FinePrint Developer's Kit

FinePrint Software
http://www.fineprint.com

November 8, 2000

Introduction

The FinePrint Developer's Kit (FPDK) is designed for independent software vendors who wish to incorporate FinePrint functionality directly into their applications. The FPDK allows you to control the FinePrint Driver and the FinePrint user interface via the FinePrint Application Programming Interface (API).

Recent changes

10/26/1999	1.	added "const" to some pointer parameters to clearly distinguish IN from OUT parameters
	2.	added an HFinePrint parameter to fpSetDestPrinterAttr
11/3/1999	3.	added PrePrint callbacks
2/9/2000	4.	the DeleteJobs parameter to fpPrintAllJobs and fpClose now works correctly. no API changes.
4/21/2000	5.	changed all occurrences of FPAPI3 to FPAPI4
7/18/2000	6.	renamed fpDeferJobs to fpSetDeferAll; rewrote documentation
	7.	added new API call fpSetShowDlg
11/8/2000	8.	some minor documentation changes

Contents

The FPDK contains the following:

- documentation for the FinePrint API
- header files for the FinePrint API
- redistributable DLLs which implement the FinePrint API
- 32-bit import libraries for the FinePrint API DLLs
- 32-bit static link libraries for the FinePrint API
- sample program which shows how to use the FinePrint API

The following files are included with the FPDK:

FPDK.DOC FPDK.PDF	documentation (same content in different formats)
FPDEFS.H FPAPI.H	C/C++ header files
FPDEFS.BAS FPAPI.BAS	VB header files
FPAPI4A.LIB FPAPI4U.LIB	import libraries (ANSI and Unicode)

FPAPI4AS.LIB FPAPI4AM.LIB FPAPI4US.LIB FPAPI4UM.LIB	static link libraries (ANSI and Unicode, single- and multi-threaded)
FPAPI4A.DLL FPAPI4U.DLL	redistributable DLLs (ANSI and Unicode)
SAMPLES	C/C++ and VB sample applications which use the FinePrint API

FinePrint Architecture

This section describes the FinePrint architecture, so that you can better understand and utilize the FinePrint API.

There are three FinePrint components that are relevant to the FPDK: the printer driver, the dispatcher, and the user interface (UI).

The printer driver, also known as a "FinePrinter", is a normal Windows printer driver that appears in the user's Printers folder. It is usually (but not always) called "FinePrint Driver". Currently FinePrint only allows a single FinePrinter on a system, but this will change in the future.

The UI is the FinePrint dialog. It maintains and displays a queue of print jobs, allows the user to select job options (e.g. 2-up, no borders, double sided, …), and to send the jobs to a destination printer. Sometimes the UI is not actually displayed, although it still exists and is active; for example, when jobs are deferred, the UI window is minimized in the Windows task bar, but it still exists and maintains its job queue.

The dispatcher is a background process that manages the flow of print jobs between the driver and the UI. It maintains its own queue of completed print jobs, which it hands off to the UI when the UI is ready to accept them.

The normal flow of print jobs within FinePrint is as follows. First, an application prints to a FinePrinter using the normal Win32 print calls (CreateDC, StartDoc, StartPage, etc.). The FinePrinter spools the job and hands it off to the dispatcher, who holds the job in its queue until the UI is ready to accept the job. At that time the dispatcher removes the job from its queue and hands it off to the UI, where the job can be printed, deleted, or deferred for later printing.

API Functions - Index

The FinePrint API consists of the following functions:

fpClearCallbackDll
fpClose
fpCloseStationery
fpCreateStationery
fpDeleteStationery
fpDisplayDialog
fpGetFinePrinterName

fpGetJobCount
fpGetLayoutAttr
fpGetStationeryAttr
fpGetVersion
fpGetVersionReq
fpOpen
fpOpenStationery
fpPrintAllJobs
fpSaveSettings
fpSetCallbackDll
fpSetDeferAll
fpSetDestPrinterAttr
fpSetShowDlg
fpSetLayoutAttr
fpSetStationeryAttr
fpWaitForJob

Static Linking Versus DLL

You can use the FinePrint API as a static link library which you link into your executable, or as a DLL which you call dynamically from your executable. We recommend that you use the DLL version in all cases except one: if you are implementing a callback function via fpSetCallbackDll, and if your callback function itself uses the FinePrint API, then the DLL in which your callback function resides must use the static library version of the API.

