



Adobe PDF Library Overview

Technical Note #5189

Version: PDF Library 6.0.1, 6.1.1



ADOBE SYSTEMS INCORPORATED

Corporate Headquarters

345 Park Avenue

San Jose, CA 95110-2704

(408) 536-6000

<http://partners.adobe.com>

April 2004

Copyright 2004 Adobe Systems Incorporated. All rights reserved. Adobe® PDF Library Overview.

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

PostScript is a registered trademark of Adobe Systems Incorporated. All instances of the name PostScript in the text are references to the PostScript language as defined by Adobe Systems Incorporated unless otherwise stated. The name PostScript also is used as a product trademark for Adobe Systems' implementation of the PostScript language interpreter.

Except as otherwise stated, any reference to a "PostScript printing device," "PostScript display device," or similar item refers to a printing device, display device or item (respectively) that contains PostScript technology created or licensed by Adobe Systems Incorporated and not to devices or items that purport to be merely compatible with the PostScript language.

Adobe, the Adobe logo, Acrobat, Acrobat Capture, the Adobe PDF logo, Distiller, PostScript, the PostScript logo and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Apple, Macintosh, and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. PowerPC is a registered trademark of IBM Corporation in the United States. ActiveX, Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. UNIX is a registered trademark of The Open Group. All other trademarks are the property of their respective owners.

This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and noninfringement of third party rights.

Portions include software under the following terms:

Portions derived from the RSA DataSecurity, Inc. MD5 Message-Digest Algorithm.

Portions of the software were developed by the University of California, Berkeley.

The author of this software is David M. Gay. Copyright (c) 1991 by AT&T. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS"; WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR AT&T MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

Portions of the software were developed at Cygnus Solutions.

Portions copyright (c) 1995-2003 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

Portions copyright (c) 1994 Hewlett-Packard Company. Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Copyright (c) 1996 Silicon Graphics Computer Systems, Inc. Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Contains an implementation of the LZW algorithm licensed under U.S. Patent 4,558,302.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA. Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

1

Introduction

About This Technical Note

This technical note provides an introduction to development using the Adobe PDF Library. It describes the Adobe PDF Library, the Adobe PDF Library Software Development Kit (SDK), support and licensing options, and basic development tasks. Developers with questions on any of these topics or those new to development with the Adobe PDF Library are encouraged to read this document.

NOTE: The PDF Library 6.0.1/6.1.1 version of this document covers two different versions of the PDF Library. The main difference between 6.0.1 and 6.1.1 is that the latter version includes the ability to safely operate in a multithreaded environment. For more information, see [Chapter 4, "Using the PDF Library 6.1.1 SDK"](#).

This document is intended for programmers who need basic information on the Adobe PDF Library and its capabilities. It assumes familiarity with C programming, common development tasks, and the use of methods exported by an object code library.

What Is the Adobe PDF Library?

Designed specifically for OEMs, ISVs, system integrators, and enterprise IT developers, the Adobe PDF Library SDK contains a powerful set of functions for developing third-party solutions and workflows around the Adobe Portable Document Format (PDF). The Adobe PDF Library is based on the core technology of the Adobe Acrobat line of products and offers complete functionality for generating, manipulating, rendering, and printing Adobe PDF documents.

The library enables Adobe PDF functionality to be seamlessly embedded within applications. It also provides reliable, accurate and Adobe-supported implementation of the latest PDF specification.

The Adobe PDF Library and the Adobe Acrobat Viewer

The Adobe PDF Library is compiled from the same source code as the Adobe PDF viewing applications (Adobe Acrobat or Adobe Reader), with some additions. There is a large degree of overlap between the functionality provided by the PDF Library Software Development Kit (SDK) and that of the Acrobat SDK. They differ in providing access to the Acrobat user interface:

- The Acrobat SDK is meant for the plug-in environment, and allows you to control and interact with the Acrobat user interface.

- The PDF Library SDK is intended for interaction between PDF and other applications, such as high volume batch processing and PDF generation applications. It does *not* export methods for creating or managing Acrobat UI elements—that is, the AcroView (AV) layer of the core API.

If you are interested in the documentation and samples in the Acrobat Software Development Kit (SDK), see:

<http://partners.adobe.com/asn/developer/acrosdk/main.html>

New in the Version 6 Release

The version 6 release of the Adobe PDF Library offers support for PDF 1.5 features. It includes API methods for creating PDF files that take advantage of new PDF 1.5 features. For specific details of the PDF 1.5 File Specification, see the *PDF Reference*. The PDF Library supports the same feature set as Acrobat 6.0, except for those features directly related to the user interface (that is, the AV layer and plug-in support).

