Using Adobe PDF Converter SDK

Version 3.2

ADOBE SYSTEMS INCORPORATED
Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000

17-023 ***November 2019***

Adobe® PDF Converter SDK Version 3.2 User Guide.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Part I: Adobe PDF Converter SDK Concepts

Part II: Adobe PDF Converter SDK Reference

Preface

This Document

Using the Adobe PDF Converter SDK describes:

- How client software interacts with the PDF Converter SDK to convert Adobe PostScript 3 files, Image files, and PPML files into PDF files
- The comparative differences between Adobe PDF Converter SDK and Acrobat Distiller
- How you can use the sample client, democonverter, as a guide to create your own clients for accessing the Adobe PDF Converter SDK
- The basic deliverables for Adobe PDF Converter SDK
- How Acrobat Distiller parameters influence Adobe PDF Converter SDK behavior
- How the Adobe PDF Converter SDK responds to font references (font policy)

This document also contains the reference chapters on the following topics:

- Functions and callbacks your client software uses to interact with the Adobe PDF Converter SDK
- Functions and callbacks your client may use to provides its own file I/O library
- Description of structures and enumerations passed by the Adobe PDF Converter SDK functions and callbacks.

This document is intended for OEM's developing software that incorporates the Adobe PDF Converter SDK.

This document replaces *Using Adobe Normalizer Server*, Version 10.0

Notational Conventions

Typefaces are used as shown below:

Format	Denotes
Regular	Examples of the PostScript, PDF, or Portable Job Ticket Format (PJTF) language. DSC and other PostScript comments.
Bold	All PostScript, PDF, or PJTF language names, such as the names of operators, keys, dictionaries, and resource categories.
`Monospaced`	All C programming language expressions, file names, and pathnames.

Documentation Problems

If you discover any errors in or have any problems with this document, please e-mail us at:

doc_problems@adobe.com

Please describe the error or problem as completely as possible and give us the document ID number (located at the foot of the cover page), the document title, and the page number or page range.

Part I

Adobe PDF Converter SDK Concepts

Part I presents an overview of the Adobe PDF Converter SDK, lists and describes the product deliverables, explains concepts such as how to use the Adobe PDF Converter SDK interfaces, and answers frequently asked questions. The chapters in Part I are:

1 About Adobe PDF Converter SDK

1.1 Overview

This chapter provides basic information about the Adobe PDF Converter SDK and compares it to Acrobat Distiller 10.0, an Adobe product that also converts PostScript content into PDF format.

NOTE: This document uses the name Distiller in place of the full product name Acrobat Distiller.

1.1.1 What It Does

The Adobe PDF Converter SDK, like Distiller, converts PostScript language streams into PDF streams, a process called *conversion*. Unlike Distiller, the Adobe PDF Converter SDK also converts image files and PPML files into PDF files and is highly customizable, allowing you to control many aspects of conversion that cannot be controlled through the Distiller interface. Conversion and the term distillation have the same meanings with regards to the functioning of the Adobe PDF Converter SDK. [1]

The Adobe PDF Converter SDK is a library of functions that works under the direction of a client. The Adobe PDF Converter SDK contains a generic sample of client software, called Democonverter. This provides a basic tool that OEMs can use to develop specific client software that includes individual printing system features.

1.1.2 What It Consumes

Summary

- EPS, provided any screen (bitmap) previews are stripped out

NOTE: A difference in terminology exists between this guide and the document entitled *Acrobat Distiller Parameters* (Technical Note 5151). Technical Note 5151 uses the terms “PDF settings file” and “Adobe PDF settings” where this guide uses “job options”.

General Description

Figure 1.1 illustrates the data flow supported by the Adobe PDF Converter SDK. The client software provides the Adobe PDF Converter SDK with job options, which

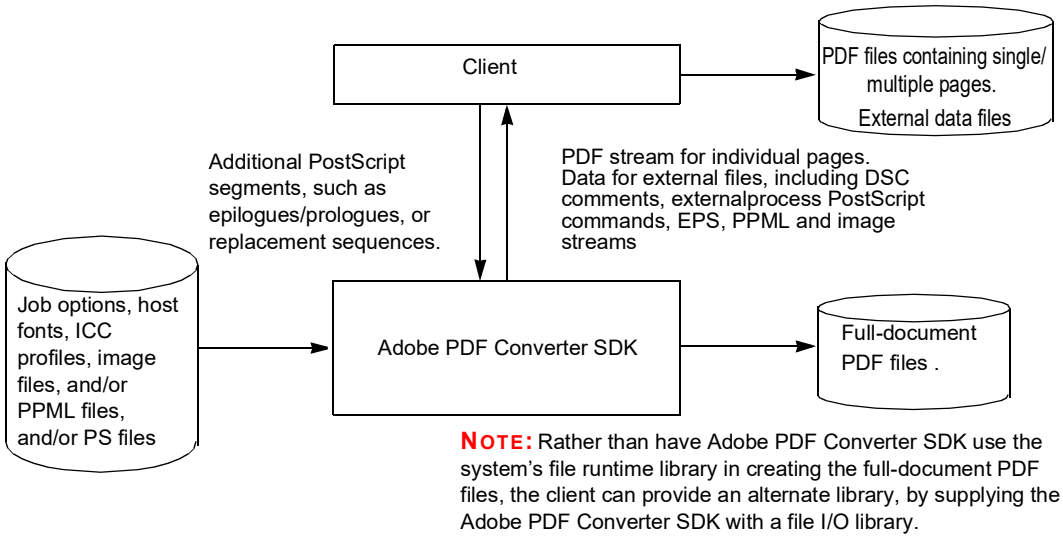
1. The term conversion was originally intended to mean segmenting a PostScript stream into page packages of some undetermined format, in order to allow pages to be independently printed. Ultimately, PDF was selected as the preferred format, causing conversion and distillation to refer to the same process.

provides instructions on how PostScript streams/files, image files and PPML files should be converted.

Job options are PostScript segments that contain PostScript **setdistillerparams** and **setpagedevice** operators that correspond to the Distiller parameters described in Technical Note 5151, *Acrobat Distiller Parameters*. Acrobat Distiller also consumes job options that the user specifies using the **Job Option** field in the user interface.

Distiller parameters specify how a PostScript file/stream should be distilled or converted.

FIGURE 1.1 Data flow for Adobe PDF Converter SDK (job option controlled)



If the client provides the Adobe PDF Converter SDK with job options, this dictates how the Adobe PDF Converter SDK obtains the streams/files for processing, as follows:

- *Job options mode* — If the Adobe PDF Converter SDK is run in job options mode, the Adobe PDF Converter SDK consumes client-provided PostScript streams, which it obtains using callbacks. The Adobe PDF Converter SDK treats such streams as a job, whose beginning and end is indicated by the client. That is, a single job can include multiple discrete PostScript streams.

1.1.3 What It Produces

Summary

The Adobe PDF Converter SDK produces files of the following format:

- PDF v1.2 - v1.7

General Description

As the Adobe PDF Converter SDK processes a job, it produces a full-document PDF file, and it can produce several data streams. More specifically, the Adobe PDF Converter SDK produces the following:

- *A PDF file for the entire job* — The Adobe PDF Converter SDK always produces a PDF file that represents the entire contents of the job. Such a file, which is called a *full-document PDF file*, can be no larger than 10 Gigabytes.
- *PDF streams for individual pages* — The Adobe PDF Converter SDK can produce PDF streams for individual pages (called *PDF page streams*), which the client stores in files, one per page. (This document uses the term *PDF page files* to mean PDF page streams the client has stored in files.)
- *PDF streams for a set of pages* — The Adobe PDF Converter SDK can produce PDF streams for a set of pages.
- *PDF streams for image files*— The Adobe PDF Converter SDK can produce PDF streams for images.
- *Data streams for external files* — The Adobe PDF Converter SDK can produce data streams that the client stores in external files. Those streams can represent any of the following inputs:
  - Conforming EPS programs. The resulting data streams are exact copies of the original EPS programs.
  - Image streams either embedded in the PostScript stream or residing in separate files. Reflecting certain Distiller parameters, such data streams may be either exact copies of the original image streams or modified versions of those streams.
- *Information about the PostScript stream* — The Adobe PDF Converter SDK provides the client with information about the PostScript stream, including page device keys and page information contained in PostScript and DSC comments. The Adobe PDF Converter SDK can also provide the client with DSC comments and with **externalcommand** PostScript operators.

1.1.4 Usage Scenarios

Products based upon the Adobe PDF Converter SDK will likely be used in one of the following general scenarios:

- *Distiller scenario* — In the Distiller scenario, the client provides the Adobe PDF Converter SDK with job options that specify how the PostScript streams are to be processed. The Adobe PDF Converter SDK then processes the job options, as described in the previous section under “Job controlled by job options.”

1.1.5 How To Control Adobe PDF Converter SDK

You can control the Adobe PDF Converter SDK using several methods:

- Specifying variables passed to the Adobe PDF Converter SDK public functions. Such variables include Job options that specify how PostScript streams/files are to be normalized.
- Specifying values returned by client-provided callbacks, as described in 7.1, "Callbacks for Transferring Data between the Adobe PDF Converter SDK and a Client", and 7.2, "Callbacks That Relay Information to the Client".
- Providing a PostScript program and other data that the Adobe PDF Converter SDK uses to initialize the PostScript Interpreter, as described in the files delivered with the Adobe PDF Converter SDK.
- Adding to the PostScript streams passed to the Adobe PDF Converter SDK, PostScript segments that initialize Distiller parameters. Adobe Technical Note #5151, *Acrobat Distiller Parameters*, describes the Distiller parameters and, in general, how you set those parameters using a PostScript program. 6.3, "Setting Distiller Parameter Values", provides specific examples of how to use PostScript segments to set Distiller parameters.

1.2 Comparing Adobe PDF Converter SDK and Distiller

This section compares the features of the Adobe PDF Converter SDK against those of Distiller.

The Adobe PDF Converter SDK and the Distiller are similar in their ability to convert PostScript streams into PDF files. However, the Adobe PDF Converter SDK allows OEMs to customize various aspects of that conversion process. In general, the Distiller does not support such customization.

1.2.1 Similarities

The Adobe PDF Converter SDK is similar to Distiller Version 10.0 in the following ways:

- Both produce PDF files and streams that are compliant with Portable Document Format, Version 1.7 and earlier.
- The Adobe PDF Converter SDK supports all Distiller parameters, except those parameters that require post processing. Examples of such post-processing parameters include **Optimize** and **DoThumbnails**. Distiller uses the PDF Library for post-processing.
- They take roughly the same time and resources to convert a file.

1.2.2 Differences

Although the Adobe PDF Converter SDK and the Distiller have the same primary function of converting PostScript streams/files into PDF format, they also have

differences that might make one a better solution for your application than the other. These differences are described in Table 1.1 (what they consume), Table 1.2 (what they produce), Table 1.3 (how they are controlled), Table 1.4 (what parameters they support), Table 1.5 (how they manage fonts), and Table 1.7 (other differences).

TABLE 1.1 Differences in what Adobe PDF Converter SDK and Distiller consume

Capability	Adobe PDF Converter SDK 3.2	Distiller 10
Support of PostScript File System Emulation	Supported	Cannot be configured to support PostScript File System Emulation.
Format of PostScript data	Accepts PostScript streams from its client.	Accepts PostScript files.

TABLE 1.2 Differences in what Adobe PDF Converter SDK and the Distiller produce

Capability	Adobe PDF Converter SDK 3.2	Distiller 10
Comment substitution	Can perform comment substitution. As the Adobe PDF Converter SDK encounters comments, it reports them to the client. The client may then direct the Adobe PDF Converter SDK to replace some or all of those comments with any PostScript expressions.	Does not provide such a capability.
Reporting page information	Notifies the client about page information found in PostScript and DSC comments, specifically the page label, plate color, and page device keys.	Does not provide such a capability.
Producing PDF for individual pages	The Adobe PDF Converter SDK can produce PDF streams for individual pages, in addition to full-document PDF files.	Not supported. Distiller only produces full-document PDF files.
Producing PDF for a set of pages	The Adobe PDF Converter SDK can produce PDF streams for a set of pages.	Not supported. Distiller only produces full-document PDF files.

TABLE 1.2 Differences in what Adobe PDF Converter SDK and the Distiller produce *(Continued)*

Capability	Adobe PDF Converter SDK 3.2	Distiller 10
Producing PDF for image files	The Adobe PDF Converter SDK can produce PDF streams for image files.	Not supported.
Producing PDF for PPML files	The Adobe PDF Converter SDK can produce PDF streams for PPML files.	Not supported.
Reporting page device keys	Reports page device keys. Can also represent such keys in PJTF objects added to the PDF file.	Does not report page device keys. However, Distiller represents such keys in PJTF objects.
Image stream sidelining	Can write image data to external files.	Does not provide such a capability.
EPS sidelining	Can set aside conforming embedded EPS programs, rather than converting them. **NOTE:** This feature may be discontinued in future versions of the Adobe PDF Converter SDK.	Does not provide such a capability.
Support of PostScript File System emulation	Allows you to implement software that uses PostScript File System emulation.	Does not provide such a capability.
Support for creation of reference Xobject	Allows you to create reference Xobjects instead of embedded Xobject.	Does not provide such a capability.

TABLE 1.3 Differences in how the Adobe PDF Converter SDK and the Distiller are controlled

Capability	Adobe PDF Converter SDK 3.2	Distiller 10
User interface	Does not provide a user interface; however, you could develop one as part of your client software.	Has a GUI that can be modified.
Support of hot folders (watched folders) and job submissions	Does not provide such support; however, you could customize your client software to provide such capabilities.	Supports hot folders and job submission.

TABLE 1.3 Differences in how the Adobe PDF Converter SDK and the Distiller are controlled

Capability	Adobe PDF Converter SDK 3.2	Distiller 10
Support of ICC profile selection from ACE default ICC profile directories.	On Windows, the Adobe PDF Converter SDK allows selection of ICC profiles from any/all of the default ICC profile directories. These directories include the Adobe Color Profiles Recommended folder, the Adobe Color Profiles folder and the system Color Profiles folder.	Searches all default ICC profile directories.
Support for ICC profiles located in non-default folders.	Supported. On Windows, such non-default folders are in addition to default ICC profile directories. On Mac, the Adobe PDF Converter SDK supports either ICC profiles located in non-default folders or those located in ACE default ICC profile directories.	Not supported

TABLE 1.4 Differences in supported Distiller parameters

Capability	Adobe PDF Converter SDK 3.2	Distiller 10
DoThumbnails	Not supported. (Distiller performs optimization as a post-processing task. The Adobe PDF Converter SDK does not support such tasks.)	Supported. This feature allows thumbnails to be embedded in the PDF, speeding up the time to display thumbnails at the expense of file size.
CompressObjects	Not supported. (Distiller performs optimization as a post-processing task. The Adobe PDF Converter SDK does not support such tasks.)	Supported
Optimize	Not supported. (Distiller performs optimization as a post-processing task. The Adobe PDF Converter SDK does not support such tasks.)	Supported. This feature optimizes a PDF file for Web viewing. It does so restructuring the file for page-at-a-time downloading and by compressing text and line art regardless of the Compression settings.

TABLE 1.4 Differences in supported Distiller parameters

Capability	Adobe PDF Converter SDK 3.2	Distiller 10
UsePrologue	Supported.	Supported
EmbedJobOptions	Not supported. (Setting the `EmbedJobOptions` Distiller parameter to TRUE in a Job Options file has no effect in Adobe PDF Converter SDK, running on Windows. When this parameter is set to TRUE, then Distiller 7.0 and later, embeds the Job Options file in the PDF as an attachment.)	Supported

TABLE 1.5 Differences in how the Adobe PDF Converter SDK and the Distiller support and manage fonts

Capability	Adobe PDF Converter SDK 3.2	Distiller 10
Control over host font cache.	Allows multiple products based on the Adobe PDF Converter SDK to use the same host font cache. Such control supports implementation of parallel conversion/distillation. (Windows only)	Not supported
Font policies	Uses the same font policies as Distiller, with the following exceptions: • The Adobe PDF Converter SDK may be directed to always embed (in the PDF file produced) fonts contained in the PostScript stream.	Can embed fonts included in the PostScript stream, only if the user has specified the names of those fonts in the list of fonts to be embedded.
Downloading CJK fonts	You can modify your client software to use the Adobe PDF Converter SDK to download CJK fonts. Such a capability would set up a connection between the back channel and a font downloader.	Cannot download CJK fonts because it does not support PostScript File System emulation.

***TABLE 1.5 Differences in how the Adobe PDF Converter SDK and the Distiller support and manage fonts** (Continued)*

Capability	Adobe PDF Converter SDK 3.2	Distiller 10
Creating substitute fonts	Allows the client to specify that missing fonts should be substituted using the ATM database or to disable such synthesis.	Always attempts to synthesize missing fonts.
Specifying a replacement font	For non-CID fonts, the job options may contain a sequence that states whether a default font can be used and, if so, which font. (the Adobe PDF Converter SDK uses a *replacement font* when the font specified in the PostScript stream can neither be found nor synthesized.)	For non-CID fonts, Distiller always uses Courier as the replacement font. You could create job options containing default font instructions; however, the Distiller UI does not support the creation of such a file.
Embedding a font that is embedded in the PostScript data	In the PDF file, the Adobe PDF Converter SDK may be directed to embed fonts that are contained in the PostScript stream, regardless of Distiller parameters that might specify otherwise.	Only embeds fonts as directed by Distiller parameters and by the user (via the UI).
PAP font support	On Windows platforms that provide “Services for Macintosh” (SFM), the Adobe PDF Converter SDK can start in “PAP mode”, listening for AppleTalk connections, and honoring font security mechanisms. **NOTE:** If your system has an AppleTalk stack, the PAP interface code can be modified to allow PAP downloading to occur on platforms other than Windows.	• On a Mac, the Distiller printer can use the Quartz printing architecture to provide font services, including downloading fonts. • On a PC, there is no PAP support (even Macintosh accessible NT Printers on NT server will not work as they are not bi-directional).
Indication of duplicate fonts.	Supported	Not supported
Control over directories searched for host fonts.	Does not attempt to find non-system ATM font folders. Instead you must specify such folders through the the Adobe PDF Converter SDK API.	Allows the user to choose which folders to look for fonts. Also finds ATM font folders automatically.

TABLE 1.5 Differences in how the Adobe PDF Converter SDK and the Distiller support and manage fonts *(Continued)*

Capability	Adobe PDF Converter SDK 3.2	Distiller 10
Ability to use or ignore TrueType fonts	Can be directed to ignore the TrueType versions of standard PostScript fonts that appear in the resource search list. Appendix A lists the standard fonts. Otherwise, the Adobe PDF Converter SDK recognizes all TrueType fonts.	Can be directed to ignore ALL TrueType fonts that appear in the font locations. Otherwise, Distiller recognizes all TrueType fonts.

TABLE 1.6 Differences in security

Issue	Adobe PDF Converter SDK 3.2	Distiller 10
Restricting the directories searched by the PostScript Interpreter.	Directories search may be constrained to a particular set.	No such limitation supported.
Security in PDF files.	Not supported.	Distiller allows the user to specify passwords required to accomplish various tasks, such as printing or modifying a document. It also allows the PDF file to be encrypted.

TABLE 1.7 Other differences

Issue	Adobe PDF Converter SDK 3.2	Distiller 10
Honor for the `/Producer` key	Honors the `/Producer` key	Does not honor the `/Producer` key set by pdfmark. **Example**: The `[/Producer (My Producer_1) /DOCINFO` pdfmark is ignored. Alternatively, Distiller hardcodes the `/Producer` key to “Acrobat Distiller 8.0”

TABLE 1.7 Other differences

Issue	Adobe PDF Converter SDK 3.2	Distiller 10
Ability to change the DSC/PostScript content being Distilled.	Windows version allows you to replace a DSC comment with other DSC/PostScript. Also allows you to add PostScript after a DSC and/or to skip existing PostScript between a DSC comment and the next DSC comment.	Not supported
Provision for invoking client-supplied code in response to the externalcommand PostScript operator.	Supported	Not supported
Control over allocation and deallocation of VM.	Supported	Not supported
Supported platforms	• Windows 32 bit (Windows 7, Windows 8.1, Windows 10) • Windows 64 bit (Windows 7, Windows 8.1, Windows 10, Windows 2008 Server R2, Windows 2012 Server) • Linux x86 64 bit (RHEL 7) *(APC SDK 3.2 is not released for Mac platform)	• Microsoft® Windows® XP, (32 bit and 64 bit); Windows Server® 2003 (32 bit and 64 bit; Windows Server 2008 or 2008 R2 (32 bit and 64 bit); Windows Vista® (32 bit and 64 bit); Windows 7 (32 bit and 64 bit)
Support of customized file I/O	Provides an API that allows a client to provide customized file I/O for full-document PDF files.	Does not provide such a capability.
PDF/X compliance test support	Adobe PDF Converter SDK does not support the `PDFX1aCheck` and `PDFX3Check` Distiller parameters. As there are more ways of PDF X and A checking available in PDF Converter SDK, a name value in an array called `CheckCompliance`, is used instead of the `PDFX1aCheck` and `PDFX3Check` Booleans.	Distiller 7.0 and later versions continue to support the `PDFX1aCheck` and `PDFX3Check` Distiller parameters.

TABLE 1.7 Other differences

Issue	Adobe PDF Converter SDK 3.2	Distiller 10
Support for generation of PDF/X-4 compatible PDF.	Allow user to generate PDF/X-4 compliant PDF.	Does not provide such a capability.

1.3 Support of UNICODE Character Strings

This section highlights the character strings for which the client can provide UNICODE character strings.

1.3.1 In Pathnames

For Files Consumed by the Adobe PDF Converter SDK

Pathnames for the following types of files may only use standard character encoding:

- Resource search lists
- External files that appear in the PostScript program with the **run** operator

For Files Produced by the Client

Pathnames for the following types of files may use either standard or UNICODE character encoding:

- Files containing PDF page streams
- Files containing streams for PDF with multiple pages
- Sidelined EPS programs
- Sidelined images
- Files containing streams for PDF with multiple pages
- Full-document PDF files. When the full-document PDF file is created by the client, and the output pathname is omitted, or when the client provides a customized file I/O library for the full-document PDF file.

For Files Produced by the Adobe PDF Converter SDK

Pathnames for the following types of files may only use standard character encoding; they must not use UNICODE.

- Scratch file directories. (The client provides the directory names, but the Adobe PDF Converter SDK determines the individual file names.)
- Full-document PDF files, when pathname is provided by the client software.

2 New in this release: APC 3.2

This chapter describes the new features in Adobe PDF Converter SDK 3.2.

2.1 New Features

- New Era Japanese Ligature support is added in Adobe PDF Converter SDK 3.2.
- APC CSL 3.2 has updated CMaps to support the New Era Japanese ligature in APC SDK.
- Algorithm for Image Data conversion in ConvertToIndexedImage() API is optimized to improve the performance in APC SDK.
- Composite font deletion mechanism is improved by adding modification to MakeFontFlush() API.

3 Architectural Overview

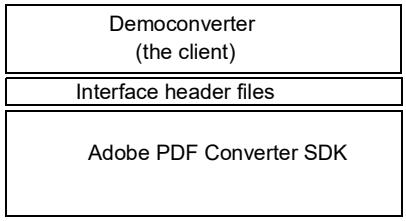
This chapter describes the relationship between the Adobe PDF Converter SDK and its client software.

3.1 Basic Architecture

Figure 3.1 illustrates the relationship between the Adobe PDF Converter SDK and it's client software. The components in the illustration are described below:

- *Democonverter* — An Adobe PDF Converter SDK front-end. Consists of example code that the OEM uses as a model for developing a specific client. Democonverter is delivered as source and executables.
- *Adobe PDF Converter SDK* — The engine for converting PostScript files/streams image and PPML files into PDF files/streams. The Adobe PDF Converter SDK is delivered in binary form.

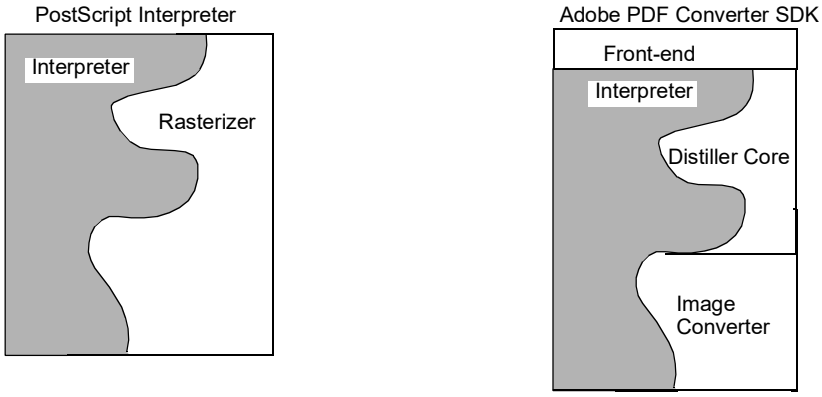
FIGURE 3.1 Democonverter architecture



3.2 Constituents of the Adobe PDF Converter SDK

The Adobe PDF Converter SDK includes the Distiller core software and the interpreter software from the PostScript Interpreter. (Despite its name, the PostScript Interpreter contains both interpreter and rasterizer software.) For both the PostScript Interpreter and the Adobe PDF Converter SDK, Figure 3.2 shows the relationship between the interpreter software and the other major components.

FIGURE 3.2 The PostScript Interpreter and the Adobe PDF Converter SDK contain interpreter software



This diagram shows the interwoven relationship between the Adobe PostScript Interpreter and the PostScript Rasterizer code. That relationship is partially duplicated between the Interpreter and the Distiller core.

3.3 Parallel Conversion

Parallel conversion uses multiple instances of the Adobe PDF Converter SDK together to process jobs more efficiently. For example, you can take advantage of the Adobe PDF Converter SDK feature of enabling and disabling individual pages in order to create a scenario where two versions of the Adobe PDF Converter SDK alternate page processing. In this way you can have each instance of the Adobe PDF Converter SDK process only a specific set of pages, such as all the odd numbered pages, from a job.

Remember these key architectural concepts when setting up and using parallel conversion:

- The Adobe PDF Converter SDK supports only a single client; in other words, it is not re-entrant. However, you can run multiple Adobe PDF Converter SDK processes from one Adobe PDF Converter SDK directory, subject to any license agreement.
- The Adobe PDF Converter SDK is not thread-safe; that is, you cannot use a single instance of democonverter launching multiple threads, each with a separate instance of the Adobe PDF Converter SDK.
- Each Adobe PDF Converter SDK process must use a unique (that is differently named) scratch file directory.
- If multiple Adobe PDF Converter SDK instances are sharing a host font cache, then initialize all instances with the same host font information. Do not change the

host font information after initialization. Alternatively, if you must change the host font information after initialization, then use separate host font files for each the Adobe PDF Converter SDK instance.

