# Technote 1131

## Creating Desktop Printers on the Fly

CONTENTS

This Technote documents the method that Apple uses to create desktop printers on the fly, without having the user go through Chooser.

Note: Apple does not recommend that your application do this behind the user's back; however, to prevent hacks and future incompatibilities we are documenting our "supported" method.

We also recommend that if you do create a DTP for your user, you switch the DTP back to your user's original selection so as not to interfere with their printing selection for other applications.

## Overview

In 1994, Apple introduced the desktop printing architecture with the StyleWriter 1200 driver and LaserWriter 8.3. Since this introduction, many developers have asked Worldwide Developer Technical Support how to create desktop printers on the fly, without having the user go through Chooser. With the release of LaserWriter 8.5.1, Apple introduced one application, Desktop Printer Utility, that creates desktop printers for the user without the Chooser.

# What is Desktop Printing?

Desktop printing is the mechanism that allows drag-and-drop document printing via a 'desktop printer' icon on the user's desktop. It is supported by Mac OS 8.x (built-in) and System 7.x (with the "Desktop Printer Extension" installed). Whenever the user selects a new printer using the Chooser, a Desktop Printer (DTP) is created to represent the newly selected default printer. Once this is done, a printing job can be initiated by simply dragging a document and dropping it on the DTP.

You can write a program (much like Apple's Chooser or Desktop Printer Utility) to create different types of desktop printers. Once a desktop printer is created by your program, the operating system--specifically the Desktop Printer Extension--treats it equally to those created by Chooser.

Each desktop printer is associated with a particular printer driver. Desktop printers can currently be created for StyleWriters and LaserWriters. A future release of the Mac OS will enable third-party printer driver developers to take advantage of desktop printing.

### Background

Once the user selects a printer with the Chooser control panel, this printer becomes the system-wide default printer driver. The file specification and the file name of the selected printer driver are recorded in the "System" file. Specifically, the `AliasRecord` of the selected printer driver is stored as resource type 'alis' with an ID of -8192. The name of the printer driver is stored as resource type 'STR ' with an ID of -8192.

If the selected printer driver is of the LaserWriter type, then one more piece of information is also needed to uniquely identify the destination printer. This piece of information is stored in the printer driver as a resource of type 'PAPA' and ID -8192. Whenever a different physical printer on the AppleTalk network is chosen in the Chooser, the 'PAPA' resource is updated to reflect the change.

One of the many things the "Desktop Printer Extension" does is to monitor the change of the resources mentioned above. When the resources change to reflect a newly selected default printer that does not yet have a desktop printer, Desktop Printer Extension will automatically make a new desktop printer to represent it.

To create a desktop printer, you need only change the three resources mentioned above.

**Warning**: While changing the content of resources 'STR ' -8192 in the System file and 'PAPA' in the LaserWriter driver, be sure not to change their length... it will cause problems if you do (see Technote 1114: "The Extended PAPA Resource").

**Note**: Some Metrowerks PowerPlant classes are used in the following sample code. Please refer to Metrowerks PowerPlant documentation for these classes. C and C++ coding styles are both used in the sample code. The code is listed for demonstration purposes only.

# Checking Your Environment

In addition to your regular environment checking, you need to perform the following checks before attempting to create a desktop printer on the fly:

### Desktop Printer Extension

The "Desktop Printer Extension" is required for any desktop printing. For System 8.0 and later, the functionality is built-into the OS. However, for 7.x systems, it is an add-on extension that can be found in the Extensions folder. The code to check for the Desktop Printer Extension follows:

```
// ------------------------------------------------------
// checking for Desktop Printer Extension
// ------------------------------------------------------
enum { kGestaltPFEFeatures = 'dtpf' };
Boolean HasDesktopExtension()
{
 OSErr err;
 Boolean hasIt = false;
 long gestaltResponse = NULL;

 err = Gestalt(kGestaltPFEFeatures, &gestaltResponse);
 if(err == noErr)
  hasIt = true;
 return hasIt;
}
```

### Finder Scripting extension

Apple events are used to communicate between the printer driver and the Desktop Printer Extension. Therefore, you need to verify that the Finder Scripting Extension is installed.

```
// ------------------------------------------------------
// checking for Finder Scripting
// ------------------------------------------------------
Boolean HasFinderScripting()
{
 long gestaltResponse = NULL;
 return ( (Gestalt(gestaltFinderAttr, &gestaltResponse ) == noErr)
  && ((response & (1L << gestaltOSLCompliantFinder)) != 0) );
}
```

### Valid Printer Driver

Each desktop printer must have a valid printer driver installed. You need to make sure that the printer driver for which you wish to create a desktop printer is installed in the system. Here is some code for this given the file type and creator of a printer driver:

| File | Type | Creator |
|---|---|---|
| LaserWriter 8 | 'PRER' | 'vgrd' |
| StyleWriter 1200 | 'PRER' | 'dblo' |
| StyleWriter 2500 | 'PRER' | 'auro' |

```
// --------------------------------------------------------------
// Find the printer driver in the system extensions folder w/ the
// specified minimum version, type and creator
// Note: This routine does not return an error code. If any error
// should happen, it'll be thrown to the caller w/ C++ exception
// handling.
// The caller should be prepared to handle errors including eofErr
// which indicates that the printer driver is not found.
// Parameters:
// inMinVersion- specify printer driver's minimum version
// inFileType - specify printer driver's file type
// inCreator - specify printer driver's creator
// outPrinterDrvrFsSpec - return the printer driver's FSSpec if
// found
// --------------------------------------------------------------
void FindPrinterDriver(unsigned short inMinVersion, OSType inFileType, OSType inCreator
void FSSpec& outPrinterDrvrFsSpec )
{
OSErr err;
const long kSearchBufSize = 16 * 1024;
short savedResFile = CurResFile();
short startupVRefNum;
long extDirID; // of extensions folder
CInfoPBRec targetPb;
CInfoPBRec maskPb;
CSParam catSearchPb;
FSSpec foundFsSpec;
StPointerBlock searchBuf(kSearchBufSize); // buffer for PBCatSearch
Boolean foundOne = false;

 err = FindFolder(-1, kExtensionFolderType, FALSE, &startupVRefNum, &extDirID);
 ThrowIfOSErr_(err);

 targetPb.hFileInfo.ioFlFndrInfo.fdType = inFileType;
 targetPb.hFileInfo.ioFlFndrInfo.fdCreator = inCreator;
 targetPb.hFileInfo.ioNamePtr = 0;
 targetPb.hFileInfo.ioFlAttrib = 0x00; // search for file only
 targetPb.hFileInfo.ioFlParID = extDirID; // in extensions folder only

 maskPb.hFileInfo.ioFlFndrInfo.fdType = 0xFFFFFFFF;
 maskPb.hFileInfo.ioFlFndrInfo.fdCreator = 0xFFFFFFFF;
 maskPb.hFileInfo.ioFlFndrInfo.fdFlags = 0;
 maskPb.hFileInfo.ioFlFndrInfo.fdLocation.h = 0;
 maskPb.hFileInfo.ioFlFndrInfo.fdLocation.v = 0;
 maskPb.hFileInfo.ioFlFndrInfo.fdFldr = 0;
 maskPb.hFileInfo.ioNamePtr = 0;
 maskPb.hFileInfo.ioFlAttrib = 0x10;  // directory bit
 maskPb.hFileInfo.ioFlParID = extDirID; // in extensions folder only

 Try_{
  catSearchPb.ioCatPosition.initialize = 0;
  while(!foundOne ){
   catSearchPb.ioCompletion = 0;
   catSearchPb.ioNamePtr = 0;
   catSearchPb.ioVRefNum = startupVRefNum;
   catSearchPb.ioMatchPtr = &foundFsSpec;
   catSearchPb.ioReqMatchCount = 1;
   catSearchPb.ioSearchBits = fsSBFlAttrib +
      fsSBFlFndrInfo + fsSBFlParID;
   catSearchPb.ioSearchInfo1 = &targetPb;
   catSearchPb.ioSearchInfo2 = &maskPb;
   catSearchPb.ioSearchTime = 0;
   catSearchPb.ioOptBuffer = (Ptr)searchBuf;
   catSearchPb.ioOptBufSize = kSearchBufSize;
   err = PBCatSearchSync(&catSearchPb);
   ThrowIfOSErr_(err);

    // found one w/ the specified type and creator
   LFile thePrDrFile(foundFsSpec);
   Try_{
    thePrDrFile.OpenResourceFork(fsRdPerm);
    // version of printer driver
    StResource versRsrc('vers', 1, true, true);
    // compare against minimum version
    if(**(unsigned short **)(Handle)versRsrc >= inMinVersion){
     // found one meeting the version requirement
     outPrinterDrvrFsSpec = foundFsSpec;
     foundOne = true; // get out of the loop
    }
    thePrDrFile.CloseResourceFork();
   }Catch_(inErr){
    thePrDrFile.CloseResourceFork();
    // don't throw
   }EndCatch_
  } // while
 }Catch_(inErr){
  // restore default resource file
  UseResFile(savedResFile);
  Throw_(inErr);
 }EndCatch_
}
```

# Creating Desktop Printers for Apple Printers

## StyleWriters

This section demonstrates how a DTP can be created for a non-LaserWriter type of printer driver, e.g., the StyleWriter printer drivers. First, we need some utility routines to change the resources in System file and printer driver's file:

```
// ---------------------------------------------------------------------------
// SetResource
// This routine change the content of an existing resource to the new content
// pointed to by inNewContentH.  The resource file containing the target resource
// must be open and set to the current resource file before this routine is
// called.
// ---------------------------------------------------------------------------
void
SetResource(ResType inResType, ResID inResID, Handle inNewContentH)
{
 // get the resource to be changed
 oldResH = Get1Resource(inResType, inResID);
 // be sure to handle error when the resource doesn't exist (oldResH == 0)

 char savedInHdlState = HGetState(inNewContentH);
  char savedRsrcHdlState = HGetState(oldResH );
 Size sz = GetHandleSize(inNewContentH);

 HUnlock(oldResH );
 SetHandleSize(oldResH, sz); // match sizes
 ThrowIfMemError_();
 HLock(oldResH );
 HLock(inNewContentH);
 BlockMove(*inNewContentH, *oldResH, sz); // copy content

 HSetState(inNewContentH, savedInHdlState);
 HSetState(oldResH , savedRsrcHdlState);
 ChangedResource(oldResH); // mark dirty
 WriteResource(oldResH);
 ThrowIfResError_();
 UpdateResFile(CurResFile());
}


// ---------------------------------------------------------------------------
// SetStrResource
// This routine changes the content of an existing 'STR ' resource to the new
// content pointed to by inStrP.  The resource file containing the target
// resource must be open and set to the current resource file before this routine
// is called.  Pay special attention to how we avoid to shrink existing resource.
// ---------------------------------------------------------------------------
void
SetStrResource(ResType inResType, ResID inResID, ConstStringPtr inStrP )
{
 // get the resource to be changed
 oldResH = Get1Resource(inResType, inResID);
 // be sure to handle error when the resource doesn't exist (oldResH == 0)

 Size sz = *inStrP + 1;
 /* If the string is longer than the handle, then
  grow the handle.
 */
 if(sz > GetHandleSize(oldResH )){
  char savedRsrcHdlState = HGetState(oldResH );
  HUnlock(oldResH );
  SetHandleSize(oldResH , sz);
  ThrowIfMemError_();
  HSetState(oldResH , savedRsrcHdlState);
 }
 BlockMoveData(inStrP, *oldResH , sz);
 ChangedResource(oldResH );
 WriteResource(oldResH);
 ThrowIfResError_();
 UpdateResFile(CurResFile());
}
```

