Technical Note TN1080

Adding Items to the Printing Manager's Dialogs

CONTENTS

This Technote, originally *Technote PR 09 - Print Dialogs: Adding Items* , discusses how to add 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 Note describes a method that an application can use to add its own items to the existing style and job dialogs.

This revised Technote reflects the current Macintosh Printing Manager and uses C code instead of the original Pascal.

You can download a complete version of the code in this Technote, **PDlogExpand**, by clicking on the icon in the Downloadables section at the end of this Note.

Updated: [Jul 12 2000]

Printing Manager Compatibility Guidelines for the Future

To begin with, you need to be aware of some guidelines that will increase your chances of being compatible with future revisions of Macintosh Printing Manager:

- Only add items to the dialogs as described in this Technical Note. Any other method will decrease your chances of compatibility 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. Even though LaserWriter 8.4 changes the print dialogs, in order to be compatible with older drivers, you still need to maintain this half-screen rule.
- Under LaserWriter 8.4.x the print dialogs have been completely redesigned. If you add items to the print dialog "legally," your items are added to a pane that has the name of your application. If you don't add them as recommended in this Note, there will be problems. Please see *develop 27 - Print Hints: The All-New LaserWriter Driver Version 8.4* for more details.

Applications should not assume that the print dialog's foreground color is black or that the background color is white when they erase their controls. Furthermore, applications should not change the foreground or background colors because other items in the dialog rely on these colors. If an application changes them, the print dialog may end up with strange colors.

Back to top

The Heart of the Print Dialogs: The TPrDlg Record

Before discussing how the dialogs work, it is important to know that at the heart of the print dialogs is a little-known data structure partially documented in Printing.h. It's a record called `TPrDlg` and it looks like this:

```
struct TPrDlg {
    DialogRecord    Dlg;            /*The Dialog window*/
    ModalFilterUPP  pFltrProc;      /*The Filter Proc.*/
    PItemUPP    pItemProc;      /*The Item evaluating proc.*/
    THPrint     hPrintUsr;      /*The user's print record.*/
    Boolean     fDoIt;
    Boolean     fDone;
    long            lUser1;     /* Four longs reserved by Apple */
                        /* to hang global data.*/
    long            lUser2;     /* More stuff needed by the */
                        /* printing dialog.*/
    long            lUser3;
    long            lUser4;
};

typedef struct TPrDlg TPrDlg;
typedef TPrDlg *TPPrDlg;
typedef TPrDlg *TPPrDlgRef;
```

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.

Back to top

How the Dialogs Work

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

```
    extern pascal Boolean PrDlgMain(THPrint hPrint, PDlgInitUPP pDlgInit);
```

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:

```
    extern pascal TPPrDlgRef PrJobInit(THPrint hPrint);
    extern pascal TPPrDlgRef PrStlInit(THPrint hPrint);
```

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.

Back to top

Adding 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 handle the necessary functions to make the items visible in your print dialog.

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, use the same method that you would for a dialog hook.

Back to top

The C Example Program

Here is an example (written in C) that modifies the job dialog.