FIGURE 3.3 Using multiple copies of Adobe PDF Converter SDK without mutex callbacks

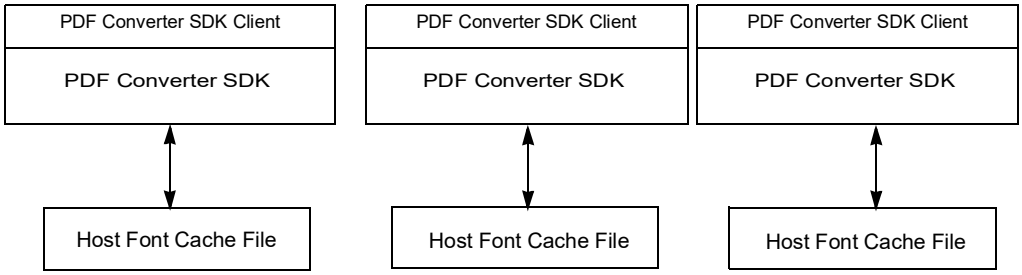
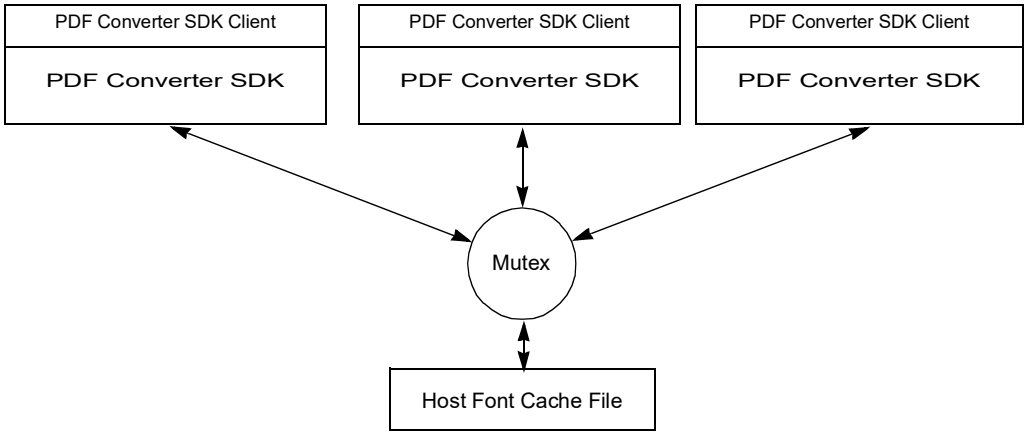


FIGURE 3.4 Using multiple copies of Adobe PDF Converter SDK with mutex callbacks



When setting up instances of the Adobe PDF Converter SDK for parallel conversion, access to host font information can be handled in two ways:

- Figure 3.3 illustrates a situation where the host fonts are not shared in a common cache file. In this case, each instance of the Adobe PDF Converter SDK uses a unique host font cache file.
- Alternatively, you can take advantage of the Windows-only `NSGetHostFontMutexProc()` and `NSReleaseHostFontMutexProc()` callbacks, illustrated in Figure 3.4. These callbacks allow you to have a single host font cache rather than having one host font cache file for each Adobe PDF Converter SDK instance. Through use of the callbacks, you can control access to the single font cache so that the first Adobe PDF Converter SDK instance to access the cache effectively locks access to the cache for other Adobe PDF Converter SDK instances. Once the cache has been released, it is accessible by the next Adobe PDF Converter SDK instance.

4 About the Deliverable Files

This chapter describes the organization and contents of the Adobe PDF Converter SDK deliverables. The directory structure has changed from previous deliveries. These deliverables include the files required to build and execute democonverter, including the object and header files for the Adobe PDF Converter SDK.

Libraries and executables do not contain debugging information.

TABLE 4.1 Adobe PDF Converter SDK first level files and directories

Directory or file	Description
Windows: ACE.dll **Linux:** libACE.so	Contains the Adobe Color Engine (ACE).
Windows: AGM.dll **Linux:** libAGM.so	Contains the Adobe Graphics Manager (AGM).
Windows: BIB.dll **Linux:** libBIB.so	The Bravo Interface Binder (BIB) linked object used by the ACE and AGM.
Windows: BIBUtils.dll **Linux:** libBIBUtils.so	The Bravo Interface Binder Utilities linked object used by the ACE and AGM.
democonverter or democonverter.build	Directory containing the Democonverter include files (Table 4.3), platform-specific library files (Table 4.4), Democonverter source files (Table 4.5), and platform-specific project or make files.
empty.ps	Dummy file required for the AppleTalk demo to run.
fonts	Directory of fonts democonverter uses as its font resource. **NOTE:** The Adobe PDF Converter SDK ships with the Courier font included. You must purchase additional fonts separately.
ICCProfiles	Directory containing the profiles that allow the Adobe PDF Converter SDK to use the Adobe Color Engine (ACE). See Table 4.2 for a description of the files in this directory.
Windows: JP2KLib.dll **Linux:** libJP2K.so	The linked object that Adobe PDF Converter SDK uses to support JPEG2000.

TABLE 4.1 Adobe PDF Converter SDK first level files and directories

Directory or file	Description
Windows: apc.dll, ACE.dll, AdobeXMP.dll, AGM.dll, AXE8SharedExpat.dll, BIB.dll, BIBUtils.dll, CoolType.dll **Linux**: libPDFL.so, libACE.so, libAdobeXMP.so, libAGM.so, libAXE8SharedExpat.so, libBIB.so, libBIBUtils.so, libCoolType.so **MacOS**: apclib, AdobeACE, AdobeXMP, AdobeAGM, AdobeAX8SharedExpat, AdobeBIB, AdobeBIBUtils, AdobeCoolType	Dynamically linked object code of Adobe PDF Converter SDK and its constituents, including the Distiller core.
Windows: AdobePDFL.dll, ImagetoPDF.dll, JP2KLib.dll **Linux**: libpdfl.so, libImagetoPDF.so, libJP2K.so **MacOS**: AdobePDFL, imagetopdf, AdobeJP2K	The linked object that Adobe PDF Converter SDK uses for converting image files to PDF files.
Linux: libc++abi.so.1, libc++.so.1, libgcc_s.so.1, libstdc++.so.6	GCC specific shared libraries.
APC.pdf	This document.

TABLE 4.1 Adobe PDF Converter SDK first level files and directories

Directory or file	Description
`Windowsps.vm` `Linux:PS.VM`	A file that Adobe PDF Converter SDK uses to initialize the PostScript Interpreter's virtual memory.
`ReadMe.html`	Late breaking information about this product.
`Resource`	Directory containing resources used by democonverter. See Table 4.4 for a description of the files in this directory.
`Resource\ColorRendering`	A directory containing several color rendering dictionaries (CRDs).
`Resource\ProcSet`	A directory containing ProcSet dictionaries used to achieve various production objectives. These dictionaries are used for historical reasons; they are not unique to the Adobe PDF Converter SDK. You shouldn't need to modify the ProcSets.
`Resource/CMap`	Directory containing CMaps, these CMaps are used by CID fonts
`Resource/Color`	Directory containing ICC Profile
`Resource/Font`	Directory containing Fonts
`Resource/Unicode`	These files are required by font to perform character-code glyph-id conversion appropriately
`Settings`	This folder contains job options files.
`startupNorm.ps`	Sample startup PostScript program that democonverter uses to initialize the PostScript Interpreter. democonverter will run without `startupNORM.ps`; however, the PostScript Interpreter's default setup is left unchanged.
`superatm.db`	Adobe Type Manager (ATM) database, which may be used by the Adobe PDF Converter SDK to substitute missing fonts. NOTE: Democonverter will run without `superatm.db`; however, the Adobe PDF Converter SDK will be unable to perform font emulation (font fauxing).

TABLE 4.2 Files in ICCProfiles

File name (Names do not vary across platforms)	Description
AdobeRGB1998.icc	A binary data file for the AdobeRGB1998 ICC profile.
AppleRGB.icc	A binary data file for the AppleRGB ICC profile.
BlackWhite.icc	A binary data file for the BlackWhite ICC profile.
CoatedFOGRA39.icc	A binary data file for CoatedFOGRA39 ICC profile.
CoatedGRACoL2006.icc	A binary data file for CoatedGRACoL2006 ICC profile.
CIERGB.icc	A binary data file for the CIERGB ICC profile.
ColorMatchRGB.icc	A binary data file for the ColorMatchRGB ICC profile.
EuropeISOCoatedFOGRA27.icc	A binary data file for EuropeISOCoatedFOGRA27 ICC profile.
EuroscaleCoated.icc	A binary data file for the EuroscaleCoated ICC profile.
EuroscaleUncoated.icc	A binary data file for the EuroscaleUncoated ICC profile.
JapanColor2001Coated.icc	A binary data file for the JapanColor2001Coated ICC profile.
JapanColor2001Uncoated.icc	A binary data file for the JapanColor2001Uncoated ICC profile.
JapanColor2002Newspaper.icc	A binary data file for JapanColor2002Newspaper ICC profile.
JapanStandard.icc	A binary data file for the JapanStandard ICC profile.
JapanWebCoated.icc	A binary data file for the JapanWebCoated ICC profile.
PAL_SECAM.icc	A binary data file for the PAL_SECAM ICC profile.
ProPhoto.icm	A binary data file for the ProPhoto.icm ICC profile
SMPTE-C.icc	A binary data file for the SMPTE-C ICC profile.
UncoatedFOGRA29.icc	A binary data file for the UncoatedFOGRA29 ICC profile.
USSheetfedCoated.icc	A binary data file for the USSheetfedCoated ICC profile.
USSheetfedUncoated.icc	A binary data file for the USSheetfedUncoated ICC profile.

TABLE 4.2 Files in ICCProfiles

File name (Names do not vary across platforms)	Description
USWebCoatedSWOP.icc	A binary data file for the USWebCoatedSWOP ICC profile.
USWebUncoated.icc	A binary data file for the USWebUncoated ICC profile.
WebCoatedFOGRA28.icc	A binary data file for the WebCoatedFOGRA28 ICC profile.
WebCoatedSWOP2006Grade3.icc	A binary data file for the WebCoatedSWOP2006Grade3 ICC profile.
WebCoatedSWOP2006Grade5.icc	A binary data file for the WebCoatedSWOP2006Grade5 ICC profile.
WideGamutRGB.icc	A binary data file for the WideGamutRGB ICC profile.
sRGB Color Space Profile.icm	A binary data file for the sRGB Color Space Profile ICC profile.

TABLE 4.3 Files in the directory democonverter/includes or democonverter.build/includes

File name	Description
ASBasic.h	Defines Adobe Standard Definitions of “Basic” parameter types.
ASEnv.h	Defines the Adobe Standard Environment, including the compilation parameter settings. Includes definitions of the allowed values for the required AS_ISP and AS_OS compiler switches. It also provides definitions of configuration parameters based on those two switches.
devcoord.h	Defines various data structures used in democonverter.
environment.h	Defines symbolic names used to declare various configuration details, including the type of the instruction set processor (type of machine), the operating system, and other information about the environment in which the resulting binary will operate.
apcif.h	Defines the interface between the client and the Adobe PDF Converter SDK.

TABLE 4.3 Files in the directory democonverter/includes or

File name	Description
opsys.h	Initializes the opsys package for products that do not use os_StartTheWorld or os_rtosStart from the task.h interface.
os_errno.h	Recasts Adobe error codes into Posix error codes.
os_pthread.h	Directly defines os_pthread_mutex_t, os_pthread_cond_t, and the various mutex and cond functions in Posix environments, in terms of the underlying Posix structures and functions.
os_time.h	Defines the real time, high-resolution clock for the operating system.
posix_environment.h	Contains macro definitions enabling certain declarations in system-dependent header files (if building for a system with some level of Posix compliance). These macro definitions are harmless on a system that has no Posix support. NOTE: This file must be included at the start of any .c file that requires it. Only package_spec.h may be included before it. This is because _POSIX_C_SOURCE must be defined before any system header files are included. Since other .h files may include a system header, the only safe practice is to include this file first.
protos.h	Contains a set of macros that optionally generate ANSI function prototypes that depend on the PROTOTYPES switch found in the file environment.h. The macros serve as good documentation, and can help ANSI compilers catch type-mismatch errors.
publictypes.h	Defines the public types that allow Distiller to build.
spdkeys.h	Defines enumerations for the page device keys that the Adobe PDF Converter SDK can recognize.

TABLE 4.4 Files in the democonverter//libs or democonverter.build/*/libsdirectory*

Directory or file name	Description
Windows:ix86win32	A directory containing the files normpap.lib and apc.lib needed by the development environment.

TABLE 4.5 Files in democonverter/sources or democonverter.build/sources

Directory or file name	Description
`Windows:demofepapwin32.c`	The source code for Democonverter, which allows font downloading over a PAP channel to the Windows Adobe PDF Converter SDK. This means Japanese fonts can be installed on a Adobe PDF Converter SDK only system.
`All platforms:demomain.c`	The source code for democonverter front-end.
`Windows:demopap.c`	The source code for the Democonverter example that allows PAP font downloading. **NOTE:** You can implement PAP support on any platform that has an Appletalk stack.
`Windows:demopap.h`	The header code for the Democonverter example that allows PAP font downloading.

`Windows:`
`norm_win_package_specs.h`
`Linux:`
 	
`norm_unix_package_specs.h`
	Macro definitions that support the header files.

5 Building and Using Democonverter

This chapter lists the supported platforms and compilers for Adobe PDF Converter 3.2. It also explains how to build and use democonverter, the sample client for the Adobe PDF Converter SDK.

5.1 Supported platforms and compilers

Table 5.1 lists the supported platforms and compilers for Adobe PDF Converter 3.2.

TABLE 5.1 Supported platforms and compilers

Platform	OS	Compiler
Windows 32 bit	Windows 7, Windows 8.1, Windows 10	Visual Studio 2017(Version 15.3.3)
Windows 64 bit	Windows 7, Windows 8.1, Windows 10, Windows 2008 Server R2, Windows 2012 Server	Visual Studio 2017(Version 15.3.3)
Linux x86 64 bit	RHEL 7	Clang 3.9, glib 2.17

*Adobe PDF Converter SDK 3.2 is not released for Mac Platform

5.2 Building Democonverter

This section explains how to build Democonverter on all supported platforms: Windows (32 and 64 bit), Linux (64 bit).

NOTE: Democonverter and Adobe PDF Converter SDK are not set up to support double-byte (CJK) fonts. If you wish to support CJK fonts, please see your Adobe Developer Support Engineer.

5.2.1 Windows

The following steps explain how to build democonverter in a Windows environment:

1. Start Microsoft Visual Studio 2017.

2. Do one of the following:

- For windows 32 bit: Open the democonverter.vcxproj project file located in **`democonverter\ix86win32\`** included with the democonverter source code..
- For windows 64 bit: Open the democonverter.vcxproj project file located in **`democonverter\ix86win64\`** included with the democonverter source code.

3. Select the release or debug build that you require.
4. Build the project.
5. Copy the resulting `democonverter.exe` to the root of the distribution and execute it.

5.2.2 Linux

The following steps explain how to build democonverter in Linux environment:

1. Do one of the following:

- For Linux 64 bit: Change the current working directory to one of the build directories, by typing one of the following:
  - cd democonverter.build/pentiumlinux_64_DEVELOP
  - cd democonverter.build/pentiumlinux_64_EXPORT

NOTE: The GNU C Compiler must be available in the path.

2. Type the make command.

5.3 Using Democonverter

5.3.1 Basic Command-Line Initialization

The following steps explain how to initialize Democonverter from a command line window:

1. Set the root directory to the SDK delivery.
2. To set up Distiller parameters with job options, type the following:

```
democonverter -efi +n -O . -B joboptionfile.joboption
postscript.ps
```

For information on the Democonverter command line format and options, type `democonverter -h` at the command prompt.

5.3.2 Specifying Pathnames in Command Lines

This section describes requirements for specifying pathnames in democonverter command lines. See `democonverter.c` for examples.

General Pathname Requirements

Pathnames must have the following characteristics:

- Case-sensitive
- Standard characters
- On Windows, absolute pathnames must include the hard drive.

Platform-Specific Separators

Pathnames in democonverter command lines must use pathname separators appropriate for the platform.

An example of the Windows command line to set the output directory follows:

```
-O diskX:\path\mydir\
-O diskX:\path\mydir\myfile
```

Or for a relative pathname:

```
-O .\path\mydir\
-O .\path\mydir\myfile
```

An example of the Linux command line to set the output directory follows:

```
-O /diskX/Users/myusername/documents/mydir/
-O /diskX/Users/myusername/documents/mydir/myfile
```

See `democonverter.c` for examples.

5.4 Democonverter PAP Font Support (Windows only)

Four separate builds of democonverter (containing both debug and release builds) and several supporting files have been added to the Windows version of the Adobe PDF Converter SDK to allow PAP font downloading. The builds are:

- democonverter - Win32 Debug
- democonverter - Win32 Release
- democonverter - Win32 NOPAP Debug
- democonverter - Win32 NOPAP Release

The supporting PAP files are:

- `normpap.lib`
- `demofepapwin32.c`
- `demopap.c`

AppleTalk Phase 2 must be loaded on the system in order for the PAP-enabled democonverter to work properly.

The parameter **APPLETALK_SUPPORT** must be set to 1 in the project to build PAP related code. If **APPLETALK_SUPPORT** is set to 0, then PAP will not be active.

To avoid major changes to `demomain.c`, a dummy file, `empty.ps` (provided) must be present in the democonverter working directory for the AppleTalk demo to run

NOTE: AppleTalk does not run on Microsoft Windows 98, Windows ME, or Windows XP Professional. If you need to create a PAP-enabled client, you must use Windows NT 4.0, Windows 2000 Server, or Windows 2003 Server.

6 Distiller Parameters

This section describes the default Distiller parameter values established by the Adobe PDF Converter SDK, how you can obtain a list of the current parameter values, and how your client software can set parameter values using the **setdistillerparams** operator.

6.1 Listing of Default Parameter Values

Example 6.1 lists the default Distiller parameter values that the Adobe PDF Converter SDK establishes with its PostScript component. You can override those values by modifying the startup file your client software provides to the Adobe PDF Converter SDK. The startup file is specified in the `startupFile` field of the `NSClientConfig` structure passed to `NormalizerServerInit()`.

NOTE: This list of defaults may change in future release, and it can even be changed by a badly behaved PostScript application. If you need to determine the current, true list of default Distiller parameter values, you can extract them by processing a PostScript job such as `showdefaults.ps` shown in Example 6.2.

EXAMPLE 6.1 Default Distiller parameter values

```
<<
 /LockDistillerParams  false
 /PDFXTrimBoxToMediaBoxOffset    [
 0
  0
  0
  0
   ]

 /DownsampleMonoImageMask  true
 /CannotEmbedFontPolicy  /Warning
 /DoThumbnails  false
 /AntiAliasMonoImages  true
 /InternalDropAllImageData  false
 /JPEG2000GrayACSImageDict   <<
  /TileWidth  256
  /TileHeight  256
  /Quality  15
 >>

 /CalRGBProfile  (sRGB IEC61966-2.1)

 /GrayImageMinResolution  150
 /MaxSubsetPct  100
 /AllowTransparency  false
 /CropGrayImages  true
 /JPEG2000ColorACSImageDict   <<
  /TileWidth  256
  /TileHeight  256
  /Quality  15
 >>

 /EncodeColorImages  false
 /CropColorImages  true
 /GrayImageMinResolutionPolicy  /OK
 /GrayImageFilter  /DCTEncode
 /CoreDistVersion  0
 /Optimize  true
 /EmitDSCWarnings  false
 /ParseDSCCommentsForDocInfo  true
 /AllowPSXObjects  true
 /CalGrayProfile  (None)

 /DSCReportingLevel  0
 /JPEG2000GrayImageDict   <<
  /TileWidth  256
  /TileHeight  256
  /Quality  15
```

```
>>

/GrayImageDownsampleThreshold  1.5
/CompressObjects  /Off
/JPEG2000ColorImageDict   <<
 /TileWidth  256
 /TileHeight  256
 /Quality  15
>>

/PDFXSetBleedBoxToMediaBox  true
/PreserveFlatness  true
/MonoImageResolution  300
/UseProlog ue  false
/GrayImageDict   <<
 /VSamples    [
 2
  1
  1
  2
    ]

 /QFactor  0.5
 /HSamples    [
 2
  1
  1
  2
    ]

 /Blend  1
>>

/CalCMYKProfile  (U.S. Web Coated (SWOP) v2)

/AutoFilterColorImages  true
/ColorImageDepth  -1
/sRGBProfile  (sRGB IEC61966-2.1)

/PreserveDICMYKValues  true
/PDFXTrapped  /False
/ColorImageAutoFilterStrategy  /JPEG
/PreserveOverprintSettings  true
/UCRandBGInfo  /Preserve
/AutoRotatePages  /None
/PDFXOutputIntentProfile  ()

/EmbedAllFonts  false
/CompatibilityLevel  1.3
```

```
/PassThroughJPEGImages  false
/StartPage  1
/AntiAliasColorImages  false
/CreateJobTicket  true
/ColorImageDownsampleType  /Average
/ColorSettingsFile  ()

/ColorImageDownsampleThreshold  1.5
/CreateJDFFile  false
/DetectBlends  true
/GrayImageDownsampleType  /Average
/ParseDSCComments  true
/PreserveEPSInfo  false
/PDFXRegistryName  ()

/MonoImageMinResolution  1200
/GrayACSImageDict   <<
 /VSamples    [
 2
  1
  1
  2
    ]

 /QFactor  0.76
 /HSamples    [
 2
  1
  1
  2
    ]

>>

/InternalOffOptimizations  0
/PDFXCompliantPDFOnly  false
/GrayImageAutoFilterStrategy  /JPEG
/ColorACSImageDict   <<
 /VSamples    [
 2
  1
  1
  2
    ]

 /QFactor  0.76
 /HSamples    [
 2
  1
```

```
  1
  2
    ]

>>

/PreserveCopyPage  true
/EncodeMonoImages  false
/MonoImageMinResolutionPolicy  /OK
/ColorImageMinResolutionPolicy  /OK
/ColorConversionStrategy  /LeaveColorUnchanged
/PreserveOPIComments  true
/NeverEmbed   [
 ]

/InternalForceUserUnit  0
/AntiAliasGrayImages  false
/GrayImageDepth  -1
/ColorImageResolution  72
/InternalDisablePathOptimizer  false
/AutoPositionEPSFiles  false
/EndPage  -1
/TransferFunctionInfo  /Preserve
/MonoImageDepth  -1
/CropMonoImages  true
/ColorImageMinResolution  150
/EncodeGrayImages  false
/EmbedOpenType  false
/ColorImageMinDownsampleDepth  1
/DownsampleGrayImages  false
/Description   <<
>>

/ConvertImagesToIndexed  false
/PresumeRGBRepresentationIsLossless  false
/PDFXOutputCondition  ()

/AllowRelativePathOps  false
/MonoImageDict   <<
 /K  -1
>>

/PassThroughFlateImages  false
/MonoImageDownsampleThreshold  1.5
/Binding  /Left
/DetectCurves  0.1
/MonoImageDownsampleType  /Average
/DownsampleMonoImages  false
/GrayImageResolution  72
```

```
/PDFX1aCheck  false
/PDFX3Check  false
/AlwaysEmbed   [
 ]

/ImageMemory  1048576
/SyntheticBoldness  1.0
/AutoFilterGrayImages  true
/SubsetFonts  true
/OffOptimizations  0
/PDFXOutputConditionIdentifier  ()

/OPM  1
/DefaultRenderingIntent  /Default
/ParseICCProfilesInComments  true
/MonoImageFilter  /CCITTFaxEncode
/EmbedJobOptions  false
/ColorImageFilter  /DCTEncode
/PDFXNoTrimBoxError  true
/GrayImageMinDownsampleDepth  2
/PDFXBleedBoxToTrimBoxOffset   [
0
 0
 0
 0
  ]

/DownsampleColorImages  false
/PreserveHalftoneInfo  true
/UseFlateCompression  true
/ASCII85EncodePages  false
/ColorImageDict   <<
 /VSamples    [
 2
  1
  1
  2
    ]

 /QFactor  0.5
 /HSamples    [
 2
  1
  1
  2
    ]

 /Blend  1
>>
```

```
/CompressPages  false
/CheckCompliance   [
None
  ]

>>
```

NOTE: Adobe PDF Converter SDK does not support the `PDFX1aCheck` and `PDFX3Check` Distiller parameters. As there are more ways of PDF X and A checking available in PDF Converter SDK, a name value in an array called `CheckCompliance` is used instead of the `PDFX1aCheck` and `PDFX3Check` Booleans.

6.2 Supported values of CheckCompliance key

```
/None
/PDFA:DRAFT
/PDFX1a:2001
/PDFX1a:2003
/PDFX3:2002
/PDFX3:2003
/PDFX4:2010
```

EXAMPLE 6.2 showdefaults.ps

```
%!PS
% program to dump a given dictionary

/string_buf 300 string def
/print_simple
{
   dup type /stringtype ne { string_buf cvs } if print
} bind def
/indent -1 def
/do_indent { 0 1 indent { pop (  ) print } for } def

/print_dict
{
   dup type /dicttype eq
   {
          do_indent (<<\n) print
          /indent indent 1 add def
          {
                 do_indent exch print_dict ( ) print print_dict (\n)
print
          } forall
```

```
            /indent indent 1 sub def
            do_indent (>>\n) print
      }
      {
            dup type dup /arraytype eq exch /packedarraytype eq or
            {
                  do_indent dup xcheck { ({\n) } { ([\n) } ifelse print
do_indent
                  /indent indent 1 add def
                  dup { print_simple (\n) print do_indent } forall
                  /indent indent 1 sub def
                  do_indent xcheck { (}\n) } { (]\n) } ifelse print
            }
            {
                  dup type /nametype eq
                  {
                        dup xcheck not { (/) print } if print_simple ( )
print
                  }
                  {
                        dup type /stringtype eq
                        {
                              (\() print print_simple (\)\n) print
                        }
                        {
                              dup type /nulltype eq
                              {
                                    pop (null) print_simple
                              }
                              {
                                    print_simple
                              } ifelse
                        } ifelse
                  } ifelse
            } ifelse
      } ifelse
} bind def

currentdistillerparams print_dict
%%EOF
```

6.3 Setting Distiller Parameter Values

You can change the value of a Distiller parameter through any of the mechanisms described in Table 6.1. Each subsequent method changes the value set by any

proceeding method. For example, Distiller parameters modified using method 3 override values established using method 2.

TABLE 6.1 Precedence of mechanisms for changing Distiller parameter values

#	Method
1.	Adding PostScript code or a PostScript file reference to the file `startupNORM.ps`, as described in "Adding PostScript Code to startupNORM.ps".
2.	Submitting an exitserver job to the Adobe PDF Converter SDK, as described in "Submitting an exitserver Job to Adobe PDF Converter SDK".
3.	Providing job options, as described in "Providing Job Options (Preferred Method)". Parameters set in this manner remain in effect through the end of the job.
4.	Augmenting the current job with PostScript code that uses **setdistillerparams** or **setpagedevice** PostScript key-word pairs, as described in "Setting Distiller Parameters from the Current Job". Parameters set in this method remain in effect through the end of the job.

NOTE: Some Distiller parameters are not supported by the Adobe PDF Converter SDK. See "Differences in supported Distiller parameters" on page 16 for more information.

6.3.1 Adding PostScript Code to startupNORM.ps

You can augment `startupNorm.ps` with PostScript code that changes any of the Distiller parameter default values except **LockDistillerParams**. Use the **setdistillerparams** PostScript key-word pair to change the Distiller parameter values. For example:

```
<< /CompressPages true >> setdistillerparams
```

`startupNorm.ps` is executed as an **exitserver** PostScript job. As a result, the Distiller parameter values it establishes remain until changed by another method.

6.3.2 Submitting an exitserver Job to Adobe PDF Converter SDK

As in Section 6.3.1, you can submit an **exitserver** PostScript job to the Adobe PDF Converter SDK, which uses the **setdistillerparams** PostScript key-word pair to change any of the Distiller parameter default values except **LockDistillerParams**. However, if this method is used during a PostScript job, changes to the parameters listed in the section "Caveats for Setting Distiller Parameters" in Adobe Technical Note 5151,

Acrobat Distiller Parameters have no effect on the current job. Rather, such settings become the new default values for the next and subsequent jobs.

6.3.3 Providing Job Options (Preferred Method)

On a per-job basis, your client may set Distiller parameters using the following method depending on the value of the `runMethod` field provided in the `NSJobParams` structure. That structure is one of the arguments passed to `NormalizerServerRunJob()`.

- *By providing job options*. If `runMethod` is set to `normRunJobOptions`, the Adobe PDF Converter SDK uses client-provided job options to set Distiller parameters for the current job. Adobe PDF Converter SDK obtains the job options from the `jobOptions` argument, or if not provided, by invoking the `bufGetJobOptions` callback.

The above method is preferred because it allows you to set all Distiller parameters and, unlike the startup PostScript or **exitserver** methods, it supports the **LockDistillerParams** parameter.

NOTE: Certain Distiller parameters should be used only in job options. For more information and a complete listing of these parameters, see the section “Caveats for Setting Distiller Parameters” in Adobe Technical Note 5151, *Acrobat Distiller Parameters*.

6.3.4 Setting Distiller Parameters from the Current Job

The PostScript document being normalized may contain **setdistillerparams** or **setpagedevice** commands to modify certain Distiller parameter values. If such code is not already part of the job, your client may add it when it responds to the `NSBufferGetPS()` or `NSProcessComment()` callbacks. The client should add the PostScript segment ahead of the PostScript stream for the job.