Because Desktop Printer Extension is implemented as a Finder extension, we need to talk to Finder in order to communicate with it. The following routines are needed for this purpose:

```
// ---------------------------------------------------------------------------
// Create the Finder address we wish to send our Apple Event to;
// Note that we mean here "Finder" in the most generic sense;
// In reality, this can be the Finder or AtEase.
// ---------------------------------------------------------------------------
OSStatus getFinderAddress(AEAddressDesc *theDesc)
{
   OSErr               result = noErr;
   ProcessInfoRec      processInfo;
   ProcessSerialNumber serialNumber;

   serialNumber.highLongOfPSN = 0;
   serialNumber.lowLongOfPSN = kNoProcess;
   while ((result = GetNextProcess(&serialNumber)) == noErr)
   {
       processInfo.processInfoLength = sizeof(ProcessInfoRec);
       processInfo.processName     = nil;
       processInfo.processAppSpec  = nil;

       result = GetProcessInformation(&serialNumber, &processInfo);
       if (result == noErr){
           if ( processInfo.processType == 'FNDR' ){
               result = AECreateDesc(typeProcessSerialNumber, (Ptr)&serialNumber,
sizeof(ProcessSerialNumber), theDesc);
               return(result);
           }
       }
   }
   return(result);
}
```

```
// ---------------------------------------------------------------------------
// Send an Apple event to the Finder. The data for the event is pointed to
// by 'dataPtr' and is 'dataSize' bytes long.
// Note that the data is sub-typed so the Finder can decipher the contents.
// ---------------------------------------------------------------------------
#define kFinderExtension       'fext'
static OSStatus sendAEToFinder( Ptr dataPtr, Size dataSize)
{
 OSStatus  err = noErr;
 // the address of the Finder as an Apple event Descriptor
 AEAddressDesc  finderAddr;
 AppleEvent      theEvent;
 AppleEvent      replyEvent;

   err = getFinderAddress(&finderAddr);
 if (err == noErr) {
       err = AECreateAppleEvent(
           kCoreEventClass,        // this is a core event
           kFinderExtension,       // for a Finder extension
           &finderAddr,            // and we send it to the Finder
           kAutoGenerateReturnID,  // we aren't getting a return
           kAnyTransactionID,      // and we don't care about the transaction #
           &theEvent);             // and we create it right here
       if (err == noErr)
       {
           err = AEPutParamPtr(
               &theEvent,          // the event to shove into
               keyDirectObject,    // direct object keyword
               kFinderExtension,   // for the Finder extension
               dataPtr,            // here's the data!
               dataSize);          // here's how long it is!

           if (err == noErr)
           {
               err = AESend(
                   &theEvent,          // send the status event
                   &replyEvent,        // no reply event because -
                   kAENoReply +        // we don't want a reply
                    kAECanInteract +   // the receiver can interact with user
                    kAEDontReconnect,  // and don't bother to reconnect on error
                   kAENormalPriority,  // just a normal event
                   kAEDefaultTimeout,  // we'll wait some reasonable amount of time
                   nil, nil);          // and don't care what happens during that.
           }

                                       // now get rid of the Apple event
           (void) AEDisposeDesc(&theEvent);
       }

                                       // and get rid of the Finder address
       (void) AEDisposeDesc(&finderAddr);
   }

                                       // Fall through and return error, if any
   return(err);
}
```

Although the Desktop Printer Extension can detect newly selected current printer automatically, it does this by checking the related resources in its idle time. This can take a while to happen; therefore, to get better performance, the following routine sends an Apple event to notify Desktop Printer Extension of the new printer:

```
// ---------------------------------------------------------------------------
// nessieCreateDTP
// Send an Apple event to Desktop Printer Extension asking it to create a new
// DTP for the current printer.
// ---------------------------------------------------------------------------
#define kPrintingExtension     'dtpx'
#define aePFECreateDTP         'pfcr'
typedef struct
{
   OSType      pfeCreator;
   OSType      extensionType;
} CreateDTPEvent;


// ---------------------------------------------------------------------------
// Notify Desktop Printer Extension/Finder the current printer has changed
// ---------------------------------------------------------------------------
OSStatus nessieCreateDTP()
{
 OSStatus err;
 CreateDTPEvent  myEvent;

 myEvent.pfeCreator = kPrintingExtension;
 myEvent.extensionType = aePFECreateDTP;
 // because Desktop Printer Extension is a Finder extension
 err = sendAEToFinder((Ptr) &myEvent, sizeof(myEvent));
    return err;
}
```

After the resources specifying the current printer have been properly set up and the notification to Desktop Printer Extension is sent, you still have to verify that the DTP is actually created. The data structure `DTPInfo` is used just for this:

```
typedef struct DTPInfo{
   short    vRefNum;    // vRefNum of the DTP folder
   long  dirID;         // directory ID of the DTP folder
   Str31    DTPName;    // name of the DTP folder
   OSType   driverType; // creator type of the print driver for this DTP
   Boolean  current;    // is this DTP currently the default printer
   Str32 printerName;   // name of the acutal printer on the net (only for LW8.4)
   Str32 zoneName;      // zone where this printer resides (only for LW8.4 DTPs)
} DTPInfo, *DTPInfoPtr, **DTPInfoHdl;
```

