New Technical Notes

Macintosh

Developer Support

Print Dialogs: Adding Items

Imaging

Revised by: March 1988
Written by: Ginger Jernigan November 1986
Lew Rollins

This technical note discusses how to add your own items to the Printing Manager's dialogs.

When the Printing Manager was initially designed, great care was taken to make the interface to the printer drivers as generic as possible in order to allow applications to print without being device-specific. There are times, however, when this type of non-specific interface interferes with the flexibility of an application. An application may require additional information before printing which is not part of the general Printing Manager interface. This technical note describes a method that an application can use to add its own items to the existing style and job dialogs.

Before continuing, you need to be aware of some guidelines that will increase your chances of being compatible with the printing architecture in the future:

- Only add items to the dialogs as described in this technical note. Any other methods will decrease your chances of survival in the future.

- Do not change the position of any item in the current dialogs. This means don't delete items from the existing item list or add items in the middle. Add items **only at the end** of the list.

- Don't count on an item retaining its current position in the list. If you depend on the Draft button being a particular number in the ImageWriter's style dialog item list, and we change the Draft button's item number for some reason, your program may no longer function correctly.

- Don't use more than half the screen height for your items. Apple reserves the right to expand the items in the standard print dialogs to fill the top half of the screen.

- If you are adding lots of items to the dialogs (which may confuse users), you should consider having your own separate dialog in addition to the existing Printing Manager dialogs.

The Heart

Before we talk about how the dialogs work, you need to know this: at the heart of the printer dialogs is a little-known data structure partially documented in the MacPrint interface file. It's a record called `TPrDlg` and it looks like this:

```
TPrDlg = RECORD   {Print Dialog: The Dialog Stream object.}
   dlg       : DialogRecord;   {dialog window}
   pFltrProc : ProcPtr;        {filter proc.}
   pItemProc : ProcPtr;        {item evaluating proc.}
   hPrintUsr : THPrint;        {user's print record.}
   fDoIt     : BOOLEAN;
   fDone     : BOOLEAN;
   lUser1    : LONGINT;        {four longs reserved by Apple}
   lUser2    : LONGINT;
   lUser3    : LONGINT;
   lUser4    : LONGINT;
   iNumFst   : INTEGER;        {numeric edit items for std filter}
   iNumLst   : INTEGER;
{... plus more stuff needed by the particular printing dialog.}
END;
TPPrDlg = ^TPrDlg;             {== a dialog ptr}
```

All of the information pertaining to a print dialog is kept in the `TPrDlg` record. This record will be referred to frequently in the discussion below.

How the Dialogs Work

When your application calls `PrStlDialog` and `PrJobDialog`, the printer driver actually calls a routine called `PrDlgMain`. This function is declared as follows:

```
FUNCTION PrDlgMain (hprint: THPrint; pDlgInit: ProcPtr): BOOLEAN;
```

`PrDlgMain` first calls the `pDlgInit` routine to set up the appropriate dialog (in `Dlg`), dialog hook `(pItemProc)` and dialog event filter `(pFilterProc)` in the `TPrDlg` record (shown above). For the job dialog, the address of `PrJobInit` is passed to `PrDlgMain`. For the style dialog, the address of `PrStlInit` is passed. These routines are declared as follows:

```
FUNCTION PrJobInit (hPrint: THPrint): TPPrDlg;
FUNCTION PrStlInit (hPrint: THPrint): TPPrDlg;
```

After the initialization routine sets up the `TPrDlg` record, `PrDlgMain` calls `ShowWindow` (the window is initially invisible), then it calls `ModalDialog`, using the dialog event filter pointed to by the `pFltrProc` field. When an item is hit, the routine pointed to by the `pItemProc` field is called and the items are handled appropriately. When the OK button is hit (this includes pressing Return or Enter) the print record is validated. The print record is not validated if the Cancel button is hit.

How to Add Your Own Items

To modify the print dialogs, you need to change the `TPrDlg` record before the dialog is drawn on the screen. You can add your own items to the item list, replace the addresses of the standard dialog hook and event filter with the addresses of your own routines and then let the dialog code continue on its merry way.