NOTE: If the job options sets the Distiller parameter **LockDistillerParams** to TRUE, any **setdistillerparams** key-word pairs appearing in the current job are ignored. **LockDistillerParams** can only be set using the job options method. That is, it cannot be set using an exitserver job. As a result, **LockDistillerParams** always reverts to FALSE at the end of a job.

7 Interactions Between Adobe PDF Converter SDK and Callbacks

This chapter describes the interactions between the Adobe PDF Converter SDK and the callbacks it invokes to transfer data and to relay other information, such as **setpagedevice** PostScript key-word pairs. Chapter 13, “Functions and Callbacks” describes the arguments and returned values declared for each callback.

A separate set of client file callbacks is used to interface with the Adobe PDF Converter SDK when the client manages I/O. For a discussion of these callbacks, see Chapter 8, “Using the NSClientFile API”.

7.1 Callbacks for Transferring Data between the Adobe PDF Converter SDK and a Client

This section describes the Adobe PDF Converter SDK-callback interaction for transferring data between the Adobe PDF Converter SDK and the client. For each callback, this section describes the functions the Adobe PDF Converter SDK expects the callback to perform. For details on the callback arguments and returned values, see Chapter 13, “Functions and Callbacks”.

7.1.1 How the Adobe PDF Converter SDK Uses Data Transfer Callbacks

Figure 7.1 contains a simplified flowchart that illustrates how the Adobe PDF Converter SDK interacts with the data transfer callbacks when **filePerPage** is FALSE in the **NSJobParams** structure, regardless of whether distillation is disabled. The parameters chosen for the example in Figure 7.1 represent the simplest type of interaction. Figure 7.2 through Figure 7.5 show the same interaction when the **filePerPage** field is TRUE with distillation enabled.

FIGURE 7.1 Flowchart showing client and the Adobe PDF Converter SDK interaction when filePerPage == FALSE and dynamicMode==FALSE

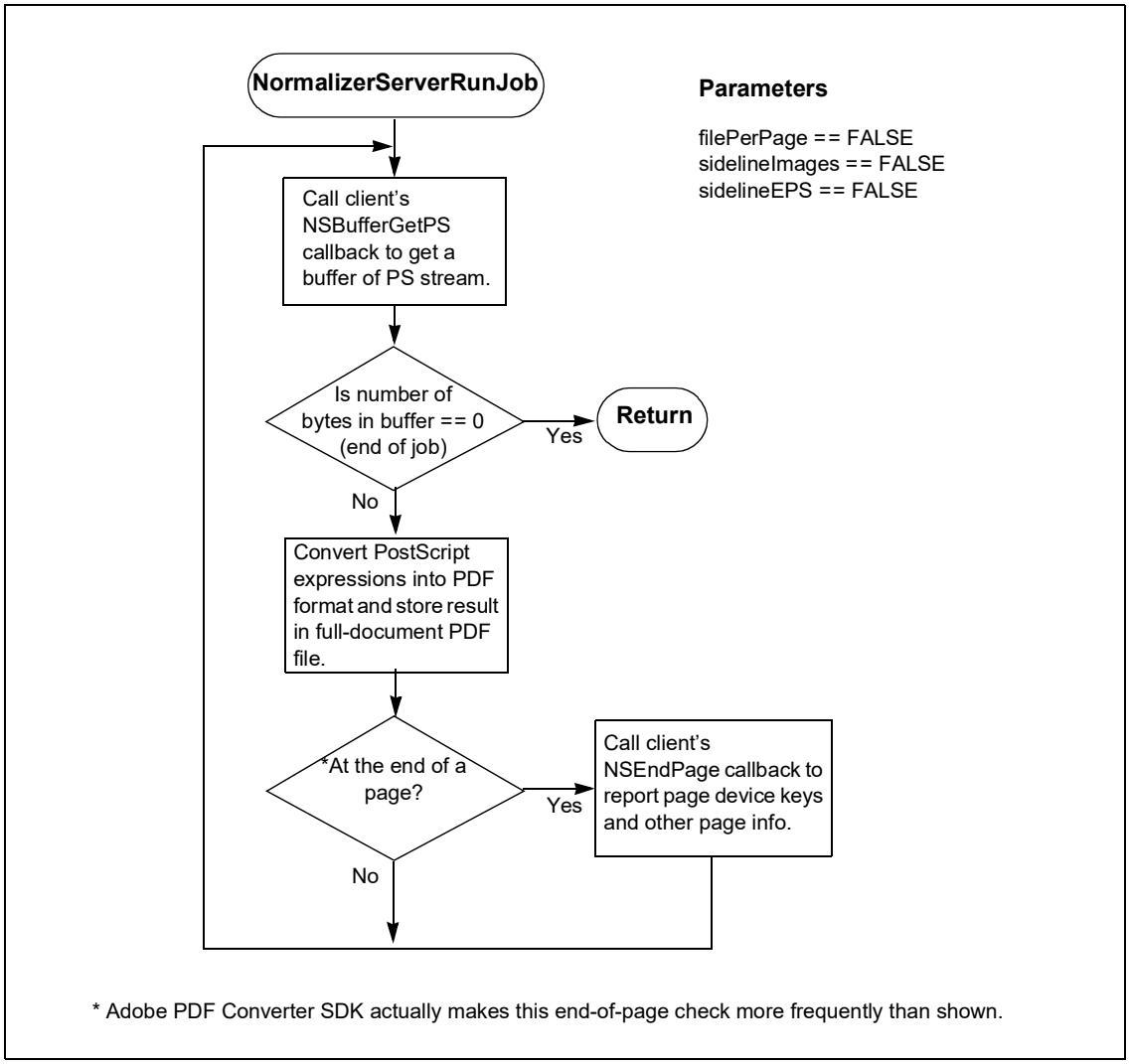


FIGURE 7.2 Flowchart showing client and the Adobe PDF Converter SDK interaction when filePerPage == TRUE (1 of 4)

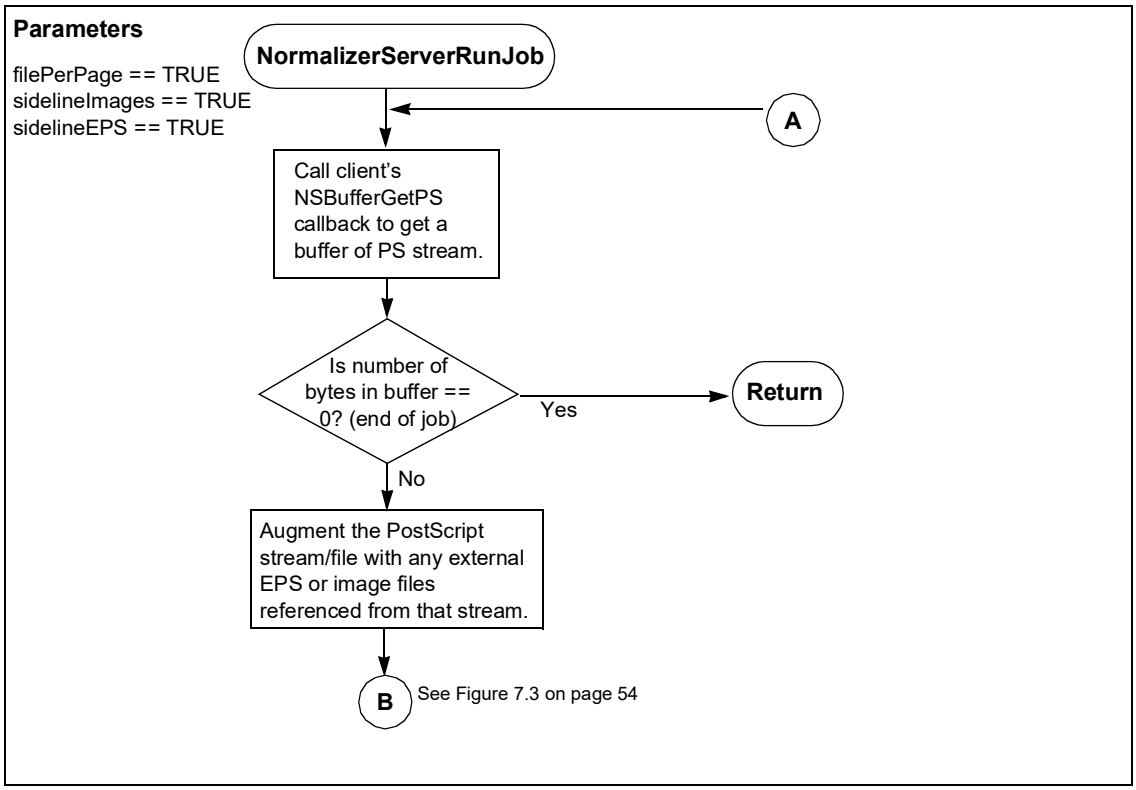
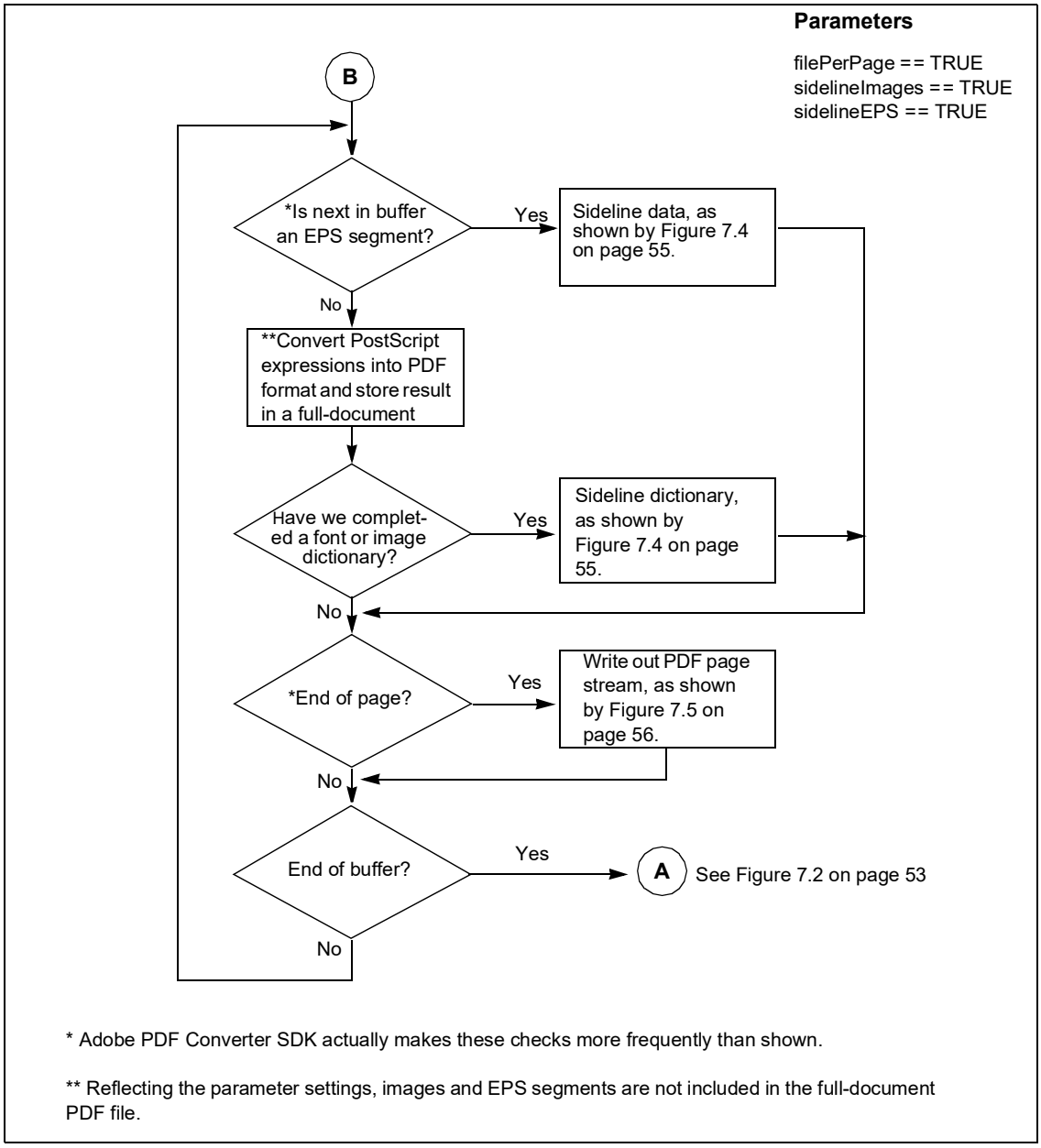


FIGURE 7.3 Flowchart showing client and the Adobe PDF Converter SDK interaction when filePerPage == TRUE (2 of 4)



* Adobe PDF Converter SDK actually makes these checks more frequently than shown.

** Reflecting the parameter settings, images and EPS segments are not included in the full-document PDF file.

FIGURE 7.4 Flowchart showing client and the Adobe PDF Converter SDK interaction when filePerPage == TRUE (3 of 4)

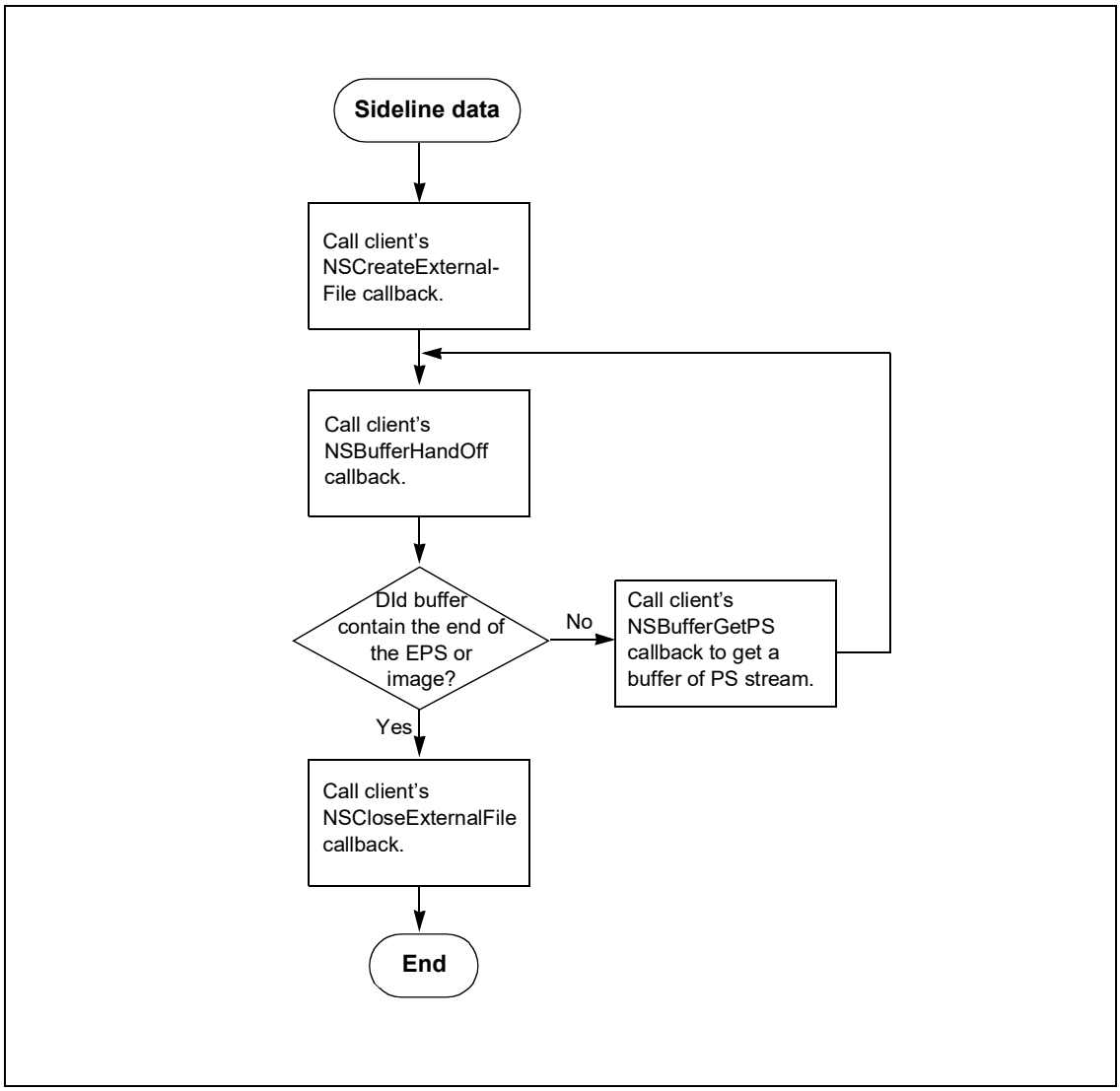
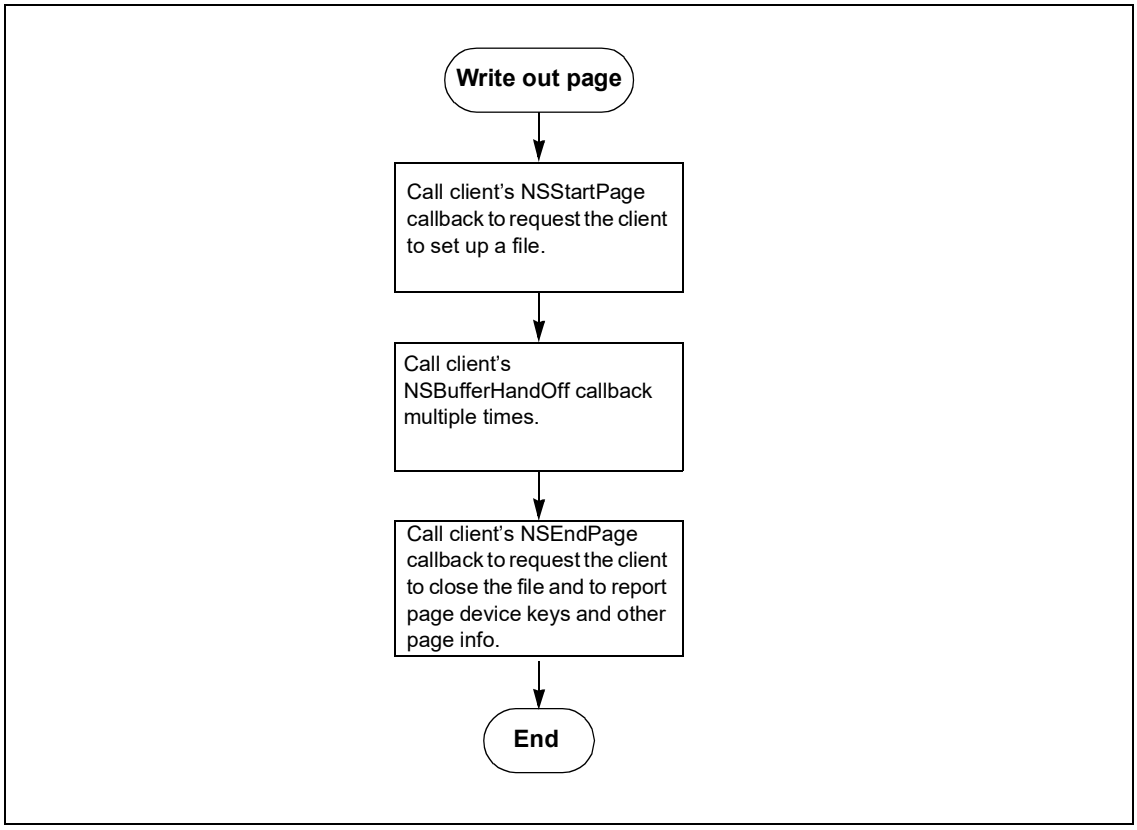


FIGURE 7.5 Flowchart showing client and the Adobe PDF Converter SDK interaction when filePerPage == TRUE (4 of 4)



The following flowcharts are applicable when dynamic mode is ON.

FIGURE 7.6 Flowchart showing client and the Adobe PDF Converter SDK interaction when dynamicMode == TRUE (1 of 4)

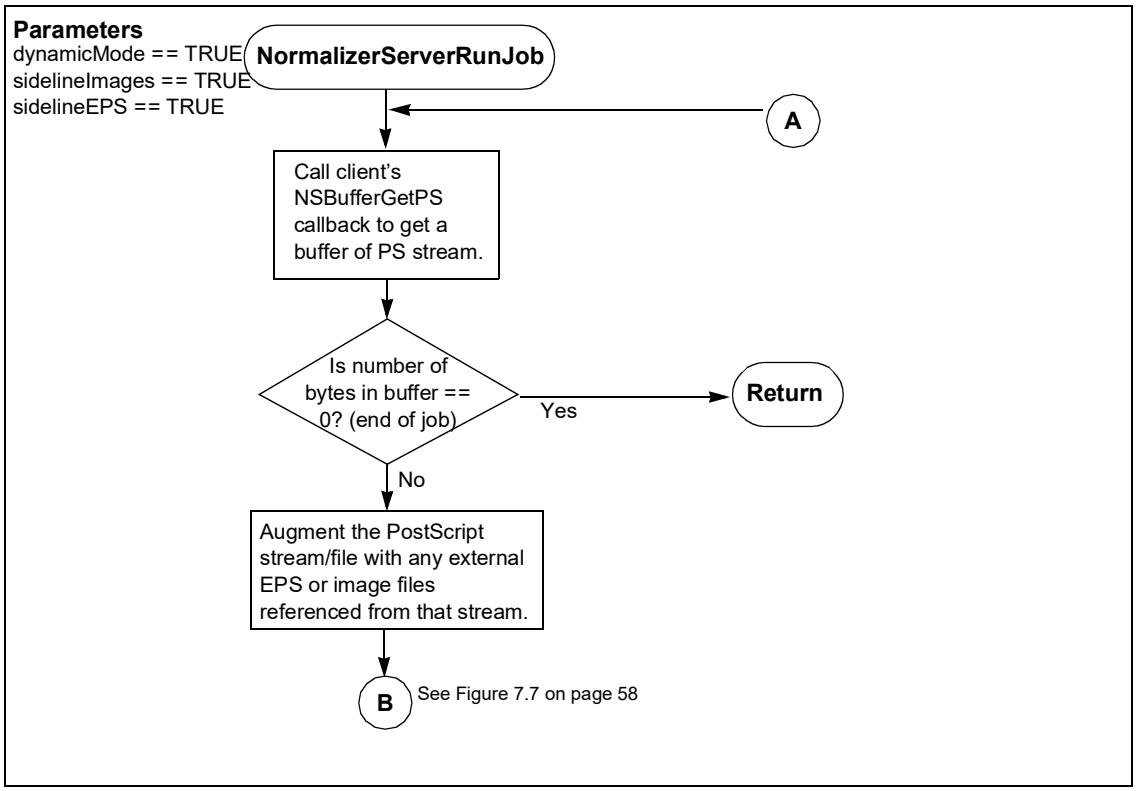
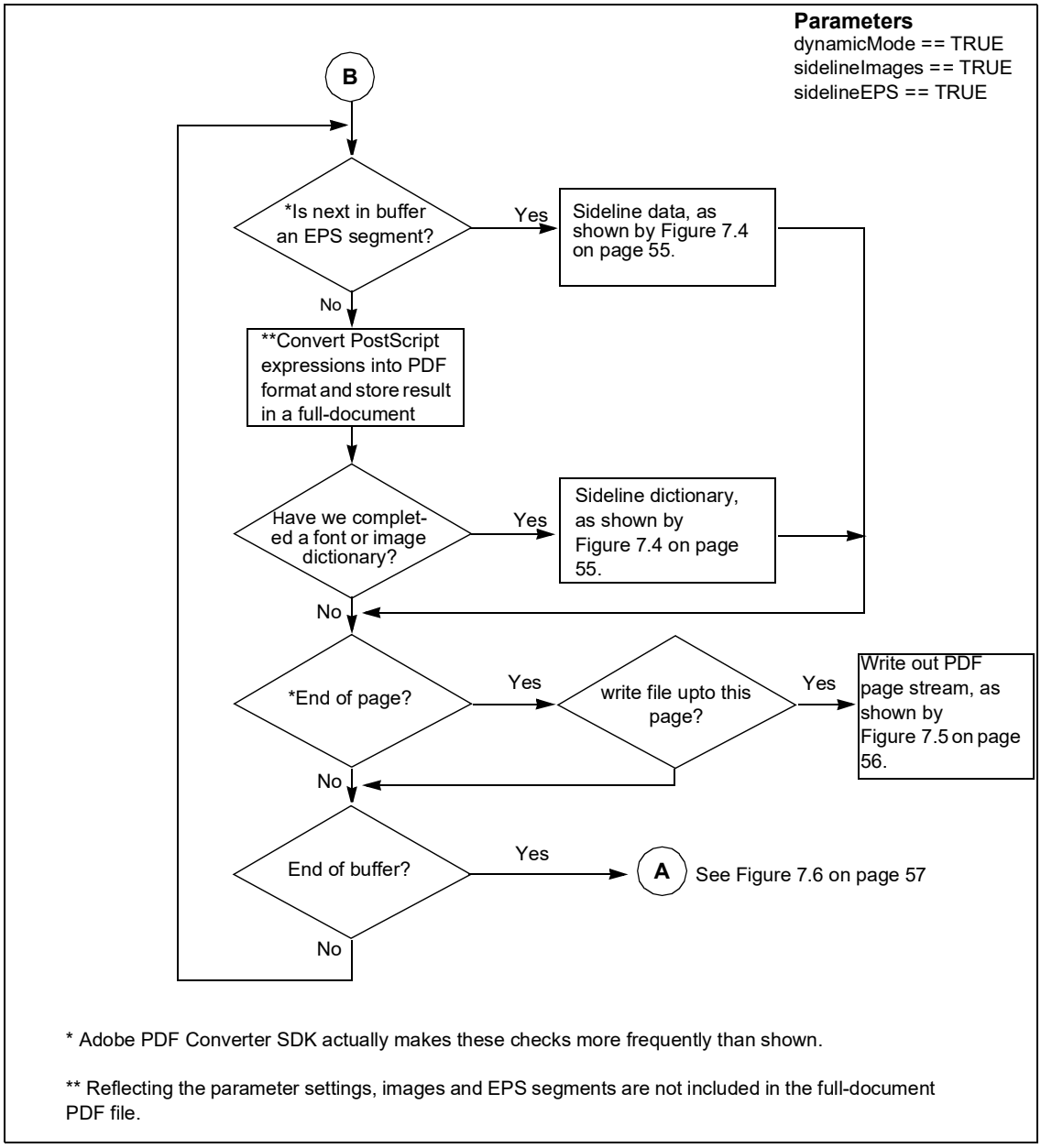


FIGURE 7.7 Flowchart showing client and the Adobe PDF Converter SDK interaction when dynamicMode == TRUE (2 of 4)



* Adobe PDF Converter SDK actually makes these checks more frequently than shown.