The way to get information about DTPs is to make a `Gestalt` call to the Desktop Printer Extension. A list of `DTPInfo` is returned by the `Gestalt` call. Each `DTPInfo` record in the list describes a single DTP. You can walk down the list and examine the content of the `DTPInfo` to decide if the DTP of interest is created. The following routines demonstrate how to do this:

```
#define errDTPNotFound -1
#define kGestaltPFEInfo  'dtpx'

enum
{
     kOldPFEGestaltStructVersion     = 0x02008000,
     // version 2.0f0 (for Mac OS 8.0, 8.1 and 7.x)
     kPFEGestaltStructVersion        = 0x03000000
     // version 3.0 (for Allegro)
};

// GestaltDTPInfo
typedef struct
{
        short   vRefNum;       // vRefNum of the DTP folder
        long    dirID;         // directory ID of the DTP folder
        Str31   dtpName;       // name of the DTP folder
        OSType  driverType;    // creator type of the print driver for this DTP
        Boolean current;       // is this DTP currently the default printer?
        Str32   printerName;   // name of the acutal printer on the net (only for LaserWriter 8
        Str32   zoneName;      // zone where this printer resides (only for LaserWriter 8 dtps)

} GestaltDTPInfo, *GestaltDTPInfoPtr;

// data associated with the pfe gestalt
typedef struct
{
        long    version;             // kPFEGestaltStructVersion
        short   numDTPs;             // number of the active dtps
        Handle  theDTPList;          // handle to a list of GestaltDTPInfo for the active dtps
        Handle  theDTPDriverList;    // handle to a list of drivers' files specs
        Handle  theDTPMoreInfoList;  // Apple's Internal use only.

} GestaltPFEInfo, **GestaltPFEInfoHdl;

// ---------------------------------------------------------------------------
// Look through the list of DTP's looking for a DTP
// that describes the printer with the network address
// 'papaH' and associated with a printer driver whose file spec is
// *printerDrvrFsSpecP.
// papaH is ignored if it's NULL.
// printerDrvrFsSpecP is ignored if it's NULL.
//
// If a match is found, then noErr is returned.
// If a match can not be found then an error is returned.
// If the caller wants information about the matched
// printer, then on entry, *'DTPInfoP' should be non-NULL
// and point to an allocated DTPInfo structure.
// ---------------------------------------------------------------------------
OSStatus
nessieFindDTP(Handle papaH, FSSpec *printerDrvrFsSpecP, DTPInfo *DTPInfoP)
{
 DTPInfoPtr found = NULL;
 GestaltPFEInfoHdl gestaltResponse;
 char papaHState;
 OSStatus err;

 if(papaH){
  papaHState = HGetState(papaH);
  HLock(papaH);
 }

 err = Gestalt(kGestaltPFEInfo, (long *)&gestaltResponse);
 if(!err && gestaltResponse != NULL){
  int numDTPs = (*gestaltResponse)->numDTPs;
  DTPInfoHdl DTPH = (*gestaltResponse)->theDTPList;
  FSSpec** driverH = (*gestaltResponse)->theDTPDriverList;

  err = errDTPNotFound;
  if((numDTPs > 0) && (DTPH != NULL) && (driverH != NULL)){
   DTPInfoPtr DTP;
   FSSpec *driverP;
   char DTPState = HGetState((Handle)DTPH);
   char driverListState = HGetState((Handle)driverH);
   HLock((Handle)DTPH);
   HLock((Handle)driverH);
   DTP = *DTPH;
   driverP = *driverH;
   if((DTP != NULL) && (driverP != NULL)){
    StringPtr printerStr;      // Points to printer's name
    StringPtr zoneStr;         // Will point to zone name (first we
                               // skip object type).
    int i;
    if(papaH){
     printerStr = (StringPtr)*papaH;          // Points to printer's name
     zoneStr = printerStr + *printerStr + 1;  // Will point to zone name
                                              // (first we skip object type).
     zoneStr += *zoneStr + 1;                 // Now points to zone name.
    }

    for(i = 0; i < numDTPs && found == NULL;
    ++i, ++DTP, ++driverP){
     // This DTP matches the one we want if it's
     // created with the specified printer driver and
     // the zone and printer names match.
     if(papaH){
      if(((printerDrvrFsSpecP == NULL) ||
       ((printerDrvrFsSpecP->vRefNum == driverP->vRefNum) &&
(printerDrvrFsSpecP->parID == driverP->parID) &&
(EqualString(printerDrvrFsSpecP->name, driverP->name, false, false)))) &&

      EqualString(printerStr, DTP->printerName, false, false) &&
EqualString(zoneStr, DTP->zoneName, false, false)){
       found = DTP;
      }
     }else{
      if((printerDrvrFsSpecP != NULL) &&
       ((printerDrvrFsSpecP->vRefNum == driverP->vRefNum) &&

       (printerDrvrFsSpecP->parID == driverP->parID) &&
      (EqualString(printerDrvrFsSpecP->name, driverP->name, false, false)))){
       found = DTP;
      }
     }
    }
    if(found != NULL){
     if(DTPInfoP) *DTPInfoP = *found;
    }
   }
   HSetState((Handle)driverH, driverListState);
   HSetState((Handle)DTPH, DTPState);
  }
 }

 if(papaH)
  HSetState(papaH, papaHState);

 return found == NULL ? errDTPNotFound : noErr;
}


// ----------------------------------------------------------------------------
// Given a FSSpec of a printer driver, this routine can decide if its DTP exists
// ----------------------------------------------------------------------------
Boolean HasDTP(DTPInfo *DTPInfoP, FSSpec *prDriverFsSpecP)
{
 OSStatus err;
 err = nessieFindDTP(NULL, prDriverFsSpecP, DTPInfoP);
 return(!err);
}
```

With the help of the routines above, we are now ready to create our first DTP. This example demonstrates how to create a DTP for a non-LaserWriter type of printer driver. For LaserWriter (LW) DTP, you need to call the routine named `LaserWriterExtra()` which is commented out here. (See the LaserWriter sections for more detail.)

After verifying the environment, we try to locate the printer driver. In the System file, we then change the two resources specifying the current printer driver, notify Desktop Printer Extension of the change, and wait for the new DTP to be made. Note how we stay in the wait loop. Within this loop, make sure you yield control to Finder by `EventAvail` or `WaitNextEvent`. Without this, Desktop Printer Extension/Finder would NOT have an opportunity to create the DTP for you:

```
#define kWaitForDTPTime   (60 * 20)
// ---------------------------------------------------------------------------
// CreateDTP
// This routine demonstrates how to create a DTP for StyleWriter1200
// ---------------------------------------------------------------------------
void
CreateDTP()
{
 OSErr err;
 short savedResFile = CurResFile();
 FSSpec printerDriverFSSpec;
 AliasHandle theAlias;
 EventRecord eventRec;
 unsigned long endTime = TickCount() + kWaitForDTPTime;
 DTPInfo DTPInfo;

                    // checking the environment
                    // make sure we have Desktop Printer Extension
                    // and Finder Scripting Extension
 if(HasDesktopExtension() && HasFinderScripting()){
  Try_{
   FindPrinterDriver(0x0120,
    'PRER', 'dblo', FSSpec& // StyleWriter1200
    printerDriverFSSpec);

   /* if you are creating DTP for LaserWriter type of
      printer driver, you'll need to write some more
      code here.
      Let's call LaserWriterExtra();
   */

   UseResFile(0);   // system file
                    // make alias for printer driver
   err = NewAlias(nil, &printerDriverFSSpec, &theAlias);
   ThrowIfOSErr_(err);
                    // set current printer to our printer driver
   SetResource('alis', -8192, theAlias);
   SetStrResource('STR ', -8192, printerDriverFSSpec.name);
   DisposeHandle(theAlias);

                    // we should notify Desktop Printer Extension
                    // that we have changed the default printer
   err = nessieCreateDTP();
                    // verify that the DTP is created
   do{
```

```
                    // give Desktop Printer Extension a chance
                    // to handle the request to create DTP
    EventAvail(everyEvent, &eventRec);
    newDTPMade = HasDTP(&DTPInfo,  &printerDriverFSSpec);
   }while((TickCount() < endTime) && (newDTPMade == FALSE));
                    // until either the DTP is made or time-out
  }Catch_(inErr){
                    // handle error…
  }EndCatch_
 }
 UseResFile(savedResFile);
}
```

## Creating A Desktop Printer for the LaserWriter

If you are writing code to create a DTP for the LaserWriter, you simply need to make some minor modifications to the routine `CreateDTP()`.

Instead of passing the creator for StyleWriter, use LaserWriter's creator where `FindPrinterDriver` is called. You should also call the routine `LaserWriterExtra()`, which is commented out in the example above. `LaserWriterExtra()` should change the 'PAPA' -8192 resource in the LaserWriter printer driver to point to your new printer.