Here are some of the advantages and disadvantages of the two approaches.

Advantages of DLL:

- works with C/C++ and VB
- FinePrint code is totally separate from your executable (e.g. no conflicts with global variable names, function names, etc.)

Disadvantages of DLL:

- you must modify your install program to install the DLL
- possible version conflict if there are multiple copies of the DLL on the system
- fpSetCallbackDll callback functions which use the FinePrint API cannot use the DLL version of the FinePrint API

Advantages of static library:

- no version conflicts, since your executable contains a complete working copy of the FinePrint API
- fpSetCallbackDll callback functions can use the FinePrint API

Disadvantages of static library:

- the FinePrint API becomes part of your executable. Can get name conflicts with global variable names, function names, etc.
- makes your executable file larger
- works only with C/C++, not VB

ANSI Versus Unicode

The FinePrint API has separate libraries and DLLs for ANSI and Unicode clients. All string parameters to API functions are defined as LPTSTRor LPCTSTR, so you must include <tchar.h> when you build your application.

Single- Versus Multi-Threaded

The static library version the FinePrint API has separate libraries for single- and multi-threaded clients. The DLL version of the API is single-threaded and can be called from either single- or multi-threaded applications.

The Matrix

The following table tells you which FinePrint API files to use in your application:

	ANSI	**Unicode**
dynamic linking	link with FPAPI4A.LIB; calls FPAPI4A.DLL at runtime	link with FPAPI4U.LIB; calls FPAPI4U.DLL at runtime
static linking, single-threaded	link with FPAPI4AS.LIB	link with FPAPI4US.LIB
static linking, multithreaded	link with FPAPI4AM.LIB	link with FPAPI4UM.LIB

How To Use The FinePrint API

To use the FinePrint API, just include the header file FPAPI.H in your application and link with the appropriate library. If you are using the DLL version of the API, then you will need to distribute either FPAPI4A.DLL or FPAPI4U.DLL with your application, and you should install the DLL into your private application directory, where your EXE is located. You should not install the DLL into the Windows or Windows System directory, because of possible version conflicts.

Users of your application must have FinePrint installed on their system in order for the FinePrint API to work. Neither the FPDK nor the FinePrint API includes a working version of FinePrint.

API Functions – Reference

This section documents each FinePrint API call in detail.

void fpClearCallbackDll (HFinePrint hfp)

Removes the currently installed FinePrint callback DLL.

hfp (HFinePrint) is a session handle returned by fpOpen.

See the section “FinePrint Callback DLLs” for more details.

fpe = fpClose (hfp, fDeleteJobs)

Ends a FinePrint session.

hfp (HFinePrint) is a session handle returned by fpOpen.

fDeleteJobs (BOOL) is nonzero to clear the FinePrint UI’s print queue, or zero to leave the queue unchanged.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

fpe = fpCloseStationery (hfp, hStat)

Closes a stationery opened by fpCloseStationery or fpOpenStationery.

hfp (HFinePrint) is a session handle returned by fpOpen.

hStat (HFpStat) is a stationery handle returned by fpCreateStationery or fpOpenStationery.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

fpe = fpCreateStationery (hfp, pszStat, phStat)

Creates a new stationery.

hfp (HFinePrint) is a session handle returned by fpOpen.

pszStat (LPCTSTR) is the name of the new stationery. If a stationery with the same name already exists, it is deleted and re-created with default properties.

phStat (HFpStat *) receives the stationery handle of the new stationery.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

If you want to select the stationery as the current stationery, you must explicitly call fpSetLayoutAttr with the eliStationery code.

fpe = fpDeleteStationery (hfp, pszStat)

Deletes a stationery.

hfp (HFinePrint) is a session handle returned by fpOpen.

pszStat (LPCTSTR) is the name of an existing stationery. If the stationery does not exist, then the call will fail.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

fpe = fpDisplayDialog (hfp, pdwDlg)

Displays the FinePrint UI.

hfp (HFinePrint) is a session handle returned by fpOpen.

pdwDlg (DWORD *) receives the dialog result:
zero if an error occurred or if the UI is already open
IDOK if the user clicked OK
IDCANCEL if the user clicked Cancel
IDDEFER if the user clicked Defer
IDDEFERALL if the user clicked Defer All