PDF Library 6.0 supports these new features:

- A revamped Cos object system that reduces the size of tagged PDF files, provides faster access to objects, and uses 32-bit addressing throughout. The new Cos object system allows compressed object and Xref streams, implemented through object collections.
- Optional content, which allows PDF documents to be organized into layers whose visibility is controlled individually.
- The JPEG2000 image data compression and packaging standard, including the JPX file format, and progressive drawing of images. A new image filter, JPXDecode, is included.
- A digital prepress facility that allows public access to color management features, detection of transparency and control of the degree of rasterization in composite and separations printing, generation of color separations, and font embedding policy support.
- An improved WordFinder and PDWord utility, which allows creation of customized WordFinders for which you can specify an algorithm, Unicode output intent, and other characteristics to assist with text extraction.

The 6.0.1 release of the SDK fixes bugs and adds the following new snippets and samples:

- **ACEEnumProfilesSnip**
- **ACEEnumSettingsSnip**
- **ACEGetWorkingSpaceSnip**
- **ACETransPDETextColorSnip**
- **AddTagSnip**
- **ASFileIteratorSnip**
- **ClassMapSnip**
- **CreateDocStructSnip**
- **ExploreStructSnip**

- **ImageInfoSnip**
- **ObjShiftSnip**
- **PDEPathDrawCurveSnip**
- **PDEPathDrawExplorerSnip**
- **PDEPathDrawLineSnip**
- **PDEPathDrawRectSnip**
- **PDOCCToggleIntentSnip**
- **PDPageSetTransparencySnip**
- **RoleMapSnip**

HideContentSnip was removed from the SDK. The new snippet **PDPageSetTransparencySnip** is a modified version of **HideContentSnip**.

The 6.1 release of the SDK includes all of the 6.0.1 updates, supports the ability to perform thread-safe operations with PDFL, and adds the following new samples:

- **MTInMemFS**
- **MTSerialNums**
- **MTTextExtract**

The 6.1.1 release of the SDK includes:

- Support for two additional platforms/compilers: Macintosh OS X/CodeWarrior 9.1, and IBM AIX 5.1/xlC 6.0.
- Macintosh samples have all been converted into app packages.
- The Macintosh Mach-O runtime architecture is used. The CFM-based runtime architecture is no longer supported.

Licensing

If you wish to license the Adobe PDF Library, or have any questions regarding the ASN Developer program, please call (800) 685-3510, (206) 675-6145 or email asndeveloper@adobe.com

For licensing information, see <http://partners.adobe.com/asn/tech/pdf/pdflibrary.jsp>.

2

About the PDF Library SDK

This chapter helps you get started with development using the Adobe PDF Library Software Developers Kit (SDK). It describes the contents of each directory in the SDK installation, lists available code samples, and provides platform-specific information on how to set up the development environment.

Supported Environments

The Adobe PDF Library is supported for the platforms, operating systems and compilers listed in the tables below. There are two separate tables: one for PDF Library 6.0.1, and one for PDF Library 6.1.1. The supported compilers and integrated development environments (IDEs) are as follows:

TABLE 2.1 *Supported Environments for PDF Library 6.0.1*

Platform	Operating System	Compiler
Macintosh PowerPC	Mac OS X 10.2.5 or higher	CodeWarrior Pro 8.3 (CFM)
Windows Intel 32-bit	Windows 98/NT/2000 /XP	Microsoft Visual C++ 6.0 with service pack 5.
SunOS 32-bit	Solaris 2.6 or higher	gcc 2.95.2
IBM AIX	4.3.3	gcc 2.95.2 (Red Hat GnuPro)
Linux	Linux Red Hat 6.2 (kernel version 2.2.14)	gcc 2.95.2

TABLE 2.2 *Supported Environments for PDF Library 6.1.1*

Platform	Operating System	Compiler
Macintosh PowerPC	Mac OS X 10.3 or higher	CodeWarrior Pro 9.1 (Mach-O)
Windows Intel 32-bit	Windows 2000 /XP	Microsoft Visual C++ 6.0 with service pack 5.
SunOS 32-bit	Solaris 2.8 or higher	gcc 3.2

TABLE 2.2 Supported Environments for PDF Library 6.1.1

Platform	Operating System	Compiler
IBM AIX	5.1	xlC 6.0
Linux	Linux Red Hat 7.2	gcc 3.2

For Mac OS and Windows, later versions of the same development environment may be supported after they have been tested. For PDFL 6.0.1 on UNIX, gcc 2.95.2 is the only compiler supported.

These specific development environments are supported because the **struct** alignment and calling conventions are the same as those used in the development of the Adobe PDF Library. While it may be possible to use the library in other development environments, such use is not supported by Adobe Developer Support. The project files for the sample applications are created and supported only in the listed compiler versions.