The routine LaserWriterExtra() varies between LaserWriter 8.5.1 and the earlier versions. For 8.5.1, this routine can change further depending on the type of the DTP.

### Creating A Desktop Printer for pre-8.5.1 LaserWriter

Only an AppleTalk (PAP) DTP can be created for pre-8.5.1 LW. These drivers support only the original 103-byte long 'PAPA' resource. The `LaserWriterExtra()` might look something like this:

```
// ---------------------------------------------------------------------------
// LaserWriterExtra for pre-8.5.1 LW
// point current printer to target
// ---------------------------------------------------------------------------
LaserWriterExtra(FSSpec *printerDrvrFsSpecP,
   ConstStr32Param inZone,
   ConstStr32Param inPrinterName){
 OSErr err;
 char papaState;
 short refNum;
 Handle papaH;
 short savedResFile = CurResFile();

 // open printer driver resource fork
 err = FSpOpenResFile(printerDrvrFsSpecP, fsWrPerm);
 // handle error…

 // get PAPA resource from printer driver
 papaH= Get1Resource('PAPA' , -8192);
 // handle error…

    papaState = HGetState(papaH);
 HLock(papaH);
 // change PAPA resource content
 NBPSetEntity(*papaH, (ConstStr32Param)StringPtr(inPrinterName),
   (ConstStr32Param)("\pLaserWriter"),
   (ConstStr32Param)inZone);
 HSetState(papaH, papaState );
 ChangedResource(papaH); // mark dirty
 WriteResource(papaH);   // update resource
 err = FSClose(refNum);
 UseResFile(savedResFile);
}
```

### For LaserWriter 8.5.1 and Greater

LaserWriter 8.5.1 supports extended PAPA (1024 bytes in length). Using the extended PAPA, different types of desktop printers can be created to represent different types of virtual or physical printers, in addition to the original AppleTalk "PAP" printer. Printing to these different types of DTPs will have different effects. A "Hold" DTP is one that can only accept spool files. "PostScript Translator" DTP is one that can translate the printed document into a PostScript document. "Custom" DTP can launch a pre-specified application and instruct it to open the converted PostScript document. "LPR" DTP allows printing to a printer/spooler understanding LPR protocol. "Infrared" DTP allows printing to a LaserWriter with infrared communication port. For extended PAPA details, see Technote 1115: "The Extended 'PAPA' Resource".

`PrintingLib` provides some routines for fabricating extended PAPA so programmers don't have to understand its internal structure See Technote 1129: "The Settings Library" for details.

Before changing the PAPA resource in LaserWriter driver to create a DTP, you must also set additional parameters to fully specify a DTP. Examples of these parameters for currently defined DTPs are PPD files for all types of DTP, domain name address for LPR DTPs, destination folder for Translator DTPs and target application for Custom DTPs.

The parameters are listed below:

| Collection Tag | Tag ID | DTP Type | parameter |
|---|---|---|---|
| 'pppd' | 1 | all | parsed PPD FSSpec |
| 'ppd ' | 1 | all | additional PPD specification |
| 'LpIa' | 1 | LPR | domain address of printer |
| 'Svap' | 1 | Translator | save to file preference |
| 'Pdka' | 1 | Translator | destination folder alias |
| 'TGap' | 1 | Custom | post-process application alias |
| 'pslv' | 1 | Custom | postscript level |
| 'bnok' | 1 | Custom | use of binary data |
| 'jobt' | 1 | Custom | type of postscript |
| 'font' | 1 | Custom | font handling |
| 'CsDs' | 1 | Custom | post-process application description |

These parameters are stored as hints within collections, which are then saved in LaserWriter's preference file. These hints are identified by collection tags and IDs. The meaning and data structure associated with each parameter are explained below with more details in the sections where they are used.

`PrintingLib` provides some routines for accessing these collections. See *The Settings Library* documentation for details. These routines should be used to access the collections, and some of the routines might require the name of the printer driver's preference file. For LaserWriter, the name of the preference file is stored as 'STR ' resource, ID -8185 within the LaserWriter file.

A routine to add a hint to a DTP's collection might look like this:

```
#define kNoCollection -2
OSErr psStorePrinterName(Collection prInfo, Handle papa);
/* store a normalized EntityName into collection prInfo */


/*
 add a hint to a DTP's collection
 The memory block pointed by 'bufP' of length 'size' is added as a hint of 'tag' and
   'id' to the DTP
whose PAPA is specified by 'papaH'.
 Note: the resource fork of LaserWriter must be open before this routine    is called.
*/
OSErr addHint(Handle papaH, CollectionTag tag, long id, long size, void *bufP){

 Handle lwPrefFileNameH;
 char savedHandleState;
 Collection DTPCollection;

 // we need the name of LW's pref. file
 lwPrefFileNameH= GetString(-8185 );
 // don't forget to handle error
 savedHandleState = HGetState(inHandle);
 HLock(lwPrefFileNameH);
 // use routine in SettingsLib to get the collection and add item
 DTPCollection= psGetPrefsPrinterInfo((StringPtr)(*lwPrefFileNameH), papaH);
 if(DTPCollection== nil)
  // don't forget to handle error
  err = kNoCollection;

 err = psStorePrinterName(DTPCollection, papaH);
 // handle error
 err = AddCollectionItem(DTPCollection, tag, id, size, bufP);
 // handle error
 err = psUpdatePrefsPrinterInfo(*lwPrefFileNameH, papaH, DTPCollection);
 DisposeCollection(DTPCollection);
 // clean up
 HSetState(lwPrefFileNameH, savedHandleState );
 ReleaseResource(lwPrefFileNameH);
 return (err);
}
```

