PDF Explained by John Whitington

Chapter 1. Introduction

The Portable Document Format (PDF) is the world's leading language for describing the printed page, and the first one equally suitable for paper and online use. In this chapter, we take a tour of its uses, features, and history. We look at some useful free software and resources, some of which we'll use later in this book.

A Little History

Today we take the high fidelity exchange of documents for granted, knowing that a document sent here will appear the same there and vice versa, and that it may be displayed equally on screen and on paper. This was not always so.

Page Description Languages

We could pass documents between users, and from user to printer, as a series of bitmap pictures (e.g., TIFF or PNG), one for each page. However, this doesn't allow for any structure to be retained, precludes scaling to different paper sizes or resolutions without loss of quality, involves huge file sizes, and so on.

A *page description language* like PDF is a way of describing the contents (text and graphics) of a printed or onscreen page using highly structured data, often with extra *metadata* describing various aspects of the document (such as printing information or textual annotations or how it is to be viewed or printed). This way, decisions about how the document is rasterized (converted to pixels by a printer or on screen) can be left until the end of the production process. A PDF file can contain text and associated font definitions, vector and bitmap graphics, navigation (such as hyperlinks and bookmarks), and interactive forms.

PDF is used wherever the exact presentation of the content is important (for example, for a print advertisement or book). It isn't normally suitable when the content is to be laid out or reflowed at the last moment, such as in a variable width web page—languages like HTML and CSS, which separate content from presentation, are more suitable in those circumstances.

Other page description languages

Many page description languages were created when the printing of lines of text in fixed fonts began to be replaced by digital graphics printing. The printer would then process the language to generate a bitmap at the appropriate resolution. For example, PostScript (Adobe), PCL (Hewlett Packard), and KPDL (Kyocera). Simpler languages were used for vector plotters (for example, HPGL from Hewlett Packard).

These languages varied in complexity and functionality. PostScript files, for example, are full programs—the result of executing the program is the document's visual representation. These languages often contain extra instructions to control aspects of the document other than the page content, for example, which tray paper is drawn from or whether the output is to be duplexed.

Development of PDF

PDF began as an internal project at Adobe to create a platform-neutral method for document interchange. PostScript was already popular in the print community, but wasn't practical for onscreen use with the computers of the day—especially for random access (to render page 50 of a PostScript document, one must process pages 1–49 first). The idea was to use a subset of the PostScript graphics language together with ancillary data to create a structured language for standalone documents to be viewed on (or printed from) any computer.

PDF 1.0 was announced in 1993, with Acrobat Distiller (for creating and editing PDF files) and Acrobat Reader (for viewing only), both as paid-for programs. Later on, Acrobat Reader was made available to everybody at no cost, leading to the widespread use of PDF for the exchange of documents online.

Over the next 10 years, after a slow start as prepress features were added, PDF overtook PostScript as the language of choice in the printing industry. Today, it is the only general page description language of note.

Some Advantages of PDF

When a number of formats compete to be the industry standard, the best contender is not always the victor—luck can intervene. In this case,

though, PDF had a number of singular advantages. We look at some of them here.

Random access and linearization

Unlike PostScript, any object (page, graphic, etc.) in a PDF document can be accessed at will, in constant time. This means it's no harder to read page 150 than page 1. *Linearization* is the process of arranging the objects in the file such that all those needed for a given page are located in adjacent positions. This explains why you can quickly jump to any page in a PDF being viewed in Acrobat Reader in a web browser window—the viewer doesn't need to load the whole file to begin with, it fetches from the server just the sections needed to display each new page.

Stream creation and incremental update

Stream creation is the ability inherent in the PDF format to allow files to be created in order, from beginning to end, even if the eventual file is larger than the memory available.

Incremental update means that, when editing a file, it's possible to write the changes to the end of the file without modifying any existing part—this makes saving changed versions very fast, and can be used to provide an undo mechanism (since the previous version is still intact).

Embedded fonts

Fonts used in a PDF are embedded along with the document. This means that it should always be rendered correctly, regardless of which fonts are installed on a given computer. The program creating the PDF document will remove unnecessary data from the font (such as unused characters), so the file does not become unduly large. PDF supports all common font formats, such as TrueType and Type 1.

Searchable text

Most PDF files maintain the information to map the character shapes making up the text to Unicode character codes. This means that you can copy and paste text from a document, or search the text easily. More recent developments in PDF allow the logical order of the text in the document to be stored separately from the layout of the text on the page, preserving yet more structured information.

ISO Standardization

PDF was released as an open standard by the International Organization for Standardization (ISO) in 2008. The ISO-32000-1:2008 document is largely the same as the PDF file format document previously released by Adobe.

This independence lends legitimacy and oversight to the PDF standard, which should encourage its further adoption. However, with no real tools for detecting whether a file meets the standard (Adobe Reader will happily load malformed files, so many tools create them), genuine rigor is some time away.

THE PDF FILE FORMAT DOCUMENT

The PDF File Format Version 1.7 is documented in ISO 32000-1:2008, which is available on CD or as a PDF for 380 Swiss Francs at the International Organization for Standardization.

The almost identical Adobe Document “Adobe PDF Reference, Sixth Edition, version 1.7” is available in PDF format at the Adobe PDF Technology Center. Adobe extensions, which do not yet form part of the ISO standard, are published at the same location.

Unfortunately, the PDF File Format is no longer available in print.

Specialized Kinds of PDF

There are several specialized variations on the PDF format—both standardized and in development. These are subsets of the PDF format. Each file is a valid PDF document, but with restrictions on the facilities used or the content itself. Two of these, PDF/A and PDF/X, are now ISO standards.

PDF/A

The PDF/A Standard (ISO 19005-1:2005) defines a set of rules for documents intended for long-term archiving in libraries, national archives, and bureaucracies. It also requires a "conforming reader" to act in certain ways, using the embedded fonts, using color management, and so forth. Briefly, the restrictions on PDF/A are:

- No encryption
- All fonts to be embedded
- Metadata is required
- JavaScript is disallowed
- Color spaces specified in a device independent manner
- No audio or video content

There are two levels of PDF/A compliance: PDF/A-1b ("level B compliance") requires exact visual reproduction of the document. PDF/A-1a ("level A compliance") requires that text can be mapped to Unicode, and that the order and structure of the text is documented, in addition to the requirement of exact visual reproduction.

PDF/X

The PDF/X Standard is a family of ISO standards for graphics exchange in the printing industry, the latest of which is PDF/X-5 (ISO 15930-8:2010).

It defines a number of restrictions:

- All fonts must be embedded
- All image data must be embedded
- Cannot contain sound, films, or non-printable annotations
- No forms
- No JavaScript
- No encryption

and a number of extra requirements:

- The file is marked as PDF/X with the subversion (e.g., PDF/X-5).
- Bleed, trim, and/or art boxes are required, in addition to the normal page size. These boxes define the size of the media, the printable area, the final cut size, and so on.
- A flag is set if the file has been *trapped*. Trapping is the process of creating small overlaps between graphical objects to mask registration problems in multiple color printing processes.
- The file must contain an *output intent*, a color profile describing how it is to be printed.