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

fpe = fpGetFinePrinterName (iPrinter, pszFinePrinter)

Returns the device name of a FinePrinter installed on the current system.

iPrinter (int) is the zero-based index of the FinePrinter. Current releases of FinePrint only support one FinePrinter per system, so this parameter must be zero.

pszPrinter (LPTSTR) points to a buffer which receives the FinePrinter name. This buffer should be at least 80 characters long.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

fpe = fpGetJobCount (hfp, pjc)

Returns the number of print jobs handled by FinePrint. Three different values are returned:

- the number of print jobs in the dispatcher's queue
- the number of print jobs in the UI's queue
- the total number of print jobs handled since dispatcher startup. A job is considered "handled" for our purposes as soon as the printing application calls the Win32 StartDoc function.

hfp (HFinePrint) is a session handle returned by fpOpen.

pjc (FpJobCount *) points to a structure which is filled in with the job counts.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

cbRequired = fpGetLayoutAttr (hfp, li, pAttr, cbAttr)

Retrieves an attribute of the current FinePrint layout.

hfp (HFinePrint) is a session handle returned by fpOpen.

li (eLayoutItem) specifies which layout item to get (N-up, borders, destination printer, etc.).

pAttr (void *) points to a buffer which receives the value of the attribute. Its contents depend on the eLayoutItem:

li	pAttr
eliLayout	eLayoutType *
eliBorders	eBorderType *
eliOrder	BOOL *
eliStationery	LPTSTR
eliForm	LPTSTR
eliDestPrinter	LPTSTR
eliMargins	eMarginType *
eliDuplex	BOOL *
eliCopies	DWORD *
eliSeparateJobs	eJobSeparation *
eliGutter	eGutterType *
eliEnableRTL	BOOL *
eliRTL	BOOL *
eliAlignText	BOOL *
eliSkipBitmaps	BOOL *

pAttr can be NULL, in which case no data is copied and the function returns the number of bytes required to contain the attribute. In this case cbAttr should be zero.

cbAttr (DWORD) specifies the size, in bytes, of the buffer pointed to by pAttr.

Returns the buffer size required for the attribute, or zero if an error occurred.

Notes:

- for string attributes, the size returned by the function includes the zero terminator.
- if the buffer passed in is too small to contain the attribute, then only as many bytes as will fit are copied. The return value of the function still specifies the full size of the attribute, not the truncated size. If a string attribute is truncated, then the returned partial text string will NOT be zero-terminated.

cbRequired = fpGetStationeryAttr (hfp, hStat, si, sia, pAttr, cbAttr)

Retrieves an attribute of a stationery's header, footer, or watermark.

hfp (HFinePrint) is a session handle returned by fpOpen.

hStat (HFpStat) is a stationery handle returned by fpCreateStationery or fpOpenStationery.

si (eStatItem) specifies which stationery item to use (header, footer, or watermark).

sia (eStatItemAttr) specifies which attribute of the stationery item to get (text, font, or color).

pAttr (void *) points to a buffer which receives the value of the attribute. Its contents depend on the eStatItemAttr:

sia	**pAttr**
esiaText	LPTSTR
esiaFont	LOGFONT *
esiaColor	COLORREF *

pAttr can be NULL, in which case no data is copied and the function returns the number of bytes required to contain the attribute. In this case cbAttr should be zero.

cbAttr (DWORD) specifies the size, in bytes, of the buffer pointed to by pAttr.

Returns the buffer size required for the attribute, or zero if an error occurred.

Notes:

- for string attributes, the size returned by the function includes the zero terminator.
- if the buffer passed in is too small to contain the attribute, then only as many bytes as will fit are copied. The return value of the function still specifies the full size of the attribute, not the truncated size. If a string attribute is truncated, then the returned partial text string will NOT be zero-terminated.

- for the font attribute, the lfHeight field of the LOGFONT is a positive value and specifies the font height in points, not in pixels. This is because the pixel height depends on the resolution of the destination printer, which can change.

dwVersion = fpGetVersion ()

Returns the version number of the currently installed version of FinePrint, or zero if an error occurred or if FinePrint is not installed. Returns a DWORD where the high WORD is the major version and the low WORD is the minor version. For example, version 3.33 would be represented as 0x00030021.

dwVersion = fpGetVersionReq ()

Returns the minimum version number of FinePrint required by the FinePrint API, in the same format as fpGetVersion. For example, if the FinePrint API requires version 3.33 or above but version 3.20 is installed, then the API will not operate.