PPD files can be specified for all different types of DTP. The following routine shows how this can be done:

```
OSStatus psGetPPDInfo(FSSpecPtr driver, Handle papaH,
long structVersion, PrinterPPDInfo *ppdInfo);
/* Return the PPD information for the printer specified by 'papaH' for the printer
 specified by 'driver'. The PPD information is returned in *'ppdInfo'.
 The caller must pass in the version of the PrinterPPDInfo structure being used,
 this should be the constant 'kPPDInfoStructVersion'. If the caller's structure is
 not compatable with the routine, then the routine returns ' errWrongStructVersion'.
*/

OSStatus psSetPPDInfo(FSSpecPtr driver, Handle papaH,
long structVersion, PrinterPPDInfo *ppdInfo);
/* Set the ppd file and the parsed ppd file for the printer driver specified
 by 'driver' and the printer specified by 'papaH'. The FSSpec's of the PPD file
 and the parsed PPD file are passed in the structure pointed to by' ppdInfo' along
 with a flag indicating whether the generic PPD should be used. If the generic PPD
 flag is set, then only the parsed PPD FSSpec need be valid. 'structVersion' indicates
 the version of the PrinterPPDInfo structure the caller is using. The caller should pass
 in the constant
*/

OSErr ppdGetParseFolder(FSSpecPtr parseFolder);
/* Fill in the FSSpec pointed to by 'parseFolder' with
 the 'vRefNum' and 'parID' of the parsed PPD folder.
 The client can place a parsed PPD file name into the
 FSSpec's 'name' field and then use the File Manager's
 Open call's to open a parsed PPD. Note that *parseFolder'
 is not the FSSpec of the parsed PPD folder.
*/

OSErr ppdParseFile (const FSSpec *ppdFileSpec, short compiledRef, short compiledResFRef,
  PPDParseErr *errInfoP);
/* Parse a PPD file, and all its includes.
 FSSpecPtr will be closed upon exit.
 The PPD is parsed into the open file with the file reference 'compiledRef'.
 If compiledResFRef is not -1, the resource fork of the ppd file is copied into it.
 If 'errInfoP' is not NULL, then any error information is returned in
 *'errInfoP'.
*/

enum PPDPresetSource {
 kPPDSourceUnknown = -1,
 kPPDSourceRSRC = 0,
 kPPDSourceGeneric = 1,
 kPPDSourceCustom = 2
};

typedef enum{
 kTriFalse = 0,
 kTriTrue,
 kTriUnknown
} TriState;

struct CustPPDInfo {
 Boolean   usePPD;
 FSSpec   fileSpec;
};
typedef struct CustPPDInfo CustPPDInfo, **CustPPDHandle;

typedef struct {
 short presetSource;  /* enum PPDPresetSource */
 TriState isSetup;
 CustPPDInfo customPPD;
} WhatPPD;

typedef struct PrinterPPDInfo{
 long  structVersion;     // Identifies the version of the rest of this structure.
 Boolean  useGenericPPD;  // True if the generic PPD should be used.
 FSSpec  ppdFile;         // FSSpec of the current PPD file.
 FSSpec  parsedPPDFile;   // FSSpec of the parsed PPD file for the printer.
}PrinterPPDInfo;
```

```
/*
This routine parse a ppd file specified ppdFsSpec
and add the corresponding hints to a DTP specified
by papaH
*/
OSErr SetPPD(FSSpec* ppdFsSpecP, Handle papaH, FSSpec* laserWriterFsSpecP)
{
 OSErr err = noErr;
 FSSpec parsedPPD;
 WhatPPD whatPPD;
 short savedResFile = ::CurResFile();
 Boolean isGeneric = false;
 short dataForkRefNum, resourceForkRefNum;
 PrinterPPDInfo ppdInfo;

 // default
 whatPPD.presetSource = kPPDSourceCustom;
 whatPPD.isSetup = kTriTrue;
 whatPPD.customPPD.usePPD = TRUE;
 whatPPD.customPPD.fileSpec = *ppdFsSpecP;

 // prepare parsed PPD FSSpec
 err = ppdGetParseFolder(&parsedPPD);
 ThrowIfOSErr_(err);
 CopyPStr(ppdFsSpecP->name, parsedPPD.name, sizeof(parsedPPD.name));

 // create parsed file and open its resource/data fork
 FSpCreateResFile(&parsedPPD, 'vgrd', 'Pref', smSystemScript);
 err = SpOpenDF(&parsedPPD, fsRdWrPerm, &dataForkRefNum);
 resourceForkRefNum = ::FSpOpenResFile(&parsedPPD, fsRdWrPerm);
 // parse ppd
 err = ppdParseFile (ppdFsSpecP, dataForkRefNum, resourceForkRefNum, NULL);
 // close resource/data fork
 CloseResFile(resourceForkRefNum);
 err = FSClose(dataForkRefNum);
 UseResFile(savedResFile);

 // associate the ppd with the printer
 err = addHint(papaH, 'pppd', 1, sizeof(parsedPPD), &parsedPPD);
 err = addHint(papaH, 'ppd ', 1, sizeof(whatPPD), &whatPPD);
 // set default hints
 err = psGetPPDInfo(laserWriterFsSpecP, papaH, 2, &ppdInfo);
 if(!err) err = psSetPPDInfo(laserWriterFsSpecP, papaH, 2, &ppdInfo);

 return (err);
}
```

## Creating A PAP Desktop Printer for the LaserWriter

The `LaserWriterExtra` for PAP DTP might look like this:

```
// ------------------------------------------------------------------------------
// LaserWriterExtra for 8.5.1 LW
// point current printer to target
// ------------------------------------------------------------------------------
LaserWriterExtra(FSSpec *printerDrvrFsSpecP,
   ConstStr32Param inZone,
   ConstStr32Param inPrinterName){
 OSErr err;
 short refNum;
 Handle papaH;
 short savedResFile = CurResFile();
 SSpec ppdFsSpec;

 // open printer driver resource fork
 err = FSpOpenResFile(printerDrvrFsSpecP, fsWrPerm, &refNum);
 // handle error…

 // get PAPA resource from printer driver
 papaH= Get1Resource('PAPA' , -8192);
 // handle error…

 // change PAPA resource content
 // See SettingsLib Spec  for psSetPapPapa
 err = psSetPapPapa(papaH, inPrinterName,
  (const Byte *)("\pLaserWriter"), inZone, 0);
 // handle error…

 // specify PPD file w/ ppdFsSpec
 err = FSMakeFSSpec(vRefNum,dirID,fileName,&ppdFsSpec);
 err = SetPPD(&ppdFsSpec, papaH, printerDrvrFsSpecP);

 ChangedResource(papaH); // mark dirty
 WriteResource(papaH);   // update resource
 err = FSClose(refNum);
 UseResFile(savedResFile);
}
```

## Creating An LPR Desktop Printer for the LaserWriter

The domain address and queue have to be specified for LPR DTP. They are specified as 'LpIa' hint. The format of the 'LpIa' is formed by appending the queue name (a Pascal String) to the end of the domain address (another Pascal String) of the printer.