Version Summary

PDF is fully backward compatible (you can load a PDF version 1.0 document into a program designed for PDF 1.7) and mostly forward compatible (programs written for PDF 1.1 or later can often load PDF 1.7 files). Forward compatibility is ensured because readers ignore content they don't understand—it's only when new compression methods or object storage mechanisms are introduced that this may be broken. Since PDF 1.5 in 2003, such changes have been minimal. PDF versions and their features are summarized in Table 1-1.

Table 1-1. Functionality in PDF versions 1.0 to 1.7 Extension Level 8

PDF version	Acrobat Reader version	Launched	Summary of new features
1.0	1.0	1993	First release.
1.1	2.0	1996	Device independent color spaces, encryption (40-bit), article threads, named destinations, and hyperlinks.
1.2	3.0	1996	AcroForms (interactive forms), films, and sounds, more compression methods, Unicode support.
1.3	4.0	2000	More color spaces, embedded (attached) files, digital signatures, annotations, masked

			images, gradient fills, logical document structure, prepress support.
1.4	5.0	2001	Transparency, 128-bit encryption, better form support, XML metadata streams, tagged PDF, JBIG2 compression.
1.5	6.0	2003	Object streams and cross-reference streams for more compact files, JPEG 2000 support, XFA forms, public-key encryption, custom encryption methods, optional content groups.
1.6	7.0	2004	OpenType fonts, 3D content, AES encryption, new color spaces.
1.7 (later ISO 32000-1:2008)	8.0	2006	XFA 2.4, new kinds of string, extensions to public-key architecture.
1.7 Extension Level 3	9.0	2008	256-bit AES encryption.
1.7 Extension	9.1	2009	XFA 3.0.

Level 5			
1.7 Extension Level 8	X	2011	Not yet known.

What's in a PDF?

A typical PDF file contains many thousands of objects, multiple compression mechanisms, different font formats, and a mixture of vector and raster graphics together with a wide variety of metadata and ancillary content. We take a brief tour of these elements here, for context—they are covered more fully in later chapters.

Text and Fonts

A PDF file can contain text drawn from multiple fonts of all popular formats (Type1, TrueType, OpenType, etc). Legacy bitmap fonts are also supported through simulation. Font files are embedded in the document, so the character shapes are always available, meaning the file should render the same on any computer. A variety of character encodings are supported, including Unicode.

Text can be filled with any color, pattern, or transparency. A piece of text may be used as a shape to clip other content, allowing complicated graphical effects whilst text remains selectable and editable.

Typically, enough information is encoded in a PDF document to allow text extraction, though the process is not always straightforward.

Vector Images

Graphical content in PDF is based on the model first used in Adobe's PostScript language. It consists of *paths* built from straight lines and curves. Each path may be filled, "stroked" to draw a line, or both. Lines can have varying thicknesses, join styles, and dash patterns.

Paths may be filled in any color, with a repeating pattern defined by other objects, or with a smooth gradient between two colors. All these options apply also to the lines of stroked paths.

Paths can be rendered using a variety of plain or gradient transparencies, with several different *blend modes* defining how semitransparent objects interact. Objects may be grouped together for the purposes of transparency, so a single transparency can be applied to a whole group of objects at once.

Paths can be used to clip other objects, so that only sections of those objects overlapping with the clipping path are shown. These clipping regions may be nested within one another.

PDF has a mechanism which allows a graphic to be defined once and then used multiple times in different contexts. This can be used, for instance, for a recurring motif, even across more than one page.

Raster Images

PDF documents can include bitmap images between 1 and 16 bits per component, in several *color spaces* (for example, three-component RGB or four-component CMYK). Images can be compressed using a variety of lossless and lossy compression mechanisms.

Images may be placed at any scale or rotation, used to create a fill pattern, and may have a *mask*, which defines how they blend with the background they are placed on.

Color Spaces

PDF can use color spaces related to particular electronic or print devices (grayscale, RGB, CMYK) and ones related to human color perception. In addition, there are color spaces for the printing industry such as *spot colors*. Mechanisms exist for simpler PDF programs (like onscreen viewers) to fall back to basic color spaces if they do not support the more advanced ones.

Metadata

PDF documents have a set of standard metadata, such as *title*, *author*, *keywords*, and so on. These are defined outside the graphical content and have no effect on the document when viewed. The creator (the program that created the content) and producer (the program that wrote the PDF file) are also recorded. Each document also has a set of unique identifiers, allowing them to be tracked through a workflow.

Since PDF 1.4, the metadata can be stored in an XML (eXtensible Markup Language) document embedded in the PDF using Adobe's Extensible Metadata Platform (XMP), described in ISO 16684-1. This defines a way to store metadata for objects in the PDF, which can be extended by third parties to hold information relevant to their particular workflows or products.

Navigation

PDF documents have two methods of navigation, when viewed on screen:

- The *document outline*, commonly known as the document's *bookmarks*, is a structured list of destinations within the document, shown alongside it. Clicking on one moves the view to that page or

position.

- Hyperlinks within the text or graphics of a document allow the user to click to move elsewhere within the document, or to open an external URL.

Optional Content

Optional content groups in PDF allow parts of the content of a page to be grouped together and shown—or not shown—based on some other factor (user choice, whether the document is on screen or printed, the zoom factor). Relationships between groups can be defined, so that they depend upon one another. One use for this is to emulate the "layers" found in graphics packages. For example, Adobe Illustrator layers are preserved when a document it produces is read with a PDF viewer.

Multimedia

PDF documents can include various kinds of multimedia elements. A lot of this breaks the portability inherent in PDF, and is often not well supported outside of Adobe products.

From PDF 1.1

Slide shows can be defined, to move automatically between pages with transition effects.

From PDF 1.2

Sounds and movies can be embedded.

From PDF 1.5

A more general system for including arbitrary media types was introduced.

From PDF 1.6

3D Artwork can be embedded.

Interactive Forms

There are two incompatible forms architectures in PDF: AcroForms, which is an open standard, and the Adobe XML Forms Architecture (XFA), which is documented but requires commercial software from Adobe.

Forms allow users to fill in text fields, and use check boxes and radio buttons. When the data is complete, it may be saved into the document (if allowed) or submitted to a URL for further processing. Embedded JavaScript is often used in conjunction with forms to deal with verification of field values or similar tasks.

Logical Structure and Reflow

Logical structure facilities allow information about the structural content (chapters, sections, figures, tables, and footnotes) to be included alongside the graphical content. The particular elements are customizable by third parties.

A *tagged PDF* is one that has logical structure based on a set of Adobe-defined elements. Files following these conventions can be *reflowed* by a reader to display the same text in a different page size or text size, for example, in an ebook reader.