Note:
The same code works for the style dialog if you globally replace "Job" with "Stl".

```
    #include <Printing.h>
    #include <Dialogs.h>
    #include <Gestalt.h>
    #include <MixedMode.h>
    #include <SegLoad.h>
    #include <Resources.h>
    #include <ToolUtils.h>
    #include <Fonts.h>
```

```
#include <Lists.h>

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

/*

This points to the following structure:

struct TPrDlg {
    DialogRecord Dlg; // The Dialog window
    ModalFilterUPP pFltrProc; // The Filter Proc.
    PItemUPP pItemProc; // The Item evaluating proc.
    THPrint hPrintUsr; // The user's print record.
    Boolean fDoIt;
    Boolean fDone;
    long lUser1; // Four longs for apps to hang global data.
    long lUser2; // Plus more stuff needed by the particular
    long lUser3; // printing dialog.
    long lUser4;
};
typedef struct TPrDlg TPrDlg;
typedef TPrDlg *TPPrDlg;
typedef TPPrDlg TPPrDlgRef;

*/

/* Declare 'pascal' functions and procedures */

extern short Append_DITL(TPPrDlg, short);
    /* Our AppendDITL function. */
extern pascal TPPrDlg MyJobDlgInit(THPrint);
    /* Our extension to PrJobInit.  */
extern pascal void MyJobItems(TPPrDlg, short);
    /* Our modal item handler.  */
extern OSErr Print(void);
pascal void MyListDraw(WindowPtr theWindow, short itemNo);
    //mxm

#define MyDITL 256  /* resource ID of my DITL to be
                    spliced onto job dialog */
#define kPanePopUp (6)              /* mxm */

ListHandle gList = nil;             /* mxm */
UserItemUPP myDrawListUPP = nil;    /* mxm */

THPrint hPrintRec;  /* handle to print record */
long prFirstItem;   /* save our first item here */
PItemUPP prPItemProc;   /* we need to store the
                        old itemProc here */

WindowPtr MyWindow;
OSErr err;
Str255 myStr;
```

```
#if PRAGMA_ALIGN_SUPPORTED
#pragma options align=mac68k
#endif

typedef struct dialog_item_struct {
    Handle handle;   /* handle or procedure pointer
                     for this item */
    Rect bounds;     /* display rectangle for this item */
    char type;       /* item type - 1 */
    char data[1];    /* length byte of data */
} DialogItem, *DialogItemPtr, **DialogItemHandle;

typedef struct append_item_list_struct {
    short max_index; /* number of items - 1 */
    DialogItem items[1]; /* first item in the array */
} ItemList, *ItemListPtr, **ItemListHandle;

typedef union signed_byte_union {
    short integer;
    char bytes[2];
} ByteAccess;


static Boolean append_exits(void);
static Boolean append_exits(void) {
    long response;

    if (Gestalt(gestaltDITLExtAttr,&response) == noErr) {
        return !!(response & (1 << gestaltDITLExtPresent));
    } else {
        return false;
    }
}


    /* This routine appends all of the items of a specified DITL
    onto the end of a specified DLOG &emdash; 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 */

short Append_DITL(TPPrDlg dialog, short item_list_ID) {
        /* handle to DITL being appended */
    ItemListHandle append_item_list;
    ItemListHandle dlg_item_list; /* handle to DLOG's item list */
    short first_item;

    dlg_item_list = (ItemListHandle)((DialogPeek)dialog)->items;
    first_item = (**dlg_item_list).max_index + 2;

    append_item_list =
        (ItemListHandle)GetResource('DITL', item_list_ID);
    if ( append_item_list == NULL ) {
        DebugStr("\pError loading DITL resource");
        return first_item;
    }

    if (append_exits()) {
        AppendDITL((DialogPtr)dialog,
        (Handle) append_item_list,
        appendDITLBottom);

            /* Make sure to release the DITL resource
            before returning */
        ReleaseResource( (Handle)append_item_list );
        return first_item;
    } else {
        Point       offset;
        Rect        max_rect;
        DialogItemPtr   item;   /* pointer to item being appended */
        short       new_items, data_size, i;
        ByteAccess  usb;
        OSErr       err;

            /* 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.
            */

        if (dialog == NULL) ExitToShell();

        max_rect = ((DialogPeek)dialog)->window.port.portRect;
        offset.v = max_rect.bottom;
        offset.h = 0;
        max_rect.bottom -= 5;
        max_rect.right -= 5;


        HLock((Handle)append_item_list);
        new_items = (**append_item_list).max_index + 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. */

        item = (**append_item_list).items;
```

```
for ( i = 0; i < new_items; i++ ) {
    OffsetRect(&item->bounds, offset.h, offset.v);
    UnionRect(&item->bounds, &max_rect,
        &max_rect);
    usb.integer = 0;
    usb.bytes[1] = item->data[0];

    switch ( item->type & 0x7F ) {
        case ctrlItem + btnCtrl :
        case ctrlItem + chkCtrl :
        case ctrlItem + radCtrl :
            item->handle =
                (Handle)NewControl((DialogPtr) dialog,
                &item->bounds,
                (StringPtr)item->data,
                true, 0, 0, 1, item->type &
                0x03, 0);
            break;

        case ctrlItem + resCtrl :
            item->handle =
                (Handle)GetNewControl(