The `LaserWriterExtra` for an LPR DTP might look like this:

```
// ---------------------------------------------------------------------------
// LaserWriterExtra for 8.5.1 LW
// point current printer to target
// where inTcpAddr is the network address of the lpr printer.
// This address can be in either name or dot format, i.e either "\plaser.rbi.com"
// or "\p204.188.109.155".
// inQNameis the name of the print queue associated with
// a spooler at inTcpAddr. If inQName is NULL then the default queue for the
// printer/spooler is used.
// ---------------------------------------------------------------------------
LaserWriterExtra(FSSpec *printerDrvrFsSpecP,
   ConstStr32Param inTcpAddr,
   ConstStr32Param inPrinterName,
   ConstStr32Param inQName){
 OSErr err;
 short refNum;
 Handle papaH;
 Handle ipAddrQueueH;
 short savedResFile = CurResFile();

 // open printer driver resource fork
 err = FSpOpenResFile(printerDrvrFsSpecP, fsWrPerm, &refNum);
 // handle error…

 // get PAPA resource from printer driver
 papaH= Get1Resource('PAPA' , -8192);
 // handle error…

 // change PAPA resource content
 // See SettingsLib Spec  for psSetLprPapa
 err = psSetLprPapa(papaH, inPrinterName, (const Byte *)("\p=LPR"),
  (const Byte *)(*inTcpAddr), (const Byte *)inQName);
 // handle error…

 // specify PPD file w/ ppdFsSpec
 err = FSMakeFSSpec(vRefNum,dirID,fileName,&ppdFsSpec);
 err = SetPPD(&ppdFsSpec, papaH, printerDrvrFsSpecP);

 // specify domain address and queue of the printer
 err = PtrToHand(inTcpAddr, &ipAddrQueueH, inTcpAddr[0] + 1);
 err = PtrAndHand(inQName, ipAddrQueueH, inQName[0] + 1);
 HLock(ipAddrQueueH);
 err = addHint(papaH, 'LpIa', 1, GetHandleSize(ipAddrQueueH), *ipAddrQueueH);
 HUnlock(ipAddrQueueH);
 DisposHandle(ipAddrQueueH);

 ChangedResource(papaH); // mark dirty
 WriteResource(papaH); // update resource
 err = FSClose(refNum);
 UseResFile(savedResFile);
}
```

## Creating An Infrared Desktop Printer for the LaserWriter

```
// ---------------------------------------------------------------------------
// LaserWriterExtra for 8.5.1 LW
// point current printer to target
// ---------------------------------------------------------------------------
LaserWriterExtra(FSSpec *printerDrvrFsSpecP,
   ConstStr32Param inPrinterName){
 OSErr err;
 short refNum;
 Handle papaH;
 short savedResFile = CurResFile();

 // open printer driver resource fork
 err = FSpOpenResFile(printerDrvrFsSpecP, fsWrPerm, &refNum);
 // handle error…

 // get PAPA resource from printer driver
 papaH= Get1Resource('PAPA' , -8192);
 // handle error…

 // change PAPA resource content
 // See SettingsLib Spec  for psSetInfraredPapa
 err = psSetInfraredPapa(papaH, inPrinterName, ("\p=Ird"));
 // handle error…

 // specify PPD file w/ ppdFsSpec
 err = FSMakeFSSpec(vRefNum,dirID,fileName,&pgsSpec);
 err = SetPPD(&ppdFsSpec, papaH, printerDrvrFsSpecP);

 ChangedResource(papaH); // mark dirty
 WriteResource(papaH); // update resource
 err = FSClose(refNum);
 UseResFile(savedResFile);
}
```

## Creating A Hold Desktop Printer for the LaserWriter

```
//
------------------------------------------------------------------------
```

```
// LaserWriterExtra for 8.5.1 LW
// point current printer to target
// ------------------------------------------------------------------------------
LaserWriterExtra(FSSpec *printerDrvrFsSpecP,
   ConstStr32Param inPrinterName){
 OSErr err;
 short refNum;
 Handle papaH;
 short savedResFile = CurResFile();

 // open printer driver resource fork
 err = FSpOpenResFile(printerDrvrFsSpecP, fsWrPerm, &refNum);
 // handle error…

 // get PAPA resource from printer driver
 papaH= Get1Resource('PAPA' , -8192);
 // handle error…

 // change PAPA resource content
 // See SettingsLib Spec  for psSetHoldPapa
 err = psSetHoldPapa(papaH, inPrinterName, ("\p=Hld"));
 // handle error…

 // specify PPD file w/ ppdFsSpec
 err = FSMakeFSSpec(vRefNum,dirID,fileName,&ppdFsSpec);
 err = SetPPD(&ppdFsSpec, papaH, printerDrvrFsSpecP);

 ChangedResource(papaH); // mark dirty
 WriteResource(papaH); // update resource
 err = FSClose(refNum);
 UseResFile(savedResFile);
}
```

## Creating A PostScript Translator Desktop Printer for the LaserWriter

For PostScript Translator DTP, the default destination folder to hold the translated file must be specified as 'Pdka' hint. A 'Svap' must also be specified to indicate this DTP is for "print to file":

```
 struct SaveAsFilePrefs{
 Boolean  saveToFile; // true for printing to file
 Boolean  restricted; // true for printing to file
};

// ------------------------------------------------------------------------------
// LaserWriterExtra for 8.5.1 LW
// point current printer to target
// ------------------------------------------------------------------------------
LaserWriterExtra(FSSpec *printerDrvrFsSpecP,
   ConstStr32Param inPrinterName,
   FSSpec* destinationFolderP){
 OSErr err;
 short refNum;
 Handle papaH;
 SaveAsFilePrefs saveToFilePref;
 AliasHandle printToDiskAliasH;
 short savedResFile = CurResFile();

 // open printer driver resource fork
 err = FSpOpenResFile(printerDrvrFsSpecP, fsWrPerm, &refNum);
 // handle error…

 // get PAPA resource from printer driver
 papaH= Get1Resource('PAPA' , -8192);
 // handle error…

 // change PAPA resource content
 // See SettingsLib Spec  for psSetFilePapa
 err = psSetFilePapa(papaH, inPrinterName, "\p=Fil");
 // handle error…

 // specify PPD file w/ ppdFsSpec
 err = FSMakeFSSpec(vRefNum,dirID,fileName,&ppdFsSpec);
 err = SetPPD(&ppdFsSpec, papaH, printerDrvrFsSpecP);

 // print to disk hint
 saveToFilePref.saveToFile = TRUE;
 saveToFilePref.restricted = TRUE;
 prefCollection.AddItem('Svap', 1, sizeof(saveToFilePref), &saveToFilePref);

 // set print to disk default folder
 err = NewAliasMinimal(destinationFolderP, &printToDiskAliasH);
 HLock((Handle)printToDiskAliasH);
 err = addHint(papaH, 'Pdka', 1,
  GetHandleSize((Handle)printToDiskAliasH), *printToDiskAliasH);
 HUnlock((Handle)printToDiskAliasH);
 DisposeHandle((Handle)printToDiskAliasH);

 // change PAPA resource content
 ChangedResource(papaH); // mark dirty
 WriteResource(papaH); // update resource
 err = FSClose(refNum);
 UseResFile(savedResFile);
}
```

