NOTE: This Technical Note has been retired. Please see the Technical Notes page for current documentation.

Technical Note FL33
Standard File Customization

CONTENTS

This note contains an example program that demonstrates how `SFPGetFile` can be customized using the dialog hook and file filter functions.

[Oct 01 1995]

Introduction

`SFPGetFile`'s dialog hook function and file filter function enable you to customize `SFPGetFile`'s behavior to fit the needs of your application. This technical note consists primarily of a short example program that

1) changes the title of the Open button to '`MyOpen`',

2) adds two radio buttons so that the user can choose to display either text files or text files and applications,

3) adds a quit button to the `SFPGetFile` dialog.

All this is done in a way so as to provide compatibility with the Macintosh File System (MFS), the Hierarchical File System (HFS) and (hopefully) future systems. If you have any questions as you read, the complete source of the demo program and the resource compiler input file is provided at the end of this technical note.

Basically, we need to do three things: add our extra controls to the resource compiler input file, write a dialog hook function, and write a file filter function.

Back to top

Modifying the Resource Compiler Input File

First we need to define a dialog in our resource file. It will be DLOG #128:

```
CONST myDLOGID = 128;
```

and its Rez description is:

```
    resource 'DLOG' (128, purgeable) {
        {0, 0, 200, 349},
        dBoxProc, invisible, noGoAway,
        0x0,
        128,
        "MyGF"
    };
```

The above coordinates (0 0 200 349) are from the standard Standard File dialog. If you need to change the size of the dialog to accommodate new controls, change these coordinates. Next we need to add a DITL in our resource file that is the same as the standard HFS DITL #-4000 except for one item. We need to change the left coordinate of UserItem #4, or part of the dialog will be hidden if we're running under MFS:

```
    /* [4] */
    /* left coordinate changed from 232 to 252 so program will
       work on MFS */
    {39, 252, 59, 347},
    UserItem {
         disabled
    };
```

None of the other items of the DITL should be changed, so that your program will remain as compatible as possible with different versions of Standard File. Finally, we need to add three items to this DITL, two radio buttons and one button (to serve as a quit button)

```
    /* [11] textButton */
    {1, 14, 20, 142},
    RadioButton {
        enabled,
        "Text files only"
    };
    /* [12] textAppButton */
    {19, 14, 38, 176},
    RadioButton {
        enabled,
        "Text and applications"
    };
    /* [13] quitButton */
    {6, 256, 24, 336},
    Button {
        enabled,
        "Quit"
    }
```

Because we've added three items, we need also need to change the item count for the DITL from 10 to 13. We also include the following in our resource file:

```
resource 'STR#' (256) {
    {/* array StringArray: 1 elements */
        /* [1] */
        "MyOpen"
    }
};
```

That's all there is to modify in the resource file.

Back to top

The Dialog Hook

We will be calling SFPGetFile as follows:

```
    SFPGetFile (wher, '', @SFFileFilter, NumFileTypes, MyFileTypes, @MySFHook,
reply, myDLOGID,nil);
```

Notice that we're passing @MySFHook to Standard File. This is the address of our dialog hook routine. Our dialog hook is declared as:

```
    FUNCTION MySFHook(MySFitem: INTEGER; theDialog: DialogPtr):INTEGER;
```

A dialog hook routine allows us to see every item hit before standard file acts on it. This allows us to handle controls that aren't in the standard SFPGetFile's DITL or to handle standard controls in non-standard ways. The dialog hook in this example consists of a case statement with MySFitem as the case selector. Before SFPGetFile displays its dialog, it calls our dialog hook, passing it a -1 as MySFitem. This gives us a chance to initialize our controls. Here we will set the textAppButton to off and the textButton to on:

```
    GetDItem(theDialog,textAppButton,itemType,itemToChange,itemBox);
     SetCtlValue(controlHandle(itemToChange),btnOff);
     GetDItem(theDialog,textButton,itemType,itemToChange,itemBox);
     SetCtlValue(controlHandle(itemToChange),btnOn);
```

and we can also change the title of an existing control. Here's how we might change the title of the Open button using a string that we get from a resource file:

```
    GetIndString(buttonTitle,256,1);
     If buttonTitle <> '' then Begin { if we really got the resource}
         GetDItem(theDialog,getOpen,itemType,itemToChange,itemBox);
         SetCtitle(controlHandle(itemToChange),buttonTitle);
    End; {if} {if we didn't get the resource, don't change the title}
```

Upon completion of our routine that handles the -1, we return a -1 to standard file:

```
MySFHook:= MySFItem; {pass back the same item we were sent}
```

We now have a SFPGetFile dialog displayed that has a quit button and two radio buttons (the textOnly button is on, the TextApp button is off). In addition, the standard Open button has been renamed to MyOpen (or whatever STR is the first string in STR# 256). This was all done before SFPGetFile displayed the dialog. Once our hook is exited, SFPGetFile displays the dialog and calls ModalDialog.