                    *(short*)(item->data + 1),
                    (DialogPtr) dialog);
            (**(ControlHandle)item->handle).contrlRect =
                item->bounds;
            break;

        case statText :
        case editText :
            err = PtrToHand(item->data + 1,
                &item->handle, usb.integer);
                break;

        case iconItem :
            item->handle =
            GetIcon(*(short*)(item->data + 1));
            break;

        case picItem :
            item->handle =
                (Handle)GetPicture(*(short*)(item->data + 1));
            break;

        default :
            item->handle = NULL;
    }

    data_size = (usb.integer + 1) & 0xFFFE;
    item = (DialogItemPtr)
        ((char*)item + data_size + sizeof(DialogItem));
}

    /* We need to subtract the short below because otherwise the
    size of the DITL count gets factored in twice, and the
```

```
                resulting DTIL has two bytes of garbage appended to it.
                This is a problem with the original TN#95 code as well. */

            err = PtrAndHand((**append_item_list).items,
                (Handle)dlg_item_list,
                GetHandleSize((Handle) append_item_list)
                - sizeof(short));

            (**dlg_item_list).max_index += new_items;
            HUnlock((Handle) append_item_list);
            ReleaseResource((Handle) append_item_list);

            max_rect.bottom += 5;
            max_rect.right += 5;
            SizeWindow((WindowPtr) dialog, max_rect.right,
                max_rect.bottom, true);

            return first_item;
        }
    }



    OSErr Print(void) {
        TPPrPort    pPrPort;
        Rect        aRect;
        TPrStatus   theStatus;
        Boolean     printingShouldNotProceed;
        PDlgInitUPP theInitProcPtr;

            /*  Allocate a print record */
        hPrintRec = (THPrint)(NewHandle(sizeof(TPrint)));
        if ( NULL == hPrintRec )
        {
            return iMemFullErr;
        }

            /*  Fill in the default values  */
        PrintDefault(hPrintRec);

            /*  Make sure the print record is valid and consistent  */
        PrValidate(hPrintRec);
        if (PrError() != noErr)
        {
            DisposeHandle( (Handle)hPrintRec );
            return PrError();
        }

            /* call PrJobInit to get pointer to the invisible
            job dialog */
        PrtJobDialog = PrJobInit(hPrintRec);
        if (PrError() != noErr)
        {
            DisposeHandle( (Handle)hPrintRec );
            return PrError();
        }

            /*  Create a UPP for the dialog init proc   */
        theInitProcPtr = NewPDlgInitProc(MyJobDlgInit);
        if ( NULL == theInitProcPtr )
        {
            DisposeHandle( (Handle)hPrintRec );
            return iMemFullErr;
        }

            /* this line does all the stuff */
        printingShouldNotProceed =
            !PrDlgMain(hPrintRec, theInitProcPtr);

            /* Get rid of our user item draw proc mxm.  Allocated
            within MyJobDlgInit() that was called by PrDlgMain() */
        if (myDrawListUPP)
        {
            DisposeRoutineDescriptor(myDrawListUPP);
        }

            /*  Make sure to free the InitProc UPP  */
        DisposeRoutineDescriptor(theInitProcPtr);

        if (printingShouldNotProceed)
        {
            DisposeHandle( (Handle)hPrintRec );
            return iPrAbort;
        }

        if (PrError() != noErr)
        {
            DisposeHandle( (Handle)hPrintRec );
            return PrError();
        }

            /* Open the printing document and page to prepare
            for drawing into the printing context */
        pPrPort = PrOpenDoc(hPrintRec, NULL, NULL);
        PrOpenPage(pPrPort, NULL);

            /* Draw something on the page  */
        aRect = (*hPrintRec)->prInfo.rPage;
        InsetRect(&aRect, 20, 20);
        PenSize(4, 4);
        FrameRect(&aRect);

            /*  We're done drawing so close up the page and
            document    */
        PrClosePage(pPrPort);
        PrCloseDoc(pPrPort);

            /*  Do we need to spool the job?    */
        if (!PrError()
        && (*hPrintRec)->prJob.bJDocLoop == bSpoolLoop)
```

```
            PrPicFile(hPrintRec, NULL, NULL, NULL, &theStatus);

            /*  Don't forget to release the print record memory */
        DisposeHandle((Handle) hPrintRec);

        return(noErr);
    } /* Print */



    pascal void MyListDraw(WindowPtr theWindow, short itemNo) {
        #pragma unused (theWindow,itemNo)

        short   itemType;
        Handle  itemH;
        Rect    itemBox,r;
        Point   pt;
        ControlRef cr;
        RgnHandle clip = NewRgn();

        if (itemNo == kPanePopUp) {
            cr = gList[0]->vScroll;
            HideControl(cr);
            return;
        }

        GetDialogItem((DialogPtr) PrtJobDialog,
            itemNo,&itemType,&itemH,&itemBox);
        r = theWindow->portRect;
        pt.h = itemBox.left;
        pt.v = itemBox.top;
        cr = gList[0]->vScroll;
        MoveControl(cr,itemBox.right-15,itemBox.top-1);
        ShowControl(cr);
        if (PtInRect(pt,&r)) {