Contents of the Software Development Kit

The Adobe PDF Library 6.x SDKs consist of:

- Core libraries that provide the functionality.
- Header files that provide access to the libraries.
- Fonts used in the library's basic operations.
- Sample applications showing how to use the library for a variety of purposes.
- Documentation of installation and development techniques and of the PDFL-specific API.

Libraries, Headers, and Fonts

The following components are shipped with the PDFL 6.x SDKs:

- The Acrobat PDF Library
 - This is a dynamic link library on the Windows platform and a shared object library on UNIX and Mac OS. In Windows, an interface library must be included in your MSVC project. The file names of these libraries are:
 - PDFL60.LIB — The interface library for the Windows PDF Library DLL.
 - PDFL60.DLL — The Windows PDF Library DLL.
 - libpdf1.so — The shared object library for most UNIX platforms.
 - PDFL60.lib — The shared object library for Mac OS.
- Helper Libraries
 - These libraries are used by the PDF Library internally. Developers are not allowed to call any method directly from these libraries.

- AGM — Used for rasterization.
 - CoolType — Used for font handling.
 - BIB, BIBUtils — Used for interfacing with other internal helper libraries.
 - ACE — Used for color management.
 - OPP — Used for overprint preview.
 - JP2K — Support for JPEG 2000.
 - ARE — Support for Adobe Raster Express. (PDFL 6.1.1 only)
 - XMP — Support for XMP. (PDFL 6.1.1 only)
- Headers and Fonts

The `include` directory of the SDK contains the required headers for accessing the API methods.

Sample Code

Samples are provided for Mac OS, UNIX, and Windows platforms in two forms: standalone sample programs, and the PDFLSnippetRunner, an environment and infrastructure for code snippets that illustrate specific functions or techniques.

Sample code is intended to demonstrate the use of the PDFL API, and is not necessarily robust enough for a final implementation. The sample code itself is platform-independent, as is the majority of the PDFL API; the only difference between the sample source code for different platforms is the line-endings.

For PDFL 6.1.1, the Macintosh samples are provided as application packages. This format is normal for double-clickable applications, but they can also be run from the command line. To run them from the command line, you can either specify the command line arguments in the CodeWarrior project file and execute within the IDE, or you can target the actual executable, which is in the `Contents/MacOS` folder inside the package. For example, from the Terminal window:

```
$ PDFLSnippetRunner.app/Contents/MacOS/SnippetRunner
```

NOTE: Since some of the PDFL 6.1.1 “MT” samples (see [“PDFL 6.1.x-specific Samples” on page 25](#)) require command line arguments (a default set is added to the project files), execution from within the IDE is preferred. Furthermore, SnippetRunner should be run from the Terminal command line since it uses the Terminal window; running SnippetRunner from the IDE or double-clicking proves of limited value.

Standalone Samples

The standalone samples include the following (PDFL 6.1.1-specific samples are described in [Chapter 4, “Using the PDF Library 6.1.1 SDK”](#)):

TABLE 2.3 **PDFL Samples**

Sample	Description
--------	-------------

TABLE 2.3 *PDFL Samples*

addelem	Shows how to modify existing pages in a PDF file. It adds a footer to each page and shifts the first line of each text run.
all	Used to compile all samples at the same time. (Not supported on Macintosh.)
Decryption	Shows how to programmatically decrypt a PDF document encrypted with Acrobat Standard Security options.
drawtomemory	Shows how to render a page to memory using the PDPageDrawContentsToMemory PDFL method, and creates a PDF file with a bitmap image rendered on the page.
fontembd	Shows font enumeration and font embedding.
helowrld	Shows the basics of creating a PDF document.
mergepdf	Shows how to merge two PDF files.
Peddler	Shows how to add hyperlinking (specifically targetting URIs) capabilities to an existing PDF document.
printpdf	Shows how to print a PDF file to a printer or to a file using the PDFL method PDFLPrintDoc .

The Snippet Runner

The **PDFLSnippetRunner** is based on the **AcroSnippetRunner** provided with the Acrobat 6 SDK, but is adapted to the PDFL environment. It has a command-line interface and prints output to the console. It is found in the **SnippetRunner** subdirectory of the **Samples** directory.

The following commands allow you to control the Snippet Runner in a command shell:

TABLE 2.4 *PDFL Snippet Runner Commands*

Command	Description
ChangeDir	Navigates around the hierarchy of snippets using standard file path semantics.
FindSnippet	Searches for a snippet based on a specified substring of the snippet identification string. If the result is a single file, prompts for execution.
SnippetInfo	Displays the description of a specified snippet.
DocName	Displays the name and file information for the current document.
SetPage	Sets the current page to the one specified.