** Reflecting the parameter settings, images and EPS segments are not included in the full-document PDF file.

FIGURE 7.8 Flowchart showing client and the Adobe PDF Converter SDK interaction when dynamicMode == TRUE (3 of 4)

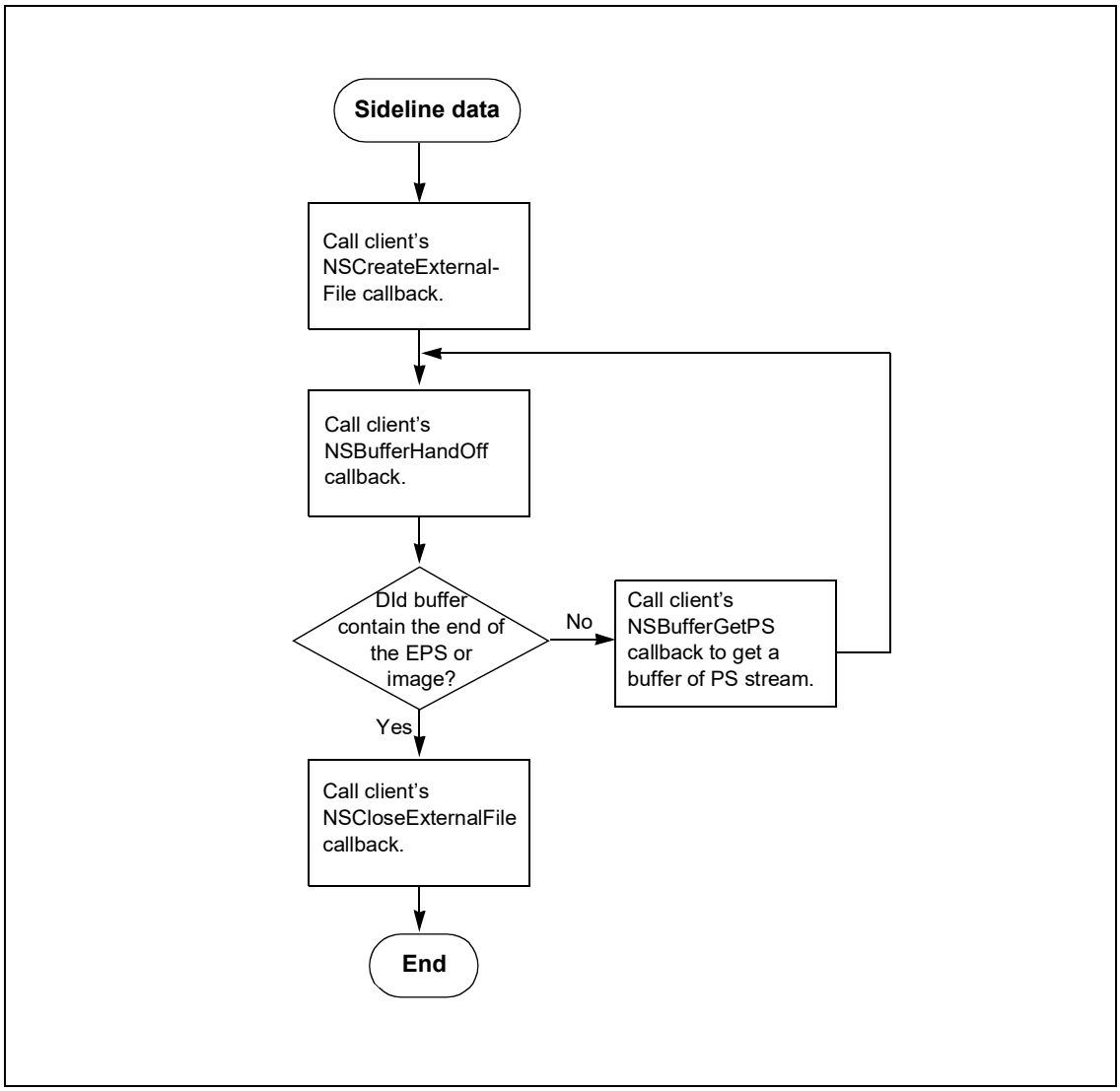
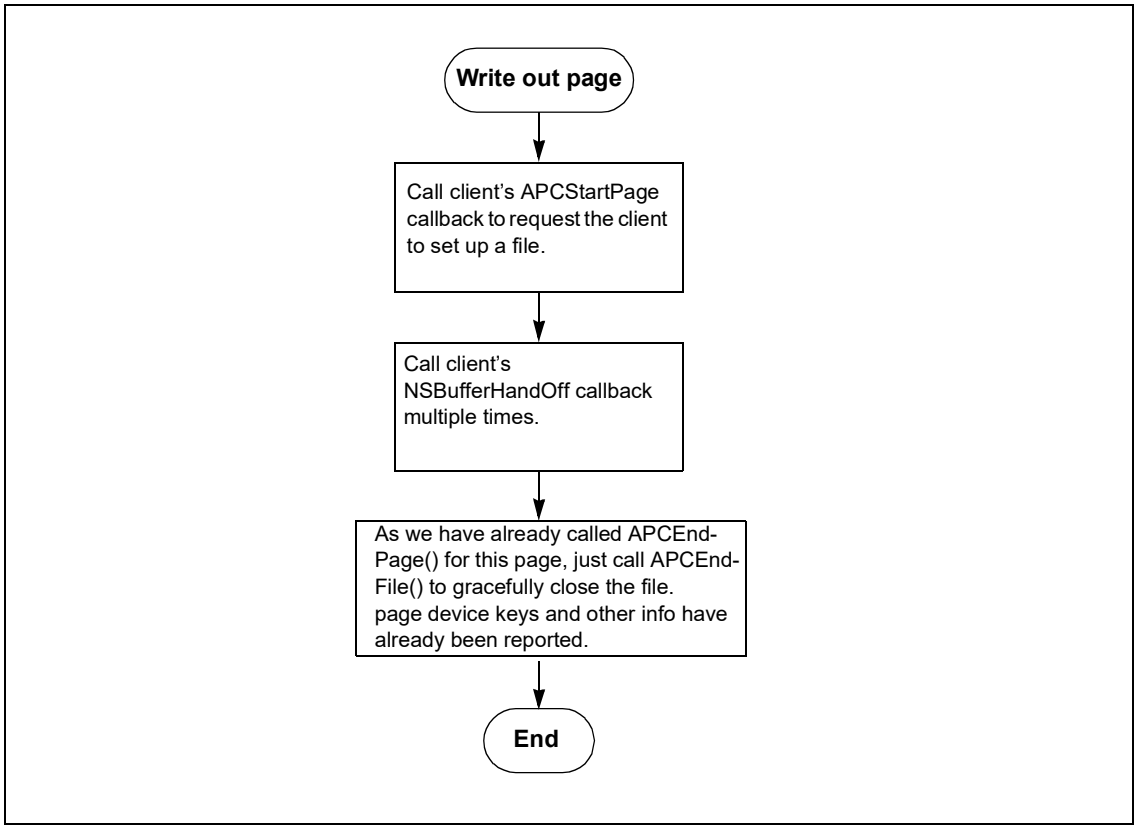


FIGURE 7.9 Flowchart showing client and the Adobe PDF Converter SDK interaction when dynamicMode == TRUE (4 of 4)



7.1.2 Private Client Data for Information about the Destination File

The data transfer callbacks pass a pointer of type **NSFileDataPtr** to a private structure, which allows the client to associate private information with each callback. The client can use this structure to store file data. Generally, the Adobe PDF Converter SDK does not request the client to have multiple files open simultaneously; however your implementation of the client should not depend on that.

7.1.3 Preparing to Transfer a PDF Page Stream/File

The Adobe PDF Converter SDK invokes the **NSStartPage()** callback to inform the client that it is ready to begin transferring a PDF stream for a page. The Adobe PDF Converter SDK invokes **NSStartPage()** only if **filePerPage** in the **NSJobParams** structure is TRUE and distillation is enabled.

In the function invoked by the **NSStartPage()** callback, the client should open a file in which to store the PDF page stream produced by the Adobe PDF Converter SDK. The callback returns an integer value indicating the success or failure of creating the requested file.

7.1.4 Completing the Transfer of Content into a PDF Page File

The Adobe PDF Converter SDK invokes the **NSEndPage()** callback to indicate that it has finished processing a page, regardless of whether distillation is enabled or **filePerPage** in the **NSJobParams** structure is TRUE. As part of this call, the Adobe PDF Converter SDK reports a linked list of key-value pairs representing the page device keys that have appeared in the page. See the **SPDKeyValue** structure in Chapter 18, "Structures and Enumerations". In addition the Adobe PDF Converter SDK provides plate color information and page label information in an **NSPageInfo** structure that appears in PostScript and DSC comments on the page.

If **filePerPage** in the **NSJobParams** structure is TRUE and distillation is enabled, the client should close the PDF page file it created for the page.

7.1.5 Preparing to Transfer the Contents of an External File

The Adobe PDF Converter SDK invokes the **NSCreateExternalFile()** callback to inform the client that it is ready to begin transferring embedded image or EPS data to an external file. In response, the client must create the file and return a **NSFileDataPtr** handle to that file.

The **sidelineType** argument to **NSCreateExternalFile()** is an enumerated value of type **NormalizerSidelineType**, which allows the client to assign file names that reflect the type of data (image or EPS) to be stored in the file.

Normally, the client places the external file in the same directory as the PDF files, and the file name is a relative path. The client must ensure that the file name provided conforms with the names described in Section 4.5 of *Portable Document Format, Version 1.3*.

The Adobe PDF Converter SDK invokes **NSBufferHandOff()** if distillation is enabled and if the Adobe PDF Converter SDK detects any of the following conditions:

- PostScript stream/file contains embedded EPS segments and **sidelineEPS** in the **NSJobParams** structure is TRUE
- PostScript stream/file contains embedded images and **sidelineImages** in the **NSJobParams** structure is TRUE

NSCreateExternalFile() returns an integer that indicates the client's success at creating the requested file.

7.1.6 Completing Transfer of the Contents of an External File

The Adobe PDF Converter SDK invokes the **NSCloseExternalFile()** callback to inform the client that it has finished transferring the PDF data into the external file. The client should respond by closing the previously opened external file.

NSCloseExternalFile() does not return a result.

7.1.7 Getting a Buffer of PostScript Stream

The Adobe PDF Converter SDK invokes the **NSBufferGetPS()** callback to request a buffer of a PostScript stream/file. The client is responsible for allocating the buffer into which the data is read. The number of bytes placed in the buffer on each call to this callback function is at the discretion of the client. To provide the best performance, you should carefully choose the number of bytes placed in the buffer. A value of four times the physical disk block size is a good starting point.

The client should not reuse or free a buffer provided in the **NSBufferGetPS()** callback until the Adobe PDF Converter SDK again calls the **NSBufferGetPS()** callback or the job completes.

NSBufferGetPS() returns an integer (0) to indicate the client's success at obtaining the next buffer of PostScript stream/file.

7.1.8 Handing Off a Buffer of PDF or Other Stream

The Adobe PDF Converter SDK invokes the **NSBufferHandOff()** callback multiple times to send the client either PDF streams for individual pages or data to be stored in external files. The Adobe PDF Converter SDK invokes **NSBufferHandOff()** if distillation is enabled and the following 1-bit fields are set in the **NSJobParams** structure in the circumstances described:

- **filePerPage** in the **NSJobParams** structure is TRUE and the Adobe PDF Converter SDK finishes interpreting a page. (the Adobe PDF Converter SDK first invokes the **NSStartPage()** callback.)
- **sidelineEPS** is TRUE and Adobe PDF Converter SDK encounters an embedded EPS segment. (Adobe PDF Converter SDK first invokes the **NSCreateExternalFile()** callback.)
- **sidelineImages** is TRUE and Adobe PDF Converter SDK detects an embedded image. (Adobe PDF Converter SDK first invokes the **NSCreateExternalFile()** callback.)

7.2 Callbacks That Relay Information to the Client

The Adobe PDF Converter SDK uses information-related callbacks to relay information to the client. These callbacks do not require a response from the client.

The following sections describe the situations in which the Adobe PDF Converter SDK calls these callbacks.

7.2.1 Passing Text Strings that Describe Fatal Errors

The Adobe PDF Converter SDK calls the **NSErrorMsg()** callback to send the client a character string that describes fatal errors. After calling **NSErrorMsg()**, the **NormalizerServerRunJob()** function returns the **NormalizerResult** enumerator, indicating the type of error.

NSErrorMsg() does not return any results.

7.2.2 Passing Text Strings that Describe PostScript Interpreter Errors

The **NSBackChanMsg()** callback implements the PostScript Interpreter's standard output device.

The Adobe PDF Converter SDK calls **NSBackChanMsg()** to send the client character strings that describe errors encountered by the PostScript Interpreter. Errors that cause a job to be terminated are reported in strings that begin with **%% [Error**. All other messages are warnings and do not signify job failure. After calling **NSBackChanMsg()** to report an error that causes a job to be terminated, the **NormalizerServerRunJob()** function returns the **normPostScriptError** enumeration value. **NSBackChanMsg()** does not return any results.

7.2.3 Reporting Progress

The Adobe PDF Converter SDK calls the **NSProgress()** callback every time the PostScript Interpreter processes the number of PostScript operations indicated in the **progressQuantum** field of the **NSClientConfig** structure. The client can use this callback to perform communication tasks or possibly to abort the current job.

NSProgress() does not return any results.

7.3 Callbacks for Modifying DSC and PostScript

The Adobe PDF Converter SDK provides two callbacks for reporting the appearance of a DSC comment and for affecting the DSC and the PostScript stream. Table 7.1 compares the tasks performed by each callback.

TABLE 7.1 Comparison of NSProcessComment() and NSExternalProcessCommentProc()

Capability	NSProcess Comment()	NSExternal ProcessCom mentProc()
Change the DSC being reported		
Add new DSC and/or PostScript		
Skip PostScript from the DSC being reported to the next DSC		
Immediately execute new PostScript		
Interpret client-provided DSC (Not supported by either callback)		
Interpret client-provided **externalcommand** operator.		
Called for each DSC comment in a sequence, where % begins a line. For example: %% comment 1 % comment 2 % comment 3		
Provision for setup and cleanup callbacks		

When the Adobe PDF Converter SDK encounters a DSC comment, it sequentially invokes the following non-NULL callbacks:

1. **NSExternalProcessCommentSetupProc()**
2. **NSExternalProcessCommentProc()**
3. **NSExternalProcessCommentCleanupProc()**
4. **NSProcessComment()**

7.3.1 Keying Off a DSC Comment to Skip PostScript or to Execute Client-Provided PostScript

The Adobe PDF Converter SDK invokes the **NSExternalProcessCommentProc()** callback to report individual DSC comments that appear in the PostScript file/stream. The arguments in **NSExternalProcessCommentProc()** allow you to direct Adobe PDF Converter SDK to do the following:

- Skip any PostScript code appearing between the DSC comment and the next DSC comment.
- Introduce PostScript code, which is immediately executed.

NSExternalProcessCommentProc() is typically used to replace one standard PostScript sequence with another, when the original sequence can be identified by a particular DSC comment.

NOTE: **NSExternalProcessCommentProc()** is provided on all platforms but has been tested only on Windows platforms.

When the Adobe PDF Converter SDK invokes your implementation of **NSExternalProcessCommentProc()**, it provides a pointer to the callback **NSPSExecuteStringProc()**. If you wish to provide a PostScript segment, invoke **NSPSExecuteStringProc()**, referencing the PostScript segment in the sole argument, **buf**. PostScript executed by **NSExternalProcessCommentProc()** is not parsed for DSC comments. You may release the buffer when **NSPSExecuteStringProc()** returns.

NOTE: Please be aware the the following limitations on the PostScript code your client software provides through **NSPSExecuteStringProc()**:

- Must not modify the PostScript graphic state.
- Must be one series of complete PS operations.

  Any operators cannot be terminated in the middle of the operator. For instance, if you want to process "0 0 moveto", you can do calling the function twice with "0 0" and "moveto". But you cannot do "0 0 mo" and "veto".
- Must not include certain PostScript operators if the Adobe PDF Converter SDK is in the middle of interpreting font resources. Such operators include save/restore, gsave/grestore. They raise a PostScript error. (A complete list of restricted operators is not available at this time.)
- **AutoPositionEPSFiles** job option may not work if the user performed any graphic operations, including the pdfmark operator, before Adobe PDF Converter SDK processes **%%EndProlog**. If **AutoPositionEPSFile** is true, the Adobe PDF Converter SDK core code checks the display list count when it encounters **%%EndProlog**, and then sets the appropriate page settings for locating the EPS file automatically.

The **NSExternalProcessCommentSetupProc()** and **NSExternalProcessCommentCleanupProc()** callbacks associated with **NSExternalProcessCommentProc()** are optional. If your client provides

NSExternalProcessCommentCleanupProc(), Adobe PDF Converter SDK invokes it regardless of the success or failure of **NSExternalProcessCommentProc()**.

7.3.2 Reporting and Allowing Substitutions for DSC Comments

The Adobe PDF Converter SDK invokes the **NSProcessComment()** callback to report individual DSC comments that appear in the PostScript file/stream. The arguments in **NSProcessComment()** allow you to direct Adobe PDF Converter SDK to replace the DSC comment with any combination of DSC comments and PostScript code. Typically, this callback is used to correct commonly mis-represented DSC comments; however, it can also be used to introduce entire segments of PostScript code.

The Adobe PDF Converter SDK invokes the **NSProcessComment()** callback to parse enclosed comments in a PostScript job provided the PostScript is a conforming document, as indicated by the appearance of **%!PS-Adobe-2.0**, **%!PS-Adobe-3.0**, or **%!PS-Adobe-3.1** in the PostScript stream/file.

NOTE: Non-conforming DSC comments may cause the DSC Interpreter (part of the PostScript Interpreter) to stop reporting comments. The DSC convention is specified in *PostScript Language Document Structuring Conventions (DSC) Specification* version 3.0, Technical Note #5001, 9/25/92. This technical note is available from http://partners.adobe.com/asn/.

The Adobe PDF Converter SDK calls the **NSProcessComment()** callback whenever it encounters comments that occur outside prologues. **NSProcessComment()** is called for each group of comments, even when those comments are part of embedded fonts, embedded EPS segments, external EPS files referenced from the PostScript stream/file, embedded image streams, or external image streams referenced from the PostScript stream/file. (Adobe PDF Converter SDK does not report comments for external fonts referenced from the PostScript stream/file.)

The Adobe PDF Converter SDK reports each comment in a sequence of comments, the first of which begins with "**%**" followed by a non-blank character and the subsequent of which begin with "**%**" followed by any character. In the following example, the Adobe PDF Converter SDK invokes **NSProcessComment()** three times to report comments 1, 2, and 3, respectively. However, it does not invoke the function for comment 4 or 5.

```
%% comment 1
%  comment 2
%  comment 3
expression % comment 4
expression
%  comment 5
```

The client can either read the comment for informational purposes or request Adobe PDF Converter SDK to replace the comment with other data. If the client chooses the latter, it provides a buffer of data, usually PostScript expressions, which Adobe PDF Converter SDK substitutes for the comment.

7.4 Callback for Responding to the externalcomm and PostScript Operator

The **NSExternalCommandProc()** callback allows interaction between the PostScript program and your client software. Adobe PDF Converter SDK invokes the **NSExternalCommand()** callback when a PostScript language program executes the key-word pair **externalcommand**. The combination of **externalcommand** in a PostScript program and an **NSExternalCommand()** implementation allows the PostScript program to communicate directly with Adobe PDF Converter SDK.

If the callback is set to NULL in the **NSClientConfig** structure, an internal callback is used, which does nothing.

The **externalcommand** operator takes two string operands: a command string and a response string. To execute **externalcommand**, the PostScript code should look something like this:

```
/commandstr 256 string def
/responsestr 256 string def
...
commandstr responsestr /CPSI /ProcSet findresource /externalcommand
get exec
```

The contents of the command string are stored in **command[0:commandLength-1]**. If the command requires a response, Adobe PDF Converter SDK stores its response in **response[0:*responseLength-1]**.

Because ***responseLength** will be the length of the PostScript substring that is pushed onto the PostScript operand stack, Adobe PDF Converter SDK must set ***responseLength** to be the exact length of the response string stored. Exceeding the initial length set by ***responseLength** is a fatal error.

In Example 7.1, the text "Test the external command" is passed to the **ExternalCommand** callback in the “command” parameter. The callback code can then put data into the “response” parameter, which ends up on the stack for the PostScript program to read.

The return value of the **ExternalCommand** callback is also returned on the stack (converted to a boolean, although the function prototype is an unsigned int) as in the PostScript command. The significance of the result your client supplies reflects the result of the query posed by the command string.

EXAMPLE 7.1 externalcommand definition passed to ExternalCommand callback

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%begin
/exc 1183615869 internaldict /externalcommand get def

/dotest
{
  (Test the external command) 256 string
  % /CPSI /ProcSet findresource /externalcommand get dup == flush

  exc  exec

  == == flush
} bind def


dotest
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%end
```

For additional information on this callback see the *CPSI Developer's Companion*.

7.5 Callbacks for handling fatal error conditions

If the Adobe PDF Converter SDK encounters a fatal error condition, it calls **NSExitProcessProc** unless you explicitly set the callback to be NULL. If the Adobe PDF Converter SDK encounters a fatal error it doesn't understand, it calls **NSCantHappenProc** unless you have explicitly set the callback to be NULL.

7.6 Callbacks for handling pageskip feature

When APC enables pageskip by calling **APCPageSkipEnable** API then APC calls **NSGetNextDeviceActivatePageNumber** to get next postscript job page number for which PDF page needs to be generated, At the end of job, APC calls **NSTotalNumberOfPages** to inform client about total number of pages in the job.

8 Using the NSClientFile API

This chapter describes the NSClientFile API, including what it is, the functions and the callbacks that are necessary for the client software to initiate actions.

The NSClientFile API is defined in the file `apcif.h`. For detailed descriptions of each NSClientFile callback, see Chapter 14, "NSClientFile API".

8.1 About the NSClientFile API

The NSClientFile API allows clients to override the Adobe PDF Converter SDK's file I/O methods for full-document PDF files.

By default the Adobe PDF Converter SDK performs its file operations using the standard C runtime library provided on each platform. This is sufficient for many purposes, but you may want to replace these methods for different reasons, such as:

- Improving performance
- Accessing unusual devices
- Accessing files with multibyte filenames

The NSClientFile API provides a method for the client to manage all I/O (including read, write, and seek) on a file used by the Adobe PDF Converter SDK. The client describes a client file using a structure that contains a file ID and a set of callbacks that the Adobe PDF Converter SDK invokes for perform file I/O operations. A *file ID* is a value used by the client to identify a particular file. The client references a client file from the `fullDocClientFile` field in the `NSJobParams` structure. The client passes the `NSJobParams` structure as an argument to the the Adobe PDF Converter SDK when calling `NormalizerServerRunJob()`.

8.2 File Size Limitations

Regardless of the method used for file I/O, the full-document PDF files produced by the Adobe PDF Converter SDK can be no larger than 10 GBytes.

8.3 Selecting File I/O Methods

The `NSJobParams` structure provides two fields each for the full-document PDF file:

- *Defined as a pathname*. If non-NULL, the Adobe PDF Converter SDK uses the standard C runtime library to access the particular file.

- *Defined as an NSClientFile API*. If the pathname is not provided, the Adobe PDF Converter SDK uses the client-provided callbacks provided in an `NSClientFile` structure to access the particular file.

It is important to note that, if the NSClientFile API is used, the client is responsible for creating the full-document PDF file and for deleting it in the event the job fails.

To be consistent with the default behavior of the Adobe PDF Converter SDK core, the client should delete the full-document PDF file if either of the following occurs:

- `NormalizerServerRunJob()` returns an error code.
- `NormalizerServerRunJob()` returns a success code but no pages have been produced. The client would recognize when no pages have been produced by the absence of calls to the `NSEndPage()` callback for the job.

8.4 Data That Describes a Client File

This section discusses the NSClientFile data that the client provides to the Adobe PDF Converter SDK. The client provides the `NormalizerServerRunJob()` function with a set of APIs in the `fullDocClientFile` field of the `NSJobParams` structure.

The type definition for a client file follows:

```
typedef struct _t_NSClientFile
{
  NSClientFileID fd;
  NSClientFileProcs procs;
} NSClientFileRec, *NSClientFile;
```

A file ID may be either a pointer or an integer and is therefore represented as a union, as follows:

```
typedef union { void *ptr; int index; } NSClientFileID;
```

Pointers to the OEM-provided callbacks are collected in the `NSClientFileProcs` structure as follows:

```
typedef struct _t_NSClientFileProcs
    {
      NSReadProc ReadProc; /* Required */
      NSWriteProc WriteProc; /* Required */
      NSSeekProc SeekProc; /* Required */
      NSCloseProc CloseProc; /* Required */
      NSTruncateProc TruncateProc; /* Required */
      NSBufsizeProc BufsizeProc; /* Optional, may be set to NULL */
    } NSClientFileProcsRec, *NSClientFileProcs;
```

The specifications of the first five callbacks have exact counterparts in the POSIX standard C runtime library. The `NSBufsizeProc()` callback is optional. You may choose to implement it for optimization purposes.

The client does not provide a callback for opening a file; rather, the client should open a file before directing the Adobe PDF Converter SDK to begin a job.

9 Font-Related Behavior

This chapter describes font-related behavior in the Adobe PDF Converter SDK, including the font policy and how font-related parameters can affect performance. It also provides some guidance on modifications you can make in your client software to change font behavior when the specified font cannot be found.

9.1 Review of Parameters That Affect Font-Related Behavior

Parameters that affect font-related behavior in the Adobe PDF Converter SDK may be defined in the following ways.

- *Client configuration parameters.* Table 9.1 reviews font-related parameters supplied in the client configuration structure `NSClientConfig`. These definitions are summarized from the detailed field descriptions. See `NSClientConfig` in Chapter 18, "Structures and Enumerations".
- *Job parameters.* Table 9.2 reviews such parameters supplied in the job data structure, `NSJobParams`. These definitions are summarized from the detailed field descriptions of the `NSJobParams` structure. See `NSJobParams` in Chapter 18, "Structures and Enumerations".
- *Distiller parameters.* Table 9.3 reviews such parameters supplied as Distiller parameters. These definitions are summarized from Technical Note #5151, *Acrobat Distiller Parameters*, found in the Acrobat SDK Documentation at http://partners.adobe.com/asn.

There may be some interaction between job parameters and Distiller parameters. In particular, the PostScript being converted may contain **setdistillerparams** instructions that override the **EmbedAllFonts** Distiller parameter, as described in Table 9.2. Further, the client may specify job parameters that cause the Adobe PDF Converter SDK to override the **CannotEmbedFontPolicy** and **EmbedAllFonts** Distiller parameters, as described in Table 9.3.

TABLE 9.1 Client Configuration Parameters Affecting Font-related Behavior

Field name	Description
`atmFile`	*(Optional)* References a location where the client stores the path name for the ATM database file. The Adobe PDF Converter SDK uses the information contained in this file to synthesize missing fonts.

TABLE 9.1 Client Configuration Parameters Affecting Font-related Behavior

Field name	Description
`hostfontSearchList`	A list of font directories to be searched for host fonts. It can include the ATM font directory, the system font directory, the printer CMAP directory, and the printer fonts directory. (The directories must be in the form described in Section 5.3.2, "Specifying Pathnames in Command Lines" on page 39.) See fuller description on page 172
`ignoreStdTTFonts`	If TRUE, Adobe PDF Converter SDK ignores the TrueType versions of certain standard PostScript fonts (Appendix A). This setting can also be changed by NormalizerNewHostFontList(). `ignoreStdTTFonts` corresponds to the Ignore TrueType versions of standard PostScript fonts button of the Acrobat Distiller-Font Locations dialog. See the fuller description on page 176.
`resourceSearchList`	*(Required) References a location containing the PostScript fonts directory. The directory name typically is of the form* `fonts\`. You must set this field for backward compatibility with PostScript programs. Use `hostfontSearchList` (below) for the `fonts\` directory and all other font directories. See the fuller description on page 168

TABLE 9.2 Job Parameters Affecting Font-related Behavior

Field name	Description
`fontAllowMM`	If `true`, Adobe PDF Converter SDK attempts to synthesize missing fonts, using metric information from the ATM font database. If FALSE, Adobe PDF Converter SDK attempts to replace any missing fonts with the font specified in the `fontDefaultName` field (described below) or, if not provided, with Courier. Multiple Master fonts cannot be embedded. As a result, if **EmbedAllFonts** is TRUE, the Adobe PDF Converter SDK does not synthesize missing fonts, regardless of the value of `fontAllowMM`. `fontAllowMM` has no Distiller parameter counterpart.

TABLE 9.2 Job Parameters Affecting Font-related Behavior

Field name	Description
`fontEmbedJobsFonts`	If `true`, the Adobe PDF Converter SDK embeds in the PDF file fonts that are embedded in the PostScript. If FALSE, the Adobe PDF Converter SDK embeds such fonts ONLY if the Distiller parameters dictate embedding. Regarding fonts that are referenced from but not embedded in the PostScript, the Adobe PDF Converter SDK embeds such fonts ONLY if the Distiller parameters dictate embedding.

TABLE 9.3 Distiller parameters affecting font-related behavior

Parameter name	Description
AlwaysEmbed	An array of font names that the Distiller should always embed. There is no job parameter counterpart to **AlwaysEmbed**.
CannotEmbedFontPolicy	Specifies how the Distiller should respond if a font cannot be found or embedded. Possible values are: • OK, ignore if a font cannot be found or embedded • Warning, warn and continue if a font cannot be found or embedded • Error, terminate the current job if a font cannot be found or embedded If the client submits a job with **EmbedAllFonts** defined as TRUE, the Adobe PDF Converter SDK defines **CannotEmbedFontPolicy** as Error. If the client submits a job with **EmbedAllFonts** FALSE, the Adobe PDF Converter SDK defines it as Warning.
EmbedAllFonts	If `true`, Adobe PDF Converter SDK attempts to embed all the document's fonts in the full-document PDF file and to embed a page's fonts in its PDF page stream. If the Adobe PDF Converter SDK is unable to embed a font, the job fails with an `invalidfont` (typically) and `NormalizerServerRunJob()` returns with the `normPostScriptError` enumeration value.
MaxSubsetPct	The maximum percentage of font glyphs used before the Distiller embeds the entire font, rather than a subset of the font. There is no job parameter counterpart to **MaxSubsetPct**.
NeverEmbed	An array of font names that the Distiller should not embed. There is no job parameter counterpart to **NeverEmbed**.

TABLE 9.3 Distiller parameters affecting font-related behavior

Parameter name	Description
PDFX1aCheck	If `true`, verifies that all fonts can be embedded in order to ensure compliance with the PDF/X-1a standard. **NOTE:** Although Distiller 7.0 and later versions continue to support the PDFX1aCheck Distiller parameter, PDF Converter SDK does not support this Distiller parameter. Instead of the PDFX1aCheck boolean, a name value in an array called `CheckCompliance`, is used.
PDFX3Check	If `true`, verifies that all fonts can be embedded in order to ensure compliance with the PDF/X-3 standard. **NOTE:** Although Distiller 7.0 and later versions continue to support the PDFX3Check Distiller parameter, PDF Converter SDK does not support this Distiller parameter. Instead of the PDFX3Check boolean, a name value in an array called `CheckCompliance`, is used.
PDFXCompliantPDFOnly	If `true`, Distiller will produce a PDF document only if the appropriate PDF/X compliance test(s) are passed. **NOTE:** This parameter is ignored if the values returned by both **PDFX1aCheck** and **PDFX3Check** are `false`.
SubsetFonts	If `true`, the Distiller embeds only those font glyphs that are used, rather than embedding the entire font. There is no job parameter counterpart to **SubsetFonts**.

9.2 Font Policy

Font policy is the set of rules that describe how the Adobe PDF Converter SDK uses font-related parameters to determine its response to **findfont** operators (or **findresource** operators that specify fonts) in the PostScript stream. Such rules dictate how the Adobe PDF Converter SDK handles the following issues:

- Where to look for fonts
- Whether to embed specific fonts
- How to respond to missing fonts
- How to respond when a font cannot be embedded

9.2.1 Where Adobe PDF Converter SDK Looks for Fonts

The Adobe PDF Converter SDK looks for fonts in the following locations (in order), stopping when it finds a match:

1. *PostScript virtual memory*. Fonts embedded earlier[1] in the PostScript stream are saved to PostScript virtual memory.
2. *PostScript (and TrueType) fonts in OEM resource folders*. Such folders are specified by the `resourceSearchList` field of the `NSClientConfig` struct. The Adobe PDF Converter SDK finds matches by comparing **FontName** in the font reference with **FontName** in the font.

   The flag `ignoreStdTTFonts` of the `NSClientConfig` struct specifies whether the Adobe PDF Converter SDK should consider the TrueType versions of certain standard PostScript fonts (Appendix A). (The flag can also be specified as an argument passed to `NormalizerNewHostFontList()`). If `ignoreStdTTFonts` is FALSE, the Adobe PDF Converter SDK considers the TrueType versions of the standard PostScript fonts, using whatever font it finds first in the OEM resource folders.
3. Same as Step 2., but compares **FontName** in the font reference against the font file's name.
4. System resources.

9.2.2 Rules for Embedding

The following sections describe the rules the Adobe PDF Converter SDK uses to determine whether to try to embed a font of a particular type.

NOTE: Fonts embedded in the PostScript file are embedded in the PDF file only if embedding rules dictate that those font should be embedded. In other words, the Adobe PDF Converter SDK does not embed a font in the PDF file simply because that font is embedded in the PostScript file.

In general, if the Adobe PDF Converter SDK cannot embed a font that should be embedded, it responds as specified in the Distiller parameter **CannotEmbedFontPolicy**; however, some fonts must be embedded to produce a readable PDF file, regardless of Distiller parameters. Adobe PDF Converter SDK terminates a job with error when it cannot embed a font that must be embedded. The following sections identifies those fonts that must be embedded.

1. Adobe PDF Converter SDK serially evaluates the PostScript stream. It cannot find fonts referenced at the beginning of the stream but embedded at the end of the stream. PostScript streams with such problems are considered ill-formed.

Type 1 and Type 2 Fonts

Adobe PDF Converter SDK embeds a Type1 font if the font is NOT in the **NeverEmbed** list, the font is found (either embedded in the PostScript or available on the host system), and at least ONE of the following conditions is true:

- `fontEmbedJobsFonts` is true and the font is embedded.
- **EmbedAllFonts** flag is *true*.
- The font is in the **AlwaysEmbed** list.
- The PostScript file uses the font for characters that are not included in the Standard Latin Character Set.
- The font contains many glyph definitions (**CharStrings** dict length > 229 and disableAutoT1Embed parameter in NSJobParams is false). Such fonts are usually used for non-standard glyphs.
- The font contains few glyph definitions (**CharStrings** dict length < 115). Such fonts are used for logos or special glyph sets, such as an all-capital letter font.

Type 3 Fonts

Type 3 fonts are always embedded. Type 3 fonts are used to represent non-standard characters for which Acrobat is unable to create substitute fonts.

Determining Whether a Font is Embeddable

Adobe PDF Converter SDK may be unable to embed a font with any of the following characteristics:

- It is a faux font generated by the PostScript job. (*Faux fonts* are fonts reproduced using font metrics obtained from the ATM database.)
- Its permissions do not allow embedding, which would happen if license restrictions were unsatisfied. License restrictions apply to certain TrueType™, OpenType, and CJK fonts. Such permissions are expressed in a font's **fsType** field.

9.2.3 Response to Missing Fonts

If the font is not embedded in the PostScript or if the Adobe PDF Converter SDK cannot find the font on the host system, the Adobe PDF Converter SDK determines whether to emulate the font using parameters in the ATM database or to instead use a substitute font. More specifically, if all the following conditions are satisfied, the Adobe PDF Converter SDK tries to duplicate the font using parameters from the ATM database.

- `fontAllowMM` true
- **EmbedAllFonts** false
- font not in **AlwaysEmbed** list

- font is in the ATM database

Otherwise, the Adobe PDF Converter SDK uses the substitute font specified in **SubstituteFont** or, if not provided, Courier.

9.2.4 Response to Unembeddable Fonts

If the Adobe PDF Converter SDK is unable to embed a font that appears in the **AlwaysEmbed** list, it responds as directed in the Distiller parameter **CannotEmbedFontPolicy**. If that parameter allows the Adobe PDF Converter SDK to complete the job (**CannotEmbedFontPolicy** is **OK** or **Warning**), the Adobe PDF Converter SDK produces a PDF file that provides information about the un-embeddable font, including, if possible, information that allows Acrobat to create a font substitution.

9.3 PostScript SubstituteFont Key Influences Font Policy

The initialization PostScript or job options you supply to the Adobe PDF Converter SDK can establish whether the Adobe PDF Converter SDK should use a default font and what that font should be. It does so by setting the value of the **SubstituteFont** PostScript key-word pair. Adobe PDF Converter SDK uses that key to determine what to do if it cannot find or replace (faux) the specified font.

Example 9.1 includes the section of Democlient code that establishes the default font to be used when the Adobe PDF Converter SDK cannot find the specified font. It sets the value of the PostScript **SubstituteFont** key-word pair to one of the following values:

- *Null value*. Specifies that default fonts should not be used. If any fonts are missing, the job fails with an **invalidfont** (typically) and `NormalizerServerRunJob()` returns with the `normPostScriptError` enumeration value.
- *A font name (usually Courier)*. Specifies the name of the font to use in place of fonts that cannot be found.

The PostScript code may be included in any of the following:

- Startup PostScript
- job prologue (if supported by the Adobe PDF Converter SDK client implementation)
- exitserver job
- Job options

The section of code included in Example 9.1 creates job options that reflects settings established through the Democlient UI.

EXAMPLE 9.1 Democlient code that establishes a new default font

```
else if(NULL == demoClientData.jobOptionFileName)
{
   /* Set pagedevice parameters and default font. */
   /* Note the setpagedevice must come after the setdistillerparams*/

   sprintf(jobOptBuf,
      "<< /PreserveOPIComments %s \
       /EmbedAllFonts %s \
       /CannotEmbedFontPolicy %s \
      >> setdistillerparams \n\
      << /PageSize [ %d %d ] /HWResolution [ %d %d ] >>
      setpagedevice \n",
      demoClientData.preserveOPI ? "true" : "false",
      demoClientData.fontEmbedAll ? "true" : "false",
      demoClientData.fontEmbedAll ? "/Error" : "/Warning",
      demoClientData.pageWidth, demoClientData.pageHeight,
      demoClientData.pageXRes, demoClientData.pageYRes);

   /* Set font policies */
   if (!demoClientData.fontAllowDefault)
   {
      /* No default font allowed. */
      sprintf(jobOptBuf,
      "%s $error /SubstituteFont {} put\n",
      jobOptBuf);
   }
   else
   {
      /* Default font allowed.
      sprintf(jobOptBuf,
      "%s $error /SubstituteFont { pop /%s } put\n",
      jobOptBuf,
      demoClientData.fontDefaultName == NULL
         ? "Courier" : demoClientData.fontDefaultName);
   }

   pJobParams->jobOptions = jobOptBuf;
}
```

10 Frequently Asked Questions

This chapter addresses frequently asked questions that are not directly answered in the preceding chapters.

10.1 Locations of ICC Profile Folders (Windows)

Question. What are the locations of the ICC profile folders on Windows.

Answer. The following describes the typical locations of these folders, although the exact location may vary because the `Program Files` folder can be moved or be on a different drive if the OS is not on the `C:` drive.

- ICCPROFILES_USE_ADOBE_STANDARD_ONLY

  Setting `iccProfilesStandardFolders` to the above value references Adobe ACE folder.

- ICCPROFILES_ADOBE_COLOR_RECOMMENDED

  Setting `iccProfilesStandardFolders` to the above value references the Adobe recommended color profiles folder, which is typically located in the folder:

  ```
  C:\Program Files\Common Files\
    Adobe\Color\Profiles\Recommended
  ```

- ICCPROFILES_ADOBE_COLOR

  Setting `iccProfilesStandardFolders` to the above value references the Adobe color profiles folder, which is typically located in the folder:

  ```
  C:\Program Files\Common Files\Adobe\Color\Profiles
  ```

- ICCPROFILES_SYSTEM_COLOR

  Setting `iccProfilesStandardFolders` to the above value references the system color profiles folder, which is typically located in the folder:

  ```
  C:\WINDOWS\SYSTEM32\COLOR (on Windows XP)
  ```

  ```
  C:\WINNT\STYSTEM32\COLOR (on all Windows 2000 and Windows
  2003 server)
  ```

10.2 Unexpected Failure

Question. What causes democonverter to fail with the following error messages?

```
EPS sidelining is off
File-per-page is off
Image sidelining is off
Resolution is 2400
Embed All Fonts is ON
%%[ Error: ioerror; OffendingCommand: setdistillerparams ]%%
%%[ Flushing: rest of job (to end-of-file) will be ignored ]%%
%%[ Warning: Empty job. No PDF file produced. ] %%
Normalizer init failed (2)
```

Answer. The above error messages may be caused by the Adobe PDF Converter SDK exceeding the space available in its scratch directory. The Adobe PDF Converter SDK stores temporary files such as images to a scratch directory specified in the `scratchFileDirectory` field of the `NSClientConfig` structure.

If you experience the above error messages, make sure there is space available in the scratch file directory. If running democonverter, use the `-S` command to increase the size of the scratch directory.

10.3 Full-document PDF File

Question. I'm using the PDF page streams produced by the Adobe PDF Converter SDK, so I don't need the full-document PDF files. Is there any way I can direct the Adobe PDF Converter SDK to stop producing full-document PDF files?

Answer. Yes. This can be done in PDF Converter SDK. For this, a new one-bit field is added to the NSJobParams struct.

fulldocfileCreation:1

Full document file creation cannot be disabled if the file per page option is off. Use '+/-w' with democonverter to enable/disable creation of a full document PDF.

10.4 Warning Message

Question. I see the following message every time the Adobe PDF Converter SDK starts:

```
%%[ Warning: Empty job. No PDF file produced. ] %%
```

Why does the Adobe PDF Converter SDK produce that message and is there any way to suppress it?

Answer. The Adobe PDF Converter SDK produces the message you describe when it executes the file `startup.ps`, which of course contains no showpages; the message has nothing to do with any actual jobs.

There is no way to suppress the message in the current version of the Adobe PDF Converter SDK. However, your product code could prevent it from being reported.

10.5 Offending command warning

Question: I get the following (or similar) error on startup:

```
%%[ Error: undefined; OffendingCommand: setdistillerparams; ErrorInfo:
CalCMYKProfile U.S. Web Coated (SWOP) ]%%
%%[ Flushing: rest of job (to end-of-file) will be ignored ]%%
Error accessing color profile: U.S. Web Coated (SWOP)
%%[ Warning: Empty job. No PDF file produced. ] %%
Normalizer Server init failed (2)
```

Answer: The Adobe PDF Converter SDK is searching the ICC color profile directories and cannot find the file `USWebCoatedSWOP.icm`. This is an indication that the files located in the `ICCProfiles` directory of the democonverter deliverables are not correctly referenced by the Adobe PDF Converter SDK.

To correct this problem, update the values specified for **`iccProfileDirList`** and/or **`iccProfilesStandardFolders`** to include the ICC profiles provided in the `ICCProfiles` directory included with the democonverter deliverables. For more information on **`iccProfileDirList`** and **`iccProfileStandardFolders`** see page 174.

10.6 Error message #8

If the Adobe PDF Converter SDK fails to initialize and returns error #8:

`normIncorrectInterfaceVersion`

then the value of the **`interfaceVersionNum`** being passed into the library is incompatible with that in the library.

It is likely that you have not updated your header file and product code to use the current version of the Adobe PDF Converter SDK.

10.7 Error Message Processing PostScript that Contains a Screen Preview

Question. When processing a an EPS file that contains a screen (bitmap) preview, I see the message:

```
%%[ Error: undefined; OffendingCommand: ÅÐÓÆpÔ  ]%%
```

Why does this happen?

Answer. Errors in the log file were found as a result of running through NormTester, using test file `Test_Set1\cs09_01.indd1_5.eps`. The errors were not found when running the same file through Distiller 5.0.

Here is the Log file after running the above file through the Adobe PDF Converter SDK:

```
%%[ Error: undefined; OffendingCommand: ÅÐÓÆpÔ  ]%%
%%[ Flushing: rest of job (to end-of-file) will be ignored ]%%
```

The first part of the EPS is a screen preview, followed by the intended image. Adobe Photoshop® 7.0 produces such EPS segments. The first four bytes of the EPS file contain 0xC6D3D0C5, which indicates the presence of the screen preview. The intended image begins with the statement **%!PS**.

Acrobat Distiller has explicit code that filters out screen previews from the files it processes. That is, Distiller bypasses the screen-preview portion of the EPS file, from the beginning of the file to the statement just before %!PS.

The Acrobat Distiller screen-preview filtering code is not in the Distiller core. As a result, the Adobe PDF Converter SDKthe Adobe PDF Converter SDK does not provide the EPS screen-preview filter present in Acrobat Distiller.

NOTE: The Adobe PDF Converter SDK can successfully process PostScript streams/files that contain embedded EPS segments that begin with screen previews.

You can implement your own filter to strip out screen previews in EPS files. The Adobe PDF Converter SDK reads such files without the help of the client software.

10.8 Requirement for "iccprofiles" Folder

Question. The Adobe PDF Converter SDK deliverables contain an "`ICCProfiles`" folder. Are these folders required if the command-line to be passed to democonverter is given as follows:

```
democonverterOS9 -efij +n -O <some dir>: -H  <font folders>
     -B <jobOption> somejob.ps
```

The command-line options used above and in the answer are summarized below:

-e Turn off EPS sidelining off|

+n Use original PostScript file name in resulting PDF file name, as in test.ps => test.pdf.

-O Set output directory to <dir>.

-H Add host fonts directory <dir>.

-B Send <file> as JobOptions before each job.

-r0 Use only profiles in folders specified by the **-P** parameter.

-P Add ICC profile directory <dir>.

Adobe PDF Converter SDK needs the ICC profiles. The Adobe PDF Converter SDK cannot run without having access to ICC profiles. You can provide the Adobe PDF Converter SDK with the ICC profile search locations using the **-P** option or using one of the Color folders accessed by ACE.

The command line you provide above will work only if ICC profiles have been installed by other Adobe applications (unless you've copied the contents of the ICC profiles folder to one of the Color folders). Otherwise, use **-r0 -P ICCProfiles** (for Windows) on the democonverter command line to use only the ICC profiles folder included in the Adobe PDF Converter SDK deliverables.

11 Restricting PostScript File System Access

This chapter provides information on Adobe PDF Converter SDK's security measures for restricting the permissions of PostScript files when accessing the file system. Using these measures, you can customize the level of accessibility to directories on any machine running the Adobe PDF Converter SDK.

11.1 Specifying Directories for Restricted Access

The `NSClientConfig` struct includes two fields, `fileSecurityDirList` and `fileSecurityWorkingDir`.

- `fileSecurityWorkingDir` — The full pathname of the PostScript interpreter's current working directory during PostScript interpretation. The path may end with the file system separator character but it is not required.
- `fileSecurityDirList` — The list of directories (full pathnames) for which the PS interpreter will allow access for the PostScript "file" operators. The list is a linked list of `NORMSearchList` elements terminated with a NULL next pointer. Each "path" field in the `fileSecurityDirList` must end with the filesystem separator character.

These fields specify a set of directories in the underlying file system (the %os% storage device) for which the PostScript interpreter will allow access for the PostScript "file" operators. When the fields contain valid directory locations, access is restricted to the set of directories specified, and access to any other directories is denied. If both fields are NULL, no access restrictions will be applied beyond what is enforced by the underlying file system.

NOTE: See "File and Path Locations" on page 89 for information on using relative pathnames in `fileSecurityWorkingDir or fileSecurityDirList`.

11.2 Access Strings

The "extension" field of each `NORMSearchList` element must be a pointer to a string specifying the access allowed for the directory in the "path" field. The access string must be one of the following:

rw	Read-write for the directory.
ro	Read-only for the directory.

rw+ Read-write for the directory and all its subdirectories.

ro+ Read-only for the directory and all its subdirectories.

The trailing "+" (plus) sign applies the specified access restriction to the directory and all its subdirectories, even if the subdirectories are created after this procedure returns.

Without the trailing "+" (plus) sign, specified access is restricted to the directory itself. Access to any subdirectory is denied. In this case you must list subdirectories individually in the `fileSecurityDirList` field to include them in the access list. This implies that subdirectories can have individual access restrictions that differ from the parent directory. For example, consider the two files `/dir1/file1` and `/dir1/dir2/file2`. If only `/dir1` is listed in `fileSecurityDirList`, access to `file1` may be granted, but access to `file2` will be denied. To allow access to both files `fileSecurityDirList` must include both `/dir1` and `/dir1/dir2`.

Each directory in the list can be a subdirectory of any other directory in the list. This allows for some quite sophisticated access control. Consider the following set of security settings composed of directories and their associated access restrictions.

```
"ro"  /dir1
"rw+" /dir1/dir2
"ro+" /dir1/dir2/dir3
"rw"  /dir1/dir2/dir3/dir4
"rw+" /dir1/dir2/dir3/dir4/dir5
```

This set of security settings produces varying results depending on the location of the file being accessed.

Accessing the file:	With the access level:	Results in:
/dir1/*file*	rw	Access denied
/dir1/*file*	ro	Access granted
/dir1/dir2/*file*	rw	Access granted
/dir1/dir2/*file*	ro	Access granted
/dir1/dir2/dir3/*file*	rw	Access denied
/dir1/dir2/dir3/*file*	ro	Access granted
/dir1/dir2/dir3/dir4/*file*	rw	Access granted
/dir1/dir2/dir3/dir4/*file*	ro	Access granted
/dir1/dir2/dir3/dir4/dir5/*file*	rw	Access granted
/dir1/dir2/dir3/dir4/dir5/*file*	ro	Access granted

11.2.1 Adobe PDF Converter SDK Folder Security Settings

If you are using file access security measures, then you must set the correct permissions for certain the Adobe PDF Converter SDK directories. The Adobe PDF Converter SDK requires full read-write access to the following directories and their subdirectories (rw+ permission):

- The current working directory.
- The Adobe PDF Converter SDK scratch directory.
- The output directory.

These settings must be implemented in order to use the file security capability.

11.3 Processing the Security Settings

When processing security settings, the Adobe PDF Converter SDK faces two distinct cases.

- A PostScript language program attempts to access a file outside of the directories specified in `fileSecurityDirList`. In this case, the PostScript interpreter raises an **UndefinedFileName** exception.
- A PostScript language program attempts to access a file in one of the directories specified in `fileSecurityDirList`. In this case, the PostScript interpreter checks the requested file access against the allowed access for the directory. If access is denied an **InvalidFileAccess** exception is raised. If access is allowed, to access to the requested file is granted.

File and Path Locations

At the PostScript language level filenames can be specified either as fully qualified pathnames or as relative pathnames. If a pathname is relative, the interpreter's current working directory, specified in the `fileSecurityWorkingDir` field, is prepended to the pathname before checking against the `fileSecurityDirList` directories. As such, you can disallow relative pathnames by simply not listing the interpreter's current working directory in `fileSecurityDirList`.

IMPORTANT: *If you wish to use relative pathnames to access files from the PostScript steam you MUST include the working directory of the Adobe PDF Converter SDK in* `fileSecurityDirList`*.*

Memory Allocation

You must allocate the storage for `fileSecurityWorkingDir` and `fileSecurityDirList`. The storage is NOT copied, so the client must not free the storage until the next call to this procedure.

Changing Access Restrictions During Processing

You can call `PSRIPSetPostScriptFileAccessRestrictions()` at any time to change the file access restrictions.

To revert to the default (no access restrictions) call this procedure with both `fileSecurityWorkingDir` and `fileSecurityDirList` set to NULL.

To disallow all access to all directories, call this procedure with `fileSecurityDirList` set to NULL and `fileSecurityWorkingDir` set to any non-NULL value.

11.4 Example Security Settings

An example of updating the Adobe PDF Converter SDK with enhanced security settings is provided below. Note that in a true implementation, all paths should be obtained dynamically and any allocated memory must be made available after it is used.

In order to test the file system security, replace the two file security code segments within `democonverter.c`:

```
demoConfig.fileSecurityWorkingDir = NULL;
demoConfig.fileSecurityDirList = NULL;
```

with the following code ...

```
demoConfig.fileSecurityWorkingDir =
      "c:\\norm\\democonverter\\ix86win32\\Debug";

demoConfig.fileSecurityDirList =
      (NORMSearchList)malloc(sizeof (NORMSearchListRec));

demoConfig.fileSecurityDirList->path =
      "c:\\norm\\democonverter\\ix86win32\\Debug\\";
      /* note trailing slashes */

demoConfig.fileSecurityDirList->extension = (char *) "ro+";

demoConfig.fileSecurityDirList->matchingOnly = FALSE;
      /* ignored in this case */

demoConfig.fileSecurityDirList->next =
      (NORMSearchList)malloc(sizeof (NORMSearchListRec));

demoConfig.fileSecurityDirList->next->path ="c:\\temp\\";
      /* note trailing slashes */
```

```
demoConfig.fileSecurityDirList->next->extension =
      (char *) "rw+";
demoConfig.fileSecurityDirList->next->matchingOnly = FALSE;
      /* ignored in this case */

demoConfig.fileSecurityDirList->next->next =
      (NORMSearchList)malloc(sizeof (NORMSearchListRec));

demoConfig.fileSecurityDirList->next->next->path =
      "c:\\norm\\democonverter\\ix86win32\\Debug\\HostFontCache
      \\";
      /* note trailing slashes */

demoConfig.fileSecurityDirList->next->next->extension =
      (char *) "rw+";

demoConfig.fileSecurityDirList->next->next->matchingOnly =
      FALSE;
      /* ignored in this case */

demoConfig.fileSecurityDirList->next->next->next = NULL;
```

11.4.1 Example PostScript file to verify security settings

Once you update your security settings you should run a test PostScript file to verify that the settings are working correctly. The PostScript code below illustrates a simple test of the security settings specified above.