`TPrDlg` record for you and return a pointer to that record. Then call `PrDlgMain` directly, passing in the address of your own initialization function. The example code's initialization function adds items to the dialog item list, saves the address of the standard dialog hook (in our global variable `prPItemProc`) and puts the address of our dialog hook into the `pItemProc` field of the `TPrDlg` record. Please note that your dialog hook **must** call the standard dialog hook to handle all of the standard dialog's items.

Note: If you wish to have an event filter, handle it the same way that you do a dialog hook.

Now, here is an example (written in MPW Pascal) that modifies the job dialog. The same code works for the style dialog if you globally replace 'Job' with 'Stl'. Also included is a function (`AppendDITL`) provided by Lew Rollins (originally written in C, translated for this technical note to MPW Pascal) which demonstrates a method of adding items to the item list, placing them in an appropriate place, and expanding the dialog window's rectangle.

The MPW Pascal Example Program

```
PROGRAM ModifyDialogs;

	USES
		{$LOAD PasDump.dump}
		MemTypes,QuickDraw,OSIntf,ToolIntf,PackIntf,MacPrint;

	CONST
		MyDITL        = 256;
		MyDFirstBox   = 1;          {Item number of first box in DITL}
		MyDSecondBox = 2;

	VAR
		PrtJobDialog: TPPrDlg;     { pointer to job dialog }
		hPrintRec    : THPrint;    { Handle to print record }
		FirstBoxValue,             { value of first additional box }
		SecondBoxValue: Integer;   { value of second addtl. box }
		prFirstItem,               { save our first item here }
		prPItemProc : LongInt;     { need to store old itemProc here }
		itemType     : Integer;    { need for GetDItem/SetDItem calls }
		itemH        : Handle;
		itemBox             : Rect;
		err                 : OSErr;
{-------------------------------------------------------------------------}

	PROCEDURE _DataInit;
		EXTERNAL;

{-------------------------------------------------------------------------}
```

```
	PROCEDURE CallItemHandler(theDialog: DialogPtr; theItem: Integer;
				theProc: LongInt);
		INLINE $205F		{ MOVE.L (A7)+,A0}
		,$4E90;			{ JSR (A0) }

{ this code pops off theProc and then does a JSR to it, which puts the
real return address on the stack. }

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

	FUNCTION AppendDITL(theDialog: DialogPtr; theDITLID: Integer): Integer;
	{ version 0.1 9/11/86 Lew Rollins of Human-Systems Interface Group}
	{ this routine still needs some error checking }

{ This routine appends all of the items of a specified DITL
onto the end of a specified DLOG — We don't even need to know the format
of the DLOG }

{ this will be done in 3 steps:
1. append the items of the specified DITL onto the existing DLOG
2. expand the original dialog window as required
3. return the adjusted number of the first new user item
}
	TYPE
		DITLItem = RECORD   { First, a single item }
			{ Handle or procedure pointer for this item }
				itmHndl: Handle;
			{ Display rectangle for this item }
				itmRect: Rect;
			{ Item type for this item — 1 byte }
				itmType: SignedByte;
			{ Length byte of data }
				itmData: ARRAY [0..0] OF SignedByte;
		END;   {DITLItem}
		pDITLItem     = ^DITLItem;
		hDITLItem     = ^pDITLItem;

		ItemList = RECORD   { Then, the list of items }
			 { Number of items minus 1 }
				dlgMaxIndex: Integer;
			{ Array of items }
				DITLItems: ARRAY [0..0] OF DITLItem;                END;
{ItemList}
		pItemList     = ^ItemList;
		hItemList     = ^pItemList;

		IntPtr        = ^Integer;

	VAR
		offset : Point;     { Used to offset rectangles appended items}
		maxRect: Rect;      { Used to track increases in window size }
		hDITL : hItemList;  { Handle to DITL being appended }
		pItem : pDITLItem;  { Pointer to current item being appended }
		hItems: hItemList;  { Handle to DLOG's item list }
		firstItem: Integer; { offset to first item is to be appended }
		newItems: Integer;  { Count of new items }
		dataSize : Integer; { Size of data for current item }
		i : Integer;        { Working index }
		USB: RECORD         { needed because itmData[0] is unsigned}
			CASE Integer OF
				1:
					(SBArray: ARRAY [0..1] OF SignedByte);
```

```
                        2:
                                (Int: Integer);
            END;    {USB}

      BEGIN         {AppendDITL}
{
```
Using the original DLOG