TABLE 2.4 *PDFL Snippet Runner Commands*

Command	Description
OpenDoc	Opens the specified document. This document becomes the new current document. If the previously open document was modified, prompts to save changes.
CloseDocument	Evicts the current document from the application. If it was modified, prompts to save changes.
SaveDoc	Saves the document to the specified file.
RevertDoc	Reverts the current document to its last saved condition.
Run	Runs the specified snippet. The snippet must exist in the current directory—you cannot specify a path.
ls	Lists the snippets (and hierarchy nodes) in the current directory.
pwd	Prints the current directory.
Help	Lists all the commands with their synopses, or provides help for a specified command.
Quit	Quits the application.
WhichPage	Displays the current selected page.

The snippets provided include most of those available for the Acrobat SDK. Snippets include the following:

TABLE 2.5 *PDFL Snippets*

Snippet	Description
ACEEnumProfilesSnip	Enumerates the profiles of a given type installed on the system. Makes a profile list, gets a count of how many profiles are in the list, gets the description for each item in the list, and dumps the information in the DebugWindow.
ACEEnumSettingsSnip	Demonstrates how to access color Setup files programmatically using the Adobe Color Engine (ACE).
ACEGetWorkingSpaceSnip	Gets working space profiles (RGB, Gray, and CMYK) information such as size, description and color spaces information, and dumps the information in the DebugWindow.

TABLE 2.5 PDFL Snippets

Snippet	Description
ACETransPDETextColorSnip	Takes an input profile (DeviceCMYK) and an output profile (sRGB), creates a color transform from them, and then applies the transform to a single color of a PDF text object.
AddImageMetadataSnip	Shows how to access and write metadata for a component-level Cos object such as a page or an image. Writes an XMP metadata file to the Cos dictionary of an image.
AddImageSnip	Creates an image XObject resource from a JPEG file and adds it to the displayed page.
AddPageMetadataSnip	Shows how to work with XML metadata at the page level. Writes a variety of information about the contents of the page to the page's metadata stream.
AddStructureSnip	Shows how to work with logical structure within a PDF document. Tags PDEText elements within the displayed page.
AddTagSnip	Shows how to add a partial structure tree to an un-tagged PDF, add tags, and add three types of content items (marked content, structure element, and complete PDF objects).
AddTextSnip	Adds a string of English, Chinese, Korean, or Japanese text to the bottom left corner of the current viewing page. It demonstrates font embedding/subsetting and CJK double-byte fonts support in Acrobat. Fonts used in this sample are Times-Roman for English, SimSun for Chinese, Batang for Korean, and MS Mincho for Japanese.
AddXObjectStructureSnip	Shows how to work with logical structure within a PDF document. Tags image XObjects within the displayed page.
ASDateSnip	Gets today's date, and adds different time spans to a date.
ASFileIteratorSnip	Iteratively visits all files in a given folder and its sub-folders, outputting the filenames to the DebugWindow.

TABLE 2.5 **PDFL Snippets**

Snippet	Description
ASGetConfigurationSnip	Checks for the current configuration of Acrobat, and displays an alert with the return value.
ClassMapSnip	Demonstrates that an application using PDFL that processes logical structure can attach additional information (attributes) to any structure element.
ConvertOCGsToRadioButSnip	Creates a radio-button relationship among the OCGs in the document, so that only one OCG in the document can be on at any one time.
CosCryptGetVersionSnip	Reports the version of the current encryption algorithm, and the maximum number of bytes that can be used for the key when encrypting or decrypting with this version.
CosObjCompressionSnip	Shows the use of new Cos layer methods to perform full compression of indirect objects in a document to reduce PDF file size.
CosObjDecompressionSnip	Shows the use of new Cos layer methods to perform decompression of compressed indirect objects in a document so as to restore backward viewer compatibility of the document.
CosObjectExplorerSnip	Prints out either a shallow or a more complete description of a Cos object.
CreateAnnotOCSnip	Create an optional content group and associates it to any link annotations found on the first page of the front document through an optional content membership dictionary.
CreateContentXORSnip	Shows an exclusive OR relationship between optional content groups. When the content of a page is turned off through the layers panel UI, an alternative text object is turned on.
CreateDocStructSnip	Shows how to create a document structure tree that conforms to tagged PDF conventions.
CreateImageContentOCSnip	Converts images within the first page of the front document to optional content. It iterates through images found on the first page of the document and associates them with a newly created optional-content group (OCG).