When the user clicks on an item in the dialog, our hook is called again. We can then take appropriate actions, such as highlighting the textButton and un-highlighting the textAppButton if the user clicks on the textButton. At this time, we can also update a global variable (textOnly) that we will use in our file filter function to tell us which files to display. Notice that we can redisplay the file list by returning a 101 as the result of MySFHook. (Standard File for Systems newer than 4.3 will also read the low memory globals, CurDirStore and SFSaveDisk, and switch directories when necessary if a 101 is returned as the result. Thus, you can point Standard File to a new directory, or a new disk.) For example, when the textButton is hit we turn the textAppButton off, turn the textButton on, update the global variable textOnly, and tell SFPGetFile to redisplay the list of files the user can choose from:

```
    if not textOnly then Begin  {if textOnly was turned off, turn it on now}
         GetDItem(theDialog,textAppButton,itemType,itemToChange,itemBox);
         SetCtlValue(controlHandle(itemToChange),btnOff);
         GetDItem(theDialog,textButton,itemType,itemToChange,itemBox);
         SetCtlValue(controlHandle(itemToChange),btnOn);
         textOnly:=TRUE;     {toggle our global variable for use in the filter}
         MySFHook:= reDrawList;{101}       {we must tell SF to redraw the list}
     End;  {if not textOnly}
```

If our quit button is hit, we can pass `SFPGetFile` back the cancelbutton:

```
    MySFHook:= getCancel;
```

If one of `SFPGetFile`'s standard items is hit, it is very important to pass that item back to `SFPGetFile`:

```
    MySFHook:= MySFItem; {pass back the same item we were sent}
```

Back to top

The File Filter

Remember, we called `SFPGetFile` as follows:

```
    SFPGetFile (wher, '', @SFFileFilter, NumFileTypes,
                 MyFileTypes, @MySFHook, reply,myDLOGID,nil);
```

Notice that we're passing `@SFFileFilter` to `SFPGetFile`. This is the address of our file filter routine. A file filter is declared as:

```
    FUNCTION SFFileFilter (p: ParmBlkPtr): BOOLEAN;
```

A file filter routine allows us to control which files `SFPGetFile` will display for the user. Our file filter is called for every file (of the type(s) specified in the typelist) on an MFS disk, or for every file (of the type(s) specified in the typelist) in the current directory on an HFS disk. In addition, `SFPGetFile` displays HFS folders for us automatically. Our file filter selects which files should appear in the dialog by returning FALSE for every file that should be shown and TRUE for every file that shouldn't.

For example, using our global variable `textOnly` (which we set in our dialog hook, remember?):

```
    FUNCTION SFFileFilter(p:parmBlkPtr):boolean;

    Begin {SFFileFilter}
        SFFileFilter:= TRUE;                        {Don't show it -- default}

        if textOnly then
           if p^.ioFlFndrInfo.fdType = 'TEXT' then
               SFFileFilter:= FALSE                          {Show TEXT files only}
           else Begin
           End  {dummy else}
       else
           if (p^.ioFlFndrInfo.fdType = 'TEXT') or
                    (p^.ioFlFndrInfo.fdType = 'APPL') then
                         SFFileFilter:= FALSE;       { show TEXT or APPL files}
    End;  {SFFileFilter}
```

SFPGetFile calls the file filter after it has called our dialog hook. Please remember that the filter is passed every file of the types specified in the typelist (MyFileTypes). If you want your application to be able to choose from all files, pass SFPGetFile a -1 as numTypes. For information about parameters to SFPGetFile that haven't been discussed in this technical note, see the Standard File Package chapter of *Inside Macintosh* .

That's all there is to it!! Now that you know how to modify SFPGetFile to suit your needs, please don't rush off and load up the dialog window with all kinds of controls and text. Please make sure that you adhere to Macintosh interface standards. Similar techniques can be used with SFGetFile, SFPutFile and SFPPutFile.