You do not need to check the version numbers yourself, since fpOpen will do it for you and will fail if the versions are incompatible. But in case that happens, you can call fpGetVersion and fpGetVersionReq to find out exactly what the mismatch is.

fpe = fpOpen (pszFinePrinter, phfp)

Initializes the FinePrint API and creates a FinePrint session using the specified FinePrinter.

pszFinePrinter (LPCTSTR) is the name of a FinePrinter, e.g. “FinePrint Driver”. A NULL value will cause the FinePrint API to use the first installed FinePrinter on the system, which in current releases is the only FinePrinter.

phfp (HFinePrint *) receives a session handle that must be passed to most subsequent FinePrint API functions.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

fpe = fpOpenStationery (hfp, pszStat, phStat)

Opens an existing stationery.

hfp (HFinePrint) is a session handle returned by fpOpen.

pszStat (LPCTSTR) is the name of the stationery. If the stationery does not exist, then an error occurs.

phStat (HFpStat *) receives the stationery handle of the new stationery.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

If you want to select the stationery as the current stationery, you must explicitly call fpSetLayoutAttr with the eliStationery code.

fpe = fpPrintAllJobs (
hfp,
pszOutputFile,
fShowProgress,
fDeleteJobsWhenDone)

Instructs the FinePrint UI to print all the jobs in its queue.

hfp (HFinePrint) is a session handle returned by fpOpen.

pszOutputFile (LPCTSTR) specifies the name of the output file for a Print to File operation. If this parameter is non-NULL, then FinePrint will print to the destination printer on FILE:, producing the specified output file; if it is NULL, then FinePrint will print normally to the destination printer, to its usual printer port.

fShowProgress (BOOL) is nonzero to display a thermometer progress window while printing, or zero to suppress the progress window.

fDeleteJobsWhenDone (BOOL) is nonzero to remove the jobs from the FinePrint UI queue after they are printed, or zero to leave them in the queue. This is the same distinction as the user clicking OK versus Print on the UI; OK means "print jobs and delete them", and Print means "print jobs but leave them around".

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

fpe = fpSaveSettings (hfp)

Instructs the FinePrint UI to save its current settings (layout, borders, destination printer, etc.) to the registry. The settings will become the defaults when FinePrint is next used.

hfp (HFinePrint) is a session handle returned by fpOpen.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

NOTE: you should not call fpSaveSettings after calling fpPrintAllJobs with the fDeleteJobsWhenDone flag set to TRUE. This is because fpPrintAllJobs destroys

the UI (without saving settings) when it deletes all the print jobs. If you want to save settings, you should do it before you delete all the print jobs.

fpe = fpSetCallbackDll (hfp, pszCallbackDll, pcEntryPoints)

Installs a FinePrint callback DLL. Only one callback DLL is active at any given time, so this call replaces any currently installed DLL with the new one.

hfp (HFinePrint) is a session handle returned by fpOpen.

pszCallbackDll (LPCTSTR) is the path name of the callback DLL.

pcEntryPoints (DWORD *) receives the number of entry points successfully found in the callback DLL.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

See the section "FinePrint Callback DLLs" for more details.

fpe = fpSetDeferAll (hfp, fDeferAll)

Sets the FinePrint UI's "Defer All" setting. This is the same setting that is affected when you hold down the SHIFT key while printing from the UI. This function is particularly important if you disable the FinePrint UI using fpSetShowDlg; see the fpSetShowDlg documentation for details.

hfp (HFinePrint) is a session handle returned by fpOpen.

fDeferAll (BOOL) is nonzero to set the UI to Defer All mode, or zero for normal Defer mode.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

fpe = fpSetDestPrinterAttr (hfp, pszDestPrinter, pszSetting, dwValue)

Stores a printer-specific registry setting for the given printer. This call affects both the current FinePrint session and future sessions.

hfp (HFinePrint) is a session handle returned by fpOpen.

pszSetting (LPCTSTR) is the setting to modify, e.g. "DuplexSupport". Rather than use an explicit string constant, you should use one of the #defined strings in FPDEFS.H.

dwValue (DWORD) is the value of the setting.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

fpe = fpSetLayoutAttr (hfp, li, pAttr)