Security

PDF documents can be encrypted for security, using RC4 or AES encryption methods. There are two passwords—the *owner password* and the *user password*. The owner password unlocks the file for all changes, the user password just allows a range of operations selected by the owner when the file was originally encrypted (for example, allowing or disallowing printing or text extraction). Frequently the user password is blank, so the file appears to open as normal, but functionality is restricted.

Starting with PDF 1.3, digital signatures can be used to authenticate the identity of a user or the contents of the document.

Compression

Images and other data streams in PDF can be compressed using a variety of lossless and lossy methods defined by third parties. By compressing only these streams (rather than the whole file), the structure of the PDF objects is always available without decompressing the whole file, and compressed sections can be processed only when needed. There are several groups of compression methods:

- Compression for bi-level (e.g., black and white) images. PDF supports the standard lossless fax encoding methods for bi-level images and, from PDF 1.4, the JBIG2 standard (both lossy and lossless), which provides better compression for the same class of images.

- Image filters suitable for photographic data such as JPEG and, from PDF 1.5, JPEG2000 in both lossy and lossless variants.

- Lossless compression mechanisms suitable for image data and general data compression, such as Flate (the zip algorithm), Lempel-Ziv-

Welch (LZW), and run length encoding.

Who Uses PDF?

PDF is used in a wide variety of industries and professions. We describe some here, explaining why PDF is suitable for each.

The Printing Industry

PDF has support for the color spaces, page dimension information (such as media, crop, art, and bleed boxes), trapping support, and resolution-independence required for commercial printing. Together with other technologies, PDF is the key part of the publishing-for-print workflow. The extensibility of PDF metadata allows various schemes for including extra data along with the document, and for keeping it with the document throughout the publishing process—parts of the workflow that don't understand a particular piece of metadata will at least preserve it.

Ebooks and Publishing

This book was created using the *DocBook* system, which takes a structured document in XML format, typesets it, and produces a PDF complete with hyperlinks and bookmarks, together with a more traditional PDF suitable for printing.

PDF is one of the competing ebook formats. To support display on a wide range of screens, PDF documents may be tagged with reflow information, allowing lines of text to be displayed at differing widths on each device. This is at odds with the other uses of PDF, where fixed text layout is a requirement.

PDF Forms

PDF forms are especially useful when existing paper-based systems are being transitioned to electronic ones, or must exist alongside them. A PDF form (filled in online then printed out) looks the same as one filled in manually on paper, and may be processed by existing human and computer systems in the same way.

Automatic submission of forms from within the PDF viewer, the use of JavaScript to add intelligence (making sure figures add up in a tax form, for example), and the use of digital signatures to sign filled-in forms are all compelling reasons to use PDF for electronic forms.

Document Archiving

Through PDF/A, PDF is the ideal format for long-term archiving, combining accurate representations of scanned and electronic content, together with Unicode language support, and compression mechanisms for all sorts of data including the important CCITT Fax and JBIG2 methods for monochrome images. Being an ISO standard (and one that is near-ubiquitous) guarantees that these documents can be read long into the future.

PDF can be used for Optical Character Recognition (OCR), allowing searchable text to be created from the original, the exact visual representation being retained alongside the recognized text.

As a File Format

PDF is not, at first sight, suitable for use as an editable vector graphics format. For example, a circle won't remain editable as a circle, since it will have been converted to a number of curves (there is no circle element in PDF).

However, if appropriate use is made of its extensibility to store auxiliary

data, it makes a good solution. Adobe Illustrator, for example, now uses an extended form of PDF as its file format. The file can be viewed in any PDF viewer, but Illustrator can make use of the extended data when it is loaded back into the program.

Useful Free Software

In this book, we use various pieces of software to help us with examples. Luckily, everything you need is freely available. You'll need a PDF viewer:

- *Acrobat Reader* is Adobe's own PDF viewer. It supports all versions and features of PDF and comes with a browser plug-in on most platforms. It's available for Microsoft Windows, Mac OS X, Linux, Solaris, and Android.

- *Preview* is the pre-installed PDF viewer and browser plug-in for PDF documents on Mac OS X. It's highly capable and very fast, but doesn't support everything that Acrobat Reader does. Many people stick with Preview as the default application for PDF files, but install Acrobat Reader as well.

- *Xpdf* is an open source PDF viewer for Unix. It supports a reasonable subset of PDF.

- *gv* is a PostScript and PDF viewer frontend for GhostScript (see below). It can render the textual and graphical content of almost all documents. However, it lacks most of the interactive features of other PDF viewers.

There are two key command-line tools:

- *pdftk* is a multiplatform command-line tool for processing PDF files in various ways. It can be downloaded in pre-built form for Microsoft Windows, Mac OS X, and Linux, as well as in source code form.

- *Ghostscript* is a set of tools including an interpreter for PostScript and PDF. It can be used to render PDF files, and to process them in various ways from the command line. It is available in binary form for Microsoft Windows, and in source code form for all platforms.

A full discussion of Adobe and open source PDF software is in Chapter 10.

Chapter 2. Building a Simple PDF

In this chapter, we'll build PDF content manually in a text editor. Then we'll use the free *pdftk* program to turn it into a valid PDF file and look at the output in a PDF viewer.

This example, together with all the PDF files in this book, can be downloaded from the web page for this book.

We'll be looking at a lot of new concepts all at once, so don't worry if it seems overwhelming—we'll come back to all of this in future chapters.

PDFTK—THE PDF TOOLKIT

pdftk is a free, open source command-line tool for Microsoft Windows, Mac OS X, and Unix. We're going to use it in this chapter (and throughout this book) to turn PDF content we've written in a text editor into a valid PDF file. *pdftk* can also be used to:

- Merge and split PDF documents
- Rotate PDF pages
- Decrypt and encrypt
- Fill PDF forms with data
- Apply watermarks and stamps
- Print and change PDF metadata
- Attach files to PDF documents

Source and binary packages for *pdftk* can be found at PDF Labs.

The creator of *pdftk*, Sid Steward, is also the author of O'Reilly's *PDF Hacks*—a selection of tools and tips for working with PDF.

Basic PDF Syntax

A PDF file contains at least three distinct languages:

- The *document content*, which is a number of objects with links between them forming a *directed graph*. These objects describe the structure of the document (pages, metadata, fonts, and resources).
- The *page content*, described using a series of operators for placing text and graphics on a single page.

- The *file structure*, consisting ...

Chapter 3. File Structure

In this chapter, we describe the layout and content of the PDF file's four main sections, and the syntax of the objects which make up each one. We also outline the process of reading a PDF file into a high level data structure, and the converse operation of writing that structure to a PDF file.

File Layout

A simple valid PDF file has four parts, in order:

1. The *header*, which gives the PDF version number.

2. The *body*, containing the pages, graphical content, and much of the ancillary information, all encoded as a series of *objects*.

3. The *cross-reference table*, which lists the position of each object within the file, to facilitate random access.

4. The *trailer* including the *trailer dictionary*, which helps to locate each part of the file and lists various pieces of metadata which can be read without processing the whole file.

For reference, we reproduce the "Hello, World" PDF from Chapter 2 as Example 3-1. The first line of each of the four sections has been annotated.

