle dialog...>
    END;
END.
```

So this is straight forward. The procedure Feeder is defined as having one parameter (r), and the C directive is used so that the stack is handled correctly.

If you are using some other development environment that doesn't support the C directive, you have to do a little more work, making the definition look like this:

```
UNIT FeederSample;
    INTERFACE
        USES Types, Quickdraw;

    PROCEDURE Feeder;

    IMPLEMENTATION

    FUNCTION StealRectalParam: Rect;
        INLINE    $2EAE, 0008;        { MOVE.L        8(A6),(A7) }

    PROCEDURE Feeder;
    VAR
        r:    Rect;
    BEGIN
        r := StealRectalParam;

        <Code to present and handle dialog...>
    END;
END.
```

First of all, a unit is defined, and the proper interfaces are included. The definition of the `Feeder` procedure in the `INTERFACE` section is required to make the label available to external modules. In the `IMPLEMENTATION` section, one starts with the `StealRectalParam` function, which is used to get the rectangle passed by the Printing Manager without actually removing it from the stack. If you declared the rectangle as a parameter to the `Feeder` procedure, `Feeder` would remove the parameter before returning, then when the caller tried to remove the parameter again, the stack would be invalid and would cause a crash.

To solve this problem, define the `Feeder` procedure with no parameters. This way, the `Feeder` procedure leaves the parameter right where the caller left it. To get the parameter without removing it from the stack, use the `StealRectalParam` function, which moves the parameter from its normal location (off of `A6`) into the location pointed to by the stack pointer. Since `StealRectalParam` is a function, the stack pointer is already pointing to the return

value. When `StealRectalParam` returns, the `Feeder` routine gets the rectangle parameter, without having removed it from the stack.

Tickled Link

Now you have the jump table and the code, but you still need to link them together. This step is pretty simple, but remember to specify the starting location of the jump table. It looks like the following:

```
Link -w -t feed -c ZzZz -rt feed=-8192 -m START -sg Feeder ∂
    Feeder.a.o ∂              # This file MUST be first.
    Feeder.p.o ∂
    "{Libraries}"Runtime.o ∂
    "{Libraries}"Interface.o ∂
    "{PLibraries}"SANELib.o ∂
    "{PLibraries}"PasLib.o ∂
    -o Feeder
```

First tell the linker to link the code into a `'feed'` resource with an ID of -8192. Next, specify that the resource begins with the code at label `START`. This label was defined by the assembly-language used to generate the jump table. Finally, tell the linker to link all of the code into a single segment named `Feeder`. Obviously, the list of libraries and object files changes depending upon the language used, but the directives to the Link command should remain the same.

Well Fed

So that should be enough to get some code into the `'feed'` resource. Now you need to actually control the feeder during the print job. To do this, you must use PostScript. Your driver should also provide a `'feed'` resource of -8191 containing PostScript code. This code is downloaded by the LaserWriter driver prior to downloading the rest of the job. For those familiar with the `'PREC'` 103 resource, the PostScript in the `'feed'` resource is downloaded **before** the `'PREC'` 103 code. Additional PostScript to be downloaded can be stored in `'feed'` -8190. The PostScript code in the `'feed'` resource should redefine (i.e., patch) the PostScript operators required to handle switching feeders. A likely candidate is the `showpage` operator called at the end of each page. As always, calling or redefining operators defined by the LaserPrep (md) dictionary is not supported. If your device is connected via the LaserWriter's serial port, you can license code from Adobe Systems, Inc. that makes it possible to access the serial port while the LaserWriter is connected over AppleTalk. For more information, contact Adobe at:

Adobe Systems, Inc.
1585 Charleston Road
Mountain View, CA 94043
(415) 961-4400

Once a user has confirmed the configuration from the dialog box, you can edit the PostScript code in the -8191 resource to reflect the choices made. However, when MultiFinder is active, you cannot add or change the size of resources in the LaserWriter driver. For this reason, you should pad the `'feed'` -8191 resource to the maximum size. This padding can be done by adding spaces at the end. If you later need to resize the resource, you can simply overwrite some of the spaces. For more information on printer drivers under MultiFinder, see the *Learning to Drive* document, which is part of "Developer Essentials," and is available on AppleLink, the Apple FTP site, and the Developer CD Series.

You probably need to provide other resources along with the `'feed'` resources; for example, you need `'DITL'` and `'DLOG'` resources for the dialog box. This is okay, but you should be sure to pick unique resource types to avoid confusing the LaserWriter driver. In the case of a Feeder button, you are a guest in someone else's house. It would be wise to avoid rearranging the furniture.

When the LaserWriter driver actually opens the connection to the printer, it looks for `'feed'` resources -8191 and -8190. If they exist, they are downloaded. For those familiar with the `'PREC' 103` method of downloading PostScript code (refer to Technical Note #192, Surprised in LaserWriter 5.2 and Newer), the `'feed'` resources are downloaded **before** the `'PREC' 103` resource. In the case of background printing, the resources are copied into the spool file. Since `'feed'` resources -8191 and -8190 are automatically downloaded by the LaserWriter, they must contain PostScript code. The format of these PostScript resources is a string of ASCII characters without any length byte or terminator. The size of the string is determined by the size of the resource; there are no special size restrictions on these resources, and their only requirement is that they contain PostScript code. To make debugging easier, you should separate lines of PostScript using a carriage return character (13 or $0D hex).

Don't Feed The Print Monster

One last important note concerns the 6.1 version of the LaserWriter driver, shipped on the Macintosh Printing Tools disk included with the Personal LaserWriter LS and StyleWriter. In this version of the driver, the Feeder button will only work when Background printing is disabled. There is a problem with the driver finding the 'feed' resources when Background printing is enabled. This problem has been solved in the 7.0 version of the driver which should be used instead of the 6.1 driver as soon as it is available. Since there is no workaround for the problem, you don't really have to do anything except for possibly noting it in your documentation. Any note should recommend upgrading to the 7.0 version of the driver as soon as possible.

Driving Miss Lasey

Now that you have the two or three `'feed'` resources, the big question is installation. How should you ship these things? There are two methods. The first method involves licensing the LaserWriter driver from Apple Software Licensing (SW.License on AppleLink). This method is only required for "turn-key" systems, where all installation is done for the user and you must ship the LaserWriter driver as part of your product. The second method, which is by and large preferred as it requires no licensing, is to ship your resources in an installer application. This application simply opens the LaserWriter file and adds the necessary resources.

Conclusion

So this should be all the information you need to implement the feed resources for your device. If you intend to drive a sheet feeder through the LaserWriter's serial port, be sure to contact Adobe Systems, Inc. for the most current implementation and licensing information. Although the Feeder button could theoretically be used for other purposes, it will always be labeled "Feeder" by the LaserWriter driver. Because of this consistency, developers should not attempt to extend its functionality beyond support for sheet feeders.

Further Reference:

- *PostScript Language Reference Manual*, Adobe Systems Inc.

- Technical Note #192, Surprised in LaserWriter 5.2 and Newer

PostScript is a registered trademark of Adobe Systems Incorporated.