Sets an attribute of the current FinePrint layout.

hfp (HFinePrint) is a session handle returned by fpOpen.

li (eLayoutItem) specifies which layout item to set (N-up, borders, destination printer, etc.).

pAttr (const void *) specifies the value of the attribute. Its contents depend on the eLayoutItem. If the value is a string, then pAttr points to the string; otherwise pAttr is not a pointer to the value but rather is the value itself.

li	pAttr
eliLayout	eLayoutType
eliBorders	eBorderType
eliOrder	BOOL
eliStationery	LPCTSTR
eliForm	LPCTSTR
eliDestPrinter	LPCTSTR
eliMargins	eMarginType
eliDuplex	BOOL
eliCopies	DWORD
eliSeparateJobs	eJobSeparation
eliGutter	eGutterType
eliEnableRTL	BOOL
eliRTL	BOOL
eliAlignText	BOOL
eliSkipBitmaps	BOOL

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

NOTE: to set the current form or stationery to <None>, pass a NULL pointer for pAttr.

fpe = fpSetShowDlg (hfp, dwShowDlg)

Sets the FinePrint UI's display mode. This is the same setting that is affected by the "Show FinePrint dialog" setting in the FinePrint printer driver properties.

hfp (HFinePrint) is a session handle returned by fpOpen.

dwShowDlg (DWORD) one of the eShowDlgType values defined in FPDEFS.H and FPDEFS.BAS. The values are:

value	meaning
ShowDlg_Early	show the FinePrint UI when the printing application calls StartDoc (before the print job begins spooling)
ShowDlg_Late	show the FinePrint UI when the printing application calls EndDoc (after the print job has completed spooling). This is the default setting.
ShowDlg_Never	do not show the FinePrint UI. In this case, the Defer All setting plays an important role; see below for details.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

When the dialog mode is set to ShowDlg_Never, the UI's Defer All setting plays an important role. If the UI is set to normal Defer mode, then any job sent to FinePrint bypasses the UI and goes immediately to the current destination printer. Therefore in this mode, it is not possible for FinePrint to combine multiple print jobs into one. However, if the UI is set to Defer All mode, then jobs sent to FinePrint are not immediately sent to the current destination printer, but are collected in the UI (which will appear minimized in the task bar). In this case your client application must call fpPrintAllJobs to instruct the UI to send the collected jobs to the destination printer.

fpe = fpSetStationeryAttr (hfp, hStat, si, sia, pAttr)

Sets an attribute of a stationery's header, footer, or watermark.

hfp (HFinePrint) is a session handle returned by fpOpen.

hStat (HFpStat) is a stationery handle returned by fpCreateStationery or fpOpenStationery.

si (eStatItem) specifies which stationery item to use (header, footer, or watermark).

sia (eStatItemAttr) specifies which attribute of the stationery item to set (text, font, or color).

pAttr (const void *) specifies the value of the attribute. Its contents depend on the eStatItemAttr. If the value is a string, then pAttr points to the string; otherwise pAttr is not a pointer to the value but rather contains the value itself.

sia	pAttr
esiaText	LPCTSTR
esiaFont	LOGFONT
esiaColor	COLORREF

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

Note: for the font attribute, the lfHeight field of the LOGFONT is a positive value and specifies the font height in points, not in pixels. This is because the pixel height depends on the resolution of the destination printer, which can change.

fpe = fpWaitForJob (
hfp,
pjcOrig,
hProcess,
cSecTimeoutStart,
cSecTimeoutPrint,
pjs)

Waits for a print job to complete.

hfp (HFinePrint) is a session handle returned by fpOpen.

pjcOrig (const FpJobCount *) is the count of FinePrint jobs *before* the job which you are waiting for was launched. Call the FPAPI function fpGetJobCount to fill in this structure. It is very important that you get the job count *before* the job starts, or else fpWaitForJob will not work properly.

hProcess (HANDLE) is the process handle of the printing application, if known, or NULL otherwise. If you launch an app to print using CreateProcess, then its process handle is returned in the PROCESS_INFORMATION structure; if you launch it using ShellExecuteEx with the SEE_MASK_NOCLOSEPROCESS flag, then its process handle is returned in the SHELLEXECUTEINFO structure.

cSecTimeOutStart (DWORD) is the maximum number of seconds to wait for the print job's StartDoc call. If the document being printed is large, or is being opened across a network, then it could take a while for the job to start after you launch the printing application. The actual value depends on the environment you are running in, but something like 15 minutes might be a safe value for an unattended application.

cSecTimeoutPrint (DWORD) is the maximum number of seconds to wait for the print job to complete, once StartDoc was called. Something like 1-2 hours might be a safe value for an unattended application.

pjs (FpJobStatus *) points to a structure which is filled in with information about the print job.