1. Remember the original window Size.
2. Set the offset Point to be the bottom of the original window.
3. Subtract 5 pixels from bottom and right, to be added
   back later after we have possibly expanded window.
4. Get working Handle to original item list.
5. Calculate our first item number to be returned to caller.
6. Get locked Handle to DITL to be appended.
7. Calculate count of new items.

```
}
            maxRect := DialogPeek(theDialog)^.window.port.portRect;
            offset.v := maxRect.bottom;
            offset.h := 0;
            maxRect.bottom := maxRect.bottom - 5;
            maxRect.right := maxRect.right - 5;
            hItems := hItemList(DialogPeek(theDialog)^.items);
            firstItem := hItems^^.dlgMaxIndex + 2;
            hDITL := hItemList(GetResource('DITL',theDITLID));
            HLock(Handle(hDITL));
            newItems := hDITL^^.dlgMaxIndex + 1;
{
```
For each item,

1. Offset the rectangle to follow the original window.
2. Make the original window larger if necessary.
3. fill in item Handle according to type.

```
}

            pItem := @hDITL^^.DITLItems;
            FOR i := 1 TO newItems DO BEGIN
                  OffsetRect(pItem^.itmRect,offset.h,offset.v);
                  UnionRect(pItem^.itmRect,maxRect,maxRect);
                  USB.Int := 0;        {zero things out}
                  USB.SBArray[1] := pItem^.itmData[0];

                  { Strip enable bit since it doesn't matter here. }
                  WITH pItem^ DO
                        CASE BAND(itmType,$7F) OF
                              { Can't do anything with user items. }
                              userItem:
                                    itmHndl := NIL;
                                    ctrlItem + btnCtrl,ctrlItem +
                        chkCtrl,ctrlItem + radCtrl:{build
                  Control)
                                          itmHndl := Handle(NewControl(theDialog,
{ theWindow }

                    itmRect, { boundsRect }

                    StringPtr(@itmData[0])^, { title }

                    true, { visible }

                    0,0,1, { value, min, max }
```

```
                        BAND(itmType,$03), { procID }

                        0)); { refCon }
                                        ctrlItem + resCtrl: BEGIN { Get resource
                                    based Control }
                                        itmHndl :=
Handle(GetNewControl(IntPtr(@itmData[1])^, { controlID }

                                theDialog)); { theWindow }
                                        ControlHandle(itmHndl)^^.contrlRect :=
itmRect; {give it the right

                                                                    rectangle}
                                        {An actionProc for a Control
                                     should be installed here}
                                        END;                {Case ctrlItem +
                                                 resCtrl}
                                        statText,editText: { Both need Handle to
                                                    a copy of their
                                        text. }
                                                err := PtrToHand(@itmData[1],
                                     ( Start of data }
                                                                                    itmHndl,
{ Address of new Handle }
                                                                                    USB.Int);
{ Length of text }
                                        iconItem:      { Icon needs resource
                                           Handle. }
                                                pItem^.itmHndl :=
GetIcon(IntPtr(@itmData[1])^); { ICON resID }
                                        picItem:       { Picture needs resource
                                           Handle. }
                                                pItem^.itmHndl :=
Handle(GetPicture(IntPtr(@itmData[1])^));{PICT resID}
                                        OTHERWISE
                                                itmHndl := NIL;
                                    END;                        {Case}

                                dataSize := BAND(USB.Int + 1,$FFFE);
                                {now advance to next item}
                                pItem := pDITLItem(Ptr(ord4(@pItem^) + dataSize +
                    sizeof(DITLItem)));
                        END;                                    {for}
                        err := PtrAndHand
                        (@hDITL^^.DITLItems,Handle(hItems),GetHandleSize(Handle(hDITL)));
                        hItems^^.dlgMaxIndex := hItems^^.dlgMaxIndex + newItems;
                        HUnlock(Handle(hDITL));
                        ReleaseResource(Handle(hDITL));
                        maxRect.bottom := maxRect.bottom + 5;
                        maxRect.right := maxRect.right + 5;
                        SizeWindow(theDialog,maxRect.right,maxRect.bottom,true);
                        AppendDITL := firstItem;
                    END;                                        {AppendDITL}

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

              PROCEDURE MyJobItems(theDialog: DialogPtr; itemNo: Integer);
{
```

```
This routine replaces the routine in the pItemProc field in the
       TPPrDlg record.  The steps it takes are:
       1. Check to see if the item hit was one of ours. This is done by "localizing"
                the number, assuming that our items are numbered from 0..n
       2. If it's one of ours  then case it and Handle appropriately
       3. If it isn't one of ours then call the old item handler
}

              VAR
                     MyItem,firstItem: Integer;
                     thePt        : Point;
                     thePart             : Integer;
                     theValue     : Integer;
                     debugPart    : Integer;

              BEGIN                                    {MyJobItems}
                     firstItem := prFirstItem; { remember, we saved this in
                                   myJobDlgInit }
                     MyItem := itemNo - firstItem + 1; { "localize" current item No }
                     IF MyItem > 0 THEN BEGIN { if localized item > 0, it's one of