TABLE 2.5 PDFL Snippets

Snippet	Description
<code>CreateTextContentOCsSnip</code>	Shows how to convert text into optional text, Converts text within the first page of the front document to optional content.
<code>ExploreMetadataSnip</code>	Shows how to access the XMP metadata stream embedded in PDF and write metadata to an XML file.
<code>ExploreStructSnip</code>	Explores the structure and content of a tagged PDF and dumps the structure information to the console or the debug window.
<code>FontInfoSnip</code>	Extracts font information from a PDF document, including the name, subtype, and whether the font is embedded.
<code>GetDocKeywordSnip</code>	Extracts the keywords from a document's Info dictionary.
<code>GetDocMetadataSnip</code>	Shows how to extract PDF document XMP metadata in XML format.
<code>ImageInfoSnip</code>	Shows how to obtain specification information of images in PDF, including size, width, height, filters, bits per component, and rendering intent.
<code>MakeBookmarkSnip</code>	Makes a bookmark named 'Current Page' for the current page at the top (visible) level of the bookmark tree.
<code>ObjShiftSnip</code>	Demonstrates translating page object positions with PDFEdit APIs.
<code>OCActionControlSnip</code>	Shows how to control the visibility of optional content using PDActions .
<code>OCGUIReorderSnip</code>	Reorders and categorizes the OCGs shown in the layers panel UI.

TABLE 2.5 PDFL Snippets

Snippet	Description
<code>OCTextAutoStateSnip</code>	Shows how to use an autostate with optional content. Creates an optional content group for the text on the first page of the front document. Sets the OCG's usage dictionary to indicate the text should be visible when the zoom factor is between 1.0 (100%) and 1.5 (150%). Updates the document's optional content properties catalog to indicate that the OCG's zoom category should be used to determine visibility for View events. The text disappears when the zoom is out of the specified range.
<code>PDCreateMasterOCGSnip</code>	Creates a parent control OCG for use in the UI. Child OCGs cannot be manipulated through the UI while the parent OCG is Off, although the parent's state does not alter the visibility of the child OCGs.
<code>PDDocDidDeletePagesNotSnip</code>	Shows how to register for and receive <code>PDDocDidDeletePages</code> notifications.
<code>PDEContentExplorerSnip</code>	Writes information about the PDE content for page zero to the DebugWindow.
<code>PDEPathDrawCurveSnip</code>	Demonstrates simple spline curve drawing with PDEPath marking APIs.
<code>PDEPathDrawLineSnip</code>	Demonstrates simple line drawing with PDEPath marking APIs.
<code>PDEPathDrawRectSnip</code>	Demonstrates simple rectangle drawing with PDEPath marking APIs.
<code>PDEPathExplorerSnip</code>	Explores PDEPath objects on the current page by breaking them down to PDEPath marking operators and operand(s).
<code>PDOConfigCreateSnip</code>	Creates an optional content configuration, with the OCGs currently in the ON state assigned the default value of ON, and presenting only these OCGs in the layers panel UI.
<code>PDOConfigExplorerSnip</code>	Uses an optional content group callback to display diagnostic information about the OCGs in a document to the DebugWindow.

TABLE 2.5 PDFL Snippets

Snippet	Description
<code>PDOCExplorerSnip</code>	Displays information about optional content configurations to the DebugWindow. Shows the configuration's name, its default state, the list of OCGs that are on by default, and the list of OCGs that are off by default.
<code>PDOCGToggleIntentSnip</code>	Toggles the intent of an optional content group between "design" and "view."
<code>PDOCSetDefaultConfigSnip</code>	Updates the document's default optional content configuration to use the specified alternative configuration.
<code>PDPPageSetTransparencySnip</code>	Changes the transparency of the elements in the content of the first page of the current document.
<code>RaiseExcepSnip</code>	Shows how to register custom exceptions with the application. The exception text associated with the custom exception is passed in as a parameter. We use the error code we get from registering the error string to raise an exception. This exception is caught by the SnippetRunner backstop exception handler.
<code>RemoveEmbeddedFontSnip</code>	Removes embedded Roman fonts from a PDF file. An embedded font is removed only if it is not multi-byte, if encoding is not "Identity," and if the charset is Roman.
<code>RoleMapSnip</code>	Enumerates all existing role maps in the PDF. Also provides an example of how to create new role maps or change existing role maps in a tagged PDF.
<code>SecureDocumentSnip</code>	Shows how to apply security settings programmatically.
<code>SetDocBaseURLSnip</code>	Shows how to set and get a PDF document metadata property. Sets the base URL from user input.
<code>SimpleSnip</code>	Shows how to create a snippet.
<code>TextChangeColourSnip</code>	Changes the <code>PDEText</code> object of the first page of the front document to specified RGB values.

TABLE 2.5 **PDFL Snippets**

Snippet	Description
TextExtractionSnip	Demonstrates how to perform text extraction with the new WordFinder creation APIs and configuration structure.