Example 3-1. A small PDF file

```
%PDF-1.1 Header starts here
%âãÏÓ
1 0 obj Body starts here
<<
/Kids [2 0 R]
/Count 1
/Type /Pages
>>
endobj
2 0 obj
<<
/Rotate 0
/Parent 1 0 R
/Resources 3 0 R
/MediaBox [0 0 612 792]
/Contents [4 0 R]
/Type /Page
>>
endobj
3 0 obj
<<
```

```
/Font
<<
/F0
<<
/BaseFont /Times-Italic
/Subtype /Type1
/Type /Font
>>
>>
>>
endobj
4 0 obj
<<
/Length 65
>>
stream
1. 0. 0. 1. 50. 700. cm
BT
  /F0 36. Tf
  (Hello, World!) Tj
ET
endstream
endobj
5 0 obj
<<
/Pages 1 0 R
/Type /Catalog
>>
endobj
xref  Cross-reference ...
```

Chapter 4. Document Structure

In this chapter, we leave behind the bits and bytes of the PDF file, and consider the logical structure. We consider the *trailer dictionary*, *document catalog*, and *page tree*. We enumerate the required entries in each object. We then look at two common structures in PDF files: *text strings* and *dates*.

Figure 4-1 shows the logical structure of a typical document.

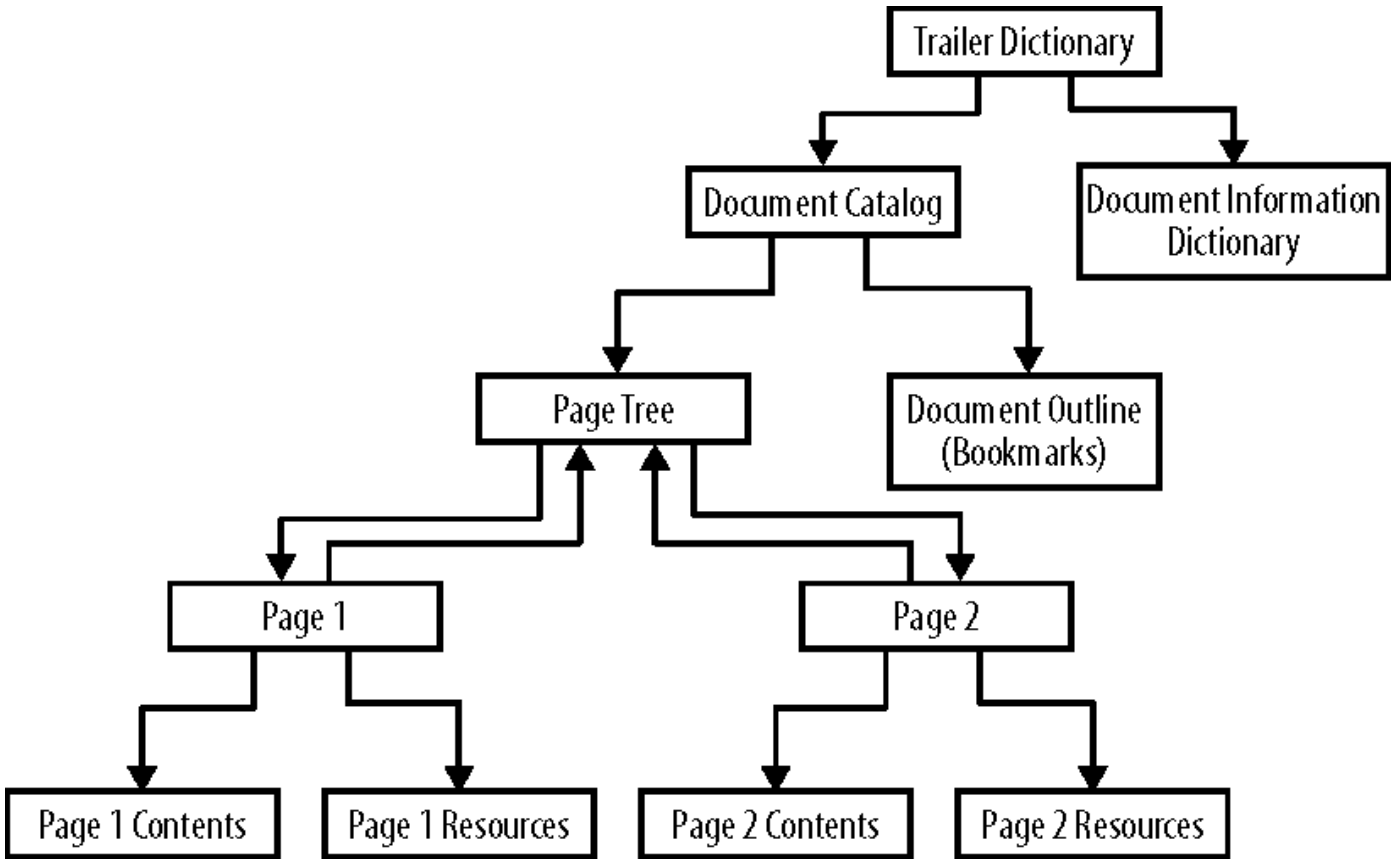

Figure 4-1. Typical document structure for a two page PDF document

Trailer Dictionary

This dictionary, residing in the file's trailer rather than the main body of the file, is one of the first things to be processed when a program wants to read a PDF document. It contains entries allowing the cross-reference table—and thus the file's objects—to be read. Its important entries are summarized in Table 4-1.

*Table 4-1. Entries in a trailer dictionary (*denotes required entry)*

Key	Value type	Value
`/Size*`	Integer	Total number of entries in the file's cross-reference table (usually equal to the number of objects in the file plus one).
`/Root*`	Indirect reference to dictionary	The *document catalog*.
`/Info`	Indirect reference to dictionary	The document's *document information dictionary*.
`/ID`	Array of two Strings	Uniquely identifies the file within a work flow. The first string is decided when the file is first created, the second modified by workflow systems when they modify the file.

Here's an example trailer dictionary:

```
<<
   /Size 421
   /Root 377 0 R
   /Info 375 0 R
   /ID [<75ff22189ceac848dfa2afec93deee03> <057928614d9711db835e0(
>>
```

Once the trailer dictionary has been processed, we can go on to read the *document information dictionary* and the *document catalog*.

Document Information Dictionary

The *document information dictionary* contains the creation and modification dates of the file, together with some simple metadata (not to be confused with the more comprehensive XMP metadata discussed in XML Metadata).

Document information dictionary entries are described in Table 4-2. A typical document information dictionary is given in Example 4-1.

Table 4-2. Entries in a document information dictionary. The types "text string" and "date string" are explained later in this chapter.