            gList[0]->rView = itemBox;
            gList[0]->rView.right -=15; /* inset for scroll bar */

            ShowControl(cr);
            LSetDrawingMode(true,gList);
            if (clip) {
                RectRgn(clip,&itemBox);
                LUpdate(clip,gList);

                InsetRect(&itemBox,-1,-1);
                FrameRect(&itemBox);
            }
        }

        /*  Free up the region memory   */
        DisposeRgn(clip);
    }
```

```
        /* 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 */

    pascal TPPrDlg MyJobDlgInit(THPrint hPrint) {
        #pragma unused(hPrint)

        short   firstItem;  /* first new item number */

        short   itemType, item;
        Handle  itemH;
        Rect    itemBox;

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

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

            /* now we'll set up our DITL items -
            - The radio buttons */

        for (item = 5; item <= 7; item++)
        {
            GetDialogItem((DialogPtr) PrtJobDialog,
                firstItem + item -1,&itemType,
                &itemH,&itemBox);
            SetControlValue((ControlHandle) itemH, (item == 5));
        }

            /* now we'll set up the second of our DITL
            items -- The checkbox */

        GetDialogItem((DialogPtr) PrtJobDialog,
            firstItem +2,&itemType,&itemH,&itemBox);
        SetControlValue((ControlHandle) itemH,1);


        // here is the list stuff (mxm)
        {
            Rect bounds ;
            Point theCell = {16,200};


            GetDialogItem((DialogPtr) PrtJobDialog,
                firstItem +8,&itemType,&itemH,&itemBox);

            SetRect(&bounds,0,0,1,0);
            itemBox.right -=15;

            gList = LNew(&itemBox,&bounds,
                theCell,0,(GrafPtr)PrtJobDialog,
                false,false,false,true);
            if (gList) {
                    /* this is how you need to do the scroll-bar so it'll
                    move off correctly with LaserWriter 8.4 and later

                    The theory is that we have another user item which
                    has the handle for the control item for the scrollbar
                    for the list. This way, the scroll bar gets redrawn
                    in the right order, so it realises it's off the panel
                    when it is. If you don't have this code, the scroller
                    gets drawn, then the list gets moved, so you get a
                    phantom scroller in the dialog. -DaveP 9/3/96 */
                short scrollType;
                Handle scrollHandle;
                Rect scrollRect;
                GetDialogItem((DialogPtr)PrtJobDialog,
                    firstItem + 9, &scrollType,
                    &scrollHandle, &scrollRect);
                SetDialogItem((DialogPtr)PrtJobDialog,
                    firstItem + 9, ctrlItem,
                    (Handle) (**gList).vScroll, &scrollRect);
            }
            LAddRow(9,0,gList);
            SetPt(&theCell,0,0);
                /* Lets add some data */
            LSetCell("Testing 1",9,theCell,gList);
            theCell.v ++;
            LSetCell("Testing 2",9,theCell,gList);
            theCell.v ++;
            LSetCell("Testing 3",9,theCell,gList);
            theCell.v ++;
            LSetCell("Testing 4",9,theCell,gList);
            theCell.v ++;
            LSetCell("Testing 5",9,theCell,gList);
            theCell.v ++;
            LSetCell("Testing 6",9,theCell,gList);
            theCell.v ++;
            LSetCell("Testing 7",9,theCell,gList);
            theCell.v ++;
            LSetCell("Testing 8",9,theCell,gList);
            theCell.v ++;
            LSetCell("Testing 9",9,theCell,gList);

            LActivate(true,gList);
            myDrawListUPP =  NewUserItemProc(MyListDraw);

            SetDialogItem((DialogPtr) PrtJobDialog,
                firstItem + 8,itemType,
                (Handle)myDrawListUPP,&itemBox);

        }

            /* 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 = PrtJobDialog->pItemProc;

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


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

    } /*myJobDlgInit*/




     static pascal Boolean testFilter(DialogPtr theDialog
        EventRecord *theEvent, short *itemHit)
    {
        #pragma unused(theDialog)
        #pragma unused(theEvent)
        #pragma unused(itemHit)

            /* This gets called a lot if you uncomment it */
        //  DebugStr("\pMy filter proc got called");
        return false;
    }

    static pascal void testDialogUserItem(WindowPtr theWindow,
            short itemNo);
    static pascal void testDialogUserItem(WindowPtr theWindow,
            short itemNo) {
        if (itemNo != 4)
            return;
        else {
            short oldFont = ((GrafPtr)theWindow)->txFont;
            short oldSize = ((GrafPtr)theWindow)->txSize;
            short itemType;
            Handle h;
            Rect r;

            SetPort(theWindow);
            TextFont(GetAppFont());
            TextSize(9);
            GetDialogItem((DialogPtr)theWindow,