Documentation of the PDFL API

The following documents are helpful in developing with the Adobe PDF Library and are included in the SDK distribution.

- *Adobe PDF Library Overview*—This document, which contains an introduction to programming with the Adobe PDF Library using the SDK.
- *Acrobat Core API Overview*—An overview of the design and structure of the Acrobat core API, which overlaps in large part with the Adobe PDF Library. The API is organized into layers as follows:
 - The Acrobat Support (AS) layer provides a variety of utility methods, including platform-independent memory allocation and fixed-point math utilities.
 - The Acrobat Viewer (AV) layer deals with the viewer’s user interface. The AV layer is *not* part of the PDF Library.
 - The Cos Object System (Cos) layer provides access to the low-level building blocks of PDF files.
 - The Portable Document (PD) layer provides access to components of PDF documents.
 - The PDFEdit layer provides access to PDF page contents associated with the **Contents** entry in the page’s dictionary.
 - The PDSEdit layer provides access to the logical structure of a PDF document.
 - OS-specific functions.
- *Acrobat Core API Reference*—Lists all of the available API methods by object and level. It also lists macros and structures used in conjunction with the methods. This document covers features that are common to both Acrobat and the Adobe PDF Library. Each method description includes availability in the Acrobat plug-in API and in the Adobe PDF Library.
- *PDF Library Supplement to the Acrobat Core API Reference*—Lists API methods and related structures that are applicable to Adobe PDF Library only.
- *PDF Reference, version 1.5, fourth edition*—Provides a description of the PDF file format, as well as suggestions for producing efficient PDF files. It is intended primarily for application developers who wish to produce PDF files directly. This document also contains enough information to allow developers to write applications that read and modify PDF files. The current version covers PDF version 1.5.
- *AcroColor API Reference*—Lists API methods and related structures that are available at the public level to access the PDF color management facility.

Installing the PDFL SDK

The Adobe PDF Library SDK installation process differs depending on the target platform.

- Windows and Mac OS — An executable installer is provided for each of these platforms. Run the installer and follow the default installation instructions.
- Solaris, AIX, and Linux — A gzip compressed tar file is provided; developers determine where to decompress and install the SDK.

Updating to PDF Library Version 6.0

All PDF Library SDK 5.0 APIs are still supported in PDFL 6.0, although some are deprecated in this release and should not be used. Other than a few minor code changes, all you should need to do is recompile your application with the new SDK headers using the recommended development environment.

NOTE: In PDFL 6.1, there are two PDFL 6.1-only libraries: ARE and XMP.

For all platforms:

- Update your font and CMap resources. The PDFL SDK no longer supports the base 14 fonts. Alternative fonts are included with the PDFL SDK 6.0, as follows:
 - AdobeMingStd-Light-Acro (Traditional Chinese)
 - AdobeSongStd-Light-Acro (Simplified Chinese)
 - AdobeMyungjoStd-Medium-Acro (Korean)
 - KozMinPro-Regular-Acro and KozGoPro-Medium-Acro (Japanese)
 - AdobePiStd
- Redirect the path environmental variable as needed, if you do not install the libraries in the same directory as the executable.

For Mac OS:

- Update your project to include the following components:
 - Add two new libraries `BIBUtilsCarbonLib.shlib` and `JP2KLibCarbon.shlib`
 - Change `PDFL50Carbon.lib` to `PDFL60Carbon.lib`
- Modify your project language setting to force C++ compilation (so that both `.c` and `.cpp` samples compile properly).

For Windows:

- Update your project to include the following components:
 - Add two new libraries `BIBUtils.lib` and `JP2KImpLib.lib` (this should automatically be taken care of because the library path setting takes care of the library search)
 - Change `PDFL50.lib` to `PDFL60.lib`

For UNIX:

- Update your makefile to add two new libraries, `libBIBUtils.so` and `libJP2K.so`.

3

Developing With the Adobe PDF Library

This chapter introduces the essentials required to develop applications using the Adobe PDF Library. The sample applications that accompany the product, which should build and run directly after installation, demonstrate these concepts.

Building Applications with the Adobe PDF Library

This section details the compiler environment variables (macros) required to build applications against the Adobe PDF Library.

- On all platforms, you must define the `PRODUCT` macro for the preprocessor.

```
PRODUCT=\"Library.h\"
```

This macro is used as a trigger for conditional compilation and allows the same headers to be used for both the Acrobat plug-in API and the Adobe PDF Library.