Key	Value type	Value
`/Title`	text string	The document's title. Note that this is nothing to do with any title displayed on the first page.
`/Subject`	text string	The subject of the document. Again, this is just metadata with no particular rules about content.
`/Keywords`	text string	Keywords associated with this document. No advice is given as to how to structure these.
`/Author`	text	The name of the author of the

	string	document.
/CreationDate	date string	The date the document was created.
/ModDate	date string	The date the document was last modified.
/Creator	text string	The name of the program which originally created this document, if it started as another format (for example, “Microsoft Word”).
/Producer	text string	The name of the program which converted this file to PDF, if it started as another format (for example, the format of a word processor).

Example 4-1. Typical document information dictionary

```
<<
   /ModDate (D:20060926213913+02'00')
   /CreationDate (D:20060926213913+02'00')
   /Title (catalogueproduit-UK.qxd)
   /Creator (QuarkXPress: pictwpstops filter 1.0)
   /Producer (Acrobat Distiller 6.0 for Macintosh)
   /Author (James Smith)
>>
```

The *date string* format (for /CreationDate and /ModDate) is discussed in the section Dates. The *text string* format (which describes how different encodings can be used within the string type) is described in Text Strings.

Document Catalog

The *document catalog* is the root object of the main object graph, from which all other objects may be reached through indirect references. In Table 4-3, we list the document catalog dictionary entries which are required, and some of the many optional ones, so as to introduce brief PDF topics we don't cover elsewhere in these pages.

*Table 4-3. The document catalog (*denotes required entry)*

Key	Value type	Value
`/Type*`	name	Must be `/Catalog`.
`/Pages*`	indirect reference to dictionary	The root node of the page tree. Page trees are discussed in Pages and Page Trees.
`/PageLabels`	number tree	A number tree giving the page labels for this document. This mechanism allows for pages in a document to have more complicated numbering than just 1,2,3.... For example, the preface of a book may be numbered i,ii,iii..., whilst the main content starts again at 1,2,3....These page labels are displayed in PDF

		viewers—they have nothing to do with printed output.
/Names	dictionary	The name dictionary. This contains various *name trees*, which map names to entities, to prevent having to use object numbers to reference them directly.
/Dests	dictionary	A dictionary mapping names to destinations. A destination is a description of a place within a PDF document to which a hyperlink sends the user.
/ViewerPreferences	dictionary	A *viewer preferences dictionary*, which allows flags to specify the behavior of a PDF viewer when the document is viewed on screen, such as the page it is opened on, the initial viewing scale and so on.
/PageLayout	name	Specifies the page layout to be used by PDF viewers. Values are /SinglePage,

		/OneColumn, /TwoColumnLeft, /TwoColumnRight, /TwoPageLeft, /TwoPageRight. (Default: /SinglePage). Details are in Table 28 of ISO 32000-1:2008.
/PageMode	name	Specifies the page mode to be used by PDF viewers. Values are /UseNone, /UseOutlines, /UseThumbs, /FullScreen, /UseOC, /UseAttachments. (Default: /UseNone). Details are in Table 28 of ISO 32000-1:2008.
/Outlines	indirect reference to dictionary	The outline dictionary is the root of the *document outline*, commonly known as the bookmarks.
/Metadata	indirect reference to stream	The document's XMP metadata—see XML Metadata.

Pages and Page Trees

A *page tree*, built from *page dictionaries*, brings together instructions for drawing the graphical and textual content (which we consider in Chapter 5 and Chapter 6) with the resources (fonts, images, and other external data) which those instructions make use of. It also includes the page size, together with a number of other *boxes* defining cropping and so forth.

The entries in a page dictionary are summarized in Table 4-4.

*Table 4-4. Entries in a page dictionary (*denotes required entry)*

Key	Value type	Value
/Type*	name	Must be /Page.
/Parent*	indirect reference to dictionary	The parent node of this node in the page tree.
/Resources	dictionary	The page's resources (fonts, images, and so on). If this entry is omitted entirely, the resources are inherited from the parent node in the page tree. If there are really no resources, include this entry but use an empty dictionary.
/Contents	indirect reference	The graphical content of the page in one or more sections. If this entry is

	to stream or array of such references	missing, the page is empty.
/Rotate	integer	The viewing rotation of the page in degrees, clockwise from north. Value must be a multiple of 90. Default value: 0. This applies to both viewing and printing. If this entry is missing, its value is inherited from its parent node in the page tree.
/MediaBox*	rectangle	The page's *media box* (the size of its media, i.e., paper). For most purposes, the page size. If this entry is missing, it is inherited from its parent node in the page tree.
/CropBox	rectangle	The page's crop box. This defines the region of the page visible by default when a page is displayed or printed. If absent, its value is defined to be the same as the media box.

The *rectangle* data structure for the *media box* and the other boxes is an array of four numbers. These define the diagonally opposite corners of the rectangle—the first two elements of the array being the *x* and *y* co-ordinates of one corner, the latter two elements being those of the oth-

er. Normally, the lower-left and upper-right corners are given. So, for example:

```
/MediaBox [0 0 500 800]
/CropBox [100 100 400 700]
```

defines a 500 by 800 point page with a crop box removing 100 points on each side of the page.

The pages are linked together using a *page tree*, rather than a simple array. This tree structure makes it faster to find a given page in a document with hundreds or thousands of pages. Good PDF applications build a *balanced tree* (one with the minimum height for the number of nodes). This ensures that a particular page can be located quickly. The nodes with no children are the pages themselves. An example page tree structure for seven pages is shown in Figure 4-2.

This would be written in PDF objects as shown in Example 4-2. The entries in an intermediate or root page tree node (i.e., not a page itself) are summarized in Table 4-5.