```
%!PS

(%os%/k/fred.txt)
(w) { file } stopped
{
   (Failed to Create file: ) print
   pop print (\n) print flush
}
{
    dup (Test Text\n) writestring
    closefile
} ifelse

%%EOF
```

Part II

Adobe PDF Converter SDK Reference

Part II describes the functions, callbacks, and structures of the Adobe PDF Converter SDK. The chapters in Part II are:

Each chapter entry is presented alphabetically by function, callback, or structure name.

13 Functions and Callbacks

APCAddImage()

```
NormalizerResult APCAddImage(char * imagePath, int pageWidth,
int pageHeight);
```

Description

Adds the specified image to the PDF document generated by APCCreateDoc(). This API takes the full path name of the image file, the desired PDF page width and height, on which the image is to be placed. If either the page width or the height is invalid, the PDF page dimensions are set to image dimensions.

Parameters

`imagePath`	Specifies the full path name of the input Image file.
`pageWidth`	Specifies the desired PDF page width.
`pageHeight`	Specifies the desired PDF page height.

Return Value

One of the NormalizerResult enumerators.

Header File

`apcif.h`

APCConfigureImageJob()

```
void APCConfigureImageJob(ImageJobConfigP config);
```

Description

This function should be called before first call to APCAddImage() function. Call this function to configure the image to pdf conversion.

Parameters

config pointer to the struct (of ImageJobConfig type) having settings required for the configuration.

Return Value

None

Header File

`apcif.h`

APCConvertPPMLToPDF()

```
NormalizerResult APCConvertPPMLtoPDF(char *inputFileName, char
*outputFileName, NSServerDataPtr serverData, NSJobParams *
jobParams)
```

Description

Converts a PPML file to PDF. Input PPML file should be as per 2.1 specifications

Parameters

`inputFileName`	path to input PPML file
`outputFileName`	path to output PDF file
`serverData`	An opaque pointer to the Adobe PDF Converter SDK's private data
`jobParams`	A pointer to the NSJobParams structure, which specifies how the PostScript (coming in PPML file) should be converted. The client deallocates this structure on job completion (when APCConvertPPMLtoPDF() returns).

Return Value

Returns one of the NormalizerResult enumerators

Header File

`apcif.h`

APCCreateDoc()

```
NormalizerResult APCCreateDoc(char * fullPDFFilePathName);
```

Description

Creates an empty PDF document. The client can add multiple image files of varying formats to this document. Only one document can be created at a time. To create a new document, save the existing one using the API, APCStartFile().

Parameters

`fullPDFFilePathName`	Specifies the full path name of the PDF file to which the PDF stream of images needs to be saved.

Return Value

One of the NormalizerResult enumerators.

Header File

`apcif.h`

APCDisableOptimizationPStoPDF()

```
NormalizerResult APCDisableOptimizationPStoPDF (unsigned int
optimization);
```

Description

APCDisableOptimizationPStoPDF() function disables some specific optimizations used during PS to PDF conversion depending on value of integer **optimization** passed as an argument. By default all optimizations are enabled. The client needs to call this API after APCInit() and before calling `NormalizerServerRunJob()`.

Parameters

`Optimization`	An integer value to disable specific optimizations. **0** - No change, all optimizations are enabled **1** - Disable merging of Type42 fonts **2** - Disable optimization which checks whether object is partially clipped or not **3** - Disable optimizations 1 and 2 above **4** - Disable clipping of objects outside of PageSize **5** - Disable optimizations 1 and 4 above **6** - Disable optimizations 2 and 4 above **7** - Disable optimizations 1, 2 and 4 above

Return Value

One of the NormalizerResult enumerators. The specific optimization has been disabled if the return value is *normOK*.

Header File

`apcif.h`

APCDisableProcessingInPageSkip()

```
void APCDisableProcessingInPageSkip();
```

Description

This is new API which has been added in version 2.1. Call the **APCDisableProcessingInPageSkip** function to disable the minimal processing of some operators in the PageSkip mode. Note that now some operators like string width might return incorrect values. This function should be called after APCInit() and before calling `NormalizerServerRunJob()`.

Parameters

None

Return Value

None

Header File

`apcif.h`

APCEnableDynamicGeneration()

```
NormalizerResult APCEnableDynamicGeneration(normBool
dynamicMode);
```

Description

The **APCEnableDynamicGeneration()** function enables or disables dynamic generation of PDF files per job, depending on the value of **dynamicMode**. **dynamicMode** is set to false, by default. The client needs to call this API after APCInit() and before calling **NormalizerServerRunJob()**.

Parameters

dynamicMode	A boolean value to set dynamic mode to ON/OFF.

Return Value

One of the NormalizerResult enumerators. The client can submit additional requests to Adobe PDF Converter SDK only if the return value is *normOK*.

Header File

apcif.h

APCEnableProcessingInPageSkip()

```
void APCEnableProcessingInPageSkip ();
```

Description

This is new API which has been added in version 2.1. **`APCEnableProcessingInPageSkip()`** enables the processing of a set of ps operators (show, stringwidth etc.) during PageSkip Mode for page which are to be skipped. By default, it is disabled. This function should be called after **`APCInit()`** and before calling **`NormalizeServerRunJob()`**. This API is useful only if the PageSkip feature is enabled. Normally, Pageskip mode skip the processing of ps operators (show, stringwidth etc.) in pages which are not converted and just interpreted. But sometimes the postscript is dependent on this variable for further processing which may result in some problem.

For example: let's assume in postscript file, there is a loop in skipped page which is dependent on value of stringwidth for deciding to break. If this stringwidth is not processed this loop may go in infinite loop.

If **`APCEnableProcessingInPageSkip()`** is called, then most of the processing is skipped and only the relevant processing is executed which returns the correct value.

Note: Enabling PageSkip disables processing of PostScript operators by default. So a **`APCEnableProcessingInPageSkip()`** should follow **`APCEnablePageSkip()`** if the processing is desired.

Parameters

None

Return Value

None

Header File

`apcif.h`

APCEndFile()

```
void (*APCEndFile)(NSClientDataPtr clientData);
```

Description

The `APCEndFile()` callback indicates that PDF Converter SDK has finished processing a set of pages, regardless of whether distillation for the current page is enabled or not and is only called if the Dynamic Mode is ON. The client needs to close the PDF file it created for the pages.

Parameters

`clientData`	A pointer to the client's private data.

Return Value

N/A

Header File

`apcif.h`

APCEndPage()

```
normBool(*APCEndPage)(NSClientDataPtr clientData,
SPDKeyValue *pageDevFeatureList, NSPageInfo *pageInfo);
```

Description

The `APCEndPage()` callback indicates that PDF Converter SDK has finished processing a page, regardless of whether distillation is enabled. As part of this call, PDF Converter SDK reports the page device keys that appear on the page as well as the plate color information and page label information, from the PostScript and DSC comments on the page. In short, this call back is a replacement for NSEndPage() call back and needs to be implemented in case the client wants to use the dynamic feature of PDF Converter SDK. This call back is only called by the Converter when dynamic Mode is enabled for the job. If the dynamic mode is off, NSEndPage() is called. If dynamicMode is set to TRUE and distillation is enabled, the client needs to notify the PDF Converter SDK about whether or not it wants the PDF file up to the current page.

Parameters

`clientData`	A pointer to the client's private data.
`pageDevFeatureList`	A pointer to the `SPDKeyValue` structure used to represent key-value pairs internally.
`pageInfo`	A pointer to the `NSPageInfo` structure that contains information about a page.

Return Value

The boolean value, true, notifies the Converter SDK to create a PDF file up to the current page. The value, false, indicates that the client does not want the PDF file created yet.

Header File

`apcif.h`

APCInit()

```
NormalizerResult APCInit(APCClientConfig *config,
NSServerDataPtr *pServerData);
```

Description

The **APCInit()** function initializes the interface between the client and the Adobe PDF Converter SDK. This API does the job of NormalizerServerInit() and also initializes the dynamic feature related call backs. The client needs to initialize with **APCInit()** instead of NormalizerServerInit() to use the dynamic feature.

Parameters

config	A pointer to the APCClientConfig Structure, which contains callbacks and other parameters used during all subsequent calls. The client deallocates this structure after shutting down Adobe PDF Converter SDK.
pServerData	The location at which Adobe PDF Converter stores a pointer to its opaque private data.

Return Value

One of the NormalizerResult enumerators. The client can submit additional requests to Adobe PDF Converter SDK only if the return value is *normOK*.

Header File

apcif.h

APCPageSkipDisable()

```
void APCPageSkipDisable();
```

Description

This is new API which has been added in version 2.1.The product should call **APCPageSkipDisable()** to disable the PageSkip feature. This function should be called after APCInit() and before calling `NormalizerServerRunJob()`.

Parameters

None

Return Value

None

Header File

`apcif.h`

APCPageSkipEnable()

```
void APCPageSkipEnable();
```

Description

This is new API which has been added in version 2.1. The product should call **APCPageSkipEnable()** to enable the PageSkip feature. This function should be called after APCInit() and before calling **NormalizeServerRunJob()**. In PageSkip mode, only those pages that are requested to print by callback **NSGetNextDeviceActivatePageNumber ()** are converted. All other pages are just interpreted. The **NSTotalNumberofPages ()** callback is called to inform the product the total number of pages in a job, after APC finishes processing of the job.

Parameters

None

Return Value

None

Header File

`apcif.h`

APCSaveDoc()

```
NormalizerResult APCSaveDoc(void);
```

Description

This API closes the document stream and saves the PDF file that was generated by APCCreateDoc().

Parameters

N/A

Return Value

One of the NormalizerResult enumerators.

Header File

apcif.h

APCSetPDFX4Setting()

```
void APCSetPDFX4Setting (PDFX4Setting setting);
```

Description

This is new API which has been added in APC SDK 3.2. The product should call this API to change the behavior of APC as described by parameters below. This API should be called after APCInit() and before calling `NormalizerServerRunJob()`.

Parameters

`PDFX4Setting`	PDF/X-4 setting structure.

Return Value

None

Header File

`apcif.h`

APCStartFile()

```
void (*APCStartFile)(NSClientDataPtr clientData,
NSFileDataPtr *pFileData);
```

Description

The `APCStartFile()` callback informs the client that PDF Converter SDK is ready to begin transferring a PDF stream for a set of pages. Adobe PDF Converter SDK calls `APCStartFile()` only if `dynamicMode` is ON. In the function invoked by the `APCStartFile()` callback, the client should open a file to store the PDF pages stream, produced by Adobe PDF Converter SDK.

Parameters

`clientData`	A pointer to the client's private data.
`pFileData`	A pointer to the location at which the client stores a pointer to a new instance of file data.

Return Value

An integer that indicates the client's success at creating the requested file. The value, 0, indicates success. A non-zero value indicates an error such as file access problems. A non-zero value typically causes the current job to fail with an I/O error and causes NormalizerServerRunJob() to return the `normPostScriptError` enumeration value.

Header File

`apcif.h`

NormalizerAddDiskStorageDevice()

```
NormalizerResult NormalizerAddDiskStorageDevice (int n,
char *prefix, int priority);
```

Description

The `NormalizerAddDiskStorageDevice()` function defines an additional PostScript language storage device, mapping it to a specific directory that is accessible to the Adobe PDF Converter SDK. `NormalizerAddDiskStorageDevice()` must be called before `NormalizerServerInit()` (or `APCInit()` when `dynamicmode is ON`).

The storage device can be used for file operations and resource access within the PostScript language. See Section 3.8.2 of the *PostScript Language Reference, Third Edition*, for more information on storage devices.

Parameters

`n`	The number of the device. In the PostScript language, this number will be part of the device's name, `%diskn%`. Legal values for `n` are integers from 1 to 99. The value of 0 is already reserved because `%disk0%` represents the current working directory of the Adobe PDF Converter SDK.
`prefix`	The directory name (including a trailing `/`) to which the device will be mapped. The syntax for `prefix` is the same as that used for the `path` member of a `NORMSearchList` structure.
`priority`	The priority with which the device will be searched during file operations. Legal values for priority range from 0 to 99. A value of 0 has the highest priority, and a value of 99 has the lowest. Setting `priority` to -1 sets the storage device's priority to the default value 2.

Return Value

One of the `NormalizerResult` enumerators.

Header File

`apcif.h`

NormalizerDisableDistilling()

```
NormalizerResult
NormalizerDisableDistilling(NSServerDataPtr serverData);
```

Description

The `NormalizerDisableDistilling()` function turns off the production of PDF output for a job, but allows the Adobe PDF Converter SDK to continue interpreting the job.

When distillation is disabled, the Adobe PDF Converter SDK stops adding PDF page objects to the full-document PDF file and stops producing streams for external files. If `filePerPage` is set to TRUE in the `NSJobParams` structure, the Adobe PDF Converter SDK also stops producing PDF page streams. When dynamic mode is ON, the pages for which the distillation is diasbled are not added to the dynamic file.

There are two purposes for disabling and enabling distillation:

- *Parallel conversion* — To support parallel conversion (distillation) where, for example, multiple instances of the Adobe PDF Converter SDK are setup to process interleaved pages of the same PostScript stream/file.
- *Selective conversion* — To produce a full-document PDF file that is missing some of the pages present in the PostScript stream/file

While distillation is disabled, the Adobe PDF Converter SDK continues invoking the `NSEndPage()` callback at the end of each page; however, it stops invoking the other data transfer callbacks, including `NSStartPage()`.

When dynamic mode is ON, the Adobe PDF Converter SDK continues invoking the **APCEndPage()** callback at the end of each page; however, it stops invoking the other data transfer callbacks.

This function should only be called either before running a job or during an `NSEndPage()` (**APCEndPage()** when dynamic mode is ON) callback. Calling it at other times will produce unpredictable results.

Parameters

`serverData`	An opaque pointer to the Adobe PDF Converter SDK's private data.

Return Value

One of the `NormalizerResult` enumerators.

Header File

`apcif.h`

Availability

All Adobe PDF Converter SDK versions.

NormalizerEnableDistilling()

```
NormalizerResult
NormalizerEnableDistilling(NSServerDataPtr serverData);
```

Description

The `NormalizerEnableDistilling()` function turns on the production of PDF output for a job, following a previous call to `NormalizerDisableDistilling()`.

This function should only be called either before running a job or during an `NSEndPage()` (**APCEndPage()** when dynamic mode is ON) callback. Calling it at other times will produce unpredictable results.

Parameters

`serverData`	An opaque pointer to the Adobe PDF Converter SDK's private data.

Return Value

One of the `NormalizerResult` enumerators.

Header File

`apcif.h`

Availability

All the Adobe PDF Converter SDK versions.

NormalizerNewHostFontList()

```
NormalizerResult NormalizerNewHostFontList(NSServerDataPtr
serverData, NORMSearchList hostfontSearchList, normBool reset,
normBool ignoreStdTTFonts);
```

Description

The `NormalizerNewHostFontList()` function modifies the directories containing host fonts that the Adobe PDF Converter SDK uses when searching for host font programs. This function replaces the deprecated function `NormalizerNewResourceList()`, which could not be called while a job was running.

NOTE: If more than one instance of the Adobe PDF Converter SDK is running on a machine, and all the instances are sharing a host font cache, then the `hostfontSearchList` for each Adobe PDF Converter SDK instance must be identical. In addition, `NormalizerNewHostFontList()` must not be used to change the `hostfontSearchList` values.

Parameters

`serverData`	An opaque pointer to the Adobe PDF Converter SDK's private data.
`hostfontSearchList`	A linked list of directory names that the Adobe PDF Converter SDK uses when searching for files containing host fonts. This linked list is initialized in the `NSClientConfig` structure (`hostfontSearchList` entry). It should contain ALL font directories. The client deallocates `searchList` when this function returns. **NOTE:** Each pathname in the referenced search list must be encoded using standard characters and must contain a platform-specific separator, such as "/" or "\".
`reset`	If TRUE, the hostfont cache is reset; otherwise, it is not.
`ignoreStdTTFonts`	If TRUE, the Adobe PDF Converter SDK ignores the TrueType versions of certain standard PostScript fonts. This parameter can also be specified in the `ignoreStdTTFonts` field of the `NSClientConfig` struct.

Return Value

One of the `NormalizerResult` enumerators.

NormalizerServerInit()

```
NormalizerResult NormalizerServerInit(NSClientConfig *config,
NSServerDataPtr *pServerData);
```

Description

The `NormalizerServerInit()` function initializes the interface between the client and Adobe PDF Converter SDK.

Parameters

`config`	A pointer to the `NSClientConfig` structure, which contains callbacks and other parameters used during all subsequent calls. The client deallocates this structure after shutting down the Adobe PDF Converter SDK.
`pServerData`	The location at which the Adobe PDF Converter SDK stores a pointer to its opaque private data.

Return Value

One of the `NormalizerResult` enumerators. The client can submit additional requests to the Adobe PDF Converter SDK only if the return value is `normOK`.

Header File

`apcif.h`

NormalizerServerRunJob()

```
NormalizerResult
NormalizerServerRunJob(NSServerDataPtr serverData,
NSJobParams *jobParams);
```

Description

The client calls the `NormalizerServerRunJob()` function to begin converting a PostScript stream into PDF expressions. After the client calls this function, the Adobe PDF Converter SDK may use client callbacks to obtain PostScript streams, to write out data to files, or to report information about the PostScript content being processed.

NOTE: `NormalizerServerRunJob()` does not re-enable the distillation process if it was disabled during a previous job. To avoid running a job with distillation erroneously disableld, the client should call `NormalizerEnableDistilling()` just before calling `NormalizerServerRunJob()`.

Parameters

`serverData`	An opaque pointer to the Adobe PDF Converter SDK's private data.
`jobParams`	A pointer to the `NSJobParams` structure, which specifies how the PostScript file should be converted. The client deallocates this structure on job completion (when `NormalizerServerRunJob()` returns).

Return Value

Returns one of the `NormalizerResult` enumerators.

If this function returns with an unsuccessful enumerator result, the Adobe PDF Converter SDK does the following:

- Closes and deletes the partially-built full-document PDF file, if the Adobe PDF Converter SDK is performing file I/O (the `NSClientFile` API is not being used).

If `NormalizerServerRunJob()` returns with an error, the client should close and delete any open page files or external files.

Header File

`apcif.h`

NormalizerServerShutdown()

```
NormalizerResult
NormalizerServerShutdown(NSServerDataPtr serverData);
```

Description

The `NormalizerServerShutdown()` function terminates the Adobe PDF Converter SDK.

Parameters

`serverData`	An opaque pointer to the Adobe PDF Converter SDK's private data.

Return Value

Returns one of the `NormalizerResult` enumerators

Header File

`apcif.h`

NormalizerSetDisk0Prefix()

```
NormalizerResult NormalizerSetDisk0Prefix(char *prefix);
```

Description

The **NormalizerSetDisk0Prefix()** function specifies the location of a string with a trailing slash (/) to which the **%disk0%** storage device will be mapped.

By default, **%disk0%** maps to the current working directory, which might be dynamic. **NormalizerSetDisk0Prefix()** can be used to specify an absolute path such as /volume/directory/NSproduct/, in which case the location of **%disk0%** is independent of the current working directory.

Parameters

prefix	A pointer to a string whose syntax is the same as that used by the **path** member of a NORMSearchList structure.

Return Value

Returns one of the NormalizerResult enumerators

Header File

apcif.h

NSBackChanMsg()

```
void (*NSBackChanMsg)(NSClientDataPtr clientData, char *buf);
```

Description

The `NSBackChanMsg()` callback implements the PostScript Interpreter's standard output device. It sends errors to the client that cause a job to be terminated. Errors are reported in strings that begin with `%%[Error`.

After calling the `NSBackChanMsg()` callback to report an error that causes a job to be terminated, the `NormalizerServerRunJob()` function returns one of the `NormalizerResult` enumerators. `NSBackChanMsg()` does not return any results.

Parameters

`clientData`	A pointer to the client's private data.
`buf`	A pointer to a character string that contains an error message. The Adobe PDF Converter SDK deallocates this structure after `NSBackChanMsg()` returns.

Return Value

None.

Header File

`apcif.h`

NSBufferGetPS()

```
int (*NSBufferGetPS)(NSClientDataPtr clientData, char **pBuf,
unsigned int *pLength, normBool usePrologue);
```

Description

The **NSBufferGetPS()** callback requests additional input PostScript data.

The client should not reuse or free a buffer provided in the **NSBufferGetPS()** callback until the Adobe PDF Converter SDK makes a new call to **NSBufferGetPS()** or as the job finishes.

See also Section 7.1.1, “How the Adobe PDF Converter SDK Uses Data Transfer Callbacks.

Parameters

clientData	A pointer to the client’s private data.
pBuf	In the location specified by **pBuf**, the client stores a pointer to a buffer of PostScript stream. The client deallocates this buffer at the next call to **NSBufferGetPS()** or when the job finishes (when **NormalizerServerRunJob()** returns).
pLength	In this location, the client stores the number of bytes in the PostScript buffer that contain information. The client sets **pLength** to 0 when it has no more PostScript stream remaining in the job.
usePrologue	The value of the **UsePrologue** Distiller parameter. If TRUE, you may choose to add a `prologue.ps` and/or an `epilogue.ps` to the PostScript stream.

Return Value

Indicates the client’s success at obtaining the next buffer of PostScript stream. A value of 0 indicates success. An error will cause the job to terminate. An end-of-file condition should not be considered an error.

Header File

`apcif.h`

NSBufferHandOff()

```
int (*NSBufferHandOff)(NSClientDataPtr clientData,
NSFileDataPtr fileData, char *buf, int length);
```

Description

The `NSBufferHandOff()` callback writes output data to a PDF file or external stream file. The Adobe PDF Converter SDK calls `NSBufferHandOff()` multiple times to send the client either PDF streams for individual pages or data to be stored in external files. The Adobe PDF Converter SDK invokes `NSBufferHandOff` if distillation is enabled and if:

- `filePerPage` is TRUE and Adobe PDF Converter SDK finishes interpreting a page.
- `sidelineEPS` is TRUE and Adobe PDF Converter SDK encounters an embedded EPS segment.
- `sidelineImages` is TRUE and Adobe PDF Converter SDK detects an embedded image.

See also Section 7.1.1, "How the Adobe PDF Converter SDK Uses Data Transfer Callbacks.

Parameters

`clientData`	A pointer to the client's private data.
`fileData`	A pointer to the client's file handle returned by `NSCreateExternalFile()`.
`buf`	A pointer to the buffer that the client should write to the file specified in `fileData`. The client is not responsible for deallocating this buffer.
`length`	The number of bytes in `buf`.

Return Value

An integer value of 0 indicates success. A nonzero value indicates an error, such as an out-of-memory or file access problem. A nonzero value typically causes the current job to fail with an **ioerror** and causes `NormalizerServerRunJob()` to return with the `normPostScriptError` enumeration value.

Header File

`apcif.h`

NSCantHappenProc()

```
typedef void (*NSCantHappenProc) (unsigned long errID);
```

Description

The `NSCantHappenProc()` callback is called under fatal error conditions. If you don't want to use this callback, it must be NULL.

Parameters

`errID`	Identifies the kind of fatal error condition.

Return Value

No value is returned.

Header File

`apcif.h`

This callback tells the client a fatal error has terminated execution.

NSCloseExternalFile()

```
void (*NSCloseExternalFile)(NSClientDataPtr clientData,
NSFileDataPtr fileData);
```

Description

The `NSCloseExternalFile()` callback indicates to the client that it may close the file. Adobe PDF Converter SDK calls this callback function when it has finished writing to an external file. The client should free the `NSFileDataPtr` handle used in the `NSCreateExternalFile()` callback.

Parameters

`clientData`	A pointer to the client's private data.
`fileData`	A pointer to the client's file handle returned by `NSCreateExternalFile()`.

Return Value

None.

Header File

`apcif.h`

NSCreateExternalFile()

```
int (*NSCreateExternalFile)(NSClientDataPtr clientData,
NormalizerSidelineType sidelineType, NSFileDataPtr *pFileData,
char *nameBuf, int maxNameLength);
```

Description

The `NSCreateExternalFile()` callback creates an external file. External files are used to store sidelined images and EPS files. The client must create the file and return a `NSFileDataPtr` handle to it. It must also provide a name for the file. The name will be written into the individual page PDF file. Normally the external stream file should be placed in the same directory as the PDF files and the filename should be a relative path.You must ensure that the filename provided will allow a RIP to open the file using `os_fopen()`.

The string placed in `nameBuf` may either be a null-terminated ASCII string or a UNICODE string. A UNICODE string must be terminated with two NULL bytes and must quote bytes that correspond to path separator characters, as described in section 7.3.1 of the *PDF Language Specification*, Version 1.3.

Parameters

`clientData`	A pointer to the client's private data.
`sidelineType`	An enumerated value of type `NormalizerSidelineType` that describes the type of data that Adobe PDF Converter SDK will store in the client-created file. Possible values are: • normSidelineImage - Image data • normSidelineEPS - EPS data
`pFileData`	A pointer to the location where the client stores a pointer to a new instance of file data.
`nameBuf`	A pointer to a character string in which the client stores the pathname of the external file. That name must be of type FILESPECIFICATION, as described in Section 7.3.1 of the *PDF Language Specification*, Version 1.3. NOTE: Adobe PDF Converter SDK uses the name referenced by `nameBuf` to create external references in the PDF content it subsequently produces.The referenced character string may be encoded using either the standard character format or the UNICODE one.
`maxNameLength`	The maximum number of characters permitted in the name not including the NULL terminator.

Return Value

An integer that indicates the client's success at creating the requested file. A value of 0 indicates success. A nonzero value indicates an error, such as file access problems. A nonzero value typically causes the current job to fail with an **ioerror** and causes `NormalizerServerRunJob()` to return with the `normPostScriptError` enumeration value.

Header File

`apcif.h`

NSDupFontNotifyProc()

```
void (*NSDupFontNotifyProc) (PNSHostFontListData entry1,
PNSHostFontListData entry2 );
```

Description

The `NSDupFontNotifyProc()` callback provides information to the client on duplicate fonts found at startup. The information provided is read-only and cannot be changed.

NOTE: This callback must be NULL if it is not used.

Parameters

`entry1`	A pointer to a `NSHostFontListData` structure.
`entry2`	A pointer to a `NSHostFontListData` structure describing the same host fonts as those located at `entry1`.

Return Value

None.

Header File

`apcif.h`

NSEndPage()

```
void (*NSEndPage)(NSClientDataPtr clientData,
SPDKeyValue *pageDevFeatureList, NSPageInfo *pageInfo);
```

Description

The `NSEndPage()` callback indicates that Adobe PDF Converter SDK has finished processing a page, regardless of whether distillation is enabled or `filePerPage` is set to TRUE. As part of this call, Adobe PDF Converter SDK reports the page device keys appearing on the page, as well as plate color information and page label information from PostScript and DSC comments on the page.

If `filePerPage` is set to TRUE and distillation is enabled, the client should close the PDF page file it created for the page.

See also Section 7.1.1, “How the Adobe PDF Converter SDK Uses Data Transfer Callbacks.

Parameters

`clientData`	A pointer to the client's private data.
`pageDevFeatureList`	A pointer to a linked list of key-value pairs representing page device keys that have appeared in the job since the previous page. The type declaration for the nodes in this list appears after this table. Using this argument, Adobe PDF Converter SDK reports only those page device keys that have changed in the PostScript expressions since the last page in the job and that are listed in `*setpagedeviceKeysList`. See the `NSClientConfig` structure. Adobe PDF Converter SDK returns NULL if there are no page device keys to report. NOTE: A job may contain several concatenated PostScript files or streams. Adobe PDF Converter SDK deallocates this list after the callback returns.
`pageInfo`	An `NSPageInfo` structure that provides plate color and page label information about a page. The structure contains pointers to the following character strings: • `plateColor`, if non-NULL, references the name of the plate color declared by the PostScript job in an `%%PlateColor:` PostScript comment. • `pageLabel`, if non-NULL, references the page label specified by the PostScript job in a `%%Page:` comment Adobe PDF Converter SDK deallocates this structure after the callback returns.

Return Value

None.

Header File

`apcif.h`

NSErrorMsg()

```
void (*NSErrorMsg)(NSClientDataPtr clientData, char * buf);
```

Description

The `NSErrorMsg()` callback sends fatal error messages from Adobe PDF Converter SDK to the client.

Parameters

`clientData`	A pointer to the client's private data.
`buf`	A pointer to the error message. Adobe PDF Converter SDK deallocates this structure after `NSErrorMsg()` returns.

Return Value

`NormalizerServerRunJob()` returns one of the `NormalizerResult` enumerators.

Header File

`apcif.h`

NSExitProcessProc()

```
void (*NSExitProcessProc)(void);
```

Description

The `NSExitProcessProc` callback is called under fatal error conditions. If you don't want to use this callback, it must be NULL.

Parameters

None.

Return Value

Do not return from this callback; exit instead.

Header File

`apcif.h`

NSExternalCommandProc()

```
unsigned int (*NSExternalCommandProc)
(NSClientDataPtr clientData, char *command,
long int commandLength, char *response, long int
*responseLength);
```

Description

The `NSExternalCommandProc()` callback allows interaction between the PostScript program and your client software. Adobe PDF Converter SDK invokes the `NSExternalCommand()` callback when a PostScript language program executes the key-word pair **externalcommand**. The combination of **externalcommand** in a PostScript program and an `NSExternalCommand()` implementation allows the PostScript program to communicate directly with Adobe PDF Converter SDK. See also Section 7.4, “Callback for Responding to the externalcomm and PostScript Operator.

If the callback is set to NULL in the `NSClientConfig` structure, an internal callback is used, which does nothing.

Parameters

`clientData`	A pointer to the client’s private data.
`command`	A pointer to the command (text) string containing the PostScript command that will pushed onto the operand stack.
`commandLength`	A pointer to the length in bytes of the PostScript command (text) string.
`response`	A pointer to a text string containing Adobe PDF Converter SDK’s response to the command.
`responseLength`	A pointer to the length in bytes of the response text string.

Return Value

A zero or non-zero integer. A zero result pushes FALSE onto the operand stack after the response string. A non-zero result pushes TRUE onto the PostScript operand stack. The significance of the result your client supplies reflects the result of the query posed by the command string.

NSExternalProcessCommentCleanupProc()

Windows only.

```
void (*NSExternalProcessCommentCleanupProc)
(NSClientDataPtr clientData);
```

Description

This function deallocates resources used by `NSExternalProcessCommentProc()`. This function is called just after `NSExternalProcessCommentProc()` executes, and is also called if a PostScript error occurs during execution of `NSPSExecuteStringProc()`.

For more information on this callback, see "Callbacks for Modifying DSC and PostScript" on page 64.

Parameters

`clientData`	A pointer to the client's private data. If the value of `clientData` is NULL, then this function is called but does not perform any operation.

Return Value

None.

Header File

`apcif.h`

NSExternalProcessCommentProc()

Windows only.