Windows

- The following macros must also be defined in the IDE project settings for applications to compile correctly on the Windows platform:

```
WIN_ENV  
WIN32  
WIN_PLATFORM
```

- The Adobe PDF Library 6.0.1 is compiled with code generation set to “Single-Threaded,” while the Adobe PDF Library 6.1.1 is compiled with code generation set to “Multithreaded.” Applications linking with the Adobe PDF Library must have code generation settings that match or there will be conflicts between the Microsoft libraries `MSVCRT` and `LIBCMT`.
- For Adobe PDF Library 6.1.1, the “Project Settings > Link > Input > Ignore libraries” settings should *not* ignore `LIBCMT` (other versions of PDFL do ignore it).
- The Adobe PDF Library is distributed as an interface library (`PDFL60.lib`) and matching DLL (`PDFL60.dll`). You should link the interface library into your application.
- The operating system must be able to access the Adobe PDF library at runtime. It does so by searching the paths specified by the `PATH` environment variable, as well as the folder in which the application was launched.

MacOS

The MacOS libraries use a precompiled header and prefix file to define the appropriate macros. See `Precompile.pch` in the `Samples:utils` directory of the Adobe PDF Library 6.0 SDK for the macros required to successfully compile the samples.

There are a number of methods for linking with and locating shared object libraries on the MacOS platform.

UNIX

- The following macros must be defined for the headers to compile correctly on the UNIX platform:

```
UNIX_ENV=1
UNIX_PLATFORM=1
```

- Before you can compile the samples, you must point the makefiles to your `gcc 2.95.2` (PDFL 6.0.1), `gcc 3.2` (PDFL 6.1.1), or `xlc 6.0` (6.1.1) compiler. Make sure the permissions on all libraries are set so that the dynamic loader can find and load the libraries.

```
chmod o+x libraryname
```

Shared objects are provided for AIX, Solaris and Linux. Alter the common makefile for each individual platform/os (i.e., `linux.mak`) under `samples/utils` directory to specify the `gcc` or `g++` and static library access path. You will need to set the environment variable `LD_LIBRARY_PATH` to the location of the libraries so that the application will find the Shared Object Libraries at run time. This accomplished with the command `setenv LD_LIBRARY_PATH path`. For more information, see your platform's manual pages for the keyword **LD**.

- Before you run your application, set the `PSRESOURCEPATH` and `ACRO_RES_PATH` environment variables to point to your fonts.

For example, to set these environment variables manually before you run your application:

```
setenv PSRESOURCEPATH /user/yourname/PSFont
setenv ACRO_RES_PATH /user/yourname/PSFont
```

Alternatively, you can define the environment variables within the application using the `putenv` system call.

Initialization and Termination

- NOTE:** PDF Library version 6.1.1 supports thread-safety, so initialization and termination are handled on a per-thread basis. See [Chapter 4, "Using the PDF Library 6.1.1 SDK"](#) for details. The information in this section applies primarily to the single-threading versions of the library.

Applications must initialize and terminate the Adobe PDF Library appropriately with the public API functions **PDFLInit** and **PDFLTerm**.

- Call **PDFLInit** to set up internal data structures, locate required resources such as fonts, and perform initialization (such as setting client-provided memory allocation routines). Calling most library functions without successfully initializing the library results in error conditions.
- Call the **PDFLTerm** routine to clean up before an application terminates or when access to PDF Library functionality is no longer needed.

Initialization Details

The **PDFLInit** function takes as a parameter a **PDFLData** structure, defined in the API header file `PDFInit.h`. See the *PDF Library Supplement to the Core API Reference* for details. You must provide valid values for the **size**, **dirList** and **listLen** members of the structure before passing it to **PDFLInit**.

- **size** denotes the size of the structure itself and can be obtained with:
`sizeof(PDFLDataRec)`.
- The **listLen** member should be the number of directories listed in **dirList**.

The Adobe PDF Library uses the directories listed in the **dirList** member to locate font resources. The following sections describe how this is done on each of the supported platforms.

Windows and MacOS

By default, the PDF Library attempts to locate fonts in subdirectories of the application directory named `Font`, `CIDFont` or `CMap`. Each of these folders, if it exists, is searched 99 levels deep for font resources. For optimization, you can set a flag, **kPDFLInitIgnoreDefaultDirectories**, in the **flags** field of the initialization structure to prevent the process from searching the default font directories.

Use the **dirList** member to specify additional locations of fonts. Each directory specified in the **dirList** array is searched 99 levels deep to locate fonts. (Note that this can affect performance.)

Here is an example showing how to pass the font paths to **dirList** for Windows:

```
pdfLibData.dirList[0]=strdup("C:\\Myfontfolder\\CMap");
pdfLibData.dirList[1]=strdup("C:\\Myfontfolder\\CIDFont");
pdfLibData.dirList[2]= strdup("C:\\Myfontfolder\\Font");
```

The paths can be either full paths or paths relative to the directory from which the executable linking in the Adobe PDF Library was launched.