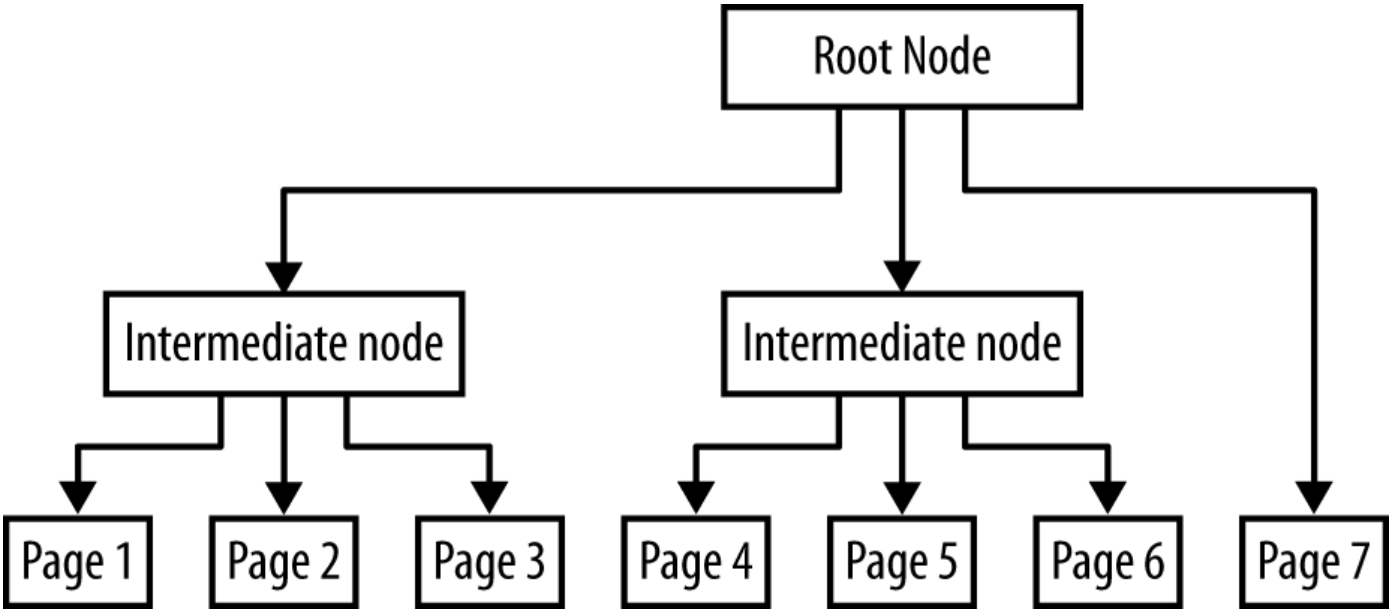

Figure 4-2. A page tree for seven pages. The exact shape of the tree is left to the individual PDF application. The PDF code for this tree is shown in Example 4-2.

Example 4-2. PDF objects used to build the page tree illustrated in Figure 4-2

```
1 0 obj  Root node
<< /Type /Pages /Kids [2 0 R 3 0 R 4 0 R] /Count 7 >>
endobj
2 0 obj  Intermediate node
<< /Type /Pages /Kids [5 0 R 6 0 R 7 0 R] /Parent 1 0 R /Count 3 >
endobj
3 0 obj  Intermediate node
<< /Type /Pages /Kids [8 0 R 9 0 R 10 0 R] /Parent 1 0 R /Count 3
endobj
4 0 obj  Page 7
<< /Type /Page /Parent 1 0 R /MediaBox [0 0 500 500] /Resources <<
endobj
5 0 obj  Page 1
<< /Type /Page /Parent 2 0 R /MediaBox [0 0 500 500] /Resources <<
endobj
6 0 obj  Page 2
<< /Type /Page /Parent 2 0 R /MediaBox [0 0 500 500] /Resources <<
endobj
7 0 obj  Page 3
```

```
<< /Type /Page /Parent 2 0 R /MediaBox [0 0 500 500] /Resources <<
endobj
8 0 obj  Page 4
<< /Type /Page /Parent 3 0 R /MediaBox [0 0 500 500] /Resources <<
endobj
9 0 obj  Page 5
<< /Type /Page /Parent 3 0 R /MediaBox [0 0 500 500] /Resources <<
endobj
10 0 obj  Page 6
<< /Type /Page /Parent 3 0 R /MediaBox [0 0 500 500] /Resources <<
endobj
```

*Table 4-5. Entries in an intermediate or root page tree node (*denotes a required entry)*

Key	Value type	Value
/Type*	name	Must be /Pages.
/Kids*	array of indirect references	The immediate child page-tree nodes of this node.
/Count*	integer	The number of page nodes (not other page tree nodes) which are eventual children of this node.
/Parent	indirect reference to page tree node	Reference to the parent of this node (the node of which this is a child). Must be present if not the root node of the page tree.

In this tree, any page can be found at most two indirect references away

from the root node.

Text Strings

Strings outside of the actual textual content of a page (e.g., bookmark names, document information etc.) are known as *text strings*. They are encoded using either *PDFDocEncoding* or (in more recent documents) Unicode. PDFDocEncoding is a based on the ISO Latin-1 Encoding. It is documented fully in Annex D of ISO Standard 32000-1:2008.

Text strings which are encoded as Unicode are distinguished by looking at the first two bytes: these will be 254 followed by 255. This is the Unicode byte-order marker U+FEFF, which indicates the UTF16BE encoding. This means a PDFDocEncoding string can't begin with þ (254) followed by ÿ (255), but this is unlikely to occur in any reasonable circumstance.

Dates

The creation and modification dates `/CreationDate` and `/ModDate` in the document information dictionary are examples of the PDF date format, which encodes a date in a string, including information about the time zone.

A date string has the format:

```
(D:YYYYMMDDHHmmSSOHH'mm')
```

where the parentheses indicate a string as usual. The other parts of the date are summarized in Table 4-6.

Table 4-6. PDF date format constituents

Portion	Meaning
YYYY	The year, in four digits, e.g., 2008.
MM	The month, in two digits from 01 to 12.
DD	The day, in two digits from 01 to 31.
HH	The hour, in two digits from 00 to 23.
mm	The minute, in two digits from 00 to 59.
SS	The second, in two digits from 00 to 59.
O	The relationship of local time to Universal Time, either +, – or Z. + signifies local time is later than UT, – earlier, and Z equal to Universal Time.
HH'	The absolute value of the offset from Universal Time in hours, in two digits from 00 to 23.
mm'	The absolute value of the offset from Universal Time in minutes, in two digits from 00 to 59.

All parts of the date after the year are optional. For example, (D:1999) is perfectly valid. Plainly, though, if you omit one part, you must omit everything which follows, otherwise the result would be ambiguous. The default values for DD and MM is 01, for all other parts, the default is zeros.

For example:

```
(D:20060926213913+02'00')
```

represents September 26th 2006 at 9:39:13 p.m, in a time zone two hours ahead of Universal Time.

Putting it Together

This is a manually-created text, to be processed into a valid PDF file by *pdftk* using the method introduced in Chapter 2. It is a three page document, with document information dictionary and page tree. Figure 4-3 shows this document displayed in Acrobat Reader. Figure 4-4 is the corresponding object graph.