                                   ours }
                            { find out which of our items was hit }
                            GetDItem(theDialog,itemNo,itemType,itemH,itemBox);
                            CASE MyItem OF
                                   MyDFirstBox: BEGIN
                                          { invert value of FirstBoxValue and
                               redraw it }
                                          FirstBoxValue := 1 - FirstBoxValue;
                                          SetCtlValue(ControlHandle(itemH),FirstBoxValue);
                                   END;                        {case MyDFirstBox}
                                   MyDSecondBox: BEGIN
                                          { invert value of SecondBoxValue and
                                 redraw it }
                                          SecondBoxValue := 1 - SecondBoxValue;
                                          SetCtlValue(ControlHandle(itemH),SecondBoxValue);
                                   END;                        {case
                                                 MyDSecondBox}
                                   OTHERWISE
                                          Debug;         { OH OH — We got an item
                                             we didn't expect }
                            END;                               {Case}
                     END                                       { if MyItem > 0 }
                     ELSE                          { chain to standard item handler,
                                                   whose address is saved

       in prPItemProc }
                            CallItemHandler(theDialog,itemNo,prPItemProc);
              END;                                             { MyJobItems }

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

       FUNCTION MyJobDlgInit(hPrint: THPrint): TPPrDlg;
{
This routine appends items to the standard job dialog and sets up the
user fields of the printing dialog record TPRDlg
This routine will be called by PrDlgMain
```

```
This is what it does:
1. First call PrJobInit to fill in the TPPrDlg record.
2. Append our items onto the old DITL. Set them up appropriately.
3. Save the address of the old item handler and replace it with ours.
4. Return the Fixed dialog to PrDlgMain.
}

            VAR
                  firstItem    : Integer; { first new item number }

            BEGIN                                       {MyJobDlgInit}
                  firstItem := AppendDITL(DialogPtr(PrtJobDialog),MyDITL);

                  prFirstItem := firstItem; { save this so MyJobItems can
                                              find it }

                  { now we'll set up our DITL items — The "First Box" }
                  GetDItem(DialogPtr(PrtJobDialog),firstItem,itemType,itemH,itemBox);
                  SetCtlValue(ControlHandle(itemH),FirstBoxValue);

                  { now we'll set up the second of our DITL items — The "Second
              Box" }
                  GetDItem(DialogPtr(PrtJobDialog),firstItem +
      1,itemType,itemH,itemBox);
                  SetCtlValue(ControlHandle(itemH),SecondBoxValue);

{ Now comes the part where we patch in our item handler.  We have to save
the old item handler address, so we can call it if one of the standard
items is hit, and put our item handler's address
in pItemProc field of the TPrDlg struct}

                  prPItemProc := LongInt(PrtJobDialog^.pItemProc);

                  { Now we'll tell the modal item handler where our routine is }
                  PrtJobDialog^.pItemProc := ProcPtr(@MyJobItems);

                  { PrDlgMain expects a pointer to the modified dialog to be
        returned.... }
                  MyJobDlgInit := PrtJobDialog;

            END;                                                {myJobDlgInit}

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

      FUNCTION Print: OSErr;

            VAR
                  bool                : BOOLEAN;

            BEGIN                                       {Print}
                  hPrintRec := THPrint(NewHandle(sizeof(TPrint)));
                  PrintDefault(hPrintRec);
                  bool := PrValidate(hPrintRec);
                  IF (PrError <> noErr) THEN BEGIN
                        Print := PrError;
                        Exit(Print);
                  END;                                      {If}

                  { call PrJobInit to get pointer to the invisible job dialog }
                  PrtJobDialog := PrJobInit(hPrintRec);
                  IF (PrError <> noErr) THEN BEGIN
```

```
                        Print := PrError;
                        Exit(Print);
                    END;                                        {If}

{Here's the line that does it all!}
                    IF NOT (PrDlgMain(hPrintRec,@MyJobDlgInit)) THEN BEGIN
                        Print := cancel;
                        Exit(Print);
                    END;                                        {If}

                    IF PrError <> noErr THEN Print := PrError;

                    { that's all for now }

            END;                                                { Print }

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

        BEGIN                                                   {PROGRAM}

            UnloadSeg(@_DataInit);      {remove data initialization code before
                                    any allocations}
            InitGraf(@thePort);
            InitFonts;
            FlushEvents(everyEvent,0);
            InitWindows;
            InitMenus;
            TEInit;
            InitDialogs(NIL);
            InitCursor;

            { call the routine that does printing }
            FirstBoxValue := 0;         { value of our first additional box }
            SecondBoxValue := 0;        { value of our second addtl. box }
            PrOpen;    { Open the Print Manager }
            IF PrError = noErr THEN
                err := Print     { This actually brings up the modified Job
                                dialog }
            ELSE BEGIN
                {tell the user that PrOpen failed}
            END;

            PrClose;    { Close the Print Manager and leave }
        END.
```