```
void (*NSExternalProcessCommentProc)
(NSClientDataPtr clientData, char * line, unsigned int
*skipPS, NSPSExecuteStringProc() executeStringProc);
```

Description

This function provides enhanced control over the parsing of a given DSC comment by allowing you to specify skip lines of PostScript or execute arbitrary PostScript if desired. This function is called just after `NSExternalProcessCommentSetupProc()` executes, and is always followed by `NSExternalProcessCommentCleanupProc()`.

For more information on this callback, see "Callbacks for Modifying DSC and PostScript" on page 64.

Parameters

`clientData`	A pointer to the client's private data.
`line`	A DSC comment string passed from Adobe PDF Converter SDK.
`skipPS`	An integer value which is used to return a value to Adobe PDF Converter SDK. When set to a non-zero value, this argument causes Adobe PDF Converter SDK to skip executing lines of PostScript until the next DSC comment is read.
`executeStringProc`	A pointer to `NSPSExecuteStringProc()` which allows you to execute arbitrary PostScript.

Return Value

None.

Header File

`apcif.h`

NSExternalProcessCommentSetupProc()

Windows only.

```
void (*NSExternalProcessCommentSetupProc)
(NSClientDataPtr clientData);
```

Description

This function sets up necessary resources needed to execute `NSExternalProcessCommentProc()`. This function is called just before `NSExternalProcessCommentProc()` executes.

For more information on this callback, see "Callbacks for Modifying DSC and PostScript" on page 64.

Parameters

`clientData`	A pointer to the client's private data. If the value of `clientData` is NULL, then this function is called but does not perform any operation.

Return Value

None.

Header File

`apcif.h`

NSFreeMemoryProc()

```
typedef void(*NSFreeMemoryProc) (NSClientDataPtr clientData,
void *freeThis, NSMemoryFreeInfoPtr memInfoPtr);
```

Description

The `NSFreeMemoryProc()` callback allows you control to free VM memory. It is only used if `NSMoreMemoryProc()` also is implemented.

NOTE: If this callback is not used, it must be set to NULL in `NSClientConfig`.

Parameters

`clientData`	ClientData pointer set by the product.
`freeThis`	A pointer to the memory to free.
`memInfoPtr`	Information regarding the memory freeing.

Return Value

None.

Header File

`apcif.h`

NSGetHostFontMutexProc()

Windows only.

```
void * (*NSGetHostFontMutexProc)(void);
```

Description

The `NSGetHostFontMutexProc()` callback either creates or acquires a machine global mutex as required. The callback then returns a handle to the mutex.

This function is used when multiple Adobe PDF Converter SDK processes are sharing the same host font file, as in the case of parallel conversion. For a description of parallel conversion, and more information on `NSGetHostFontMutexProc()`, see “Parallel Conversion” on page 26.

Parameters

None.

Return Value

Returns a pointer to a handle.

Header File

`apcif.h`

NSGetNextDeviceActivatePageNumber()

```
typedef void(*NSGetNextDeviceActivatePageNumber)
(NSClientDataPtr clientData, int *pageno)
```

Description

This is new callback which has been added in version 2.1.

The **`NSGetNextDeviceActivatePageNumber()`** callback is called when a job starts. It allows the product to provide page number of job APC should convert to postscript. The page which is specified by product would be rendered by APC. After done with the processing of specified page, this callback is called to get next page number to be converted. This way product can specify the page number which it wants to be converted through this callback, only those pages of job will be converted to postscript rest of the pages will be just skipped without converting.

Parameters

`clientData`	A pointer to the client's private data.
`pageno`	Page number which should be converted by APC to postscript.

Return Value

None.

Header File

`apcif.h`

NSMoreMemoryProc()

```
typedef char * (*NSMoreMemoryProc) (NSClientDataPtr
clientData,long int * nBytes, NSMemoryAllocInfoPtr
memInfoPtr);
```

Description

The `NSMoreMemoryProc()` callback allows you to control the allocation of VM memory. If `NSMoreMemoryProc()` is implemented, `NSFreeMemoryProc()` must also be implemented. Use this callback to control or monitor the allocation of memory.

NOTE: If this callback is not used, it must be set to NULL in `NSClientConfig`.

Parameters

`clientData`	ClientData pointer set by the product.
`nBytes`	Number of bytes to be alocated.
`memInfoPtr`	Information regarding the memory freeing.

Return Value

None.

Header File

`apcif.h`

NSProcessComment()

```
int (*NSProcessComment)(NSClientDataPtr clientData,
char *comment, char **newText);
```

Description

The `NSProcessComment()` callback parses each enclosed comment in a PostScript job. For processing PostScript jobs that claim compliance with DSC 2.0 or 3.0, Adobe PDF Converter SDK calls `NSProcessComment()` before parsing:

- Each enclosed comment that begins with a percent sign (%) followed by a non-blank character
- Each subsequent line that begins with a percent sign.

The client may simply read the comment for information purposes or use it to modify the job by substituting a new block of PostScript for the comment.

Adobe PDF Converter SDK does not report comments for external fonts referenced from the PostScript stream.

Parameters

`clientData`	A pointer to the client's private data.
`comment`	A pointer to the comment text string. The pointer references a location in the buffer previously supplied by the client in the `NSBufferGetPS()` callback.
`newText`	If the client wants to replace the comment with a new block of PostScript, `newText` is a pointer to the location where the new block is placed.

Return Value

An integer value of zero indicates Adobe PDF Converter SDK should continue processing the PostScript comment. A nonzero integer value indicates Adobe PDF Converter SDK should process the text referenced by `newText`, rather than the comments referenced by `comment`.

Header File

`apcif.h`

NSProgress()

```
int (*NSProgress)(NSClientDataPtr clientData);
```

Description

The `NSProgress()` callback is called every time the PostScript Interpreter processes the number of PostScript operations indicated in the `progressQuantum` field of the `NSClientConfig` structure.

Parameters

`clientData`	A pointer to the client's private data.

Return Value

None.

Header File

`apcif.h`

NSPSExecuteStringProc()

Windows only.

```
void (*NSPSExecuteStringProc)(char *buf);
```

Description

This function is provided by Adobe PDF Converter SDK for use in conjunction with `NSExternalProcessCommentProc()`. `NSPSExecuteStringProc` is called from `NSExternalProcessCommentProc()` as a way to pass arbitrary PostScript into the stream.

IMPORTANT: *The PostScript you supply through* `NSPSExecuteStringProc` *must not modify the PostScript graphic state.*

Parameters

`buf`	A pointer to valid PostScript. The PostScript is executed immediately. You may release the buffer when `NSPSExecuteStringProc()` returns.

Return Value

None.

Header File

`apcif.h`

NSReleaseHostFontMutexProc()

Windows only.

```
void (*NSReleaseHostFontMutexProc)(void *)
```

Description

The `NSReleaseHostFontMutexProc()` callback releases the mutex that is created acquired by the `NSGetHostFontMutexProc()` callback.

This function is used when multiple Adobe PDF Converter SDK processes are sharing the same host font file, as in the case of Parallel Conversion. For a description of parallel conversion, and more information on `NSReleaseHostFontMutexProc()`, see “Parallel Conversion” on page 26.

Parameters

Takes in the mutex handle (cast as a void) used by `NSGetHostFontMutexProc()`.

Return Value

None.

Header File

`apcif.h`

NSStartPage()

```
int (*NSStartPage)(NSClientDataPtr clientData,
NSFileDataPtr *pFileData);
```

Description

The `NSStartPage()` callback informs the client that Adobe PDF Converter SDK is ready to begin transferring a PDF stream for a page. Adobe PDF Converter SDK calls `NSStartPage()` only if `filePerPage` is TRUE and distillation is enabled.

In the function invoked by the `NSStartPage()` callback, the client should open a file in which to store the PDF page stream produced by Adobe PDF Converter SDK.

Parameters

`clientData`	A pointer to the client's private data.
`pFileData`	A pointer to the location at which the client stores a pointer to a new instance of file data.

Return Value

An integer that indicates the client's success at creating the requested file. A value of 0 indicates success. A nonzero value indicates an error, such as file access problems. A nonzero value typically causes the current job to fail with an **ioerror** and causes `NormalizerServerRunJob()` to return with the `normPostScriptError` enumeration value.

Header File

`apcif.h`

NSTotalNumberofPages()

```
typedef void(*NSTotalNumberofPages) (NSClientDataPtr
clientData, int TotalPageCount)
```

Description

This is the new callback which has been added in version 2.1.

The **`NSTotalNumberofPages()`** callback is called at the end of the job with total number of pages in job.

Parameters

`clientData`	A pointer to the client's private data.
`TotalPage Count`	Total numbers of pages in job.

Return Value

None

Header File

`apcif.h`

14 NSClientFile API

NSBufsizeProc()

```
int (*NSBufsizeProc)(NSClientDataPtr clientData,
NSClientFileID fd, int *result);
```

Description

The `NSBufsizeProc()` callback queries the client on the buffer size the Adobe PDF Converter SDK should use for reads and writes. This callback will be used once each job, before any read or write operations are performed.

Parameters

`clientData`	A pointer to the client's private data.
`fd`	Client file ID, which is the client's file identifier.
`result`	The optimal buffer size for the file. Can be set to zero if the client has no preference about the buffer size.

Return Value

The client returns a value that indicates the success or failure of the function. The value 0 indicates success; the value -1 indicates failure. (The error return is provided for consistency; there is no reason the `NSBufsizProc()` callback would ever make an error return.)

Header File

`apcif.h`

NSCloseProc()

```
int (*NSCloseProc)(NSClientDataPtr clientData,
NSClientFileID fd);
```

Description

The `NSCloseProc()` callback notifies the client to close the file associated with `fd`, and instructs the Adobe PDF Converter SDK to make no further callbacks using `fd`.

Parameters

`clientData`	A pointer to the client's private data.
`fd`	Client file ID, which is the client's file identifier.

Return Value

The client returns a value that indicates the success or failure of the function. The value 0 indicates success; the value -1 indicates failure. A failure return typically causes the current job to fail with an **ioerror** and causes the `NormalizerServerRunJob()` function to return with the `normPostScriptError` enumeration value.

Header File

`apcif.h`

NSReadProc()

```
int (*NSReadProc)(NSClientDataPtr clientData,
NSClientFileID fd, char *buf, unsigned int count);
```

Description

The `NSReadProc()` callback requests the client to read data from the client file specified by `fd`.

Parameters

`clientData`	A pointer to the client's private data.
`fd`	Client file ID, which is the client's file identifier.
`buf`	A pointer to an area where the data read from the file should be stored. Adobe PDF Converter SDK is responsible for allocating and freeing this memory.
`count`	The number of bytes that the client should read into `buf`.

Return Value

The client returns a value that indicates the success or failure of the callback. A non-negative value indicates the number of bytes the client successfully read. A value of -1 indicates failure. A failure return typically causes the current job to fail with an **ioerror** and typically causes the `NormalizerServerRunJob()` function to return with the `normPostScriptError` enumeration value.

Header File

`apcif.h`

NSSeekProc()

```
int (*NSSeekProc)(NSClientDataPtr clientData,
NSClientFileID fd, long where, int seekMethod);
```

Description

The **NSSeekProc()** callback requests the client to reposition the file pointer for **fd**.

Parameters

clientData	A pointer to the client's private data.
fd	Client file ID, which is the client's file identifier.
where	Together with **seekMethod**, determines how the file pointer should be changed.
seekMethod	Is one of the following values: • **NS_SEEK_SET** (0) indicates the file pointer should be set to the location specified by **where**. • **NS_SEEK_SET** (1) indicates the file pointer should be set to its current location plus **where**. • **NS_SEEK_END** (2) indicates the file pointer should be set to the end of the file plus **where**.

Return Value

The client returns a value that indicates the success or failure of the callback. A non-negative value indicates the number of bytes the client successfully read. A value of -1 indicates failure. A failure return typically causes the current job to fail with an **ioerror** and typically causes the **NormalizerServerRunJob()** function to return with the **normPostScriptError** enumeration value.

Header File

apcif.h

NSTruncateProc()

```
int (*NSTruncateProc)(NSClientDataPtr clientData,
NSClientFileID fd, long where);
```

NOTE: Although the `NSTruncateProc()` callback is required, Adobe PDF Converter SDK never invokes it. The NSClientFile API includes it with future development in mind.

Description

The `NSTruncateProc()` callback directs the client to terminate the file at a particular position. Any bytes occurring beyond that position are lost.

Parameters

`clientData`	A pointer to the client's private data.
`fd`	Client file ID, which is the client's file identifier.
`where`	The position just beyond which file should be truncated.

Return Value

The client returns a value that indicates the success or failure of the function. The value 0 indicates success; the value -1 indicates failure. A failure return typically causes the current job to fail with an **ioerror** and causes the `NormalizerServerRunJob()` function to return with the `normPostScriptError` enumeration value.

Header File

`apcif.h`

NSWriteProc()

```
int (*NSWriteProc)(NSClientDataPtr clientData,
NSClientFileID fd, char *buf, unsigned int count);
```

Description

The `NSWriteProc()` callback requests that data be written to the client file specified by `fd`.

Parameters

`clientData`	A pointer to the client's private data.
`fd`	Client file ID, which is the client's file identifier.
`buf`	A pointer to area where data read from the file should be stored. Adobe PDF Converter SDK is responsible for allocating and freeing this memory.
`count`	The number of bytes that the client should read into `buf`.

Return Value

The client returns a value that indicates the success or failure of the callback. A non-negative value indicates the number of bytes the client successfully read. A value of -1 indicates failure. A failure return typically causes the current job to fail with an **ioerror** and typically causes the `NormalizerServerRunJob()` function to return with the `normPostScriptError` enumeration value.

If the client returns a success indication, but the number of bytes written to the file is not the same as `count`, the job fails with an **ioerror** and `NormalizerServerRunJob()` returns with the `normPostScriptError` enumeration value.

Header File

`apcif.h`

15 Conversion of Image Files to PDF

Using the Adobe PDF Converter SDK you can convert image files to PDF files. The SDK supports the conversion of JPEG, BMP, TIFF, and PNG image files to PDF files. Image to PDF conversion also supports alpha channel in image formats and converts them to softmask in PDF file.

The following APIs are available for conversion:

APCCreateDoc()

APCConfigureImageJob()

APCAddImage()

APCStartFile()

16 Conversion of PPML Files to PDF

Using the Adobe PDF Converter SDK, you can convert PPML files to PDF files. The SDK supports the conversion of PPML 2.1 specification files to PDF files.

The API available for conversion is APCConvertPPMLToPDF().

Limitations of PPML Implementation

APCConvertPPMLToPDF() supports all PPML file conforming to PPML 2.1 specification. However, implementation of a few PPML tags is not supported.

Given below is the list of PPML tags which are not supported in APC 3.2:

- FONT
- PROCESSOR
- CONFORMANCE
- TICKET
- TICKET_REF
- TICKET_SET
- TICKET_STATE
- SEGMENT_ARRAY
- SEGMENT_REF
- HOR_TRIM_MARKS
- VER_TRIM_MARKS
- HOR_FOLD_MARKS
- VER_FOLD_MARKS
- REPEAT

NOTE: If a PPML file contains these tags then it will be processed after ignoring these tags

A key point to note is that the generated PDF from PPML contains DPart structure, DPart MetaData and GTS keys information. But generated PDF is not PDF/X-4 compliant and hence it is not marked as PDF/VT file.

17 Dynamic N Page PDF Generation

When converting a PS file/stream to a PDF file, you can intercept the conversion at a specific page and notify Adobe PDF Converter SDK to include pages only up to the point of interception, in the output PDF stream. The client indicates the page at which the conversion is to be intercepted, by implementing the `APCEndPage()` call back.

The following APIs are available for Dynamic N Page PDF generation:

APCInit()

APCEnableDynamicGeneration()

The following callbacks need to be implemented by the client for Dynamic N page PDF generation:

APCStartFile()

APCEndFile()

APCEndPage()

Following is the structure that the client needs to initialize for Dynamic N page PDF generation:

APCClientConfig Structure

17.1 Improvement in Dynamic N Page PDF Generation

The decision to create a PDF file can now be made at the time of device activation. Device activation can happen either by a call to **setpagedevice** operator or when PostScript VM graphic state is resorted with a call to restore operator. In both these cases APCEndPage() callback is called which can take decision to create new PDF file based on **SPDKeyValue** list.

This functionality is also enabled with +g option on command line.

The demo code function **DemoEndPageV2** implements a simple state mechanism where a new PDF file is created whenever value of staple key changes from 0 to 1 and vice versa.

For example, in the following PS Snippet, 3 PDF files are created. First PDF file will have 2 pages, second PDF file will have 1 page and third PDF file will have 2 pages.

```
%!PS
<< /Staple 1 >> setpagedevice
100 100 moveto
/Helvetica findfont 8 scalefont setfont
(Page 1)show
showpage
100 100 moveto
/Helvetica findfont 8 scalefont setfont
(Page 2)show
showpage
save
<< /Staple 0 >> setpagedevice
100 100 moveto
/Helvetica findfont 8 scalefont setfont
(Page 3)show
showpage
restore
100 100 moveto
/Helvetica findfont 8 scalefont setfont
(Page 4)show
showpage
<< /Staple 1 >> setpagedevice
100 100 moveto
/Helvetica findfont 8 scalefont setfont
(Page 5)show
showpage
```

18 Structures and Enumerations

This chapter provides reference information about the structures and enumerations your client software uses to interface with Adobe PDF Converter SDK.

APCClientConfig Structure

```
typedef struct _t_APCClientConfig {
/* Client's private data pointer, passed back through
callbacks. */
NSClientDataPtr clientData;
NSClientConfig* config;
/* clients's call backs */
APCEndPage endOfPageV2;
APCStartFile startOfFile;
APCEndFile endOfFile;
} APCClientConfig;
```

Description

The APCClientConfig structure contains configuration information that the client passes to Adobe PDF Converter SDK in the call to APCInit().

Header File

apcif.h

Members

clientData	A pointer to client-provided private data. This pointer is passed back in all callbacks from Adobe PDF Converter SDK to the client. This structure is opaque to Adobe PDF Converter SDK.
config	pointer to structure NSClientConfig. This structure contains configuration information that the client passes to Adobe PDF Converter SDK in the call to NormalizerServerInit().
endOfPageV2	Funtion pointer to APCEndPage callback.

`startOfFile`	Funtion pointer to APCStrartFile callback
`endOfFile`	Funtion pointer to APCEndFile callback

NormalizerResult

```
typedef enum {
    normOk,
    normParameterError,
    normPostScriptError,
    normOutOfMemory,
    normOutOfDiskSpace,
    normInternalError,
    normClientCancel,
    normNotNow,
    normIncorrectInterfaceVersion,
    normAlreadyInitialized,
    normHasNotInitialized,
    docNotCreated,/* Unable to create an empty doc   */
    fileNotFound,
    incorrectImageFormat,
    fileNotSaved,
    incorrectPDFFormat,
    unknownError,
    docAlreadyCreated
} NormalizerResult;
```

Description

All Adobe PDF Converter SDK functions return one of the `NormalizerResult` enumerators to indicate the status of the function.

Header File

`apcif.h`

Members

`normOk`	The function was successful.
`normParameterError`	The client has passed an invalid parameter.
`normPostScriptError`	The file contained a PostScript error.
`normOutOfMemory`	Memory is insufficient to complete the function.
`normOutOfDiskSpace`	Disk space is inadequate to complete the function.
`normInternalError`	An internal Adobe PDF Converter SDK error has occurred.

normClientCancel	The client cancelled the function.
normNotNow	The requested operation is not currently permitted.
normIncorrect InterfaceVersion	The interfaceVersionNum supplied in NormalizerServerInit() is not compatible with the Adobe PDF Converter SDK code being initialized.
normAlreadyInitialized	NormalizerServerInit() was called more than once.
normHasNotInitialized	NormalizerServerInit() was called before NormalizerServerShutdown() was called.
docNotCreated	The converter cannot create a doc stream either because of insufficient space or an i/o error on the client side.
fileNotFound	The image file does not exists at the image path provided by the client.
incorrectImageFormat	The image format is not supported for conversion by the converter.
fileNotSaved	The converter is unable to save the doc stream to the PDF file specified by the client due to i/o issues on the client side.
incorrectPDFFormat	Reserved for future use.
unknownError	An unknown error occured in file i/o on the client side.
docAlreadyCreated	The client made a new call to APCCreateDoc() without saving the document opened previously with APCCreateDoc().

NormalizerSidelineType

```
typedef enum {
    normSidelineFontNoLongerSupported,
       /* Font data - no longer supported (From Norm 6.0)*/
    normSidelineImage,/* Image data */
    normSidelineEPS,/* EPS file */
    normCreateRefXObj
} NormalizerSidelineType;
```

Description

The `NormalizerSidelineType` enumerators are types of data that can be sidelined to external files in the `NSCreateExternalFile()` callback.

NOTE: The argument `normSideLineFont` is no longer available.

Header File

apcif.h

NORMSearchList

NORMSearchListRec

```
typedef struct _t_NORMSearchListRec{
    char *path;
    struct _t_NORMSearchListRec *next;
    void *extension;
    normBool matchingOnly;
} NORMSearchListRec, *NORMSearchList;
```

Description

The `NORMSearchList` structure is a linked list of directory names used to search for finding resources, such as font files.

Adobe PDF Converter SDK locates a file by appending the filename to each `path` string on the search list in turn and checking whether the resulting pathname is a file that can be opened. A NULL `next` pointer identifies the end of the search list. Adobe PDF Converter SDK begins its search with the first path on the search list and terminates as soon as it finds the required or it reaches the end of the list.

If an empty search list (NULL pointer) is provided when a search list is required, Adobe PDF Converter SDK looks for the file in the current working directory only. If the search list is not empty, it looks for the file only in the directories in the list. It uses the current working directory if the `path` member of a search list entry is NULL.

Header File

`apcif.h`

Members

`path`	A string to which a filename can be appended, for example, `fonts/` **NOTE:** Path must use be encoded using the standard character format rather than the UNICODE one. It must also be terminated with a platform-specific separator, such as “/” or “\”.
`next`	A pointer to next search list element or NULL.

`extension`	A pointer to additional OS-specific information used only for Macintosh platforms. Adobe PDF Converter SDK appends `*extension` to a font name to obtain the name of the file in which the desired font program or **FontFile** object is stored.
`matchingOnly`	If TRUE, Adobe PDF Converter SDK uses the TrueType fonts in the folder to perform internal font matching for Unicode and OS/2 information. If FALSE, Adobe PDF Converter SDK uses such fonts to resolve **findfont** PostScript/PDF commands. Examples of situations in which setting the flag helps: • *Distinguishing between identically-named TrueType fonts in different folders*. If two Arial TrueType font files exist in two different folders, only one of them will be used for **findfont**. **findfont** always uses the first one found in the folder listis useful for Adobe PDF Converter SDK to match a Type42 font against both Arial TrueType font files to see if any of them match. In such a situation, set the second folder with `matchingOnly` set to TRUE. • *Matching a TrueType font to an embedded PostScript font equivalent*. If only one Arial TrueType font exists and somehow the system font folder is deleted from the **Font Locations** dialog. For a PostScript job that has a Type42 of this Arial TrueType font embedded, it is also useful to see this Arial TrueType font for `matchingOnly`, not for **findfont** because the folder is not in **Font Locations**.

NSClientConfig

```
typedef struct _t_NSClientConfig {
   NSClientDataPtr clientData;
/* Client's callbacks: */
   NSStartPage startOfPage;
   NSEndPage endOfPage;
   NSBufferHandOff bufHandOff;
   NSBufferGetPS bufGetPS;
   NSBackChanMsg backChanMsg;
   NSBackChanMsg feedBackMsg;
   NSErrorMsg errorMsg;
   NSProcessComment processComment;
   NSCreateExternalFile  createExternalFile;
   NSCloseExternalFile  closeExternalFile;
   NSProgress progress;
   NSExternalCommandProc externalCommand;
   NSDupFontNotifyProc dupFontNotify;
   NSMoreMemoryProc moreMemory;
   NSFreeMemoryProc freeMemory;
   NSCantHappenProc cantHappen;
   NSExitProcessProc exitProcess;
   NSCustomResourceDevInit customResourceDevInit; /*OBSOLETE */
   NSCustomRegisterFauxFontProc
      customRegisterFauxFontProc; /* OBSOLETE */
   NORMSearchList resourceSearchList;
   char * scratchFileDirectory;
   char * initialVMFile;
   char * atmFile;
   char * startupFile;
   char * licenseID;
   unsigned int serialnumber;
   unsigned int progressQuantum;
   int setpagedeviceKeysCount;
   int *setpagedeviceKeysList;
   unsigned int initialVMSize;
   char * productName;
   char * languageCode;
   normBool runningAsServer;
   char * versionString;
   int intinterfaceVersionNum;
   NORMSearchList hostfontSearchList;
```