The complete source of the demo program and of the resource compiler input file follows:

Back to top

MPW Pascal Source

```
{$R-}

{Jim Friedlander    Macintosh Technical Support 9/30/85}

program SFGetDemo;

USES
     MemTypes,
     QuickDraw,
     OSIntf,
     ToolIntf,
     PackIntf;
{$D+}

CONST
  myDLOGID = 128; {ID of our dialog for use with SFPGetFile}

VAR
  wher: Point; { where to display dialog }
  reply: SFReply; { reply record }
  textOnly: BOOLEAN; { tells us which files are currently being displayed}
  myFileTypes: SFTypeList; { we won't actually use this }
  NumFileTypes: integer;


{-----------------------------------------------------------------------------}
FUNCTION MySFHook(MySFitem:integer; theDialog:DialogPtr): integer;

CONST
```

```
  textButton        = 11;      {DITL item number of textButton}
  textAppButton    = 12;      {DITL item number of textAppButton}
  quitButton        = 13;      {DITL item number of quitButton}


  stayInSF         =  0;        {if we want to stay in SF after getting an
  Open hit, we can pass back a 0 from our hook (not used inthis example) }
  firstTime        = -1;        {the first time our hook is called, it is
                    passed a -1}


{The following line is the key to the whole routine -- the magic 101!!}
  reDrawList     = 101;    {returning 101 as item number will cause
                     the file list to be recalculated}
  btnOn        = 1;        {control value for on}
  btnOff         = 0;        {control value for off}

VAR
  itemToChange: Handle; {needed for GetDItem and SetCtlValue}
  itemBox:Rect;            {needed for GetDItem}
  itemType:integer;                {needed for GetDItem}
  buttonTitle: Str255;           {needed for GetIndString}

Begin {MySFHook}
  case MySFItem of

    firstTime: Begin { before the dialog is drawn, our hook gets
                     called with a -1 (firstTime) as the item so
                    we can change things like button titles,
                    etc. }

{Here we will set the textAppButton to OFF, the textButton to ON}
        GetDItem(theDialog,textAppButton,itemType,itemToChange,itemBox);
        SetCtlValue(controlHandle(itemToChange),btnOff);
        GetDItem(theDialog,textButton,itemType,itemToChange,itemBox);
        SetCtlValue(controlHandle(itemToChange),btnOn);

        GetIndString(buttonTitle,256,1); {get the button title from a resource file}
        If buttonTitle <> '' then Begin        { if we got the resource}
            GetDItem(theDialog,getOpen,itemType,itemToChange,itemBox);
                            {get handle to open button}
            SetCtitle(controlHandle(itemToChange),buttonTitle);
        End; {if}                {if we can't get the resource, we
                        just won't change the open button's
                        title}
        MySFHook:= MySFItem;        {pass back the same item we were
                         sent}
    End;  {firstTime}


{Here we will turn the textAppButton OFF, the textButton ON and redraw the list}
    textButton: Begin
        if not textOnly then Begin
            GetDItem(theDialog,textAppButton,itemType,itemToChange,itemBox);
            SetCtlValue(controlHandle(itemToChange),btnOff);
            GetDItem(theDialog,textButton,itemType,itemToChange,itemBox);
            SetCtlValue(controlHandle(itemToChange),btnOn);
            textOnly:=TRUE;
            MySFHook:= reDrawList;    {we must tell SF to redraw the list}
        End;  {if not textOnly}
    End;  {textOnlyButton}
```

```
{Here we turn the textButton OFF, the textAppButton ON and redraw the list}
    textAppButton: Begin
        if textOnly then Begin
            GetDItem(theDialog,TextButton,itemType,itemToChange,itemBox);
            SetCtlValue(controlHandle(itemToChange),BtnOff);
            GetDItem(theDialog,TextAppButton,itemType,itemToChange,itemBox);
            SetCtlValue(controlHandle(itemToChange),BtnOn);
            TextOnly:=FALSE;
            MySFHook:= reDrawList;    {we must tell SF to redraw the list}
          End;  {if not textOnly}
        End;  {textAppButton}

    quitButton:  MySFHook:= getCancel;    {Pass SF back a 'cancel button'}

{!!!!very important !!!! We pass SF's 'standard' hits back to SF}
    otherwise  Begin
                  MySFHook:= MySFItem;    { the item hit was one of SF's
                        standard items... }
    End;  {otherwise}            { so just pass it back}
  End;  {case}
End;  {MySFHook}

{-------------------------------------------------------------------------------}

FUNCTION SFFileFilter(p:parmBlkPtr):boolean; {general strategy -- check value
                           of global vartextOnly to see
                           which files to display}

Begin {SFFileFilter}
  SFFileFilter:= TRUE; {Don't show it -- default}

  if textOnly then
      if p^.ioFlFndrInfo.fdType = 'TEXT' then
          SFFileFilter:= FALSE        {Show it}
      else Begin
      End  {dummy else}
  else
      if (p^.ioFlFndrInfo.fdType = 'TEXT') or (p^.ioFlFndrInfo.fdType = 'APPL') then
          SFFileFilter:= FALSE;        {Show it}
End;  {SFFileFilter}


{-------------------------------------------------------------------------------}

Begin {main program}
   InitGraf (@thePort);
   InitFonts;
   InitWindows;
   TEInit;
   InitDialogs (nil);

   wher.h:=80;
   wher.v:=90;
   NumFileTypes:= -1;          {Display all files}

{ we don't need to initialize MyFileTypes, because we want to get a chance to filter
every file on the disk in SFFileFilter - we will decide what to show and what not to.
If you want to filter just certain types of files by name, you would set up MyFileTypes
and NumFileTypes accordingly}
```

```
   repeat
    {each time SFPGetFile is called, display will be text-only files}
       textOnly:= TRUE;

       SFPGetFile (wher, '', @SFFileFilter, NumFileTypes, MyFileTypes,
       @MySFHook,reply,myDLOGID,nil);

   until reply.good = FALSE;
{until a cancel button hit ( or a Quit button -- thanks to our dialog hook ) }
End.
```