For more details, see the functions **PDFLGetDirList_Win** and **PDFLGetDirList_Mac** in the `MyPDFLibUtils.cpp` file in the `Samples/Utils` directory.

UNIX

By default, the library only searches for fonts in the directory from which the application was launched. Use the **dirList** member to specify additional locations of font resources.

For more details, see the function **PDFLGetDirList_Unx** in the `MyPDFLibUtils.cpp` file in the `Samples/Utils` directory.

4

Using the PDF Library 6.1.1 SDK

The PDF Library 6.1.1 SDK differs from the 6.0.1 SDK primarily in that it provides the ability to safely operate in a multithreaded environment, whereas the 6.0.1 SDK does not. Otherwise, the differences are few. The differences, as well as how to use the multithreading capability of the 6.1.1 SDK, are described in this chapter.

PDF Library 6.1.1 SDK Overview

The major feature added to PDF Library 6.1.x is thread safety. When using the thread-safe PDF library, initialization and termination now additionally need to be performed for each thread that calls into the library, as well as at the process level. The interfaces for per-thread initialization/termination are the same as before (**PDFLInit** and **PDFLTerm**—see [“Initialization and Termination” on page 22](#)).

Since each thread acquires an independent PDFL memory context, clients are advised not to share PDFL data and resources among threads. This includes sharing the same PDF file.

New Samples in the 6.1.x SDKs

The 6.1.x SDKs provide the new samples shown in the table. These demonstrate the new capabilities available in PDF Library 6.1.x as a result of support for multithreading.

TABLE 4.1 *PDFL 6.1.x-specific Samples*

Sample	Description
MTInMemFS	Demonstrates use of an in-memory file system for a simple workflow within a multithreaded context. See “New APIs in PDF Library Version 6.1.x” .
MTSerialNums	Demonstrates creation of multiple threads to simultaneously generate multiple PDFs.
MTTextExtract	Demonstrates multiple threads concurrently processing multiple PDF documents.

New APIs in PDF Library Version 6.1.x

All the APIs in PDFL 6.1.x are fully compatible with 6.0.1, with the following four exceptions. The 6.1.x SDKs support the following four new APIs:

```
ASGetRamFileSys
ASRamFileSysSetLimitKB
ASSetTempFileSys
ASGetTempFileSys
```

ASGetRamFileSys is used to create a “virtual” (memory-resident) file system. This file system is flat and uses a single global “directory” that maps between filenames and files. **ASRamFileSysSetLimitKB** is used to set the in-memory file system size limit, in kilobytes. A RAM file system should be created and used by only a single thread.

NOTE: Multiple threads should be able to access a RAM file system created by another thread, but such use is NOT recommended.

Given the availability of a memory-based file system, the client may wish the PDF library to use that file system for temporary files. **ASSetTempFileSys** and **ASGetTempFileSys** allow the client to establish which file system is used for temporary files.

ASSetTempFileSys allows the client to set which file system is used for temporary files. The **ASGetTempFileSys** returns the file system to be used for temporary files. The default temporary file system is the default file system as returned by **ASGetDefaultFileSys**. If clients don't call **ASSetTempFileSys** to establish a temporary file system, no change in system behavior will occur (**ASGetDefaultFileSys** will return a “real” default file system as usual).

Both **PDDocOpenWithParams** and **PDDocSaveWithParams** allow the client to pass an **ASFileSys** in the parameter structure. A client can call **PDDocCreate** to create a new PDF file, put in contents, and then call **PDDocSaveWithParams** and specify the **ASFileSys** returned by **ASFileSysGetRamFileSys** as the file system. The client can then call **ASFileSysOpenFile** with the same **ASFileSys** parameter and the same **pathName** to get an **ASFile** on the document and then call **ASFileRead** to get the bytes and stream it off to a destination.

NOTE: The PDF Library currently has the notion of a temporary file pathname. The client can set a pathname to be used when generating temporary files. The temporary file system is orthogonal to the existing pathname mechanism.

See the **MTInMemFS** sample for further details.

Using Threads

The Adobe PDF 6.1.x libraries are thread-safe. To use threads, simply make the appropriate system call (**_beginthreadex** on Windows, and **pthread_create** on UNIX). While multiple threads cannot share PDFL datatypes, they do share the same process heap; this means that an application can share generic datatypes between threads. However,

multiple threads should not attempt to write to the same PDF document. There is not a problem, however, with multiple threads opening the same file read-only.

NOTE: On Windows, **CreateThread** is not recommended if the application is using most `stdio.h`-defined functions, including file I/O and string manipulation. It is best to use **_beginthreadex** on Windows, which performs extra bookkeeping to ensure thread-safety.