```
   char *fileSecurityWorkingDir;
   NORMSearchList fileSecurityDirList;
   char * hostFontCacheDir;
   unsigned long hostFontCacheSize;
   NORMSearchList iccProfileDirList;
   unsigned int iccProfilesStandardFolders;
   NSExternalProcessCommentSetupProc externalProcessCommentSetup;
   NSExternalProcessCommentProc externalProcessComment;
   NSExternalProcessCommentCleanupProc externalProcessCommentCleanup;
   NSGetHostFontMutexProc getHostFontMutex;
   NSReleaseHostFontMutexProc releaseHostFontMutex;
   normBool ignoreStdTTFonts;
   long int ramDiskSize;
} NSClientConfig;
```

Description

The **NSClientConfig** structure contains configuration information that the client passes to Adobe PDF Converter SDK in the call to **NormalizerServerInit()**.

Header File

apcif.h

Members

clientData	*(Optional)* A pointer to client-provided private data. This pointer is passed back in all callbacks from Adobe PDF Converter SDK to the client. This structure is opaque to Adobe PDF Converter SDK.

NOTE: The next several fields after **clientData** and before **resourceSearchList** contain the client's callbacks, **NSStartPage()** through **NSFreeMemoryProc()**, which are listed in the structure definition. For details on these callbacks, see Chapter 13, “Functions and Callbacks”.

`resourceSearchList`	*(Required)* References a location containing the PostScript fonts directory. The directory name typically is of the form `fonts\`. You must set this field for backward compatibility with PostScript programs. Use `hostfontSearchList` (below) for the `fonts\` directory and all other font directories. If `resourceSearchList` is NULL, `NormalizerServerInit` returns with the `normParameterError` enumeration as its result. The referenced character string must be encoded using a standard eight bit character encoding and must be terminated with a platform-specific separator, such as "`/`" or "`\`". **NOTE:** The `fonts\` directory must contain at least one PostScript font entry. **NOTE:** It is recommend that you set `resourceSearchList` to the default `fonts` directory supplied with Adobe PDF Converter SDK, and use `hostfontSearchList` to specify additional font locations. You should duplicate the default `fonts` folder in `hostfontSearchList`.
`scratchFileDirectory`	(*Required*) References a location containing the directory name where Adobe PDF Converter SDK can store temporary files. `scratchFileDirectory` must not be NULL. **NOTE:** The referenced character string must be encoded using standard characters and must be terminated with a platform-specific separator, such as "/" or "\".

`initialVMFile`	(*Required*) References a location containing the path name for the initial PostScript VM file. Adobe PDF Converter SDK uses that file to initialize virtual memory in the PostScript Interpreter. If `initialVMFile` is NULL or references a zero-length character string, `NormalizerServerInit()` returns with the `normParameterError` enumeration as its result. **NOTE:** The referenced character string must use standard characters.
`atmFile`	*(Optional)* References the client storage location of the path name for the Adobe Type Manager® (ATM) database file. Adobe PDF Converter SDK uses the information contained in this file to synthesize missing fonts. **NOTE:** The referenced character string must be encoded using the standard character format rather than the UNICODE one.
`startupFile`	*(Optional)* References the pathname for the startup PostScript file, which Adobe PDF Converter SDK uses to initialize the PostScript Interpreter. If `startupFile` is NULL or references a zero-length string, Adobe PDF Converter SDK instead uses default values in place of the information provided in such a startup file. Unlike CPSI, Adobe PDF Converter SDK does not call the PostScript procedure **GetOEMProcedure** until AFTER running `startupnorm.ps`. As a result, use of **makeoperator** in your startup program will result in an error. To work around this limitation, do custom startup in a separate job after initialization. **NOTE:** The referenced character string must be encoded using the standard character format rather than the UNICODE one.

`licenseID`	*(Optional)* References the license ID, which is required for copy protection of CJK fonts. NOTE: The referenced character string must be encoded using the standard character format rather than the UNICODE one.
`serialnumber`	*(Optional)* The serial number used to access CJK fonts.
`progressQuantum`	*(Optional)* The approximate number of PostScript operations that Adobe PDF Converter SDK should process before calling the `NSProgress` callback. If this field is 0, the default value of 1000 is used.
`setpagedeviceKeysCount`	*(Optional)* The number of page device keys in the `setpagedeviceKeysList` array.
`setpagedeviceKeysList`	*(Optional)* A pointer to an array of integers that represent the page device keys that Adobe PDF Converter SDK should report to the client. The integers are the enumerated page device keys listed in the file `spdkeys.h`. NOTE: This list is defined in the `demoSpdKeys` array in the file `demomain.c`.
`initialVMSize`	*(Required)* The number of bytes of memory used for the initial VM state of the PostScript Interpreter. The reference system value of 8 megabytes should be acceptable in all products. Adobe PDF Converter SDK verifies that the value provided for `initialVMSize` is within the range of 1 to 64 megabytes. If it is not, `NormalizerServerInit()` returns with the `normParameterError` enumeration as its result.
`productName`	*(Optional)* References the character string that Adobe PDF Converter SDK writes into the **Producer** field of those PDF files produced by Adobe PDF Converter SDK. The maximum length of this string is 31 characters, a value which is set in the `MAX_PRODUCTNAME` definition. NOTE: The referenced character string must be encoded using the standard character set.

`languageCode`	References the language code, which is required for the ACE engine. • See `democonverter.c` for example code to set this. • If pointer is NULL, Adobe PDF Converter SDK will default to US English (`"ENG"`). • If provided, should be three characters long. Recommend setting this to NULL unless otherwise advised by Adobe.
`runningAsServer`	When TRUE, host based (ATM fonts) CID font support is disabled except for OpenType J fonts.
`versionString`	References a string containing the build version that is set by Adobe PDF Converter SDK.
`interfaceVersionNum`	An integer representing the interface version number. To ensure that anyone upgrading from an earlier version will get a clean error rather than a crash, the value should be set to: `NORMALIZER_INTERFACE_VERSION_NUMBER`. If the interface version is incompatible with the Adobe PDF Converter SDK code, the following error is returned at initialization: `normIncorrectInterfaceVersion (=8)`

`hostfontSearchList`	A list of font directories to be searched for host fonts. It can include the ATM font directory, the system font directory, the printer CMAP directory, and the printer fonts directory. (The directories must be in the form described in Section 5.3.2, "Specifying Pathnames in Command Lines" on page 39.) Adobe PDF Converter SDK uses the `matchingOnly` field of the `hostfontSearchList` struct, but not the `extension` field. If more than one instance of Adobe PDF Converter SDK is running on a machine, and all the instances are sharing a host font cache, then the `hostfontSearchList` for each Adobe PDF Converter SDK instance must be identical. In addition, `NormalizerNewHostFontList()` must not be used to change the `hostfontSearchList` values. **NOTES:** This list must contain at least one entry.
`fileSecurityWorkingDir`	Allows you to restrict the directories in which the PostScript Interpreter may search to resolve PostScript file operators. `fileSecurityDirList` is a linked-list of `NORMSearchList` structures that specify directories in the underlying filesystem (the %os% storage device). Set to NULL for no additional security. For more information in restricting the system access of PostScript files, see Chapter 11. **NOTE:** The referenced character string must be encoded using standard characters and must be terminated with a platform-specific separator, such as "/" or "\".

`fileSecurityDirList`	Allows you to specify the PostScript Interpreter's working directory. At the PostScript language level, filenames can be specified as fully qualified pathnames or as relative pathnames.If a filename is specified as relative, the PostScript Interpreter prepends the pathname of the working directory to the relative pathname. As a result, you can disallow relative pathnames by omitting a value for `fileSecurityWorkingDir`. **NOTE:** The referenced character string must be encoded using the standard character set and must be terminated with a platform-specific separator, such as "/" or "\". For more information in restricting the system access of PostScript files, see Chapter 11.
`hostFontCacheDir`	References a string containing the pathname of the host font cache file. **NOTE:** The referenced character string must be encoded using the standard character set. **NOTE:** Prior to starting Adobe PDF Converter SDK, Mac implementations must ensure the directory specified by `hostFontCacheDir` exists.
`hostFontCacheSize`	The maximum size in bytes of the host font cache. **NOTE:** The greater of `hostFontCacheSize` and 1,536,000 bytes, the minimum size, will be used.
`jdfInitStruct`	PDF Converter SDK does not support JDF. Hence, set the value of this member to NULL.

`iccProfileDirList`	A custom list of directories to be searched for ICC profiles. Support for `iccProfileDirList` and `iccProfilesStandardFolders` is platform specific, as described below: • On Windows, Adobe PDF Converter SDK supports *both* `iccProfileDirList` and `iccProfilesStandardFolders`, with the following exception: if `iccProfilesStandardFolders` is set to `ICCPROFILES_USE_ADOBE_STANDARD_ONLY`, the standard ACE folders are used and `iccProfileDirList` is ignored. • On Linux Adobe PDF Converter SDK supports *only* `iccProfileDirList`. Using custom ICC profile folder locations allows you to use profiles without needing to install them in the system default ICC profile directories. **REQUIREMENTS**: To prevent the Adobe PDF Converter SDK initialization error `normParameterError`, you must do the following: • Specify at least one ICC profile folder in `iccProfileDirList` or at least one profile directory type (using `iccProfilesStandardFolders`). • The default ICC profiles shipped with Adobe PDF Converter SDK must be included in the specified folders. See also "Requirement for "iccprofiles" Folder" on page 84.

`iccProfilesStandardFolders` (Windows only)	Allows you to specify predefined flags that direct Adobe PDF Converter SDK to certain ICC profile locations. `iccProfilesStandardFolders` may represent a combination of the locations, which you specify by OR'ing together the types of profiles to use (described below): • `ICCPROFILES_USE_ADOBE_STANDARD_ONLY` references the standard Adobe ACE folders. • `ICCPROFILES_ADOBE_COLOR_RECOMMENDED` references the Adobe recommended color profiles folder. (Windows only) • `ICCPROFILES_ADOBE_COLOR` references the Adobe color profiles folder. (Windows only) • `ICCPROFILES_SYSTEM_COLOR` references the system color profiles folder. (Windows only) • `0` (zero). Directs Adobe PDF Converter SDK to use ONLY `iccProfileDirList`. If multiple locations are specified, the order in which they are searched is as follows: • If `ICCPROFILES_USE_ADOBE_STANDARD_ONLY` is one of the locations, ACE determines the order. • Otherwise, the order is as follows: 1. Folders specified in `iccProfileDirList` 2. `ICCPROFILES_ADOBE_COLOR_RECOMMENDED` 3. `ICCPROFILES_ADOBE_COLOR` 4. `ICCPROFILES_SYSTEM_COLOR` **NOTE:** See additional notes in the description for `iccProfileDirList`. (above)
`externalProcessCommentSetup`	This field contains the client's callback `NSExternalProcessCommentSetupProc()`, which is listed in the structure definition. For details on this callback, see Chapter 13, "Functions and Callbacks".
`externalProcessComment`	This field contains the client's callback `NSExternalProcessCommentProc()`, which is listed in the structure definition. For details on this callback, see Chapter 13, "Functions and Callbacks".

`externalProcessCommentCleanup`	This field contains the `NSExternalProcessCommentCleanupProc()` client callback, which is listed in the structure definition. For details on this callback, see Chapter 13, "Functions and Callbacks".
`getHostFontMutex`	A handle that acquires a global mutex and then effectively locks the host font cache to prevent more than one instance of Adobe PDF Converter SDK from simultaneously writing to the font cache.
`releaseHostFontMutex`	Releases the global mutex acquired by `getHostFontMutex` and unlocks the host font cache for other instances of Adobe PDF Converter SDK.
`ignoreStdTTFonts`	If TRUE, Adobe PDF Converter SDK ignores the TrueType versions of certain standard PostScript fonts (Appendix A). This setting can also be changed by `NormalizerNewHostFontList()`. `ignoreStdTTFonts` corresponds to the **Ignore TrueType versions of standard PostScript fonts** button of the **Acrobat Distiller-Font Locations** dialog. In earlier versions of Adobe PDF Converter SDK, `ignoreStdTTFonts` was always TRUE. If the destination printing system has standard PostScript fonts installed, setting this field to TRUE causes Adobe PDF Converter SDK to use the same font that will be used by the printer.
`ramDiskSize`	It is used to set maximum size (in bytes) of the ramDisk (RAM). APC creates multiple temporary files for storing PDF object data. If size of temporary file is less than ramDiskSize then such files will be stored in memory instead of writing to file. **Note**: ramDiskSize parameter does not restricts ram usage. This only restricts size of an individual temporary files. If ramDiskSize is given less than zero, the size reverts to the default value, which is 20 MB.

NSClientDataPtr

```
typedef void *NSClientDataPtr;
```

Description

`NSClientDataPtr` is a pointer to private client data. This pointer is passed back in all callback functions from Adobe PDF Converter SDK to the client. It is defined as `void` because it is opaque to the server.

Header File

apcif.h

NSClientFileID

```
typedef union {
    void *ptr;
    int index;
} NSClientFileID;
```

Description

NSClientFileID is a union of an pointer and an integer. Adobe PDF Converter SDK's clients use this file ID along with a set of **NSClientFile** API callbacks to manage the I/O on files used by PDF Converter SDK.

Header File

apcif.h

NSClientFile

NSClientFileRec

```
typedef struct _t_NSClientFile{
    NSClientFileID fd;
    NSClientFileProcs procs;
} NSClientFileRec, *NSClientFile;
```

Description

A structure that associates a file ID with the callbacks Adobe PDF Converter SDK uses to create/read/write that file. See also Chapter 14, “NSClientFile API”.

Header File

`apcif.h`

Members

`fd`	The file ID.
`procs`	The address of a `NSClientFileProcs` structure that defines callbacks used with the `NSClientFile` API.

NSClientFileProcs

```
typedef struct _t_NSClientFileProcs{
    NSReadProc ReadProc;
    NSWriteProc WriteProc;
    NSSeekProc SeekProc;
    NSCloseProc CloseProc;
    NSTruncateProc TruncateProc;
    NSBufsizeProc BufsizeProc;
} NSClientFileProcsRec, *NSClientFileProcs;
```

Description

A structure, which defines a set of `NSClientFile` API callbacks.

Header File

`apcif.h`

NSMemoryAllocInfo

```
typedef struct _t_NSMemoryAllocInfo {
normBool CanAllocateExtraMemory;
normBool WillMemoryBeFreed;
NSClientDataPtr pClientDataPtr;
} NSMemoryAllocInfo, *NSMemoryAllocInfoPtr;
```

Description

A structure, containing information about memory location.

Header File

apcif.h

Members

CanAllocateExtraMemory	Can allocate the more memory than requested. Only VM can use more memory than requested other ignores the extra memory.
WillMemoryBeFreed	Will memory be freed at the closing of application. VM frees the memory at the end of application because VM reuses that memory again and again. While other free after the work related to that memory is done.
pClientDataPtr	ClientData pointer set by the product.

NSMemoryFreeInfo

```
typedef struct _t_NSMemoryFreeInfo {
NSClientDataPtr pClientDataPtr;
} NSMemoryFreeInfo, *NSMemoryFreeInfoPtr;
```

Description

A structure, containing information about memory freeing.

Header File

apcif.h

Members

pClientDataPtr	ClientData pointer set by the product.

NSJobParams

```
typedef struct _t_NSJobParams
    char * fullDocFileName;
    unsigned int
       fontAllowMM:1,
       filePerPage:1,
       sidelineEPS:1;
       fontEmbedJobsFonts:1,
       CreateRefXObj:1, /*Create Reference XObjects from PS
       Forms rather than embedding those XObjects in the host
       PDF file*/
       fulldocfileCreation:1,  /*Allow creation of full
       document pdf file. */
       disableAutoT1Embed:1,
       TextPositionOptimization:1; /*Switch on the text
       position optimization for smaller pdf file size*/
    NSClientFile fullDocClientFile;
    char * jobOptions;
    NSBufferGetPS  bufGetJobOptions;
    NSRunMethod  runMethod;
    char * outputResourceFinalStatus;
    long maxdistilltime;
} NSJobParams;
```

Description

The **NSJobParams** structure contains job parameters that the client provides to Adobe PDF Converter SDK for each PostScript file.

Adobe PDF Converter SDK obtains default values for fields like page size, resolution, and several font-related parameters from job options, although the page size and resolution parameters are frequently over-ridden by the PostScript stream/file.

Header File

apcif.h

Members

`fullDocFileName`	Pointer to a string that specifies the pathname for the full-document PDF file created by Adobe PDF Converter SDK. Used only if `runMethod ==normRunJobOptions`. This string must be encoded using the standard character format and may contain up to 1,023 characters. Adobe PDF Converter SDK uses its default file I/O methods to create the file. If NULL, Adobe PDF Converter SDK uses the `NSClientFile` API provided by `fullDocClientFile` to output the full-document PDF file. If both `fullDocFileName` and `fullDocClientFile` are provided, `fullDocFileName` is used. NOTE: You should use the `NSClientFile` API provided by `fullDocClientFile` to output the full-document PDF file.
`fontAllowMM:1`	A (1 bit) font policy flag: • If TRUE, Adobe PDF Converter SDK attempts to synthesize missing fonts using metric information from the ATM font database. • If FALSE, Adobe PDF Converter SDK attempts to replace any missing fonts with the font specified in the `fontDefaultName` field (described below), or, if not provided, with Courier.
`filePerPage:1`	A 1-bit flag: If TRUE, Adobe PDF Converter SDK produces PDF page streams, provided the client has not disabled distillation.
`sidelineImages:1`	A 1-bit flag: If TRUE, Adobe PDF Converter SDK stores embedded or external images in external files.
`sidelineEPS:1`	A 1-bit flag: If TRUE, Adobe PDF Converter SDK stores embedded EPS files in external files without distilling them.

`fontEmbedJobsFonts:1`	A 1-bit flag. If TRUE, Adobe PDF Converter SDK embeds in the PDF file fonts that are embedded in the PostScript. If FALSE, Adobe PDF Converter SDK embeds such fonts ONLY if the Distiller parameters dictate embedding. Regarding fonts that are referenced from but not embedded in the PostScript, Adobe PDF Converter SDK embeds such fonts ONLY if the Distiller parameters dictate embedding.
`CreateRefXObj:1`	A 1-bit flag. If TRUE, Adobe PDF Converter SDK creates reference Xobjects from PS Forms rather than embedding those Xobjects in the host PDF file. This flag is applied only for PDF versions later than 1.4.
`fulldocfileCreation:1`	A 1-bit flag. If FALSE, Adobe PDF Converter SDK does not create a full document PDF.
`disableAutoT1Embed:1`	A 1-bit flag. If TRUE, Adobe PDF Converter doesn't auto embed Type 1 fonts whose **CharStrings** dict length > 229
`TextPositionOptimization:1`	A 1-bit flag. If FALSE, Adobe PDF Converter SDK does not apply text position optimization.
`fullDocClientFile`	A pointer to a structure of callbacks Adobe PDF Converter SDK can use to output the full-document PDF file, as described for `fullDocFileName` in this structure. `fullDocClientFile` points to a `NSClientFileProcs` structure that defines callbacks used with the `NSClientFile` API. You may implement this API to support either standard or UNICODE characters. The `NSClientFile` API does not provide a function for opening a file, so the client must open the file for full-document PDF object, before calling `NormalizerServerRunJob()`.
`jdfFileName`	PDF Converter SDK does not support JDF. Hence, set the value of this member to NULL.
`jdfClientFile`	PDF Converter SDK does not support JDF. Hence, set the value of this member to NULL.

`jdfInputPSFileName`	PDF Converter SDK does not support JDF. Hence, set the value of this member to NULL.
`jobOptions`	Used only when `runMethod==normRunJobOptions`. `jobOptions` is a pointer to a buffer containing a PostScript segment, which you can use to initialize job options. If non-null, the segment is processed before the job is run. If NULL, Adobe PDF Converter SDK invokes the callback provided in `bufGetJobOptions`. IMPORTANT: The PostScript code provided by `jobOptions` *MUST* call **setdistillerparams** *AND* **setpagedevice** (and in that order) or Adobe PDF Converter SDK will produce an error on initialising. See `democonverter.c` for an example of appropriate PostScript code. If both `jobOptions` and `bufGetJobOptions` are NULL then the minimal PostScript snippet: **<< >> setdistillerparams << >> setpagedevice** is executed. If both `jobOptions` and `bufGetJobOptions` are given, `bufGetJobOptions` is used.
`bufGetJobOptions`	See `jobOptions`. See `NSBufferGetPS()` for details on using the `bufGetJobOptions` callback.
`runMethod`	Possible value is `normRunJobOptions`. If `runMethod == normRunJobOptions`, the contents of the PostScript segment described by `jobOptions` or `bufGetJobOptions` are used to control the job. Since PDF Converter SDK does not support JDF, always set the value of `runMethod` to `normRunJobOptions`.
`outputResourceFinalStatus`	A string representing the value assigned to Output resources, a resource definition process. If set to NULL the status of each Output resource (if any) will be set to “Incomplete”. The memory holding the string must not be freed until the job is finished.

`maxDistillTime` `(For Unix platforms only)`	Represents the maximum number of seconds a job is allowed to run. If this job parameter is set to a positive value, then Adobe PDF Converter SDK overrides the `/JobTimeout` parameter in the PS file. If the time taken by a job exceeds the value set for the `maxDistillTime` parameter, then Adobe PDF Converter SDK aborts the current job, and stops without executing the remaining jobs.

NSFileDataPtr

```
typedef void *NSClientDataPtr;
```

Description

`NSFileDataPtr` is a pointer to data associated with a file. A pointer of this type is passed back to all callback functions associated with file I/O. Like `NSClientDataPtr`, it is opaque to Adobe PDF Converter SDK.

Header File

`apcif.h`

NSPageInfo

```
typedef struct _t_NSPageInfo
    char *plateColor;
    char *pageLabel;
} NSPageInfo;
```

Description

The `NSPageInfo` structure contains information about a page, passed by the `NSEndPage()` callback.

Header File

`apcif.h`

Members

`plateColor`	A pointer to a string. If non-null, the string is the name of plate color claimed by the PS job.
`pageLabel`	A pointer to a string. If non-null, string is the page label specified by PS job.
`JDFHandle`	PDF Converter SDK does not support JDF. Hence, set the value of this member to NULL.

NSServerDataPtr

```
void *NSServerDataPtr;
```

Description

The **NSServerDataPtr** is an opaque pointer to Adobe PDF Converter SDK's private data. It is returned to the client by NormalizerServerInit(). The client passes it into all Adobe PDF Converter SDK functions.

PDFX4Setting

```
typedef struct _t_PDFX4Setting {
unsigned short autoCorrectOPM : 1
unsigned short useOneTransferFunctionPerColorant : 1;
unsigned short unused: 14;
} PDFX4Setting;
```

Description

A structure containing PDFX-4 Setting.

Header File

`apcif.h`

Members

`autoCorrectOPM`	As per PDF/X-4 compliance Section 6.4.3.3 4 color ICC based color space shall have overprint mode set to zero If value of this key is 1 and current job is generating PDF/X-4 compliance PDF than APC will set OPM key to false for such color space. Otherwise APC will generate violation. (default value is 0).
`useOneTransferFunctionPerColorant`	As per PDF/X-4 compliance section 6.4.3.4 each separation array in a single PDF/X-4 file that have same name shall have same tint transform. If value of this key is true than APC will use first tint transform function for same separation color, Otherwise, if there are multiple tint transform function for a separation color name than APC will generate violation. (default value is 0).
`unused`	For future reference.

PNSHostFontListData

NSHostFontListData

```
typedef struct _t_NSHostFontListData {
    unsigned char *signature;
    char *fontName;
    char *pathName;
    normBool widthsOnly;
    normBool isHexName;
    normBool hasProtection;
    unsigned short int fsType;
    unsigned short int resolution;
    normBool isBound;
    normBool bindingOK;
    normBool periodic;
    normBool outlineOK;
    normBool toCache;
    normBool isTrueType;
    normBool inFontCache;
} NSHostFontListData, *PNSHostFontListData;
```

Description

The **NSHostFontListData** structure describes the characteristics of a host font list parameter to the NSDupFontNotifyProc() callback.

Header File

apcif.h

Members

signature	MD5 hash value for the font.
fontName	A pointer to the name string of the font. Example of such font name strings are: /Name Resource/CIDFont/Name Resource/CMap/Name
pathName	A host-dependent pathname to an external file.
widthsOnly	TRUE if the font is the “width-only” font.

`isHexName`	TRUE if the font name is artificially generated using a hexadecimal representation of checksums
`hasProtection`	TRUE if the fields following are valid (the font has protection information)
`fsType`	Flag that indicates whether the font is embeddable. The interpretation of each flag bit is same as FSType field in the TrueType OS/2 table.
`resolution`	Highest resolution (in dpi, inclusive) the font is allowed to be used.
`isBound`	Flag that indicates whether the font is bound to platform. The value is cached by the hostfontstodev package.
`bindingOK`	Flag that indicates whether the platform binding is validated. The %hostfont% will return NULL stream if the `isBound` is true and this flag is FALSE.
`periodic`	Flag that indicates whether the platform binding needs to be verified periodically, most typically with a Dongle. If this flag is TRUE, the platform binding is verified every 30 minutes.
`outlineOK`	Flag that indicates accessibility of glyph outline.
`toCache`	If FALSE, this font will not be cached to HostFontCache. The client can set this field to false in order to screen out the fonts that meet a certain condition from %hostfont% stodev.

SPDKeyValue

```
typedef struct _t_SPDKeyValue {
    char *keyName; /* Optional string representation of key
name */
    int key; /* Dictionary key: SPDKey */
    int type; /* Actually SPDValueType for value */
    union {
       long i;
       float f;
       char *s;
       struct { float x; float y; } p; /* ordered pair */
       struct _t_SPDKeyValue *l; /* Key/value list for */
                                 /* array or dict value */
    } value;
    struct _t_SPDKeyValue *next; /* Pointer to next item */
                                 /* in linked list */
} SPDKeyValue;
```

Description

`SPDKeyValue` is the structure used to represent key-value pairs internally.

Header File

`spdkeys.h`

```
typedef struct _t_ImageJobConfig {
    unsigned int uncompressImage;
} ImageJobConfig, *ImageJobConfigP;
```

Description

`ImageJobConfig` is the structure used to configure the image to pdf conversion

Header File

`apcif.h`

Members

`uncompressImage`

If 0, then all images in generated PDF will contain compression.

if 1. then all images will be added to the PDF without any compression.

Users may choose not to compress images to reduce conversion time. However, when uncompressimage =1, size of output PDF will be larger.

A Standard TrueType Fonts

Table A.1 lists the TrueType fonts Adobe PDF Converter SDK can use in place of PostScript fonts, provided the `ignoreStdTTFonts` field of `NSClientConfig` struct is FALSE.

TABLE A.1 Standard TrueType fonts considered by ignoreStdTTFonts

Font Name	PostScript font name
Albertus®	AlbertusMT AlbertusMT-Italic AlbertusMT-Light
Antique Olive®	AntiqueOlive-Bold AntiqueOlive-Compact AntiqueOlive-Italic AntiqueOlive-Roman
Arial®	Arial-BoldItalicMT Arial-BoldMT Arial-ItalicMT ArialMT
ITC Avant Garde Gothic®	AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique
Bauer Bodoni™	Bodoni Bodoni-Bold Bodoni-BoldItalic Bodoni-Italic Bodoni-Poster Bodoni-PosterCompressed
ITC Bookman®	Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic

TABLE A.1 Standard TrueType fonts considered by ignoreStdTTFonts (Continued)

Font Name	PostScript font name
Clarendon* (See attribution statement on page ii.)	Clarendon Clarendon-Bold Clarendon-Light
Cooper Black	CooperBlack CooperBlack-Italic
Copperplate Gothic	Copperplate-ThirtyThreeBC Copperplate-ThirtyTwoBC
Courier	Courier Courier-Bold Courier-BoldOblique Courier-Oblique
Eurostile™	Eurostile Eurostile-Bold Eurostile-BoldExtendedTwo Eurostile-ExtendedTwo
Gill Sans®	GillSans GillSans-Bold GillSans-BoldCondensed GillSans-BoldItalic GillSans-Condensed GillSans-ExtraBold GillSans-Italic GillSans-Light GillSans-LightItalic
Goudy	Goudy Goudy-Bold Goudy-BoldItalic Goudy-ExtraBold Goudy-Italic

TABLE A.1 Standard TrueType fonts considered by ignoreStdTTFonts (Continued)

Font Name	PostScript font name
Helvetica* (See attribution statement on page ii.)	Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Condensed Helvetica-Condensed-Bold Helvetica-Condensed-BoldObl Helvetica-Condensed-Oblique Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique Helvetica-Narrow-Oblique Helvetica-Oblique
Joanna®	JoannaMT JoannaMT-Bold JoannaMT-BoldItalic JoannaMT-Italic
Letter Gothic	LetterGothic LetterGothic-Bold LetterGothic-BoldSlanted LetterGothic-Slanted
ITC Lubalin Graph®	LubalinGraph-Book LubalinGraph-BookOblique LubalinGraph-Demi LubalinGraph-DemiOblique
Marigold™	Marigold
ITC Mona Lisa®	MonaLisa-Recut
New Century Schoolbook	NewCenturySchlbk-Bold NewCenturySchlbk-BoldItalic NewCenturySchlbk-Italic NewCenturySchlbk-Roman

TABLE A.1 Standard TrueType fonts considered by ignoreStdTTFonts (Continued)

Font Name	PostScript font name
Optima* (See attribution statement on page ii.)	Optima Optima-Bold Optima-BoldItalic Optima-Italic
Oxford™	Oxford
Palatino* (See attribution statement on page ii.)	Palatino-Bold Palatino-BoldItalic Palatino-Italic Palatino-Roman
Stempel Garamond* (See attribution statement on page ii.)	StempelGaramond-Bold StempelGaramond-BoldItalic StempelGaramond-Italic StempelGaramond-Roman
Tekton®	Tekton
Times* (See attribution statement on page ii.)	Times-Bold Times-BoldItalic Times-Italic Times-Roman
Times New Roman®	TimesNewRomanPS-BoldItalicMT TimesNewRomanPS-BoldMT TimesNewRomanPS-ItalicMT TimesNewRomanPSMT

TABLE A.1 Standard TrueType fonts considered by ignoreStdTTFonts (Continued)

Font Name	PostScript font name
Univers (See attribution statement on page ii.)	Univers Univers-Bold Univers-BoldExt Univers-BoldExtObl Univers-BoldOblique Univers-Condensed Univers-CondensedBold Univers-CondensedBoldOblique Univers-CondensedOblique Univers-Extended Univers-ExtendedObl Univers-Light Univers-LightOblique Univers-Oblique
ITC Zapf Dingbats®	ZapfChancery-MediumItalic

B Apache Software License, Version 1.1

```
/*
*This product includes software developed by the Apache Software Foundation
*(http://www.apache.org/).
*
*The Apache Software License, Version 1.1
*
*Portions Copyright (c) 1998-2000, 1999-2004, 1999 - 2000, 2000 - 2003 The Apache
*Software Foundation.  All rights reserved.
*
*Redistribution and use in source and binary forms, with or without modification, are
*permitted provided that the following conditions are met:
*
*1. Redistributions of source code must retain the above copyright notice, this list
*of conditions and the following disclaimer.
*
*2. Redistributions in binary form must reproduce the above copyright notice, this
*list of conditions and the following disclaimer in the documentation and/or other
*materials provided with the distribution.
*
*3. The end-user documentation included with the redistribution, if any, must include
*the following acknowledgment:
*"This product includes software developed by the Apache Software Foundation
*(http://www.apache.org/)."
*Alternately, this acknowledgment may appear in the software itself, if and wherever
*such third-party acknowledgments normally appear.
*
*4. The names "Xerces" and "Apache Software Foundation" must not be used to endorse
*or promote products derived from this software without prior written permission. For
*written  permission, please contact apache@apache.org.
*
*5. Products derived from this software may not be called "Apache", nor may "Apache"
*appear in their name, without prior written permission of the Apache Software
*Foundation.
```

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P