Back to top

MPW C Source

```
#include <Types.h>
#include <Quickdraw.h>
#include <Resources.h>
#include <Fonts.h>
#include <Windows.h>
#include <Menus.h>
#include <TextEdit.h>
#include <Events.h>
#include <Dialogs.h>
#include <Packages.h>
#include <Files.h>
#include <Controls.h>
#include <ToolUtils.h>
#define    textButton    11     /*DITL item number of textButton*/

#define    textAppButton    12    /*DITL item number of textAppButton*/

#define    quitButton    13    /*DITL item number of quitButton*/

#define    stayInSF    0    /*if we want to stay in SF after getting an
                    Open hit, we can pass back a 0 from our
                    hook  (not used in this example) */

#define    firstTime    -1    /* the first time our hook is called, it is
                    passed a -1*/


#define    reDrawList    101    /* This line is the key to the whole
                    routine the magic 101!! returning 101 as
                    item number will cause the file list to be
                    recalculated*/

#define    btnOn        1    /*control value for on*/

#define    btnOff        0    /*control value for off*/

#define     myDLOGID    128    /*resource ID of our DLOG for SFPGetFile*/

Boolean        textOnly; /* tells us which files are currently being displayed*/

main()
{    /*main program*/

        pascal short MySFHook();
        pascal Boolean flFilter();
        Point      wher;            /* where to display dialog*/
        SFReply      reply;                /* reply record */
        SFTypeList    myFileTypes;          /* we won't use this */
        short int      NumFileTypes = -1;

    InitGraf(&qd.thePort);
    InitFonts();
    FlushEvents(everyEvent, 0);
    InitWindows();
    TEInit();
    InitDialogs(nil);
    InitCursor();



   wher.h=80;
   wher.v=90;

/* we don't need to initialize MyFileTypes, because we want to get a chance to
ilter every file on  the disk in flFilter - we will decide what to show and what
not to. if you want to filter just certain types of files by name, you would set
up MyFileTypes and NumFileTypes accordingly*/

   do {
    textOnly= true;     /*each time SFPGetFile is called, initial
                    display will be text-only files*/

       SFPGetFile(&wher,"",flFilter, NumFileTypes,myFileTypes,
            MySFHook, &reply,myDLOGID,nil);

   } while (reply.good);    /*until we get a cancel button hit
                 (or a Quit button in this case ) */
} /* main */


pascal short MySFHook(short MySFItem,DialogPtr theDialog)
{

Handle  itemToChange;            /*needed for GetDItem and SetCtlValue*/
Rect    itemBox;                /*needed for GetDItem*/
short    itemType;                    /*needed for GetDItem*/
char    buttonTitle[256];        /*needed for GetIndString*/

  switch (MySFItem)
  {
    case firstTime:
        /* before the dialog is drawn, our hook gets called with a -1
         (firstTime)... as the item so we can change things like button
         titles, etc. Here we set the textAppButton to OFF, the
         textButton to ON*/
        GetDItem(theDialog,textAppButton,&itemType,&itemToChange,&itemBox);
        SetCtlValue(itemToChange,btnOff);
        GetDItem(theDialog,textButton,&itemType,&itemToChange,&itemBox);
        SetCtlValue(itemToChange,btnOn);

        /*get the button title from a resource file*/
        GetIndString((char *)buttonTitle,256,1);
        if (buttonTitle[0] != 0)    /* check the length of the p-string
                        to see if we got the resource*/
        {
```

```
    /*get a handle to the open button*/
    GetDItem(theDialog,getOpen,&itemType,&itemToChange,&itemBox);
            SetCTitle(itemToChange,buttonTitle);
          } /*if we can't get the resource, we just
                      won't change the open button's title*/

        return MySFItem; /*pass back the same item we were sent*/
        break;

/*Here we turn the textAppButton OFF, the textButton ON and redraw the list*/
    case textButton:
        if (!textOnly) {
GetDItem(theDialog,textAppButton,&itemType,&itemToChange,&itemBox);
           SetCtlValue(itemToChange,btnOff);
GetDItem(theDialog,textButton,&itemType,&itemToChange,&itemBox);
                SetCtlValue(itemToChange,btnOn);
                textOnly=true;
                    return(reDrawList);    /*must tell SF to redraw the list*/
               }                  /*if !textOnly*/
        return MySFItem;
        break;             /*Here we will turn the textButton
                        OFF, the textAppButton ON and redraw
                        the list*/
    case textAppButton:
        if (textOnly)
            {
GetDItem(theDialog,textButton,&itemType,&itemToChange,&itemBox);
                    SetCtlValue(itemToChange,btnOff);
GetDItem(theDialog,textAppButton,&itemType,&itemToChange,&itemBox);
                    SetCtlValue(itemToChange,btnOn);
                    textOnly=false;
                    return(reDrawList);    /* must tell SF to redraw the list*/
           }  /*if not textOnly*/
        return MySFItem;     /*pass back the same item we were sent*/
        break;

    case quitButton:
        return(getCancel); /*Pass SF back a 'cancel button'*/

/*!!!very important !!!! We must pass SF's 'standard' item hits back to SF*/
    default:
        return(MySFItem); /* the item was one of SF's standard items... */
  }  /*switch*/
      return(MySFItem); /* return what we got */
}  /*MySFHook*/


pascal Boolean flFilter(pb)
FileParam    *pb;

{

/* is this gross or what??? */
return((textOnly) ? ((pb->ioFlFndrInfo.fdType) != 'TEXT') :
    ((pb->ioFlFndrInfo.fdType) != 'TEXT') &&
    ((pb->ioFlFndrInfo.fdType) != 'APPL'));
}  /*flFilter*/
```