The Lightspeed C Example Program

/* NOTE: Apple reserves the top half of the screen (where the current DITL items are located). Applications may use the bottom half of the screen to add items, but should not change any items in the top half of the screen. An application should expand the print dialogs only as much as is absolutely necessary.
*/

```
/* Note: A global search and replace of 'Job' with 'Stl' will produce
      code that modifies the style dialogs */
#include <DialogMgr.h>
#include <MacTypes.h>
#include <Quickdraw.h>
#include <ResourceMgr.h>
#include <WindowMgr.h>
```

```
#include <pascal.h>
#include <printmgr.h>
#define nil 0L


static TPPrDlg PrtJobDialog;            /* pointer to job dialog */

/*     This points to the following structure:

         struct {
               DialogRecord      Dlg;          (The Dialog window)
               ProcPtr           pFltrProc;    (The Filter Proc.)
               ProcPtr           pItemProc;    (The Item evaluating proc. --
                           we'll change this)
               THPrint           hPrintUsr;    (The user's print record.)
               Boolean           fDoIt;
               Boolean           fDone;
                    (Four longs -- reserved by Apple Computer)
               long              lUser1;
               long              lUser2;
               long              lUser3;
               long              lUser4;
  } TPrDlg; *TPPrDlg;
*/

/*     Declare 'pascal' functions and procedures */
pascal Boolean PrDlgMain();             /* Print manager's dialog handler */
pascal TPPrDlg PrJobInit();             /* Gets standard print job dialog. */
pascal TPPrDlg MyJobDlgInit();    /* Our extention to PrJobInit */
pascal void MyJobItems();         /* Our modal item handler */

#define MyDITL 256                /* resource ID of my DITL to be spliced
                                            on to job dialog */


THPrint hPrintRec;                /* handle to print record */
short FirstBoxValue = 0;          /* value of our first additional box */
short SecondBoxValue = 0;         /* value of our second addtl. box */
long prFirstItem;                       /* save our first item here */
long prPItemProc;                 /* we need to store the old itemProc here */

/*-------------------------------------------------------------------------*/
      WindowPtr    MyWindow;
      OSErr  err;
      Str255       myStr;
main()
{
      Rect         myWRect;

      InitGraf(&thePort);
      InitFonts();
      InitWindows();
      InitMenus();
      InitDialogs(nil);
      InitCursor();
      SetRect(&myWRect,50,260,350,340);

      /* call the routine that does printing */
      PrOpen();
      err = Print();

      PrClose();
} /* main */
```

```
/*-------------------------------------------------------------------------*/

OSErr Print()

{
       /* call PrJobInit to get pointer to the invisible job dialog */
       hPrintRec = (THPrint)(NewHandle(sizeof(TPrint)));
       PrintDefault(hPrintRec);
       PrValidate(hPrintRec);
       if (PrError() != noErr)
              return PrError();

       PrtJobDialog = PrJobInit(hPrintRec);
       if (PrError() != noErr)
              return PrError();


       if (!PrDlgMain(hPrintRec, &MyJobDlgInit))      /* this line does all the stuff */
              return Cancel;

       if (PrError() != noErr)
              return PrError();

/* that's all for now */

} /* Print */

/*-------------------------------------------------------------------------*/