## Creating Custom Desktop Printer for LaserWriter

For an in-depth description of how a custom DTP can be used by developers and how to modify Apple's Desktop Printer Utility to support a specific post-processing application, please read Technote 1113: "Customizing the Desktop Printer Utility."

With each Custom DTP, we need to specify a post-processing application for LaserWriter to launch. The post-processing application is specified as an `AliasHandle` and stored as a 'TGap' hint.

Currently supported hints for custom DTP include 'pslv', 'bnok', 'jobt' and 'font'. The meaning of these hints and their data structures are described in *Customizing the Desktop Printer Utility*.

The following routine shows how these parameters can be added to a custom DTP's collection. For more information on the `FOntHint` and `CustomAppeDesc` structures, please see Technote 1113: "Customizing the Desktop Printer Utility."

```
 typedef struct{             // see "Custom DTPs".
 long flag;                  // kIncludeAllFontsBut or kIncludeAllFontsBut
 unsigned char name[1];      // font name
 }FOntHint;

struct CustomAppDesc{
 OSType appSignature;        // target application's signature
 Str255 docType;             // appears in the list of doc. types
                             // when "New" menu is selected
 Str255 helpText;            // lin43, appears when the type of
                             // this DTP is selected
 Str255 usage;               // appears as the DTP usage in the window
 Str255 appFileName;       // default application file name, used in
                           // error message and etc.
 short numOfHintsFollow;   // 1 based
 HintRsrcSpec hintRsrc[1]; // variable length
};
typedef struct CustomAppDesc CustomAppDesc;
typedef struct CustomAppDesc* CustomAppDescPtr;
typedef struct CustomAppDesc** CustomAppDescHdl;


/*
 This routine adds custom DTP hints to the DTP specified papaH.
 The FSSpec of the post-processing application is pointed to by ppApp.
 Note how AliasHandle for the post-processing application is
 generated and added as a 'TGap' hint.
*/
OSErr addCustomDtpParameters(FSSpec* ppApp, Handle papaH){
 OSErr err;
 long psLevel = 1;       // postscript level 1
 Byte binaryOK = 0;      // false
 char job = 0;           // psJobPostScript
 FOntHint fontHint;
 AliasHandle targetAppAliasH;
 CustomAppDesc customAppDesc;

 // postscript level 1
 err = addHint(papaH, 'pslv', 1, siezof(psLevel ), &psLevel ){
 // no Binary
 err = addHint(papaH, 'bnok', 1, siezof(binaryOK ), &binaryOK ){
 // PostScript job
 err = addHint(papaH, 'jobt', 1, siezof(job ), &job ){
 // includeAllFonts
 fontHint.flag = 1;
 fontHint.name [0] = 0;
  err = addHint(papaH, 'font', 1, siezof(fontHint), &fontHint){

 // save target application's location
 err = ::NewAliasMinimal(ppApp, &targetAppAliasH);
 // handle error
 ::HLock((Handle)targetAppAliasH);
 err = addHint('TGap', 1,
  GetHandleSize((Handle)targetAppAliasH),    *targetAppAliasH);
 ::HUnlock((Handle)targetAppAliasH);
 ::DisposeHandle((Handle)targetAppAliasH);

 // if you want Apple's Desktop Printer Utility to recognize and being able to
 // open your DTP, you must add the 'CsDs' hint

 // fill out fields in customAppDesc for post-processing application
 customAppDesc.appSignature = 'xxxx'; // of post-processing application
 /* and all other fields…
 */
 // save custom app's description
 err = addHint(('CsDs', 1, sizeof(customAppDesc), &customAppDesc);
 return err;
}


// ------------------------------------------------------------------------------
// LaserWriterExtra for 8.5.1 LW
// point current printer to target
// ------------------------------------------------------------------------------
LaserWriterExtra(FSSpec *printerDrvrFsSpecP,
   ConstStr32Param inPrinterName){
 OSErr err;
 FSSpec postProcessAppFsSpec;
 short refNum;
 Handle papaH;
 short savedResFile = CurResFile();

 // open printer driver resource fork
 err = FSpOpenResFile(printerDrvrFsSpecP, fsWrPerm, &refNum);
 // handle error…

 // get PAPA resource from printer driver
 papaH= Get1Resource('PAPA' , -8192);
 // handle error…

 // See SettingsLib Spec  for psSetCustomPapa
 err = psSetCustomPapa(papaH, inPrinterName, "\p=Cst");
 // handle error…

 // specify PPD file w/ ppdFsSpec
 err = FSMakeFSSpec(vRefNum,dirID,fileName,&ppdFsSpec);
 err = SetPPD(&ppdFsSpec, papaH, printerDrvrFsSpecP);

 // specify post-processing app. and other parameters
 err = FSMakeFSSpec(vRefNum, dirID, "\pYourPostProcessAppName",
  &postProcessAppFsSpec);
 err = addCustomDtpParameters(&postProcessAppFsSpec, papaH);

 // change PAPA resource content
 ChangedResource(papaH); // mark dirty
 WriteResource(papaH);   // update resource
 err = FSClose(refNum);
 UseResFile(savedResFile);
}
```

# Summary

As outlined here, there is some work to creating a desktop printer on the fly; however, the end result is worth it for many applications as long as you heed the warnings. Good luck!

## Further References

- Technote 1129: "The Settings Library"
- Technote 1113: "Customizing Desktop Printer Utility"
- Technote 1115: "The Extended PAPA"
- Metrowerks PowerPlant documentation
- Inside Macintosh: Macintosh Toolbox Essentials, Chapter 7.

## Downloadables

Acrobat version of this Note (K).

**To contact us, please use the Contact Us page.**
**Updated: 01-June-98**

Technotes
Previous Technote | Contents | Next Technote