Back to top

Rez Input File

```
#include "types.r"

resource 'STR#' (256) {
    {
        "MyOpen"
    }
};

resource 'DLOG' (128, purgeable) {
    {0, 0, 200, 349},
    dBoxProc,
    invisible,
    noGoAway,
    0x0,
    128,
    "MyGF"
};

resource 'DITL' (128, purgeable) {
    {
        /* [1] */
        {138, 256, 156, 336},
        Button { enabled, "Open" };
        /* [2] */
        {1152, 59, 1232, 77},
        Button { enabled, "Hidden" };
        /* [3] */
        {163, 256, 181, 336},
        Button { enabled, "Cancel" };
        /* [4] */
        {39, 252, 59, 347},
        UserItem { disabled };
        /* [5] */
        {68, 256, 86, 336},
        Button { enabled, "Eject" };
        /* [6] */
        {93, 256, 111, 336},
        Button { enabled, "Drive" };
        /* [7] */
        {39, 12, 185, 230},
        UserItem { enabled };
        /* [8] */
        {39, 229, 185, 245},
        UserItem { enabled };
        /* [9] */
        {124, 252, 125, 340},
        UserItem { disabled };
        /* [10] */
        {1044, 20, 1145, 116},
        StaticText { disabled, "" };
        /* [11] */
        {1, 14, 20, 142},
        RadioButton { enabled, "Text files only" };
        /* [12] */
        {19, 14, 38, 176},
        RadioButton { enabled, "Text and applications" };
        /* [13] */
        {6, 256, 24, 336},
        Button { enabled, "Quit" }
    }
};
```

Back to top

References

The Standard File Package

Back to top

Downloadables

	Acrobat version of this Note (K)	Download

Back to top