                4,&itemType,&h,&r);
            MoveTo(r.left,r.bottom);
            DrawString("\pThe Pooh-snif network.");
            TextFont(oldFont);
            TextSize(oldSize);
        }
    }
```

```
static void TestDialog(void);
static void TestDialog(void)
{
    DialogPtr theDialog = GetNewDialog(257,nil,(WindowPtr)-1L);
    short itemHit = 0;
            /* use NewModalFilterProc(testFilter); for testing */
    ModalFilterUPP mySubDialogFilterProc = nil;
    UserItemUPP mySubDialogUserProc =
        NewUserItemProc(testDialogUserItem);

    {       /* set up the user item to draw the outline
            around the ok button */
        short t;
        Handle h;
        Rect r;

        GetDialogItem(theDialog,
            4,&t,&h,&r);
        SetDialogItem(theDialog,
            4,t,(Handle)mySubDialogUserProc,&r);
    }

    SetDialogDefaultItem(theDialog,1);
    while((itemHit != 1) && (itemHit != 2)) {
            /* a WAY bad way of seeing if we're
            done, but I'm in a hurry */
        ModalDialog(mySubDialogFilterProc, &itemHit);
    }
    DisposeDialog(theDialog);
    if (mySubDialogFilterProc != nil)
        DisposeRoutineDescriptor(mySubDialogFilterProc);
    if (mySubDialogUserProc != nil)
        DisposeRoutineDescriptor(mySubDialogUserProc);
}

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

pascal void MyJobItems( TPPrDlg theDialog, short itemNo )
{
    short   myItem;
    short   firstItem, item, itemType, theValue;
    Handle  itemH;
    Rect    itemBox;

        /* remember, we saved this in myJobDlgInit */
    firstItem = prFirstItem;
    //  DebugStr("\p Looking at the item...");
        /* "localize" current item No */
    myItem = itemNo-firstItem+1;
        /* if localized item > 0, it's one of ours */
    if (myItem > 0) {

            /* find out which of our items was hit */

         switch (myItem)
```

```
            {
                case 1:         /*    Static text.   */
                    break;
                case 2:         /*  Edit text.       */
                    break;
                case 3:         /*  Check box.       */
                    GetDialogItem((DialogPtr) theDialog,
                        firstItem +2,&itemType,
                        &itemH,&itemBox);
                    theValue = GetControlValue((ControlHandle) itemH);
                    SetControlValue((ControlHandle) itemH,
                        theValue != 1);
                    break;
                case 4:       /* Push button.    */
                    TestDialog();
                    break;
                case 5:         /*   Radio buttons   */
                case 6:
                case 7:
                    for (item = 5; item <= 7; item++) {
                        GetDialogItem((DialogPtr) theDialog,
                            firstItem +item -1,&itemType,
                            &itemH,&itemBox);
                        SetControlValue((ControlHandle) itemH,
                            item == myItem);
                    }
                    break;
                case 8:          /*  Edit text.      */
                case 9:          /*  Our List.       */
                    {   Point pt;
                        GetMouse(&pt);
                        LClick(pt,0,gList);
                    }
                    break;
                case 10:
                        /* our list's scroll bar. We don't need
                        to do anything, but we need to have some
                        code here so we don't hit the default case */
                    break;
                default:
                    Debugger(); /* OH OH */
            }
        } else {
                /* chain to standard item handler, whose address
                is saved in prPItemProc */
            CallPItemProc(prPItemProc,(DialogPtr)theDialog, itemNo);

            if (theDialog->fDone) {
                /* we're going away, need to clean up our
                scroll bar and list */
                if (gList) {
                    (**gList).vScroll = 0L;
                    LDispose(gList);
                }
            }
        }
    } /* MyJobItems */



    void main(void)
    {
        Rect    myWRect;

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

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

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

Back to top

Summary

That's all there is to it. Now you can add items to the print dialogs.

Back to top

References

Inside Macintosh: Imaging With QuickDraw (Chapter 9)

Inside Macintosh: Macintosh Toolbox Essentials (Chapter 6)

PDlog Expand sample code, included with this Technote.

StdFileSaver sample code, available on the *Developer CD Series: Tool Chest Edition.*

develop 27 *- Print Hints: The All-New LaserWriter Driver Version 8.4* by Dave Polaschek.

Back to top

Change History

01-November-1996	Originally written as *Technote PR 09 - Print Dialogs: Adding Items.*
01-February-2000	Accompanying code revised.
12-July-2000	Accompanying code revised from original Pascal.

Back to top

Downloadables

	Acrobat version of this Note (K).	Download
	Printing Sample Code	Download

Back to top