Example 4-3. A three page document with document information dictionary

```
%PDF-1.1 Header
1 0 obj Top-level of page tree: has two children—page one and an intermediate page tree nod
<< /Kids [2 0 R 3 0 R] /Type /Pages /Count 3 >>
endobj
4 0 obj Contents stream for page one
<< >>
stream
1. 0.000000 0.000000 1. 50. 770. cm BT /F0 36. Tf (Page One) Tj E
endstream
endobj
2 0 obj Page one
<<
  /Rotate 0
  /Parent 1 0 R
  /Resources
    << /Font << /F0 << /BaseFont /Times-Italic /Subtype /Type1 /T
  /MediaBox [0.000000 0.000000 595.275590551 841.88976378]
  /Type /Page
  /Contents [4 0 R]
>>
endobj
```

```
5 0 obj  Document catalog
<< /PageLayout /TwoColumnLeft /Pages 1 0 R /Type /Catalog >>
endobj
6 0 obj  Page three
<<
  /Rotate 0
  /Parent 3 0 R
  /Resources
    << /Font << /F0 << /BaseFont /Times-Italic /Subtype /Type1 /Ty
  /MediaBox [0.000000 0.000000 595.275590551 841.88976378]
  /Type /Page
  /Contents [7 0 R]
>>
endobj
3 0 obj  Intermediate page tree node, linking to pages two and three
<< /Parent 1 0 R /Kids [8 0 R 6 0 R] /Count 2 /Type /Pages >>
endobj
8 0 obj  Page two
<<
  /Rotate 270
  /Parent 3 0 R
  /Resources
    << /Font << /F0 << /BaseFont /Times-Italic /Subtype /Type1 /Ty
  /MediaBox [0.000000 0.000000 595.275590551 841.88976378]
  /Type /Page
  /Contents [9 0 R]
>>
endobj
9 0 obj  Content stream for page two
<< >>
stream
q 1. 0.000000 0.000000 1. 50. 770. cm BT /F0 36. Tf (Page Two) Tj
1. 0.000000 0.000000 1. 50. 750 cm BT /F0 16 Tf ((Rotated by 270 d
endstream
endobj
7 0 obj  Content stream for page three
<< >>
stream
1. 0.000000 0.000000 1. 50. 770. cm BT /F0 36. Tf (Page Three) Tj
endstream
```

```
endobj
10 0 obj  Document information dictionary
<<
  /Title (PDF Explained Example)
  /Author (John Whitington)
  /Producer (Manually Created)
  /ModDate (D:20110313002346Z)
  /CreationDate (D:2011)
>>
endobj xref
0 11
trailer  Trailer dictionary
<<
  /Info 10 0 R
  /Root 5 0 R
  /Size 11
  /ID [<75ff22189ceac848dfa2afec93deee03> <75ff22189ceac848dfa2afe
>>
startxref
0
%%EOF
```

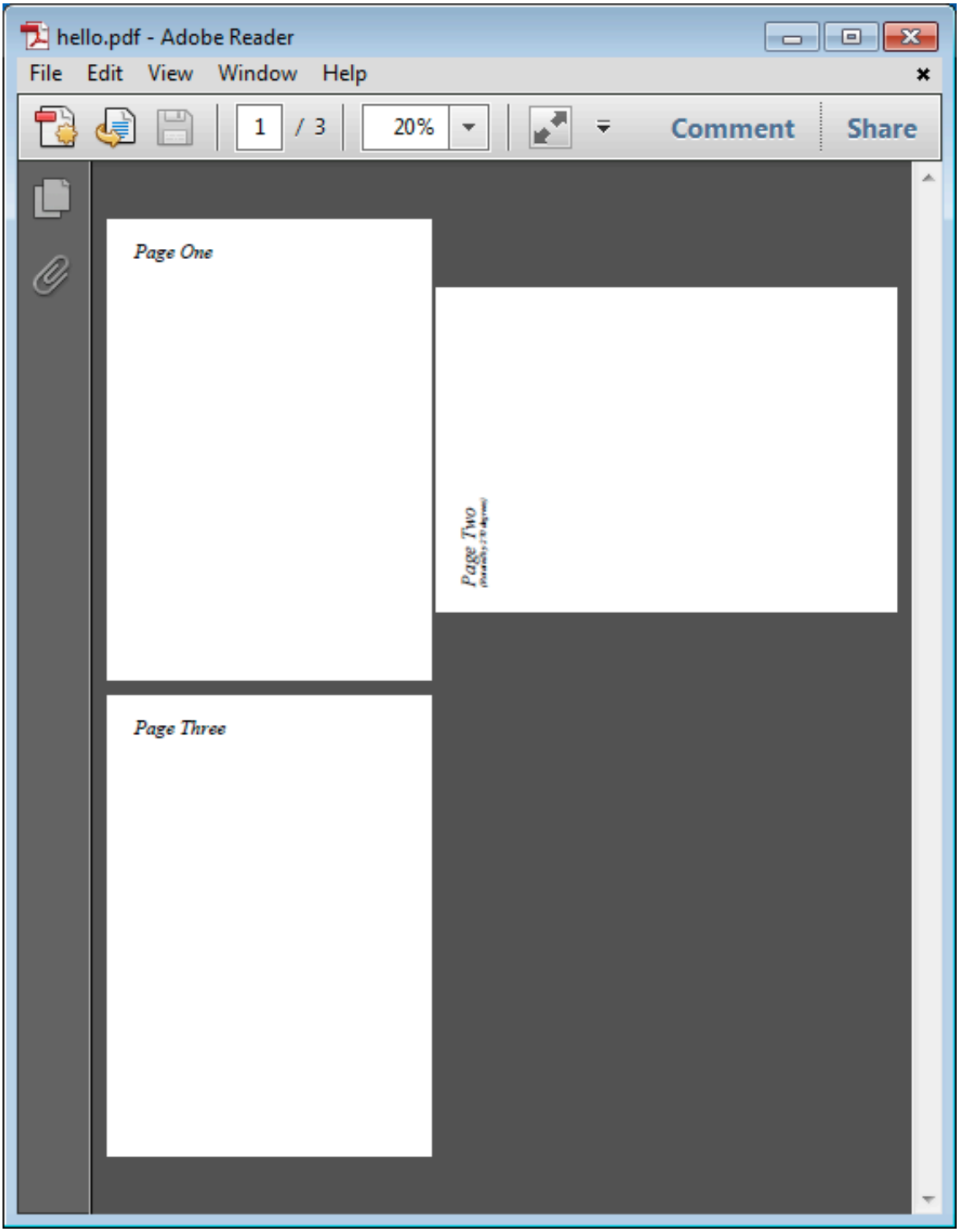

Figure 4-3. Example 4-3 converted to a valid PDF with pdftk and displayed in Acrobat Reader