Returns zero for success, or a FinePrint error code (type FpError) otherwise. See FPDEFS.H for a list of codes.

ApiSamp: A Sample FinePrint API Client Application

The FPDK includes a sample Win32 application called ApiSamp, which calls many of the FinePrint API functions. You should refer to APISAMP.CPP for examples on how to structure your application's use of the FinePrint API. APISAMP requires Microsoft Visual Studio 6/Visual C++ 98.

FinePrint Callback DLLs

FinePrint has the ability to call back into the client application at certain predefined times in the print process. Callback functions are optional. If they exist, they are called from the FinePrint Dispatcher at any or all of the following times:

- when a Windows application starts printing to a FinePrinter ("StartDoc callback")
- when a Windows application stops printing to a FinePrinter ("EndDoc callback")
- when FinePrint is about to print to a physical printer ("PrePrint callback")
- after FinePrint has printed to a physical printer ("OnPrint callback")

The dispatcher passes information about the print job to the callback functions, and the callback functions can pass information back to the dispatcher about how to handle the job.

Callback functions must reside in a DLL, which is dynamically loaded by the dispatcher. Therefore, callback functions run in the context of the dispatcher, not in the context of the main client application (if there is one).

The simplest implementation is to have the callback function be a standalone DLL that does not require access to another application. If you want to communicate between the callback function and another application (e.g. your main application), then you must implement your own interprocess communication mechanism such as named pipes.

The dispatcher locates callback functions in a callback DLL via LoadLibrary and GetProcAddress. Any callback functions that you implement must be exported by name in your DLL. If your callback DLL is written in C++, then you need to make sure that the undecorated name is exported; one way to accomplish this is to declare the callback function extern "C" and to include its name in the DLL's DEF file.

Among the information that the dispatcher passes to the callback function are strings (e.g. the name of the print job). You can write your callback functions using either ANSI or Unicode strings. The name of the callback function tells the dispatcher what kind of strings it expects, and the dispatcher marshals the strings accordingly. The possible callback function names are:

fpOnStartDocA	StartDoc callback using ANSI strings
fpOnStartDocW	StartDoc callback using Unicode strings
fpOnEndDocA	EndDoc callback using ANSI strings
fpOnEndDocW	EndDoc callback using Unicode strings
fpPrePrintA	PrePrint callback using ANSI strings
fpPrePrintW	PrePrint callback using Unicode strings
fpOnPrintA	OnPrint callback using ANSI strings
fpOnPrintW	OnPrint callback using Unicode strings

StartDoc and EndDoc callback functions receive a pointer to a structure (FpDocCallbackA or FpDocCallbackW) with the following members:

idJob	DWORD	print job identifier
dwShowDlg	DWORD (eShowDlgType)	FinePrint dialog mode. This is a value from the eShowDlgType enumeration. The StartDoc callback can modify this field to control the display of the FinePrint dialog.
szFinePrinter	char[] or WCHAR[]	FinePrinter name (e.g. "FinePrint Driver")
szJobName	char[] or WCHAR[]	print job name (e.g. "Microsoft Word – FOO.DOC")
szFpFile	char[] or WCHAR[]	FinePrint file name (e.g. "c:\temp\ ~fp100.tmp"). The file name is not known at StartDoc time, so this string will be empty in a StartDoc callback.

StartDoc and EndDoc callback functions return any nonzero value to continue with the print job or FALSE to abort the job.

PrePrint and OnPrint callback functions receive a pointer to a structure (FpPrintCallbackA or FpPrintCallbackW) with the following members:

pc	PageCount structure	specifies how many pages were printed. See FPDEFS.H for the structure definition.
szFinePrinter	char[] or WCHAR[]	FinePrinter name (e.g. "FinePrint Driver")

PrePrint callback functions return any nonzero value to continue with the print job or FALSE to abort the job. OnPrint callback functions can return any value, as the return value is ignored.