pascal TPPrDlg MyJobDlgInit (hPrint)
THPrint hPrint;
/* this routine appends items to the standard job dialog and sets up the
       user fields of the printing dialog record TPRDlg
       This routine will be called by PrDlgMain */
{
       short         firstItem;          /* first new item number */

       short         itemType;           /* needed for GetDItem/SetDItem call */
       Handle itemH;
       Rect          itemBox;

       firstItem = AppendDITL (PrtJobDialog, MyDITL); /*call routine to do
                                     this */

       prFirstItem = firstItem; /* save this so MyJobItems can find it */

/* now we'll set up our DITL items -- The "First Box" */
       GetDItem(PrtJobDialog,firstItem,&itemType,&itemH,&itemBox);
       SetCtlValue(itemH,FirstBoxValue);

/* now we'll set up the second of our DITL items  -- The "Second Box" */
       GetDItem(PrtJobDialog,firstItem+1,&itemType,&itemH,&itemBox);
       SetCtlValue(itemH,SecondBoxValue);

/* Now comes the part where we patch in our item handler.  We have to save
       the old item handler address, so we can call it if one of the standard items is
       hit, and put our item handler's address in pItemProc field of the TPrDlg struct
*/

       prPItemProc = (long)PrtJobDialog->pItemProc;
```

```
/* Now we'll tell the modal item handler where our routine is */
       PrtJobDialog->pItemProc = (ProcPtr)&MyJobItems;

/* PrDlgMain expects a pointer to the modified dialog to be returned.... */
       return PrtJobDialog;

} /*myJobDlgInit*/


/*-------------------------------------------------------------------------*/

/* here's the analogue to the SF dialog hook */

pascal void MyJobItems(theDialog,itemNo)
TPPrDlg      theDialog;
short  itemNo;

{ /* MyJobItems */
       short  myItem;
       short  firstItem;

       short        itemType;           /* needed for GetDItem/SetDItem call */
       Handle itemH;
       Rect         itemBox;

       firstItem = prFirstItem; /* remember, we saved this in myJobDlgInit */
       myItem = itemNo-firstItem+1;     /* "localize" current item No */
       if (myItem > 0)     /* if localized item > 0, it's one of ours */
       {
              /* find out which of our items was hit */
              GetDItem(theDialog,itemNo,&itemType,&itemH,&itemBox);
              switch (myItem)
              {
                     case 1:
                            /* invert value of FirstBoxValue and redraw it */
                            FirstBoxValue ^= 1;
                            SetCtlValue(itemH,FirstBoxValue);
                            break;

                     case 2:
                            /* invert value of SecondBoxValue and redraw it */
                            SecondBoxValue ^= 1;
                            SetCtlValue(itemH,SecondBoxValue);
                            break;
                     default: Debugger(); /* OH OH */
              } /* switch */
       } /* if (myItem > 0) */
       else /* chain to standard item handler, whose address is saved in
                     prPItemProc */
       {
              CallPascal(theDialog,itemNo,prPItemProc);
       }
} /* MyJobItems */
```

The Rez Source

```
#include "types.r"

resource 'DITL' (256) {
	{	/* array DITLarray: 2 elements */
		/* [1] */
		{8, 0, 24, 112},
		CheckBox {
			enabled,
			"First Box"
		};
		/* [2] */
		{8, 175, 24, 287},
		CheckBox {
			enabled,
			"Second Box"
		}
	}
};
```

Further Reference:

- The Printing Manager
- The Dialog Manager