A callback function can find out more details about current FinePrint settings by calling back into the FinePrint API. For example, if an OnPrint callback function wants to know which physical printer was printed to, it simply calls fpOpen (passing the szFinePrinter structure member from the FpPrintCallback structure), calls fpGetLayoutAttr with eliDestPrinter, and then

calls fpClose. A callback DLL of this type must link with the static library version of the FinePrint API, as described in the section "Static Linking Versus DLL".

NOTE: a PrePrint callback function should pass FALSE as the fDeleteJobs parameter to fpClose. Otherwise it can hang the system. For example, suppose a user prints something to FinePrint, then clicks OK in the UI to send the job to a physical printer. Your PrePrint callback function gets control, calls fpOpen, does some stuff, then calls fpClose. If at that time you tell fpClose to delete the print jobs, it will tell the UI to delete them, and it will wait until the UI has done that before it returns. But in the meantime, the UI is waiting for your PrePrint callback function to return, which is waiting for fpClose, which is waiting for the UI…so you have a classic deadlock scenario. The UI is waiting for your callback function to complete, and your callback function is waiting for the UI to complete.

Callback DLLs are installed by the fpSetCallbackDll API function. They can also be automatically loaded each time the dispatcher starts up (e.g. at system startup) if they are specified in the system registry. In the HKEY_LOCAL_MACHINE\Software\FinePrint2000 key, create a string value called "CallbackDll" and set it to the full path name of the desired callback DLL.

Once installed (by any method), a callback DLL remains active until fpClearCallbackDll is called or until the dispatcher exits. This means that a callback DLL installed by a client application will remain active even if the client application exits. The dispatcher maintains only one callback DLL for the entire system.

Here is a more detailed summary of exactly when the callback functions are called:

- StartDoc callbacks are called just after the FinePrinter has received a StartDoc notification from GDI. No document pages have yet been printed by the application. The FinePrint UI has not yet been displayed.
- EndDoc callbacks are called just after the FinePrinter has received an EndDoc notification from GDI. The application has printed all the document pages that it intends to print. If the FinePrint UI is set to "before spooling", then it has already been displayed and the user has already selected FinePrint settings; for any other UI setting, the FinePrint UI has not yet been displayed. In all cases, the callback function can make modifications to FinePrint settings by calling fpOpen and the other FinePrint API functions.
- PrePrint callbacks are called after all FinePrint settings have been chosen, just before FinePrint prints to the selected physical printer.
- OnPrint callbacks are called after FinePrint has finished printing to a destination printer.

Handy Tips

The following sections contain tips that might help you as you code to the FinePrint API.

How to suppress the Printer Options dialog when FinePrint first prints to a printer

This dialog only appears when FinePrint needs to know the duplex capabilities of a destination printer, so if you set the duplex caps before you print, the dialog will not be presented. To set the duplex caps, you would call

```
fpSetDestPrinterSetting (pszDestPrinter, TEXT ("DuplexSupport"), dwValue),
```

where dwValue is a member of the DuplexSupport enum in FPDEFS.H.

How to figure out when a print job to FinePrint has finished

If you launch an application to print to the FinePrint Driver (e.g. suppose you launch Word or Notepad to print a text file), then you will need to know when that application has finished printing. You should not call fpPrintAllJobs or fpClose while an app is still printing to the FinePrint Driver, because those API calls will clear the FinePrint job queue and your results are not guaranteed if a job is being sent to the driver at the same time. Another reason to wait is that you may have more documents to print, and it is not advisable to launch multiple printing apps at the same time in Windows (especially Windows 95/98, which is notoriously unstable because of its 16-bit GDI code base).

It is difficult to determine when an application has finished printing, because:

- Win32 APIs do not directly support this functionality
- there is a lot of variation in application behavior. For example, when you print using Notepad, you always get a fresh copy of Notepad, and that copy always exits when its job is done; either because the print job finished, or because a fatal error occurred. However, when you print using Word, you may or may not get a fresh copy of Word (depending on whether or not Word is already running), and Word will only exit if it was a fresh copy, and if the print job completed successfully.

You can use the FinePrint API call fpWaitForJob to tell you when a print job to a FinePrinter has completed. Of course, if your own application is the printing application, then you don't need to call fpWaitForJob; as soon as you have called the Win32 function EndDoc, your job is done.