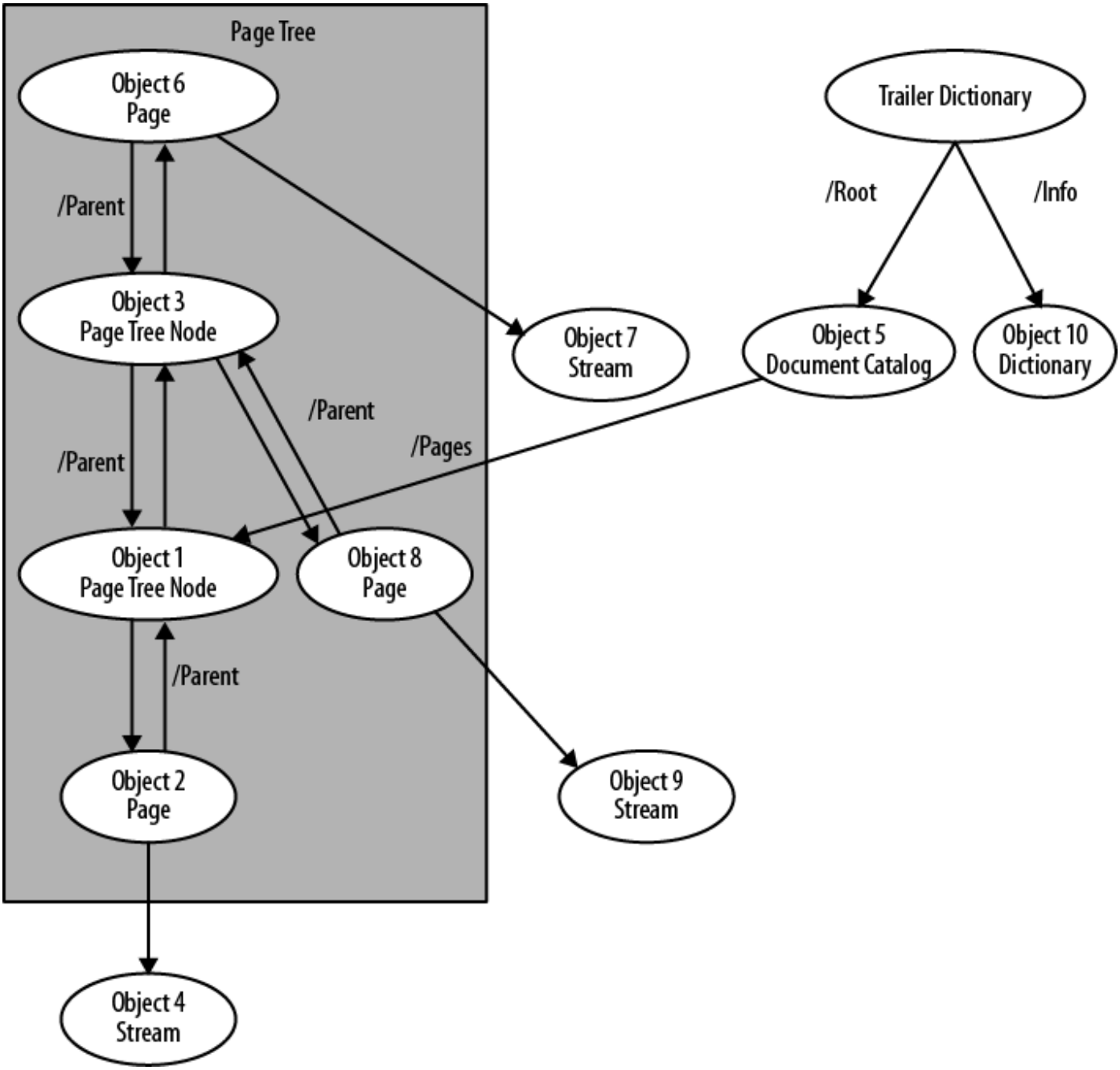

Figure 4-4. Object graph for Example 4-3

Chapter 5. Graphics

In this chapter, we'll run through the main ways to build graphics in the content stream of a PDF page. All of the examples are based on the same PDF we created manually in Chapter 2 and processed into valid PDF documents with *pdftk* in the same fashion. All the examples are included in the online resources.

Looking at Content Streams

A PDF page is made up of one or more *content streams*, defined by the `/Contents` entry in the page object, together with a shared set of resources, defined by the `/Resources` entry. In all our examples, there

will only be a single content stream. Multiple content streams are equivalent to a single stream containing their concatenated content.

Here's an example page, with no resources and a single content stream:

```
3 0 obj
<<
  /Type /Page
  /Parent 1 0 R
  /Resources << >>
  /MediaBox [ 0 0 792 612 ]
  /Rotate 0
  /Contents [ 2 0 R ]
>>
endobj
```

Here's the associated content stream, consisting of the *stream dictionary* and the *stream data*.

```
2 0 obj
<< /Length 18 >> Stream dictionary
stream
200 150 m 600 450 l S Stream data
endstream
endobj
```

We'll discover what the `m`, `l` and S operators do in a moment. The numbers are measurements in *points*—a point (or pt) is 1/72 inch. The result of loading this document into a PDF viewer (after processing with *pdftk* as per Chapter 2) is shown in Figure 5-1.

The full manually created file (before processing with *pdftk*) is shown in Example 5-1. We're going to be using variations on this file for the rest of this chapter. For the ...

Chapter 6. Text and Fonts

In the previous chapter, we saw how a series of graphics operators can be used to draw content on a page, by reference to their operands and a stack-based graphics state.

In this chapter, we look at the operators and state for selecting characters from fonts and printing them on the page. Then, we see how fonts and their metrics are defined and embedded in PDF documents. Finally, we discuss the complex task of general-purpose text extraction from a document.

Text and Fonts in PDF

It would be possible to define a page description language where none of the text layout had been performed, and plain text was supplied along with boxes and columns to be filled on-the-fly, just like a desktop publishing package. Conversely, it would be possible to define a page description language without fonts or text as such at all, just relying on text being converted to outline shapes as the document is produced, having been layed out in, for example, a word-processor.

PDF adopts a middle ground—the ideas of a font and of small-scale text layout are retained, but the large-scale paragraph layout must be done in advance. This has the following advantages:

- Complete control over layout, because large-scale layout (paragraphs, line-breaks) are the job of the program producing the PDF. The document will look as it is supposed to.

- Predictable small-scale text layout, such as fixed character spacing, is supported, so the position of each character need not be explicitly stated.

- Space ...

Chapter 7. Document Metadata and Navigation

In this chapter, we discuss four topics related not to the visual appearance of a PDF document, but to the ancillary data which may also be included for interactive, onscreen use of documents, and the metadata used to carry extra information with a document for use by programs in a PDF workflow.

Destinations

Data structures defining a position within a file. They can be used to specify where a *bookmark* or *hyperlink* points to.

Bookmarks (properly called the *document outline*) are used as a table of contents for the document.

XML Metadata

A stream containing an XML file in a specified format, containing some of the same metadata as the document information dictionary, together with additional fields.

File Attachments

Allow whole files to be encapsulated in a document, much like an email attachment.

Annotations

Allow text and graphics to be applied on top of a PDF page, separate from the main page content, for display by onscreen readers. One particular kind of annotation is the *hyperlink*, which allows a user to click somewhere on a page and be redirected to a destination elsewhere in the file.

Bookmarks and Destinations

A document's *bookmarks* (properly called the *document outline*) are a tree of entries (typically titles of chapters, sections, paragraphs etc.) which can be clicked on in a PDF viewer to move around the document. Each entry has some text and a *destination* describing where it links to.

Destinations

A destination defines a place ...

Chapter 8. Encrypted Documents

PDF documents can be encrypted using a variety of industry-standard schemes which have increased in complexity and security over the years, starting with PDF version 1.1. The PDF standard provides, in addition, a general mechanism for encapsulating third-party encryption and security policies.

Encryption applies, with a few exceptions, to streams and strings in the file, but does not encrypt numbers or other PDF data types, nor does it encrypt the file as a whole. Thus, the document's object structure remains visible to applications without the need for decryption, but the substantive content of the document is safeguarded.

The more modern PDF encryption methods allow the file's XMP metadata stream (XML Metadata) to be left unencrypted so it may be extracted and read by programs which don't know how to open encrypted PDF files, or if the password is not known.

Introduction

Due to the complexity of encrypted documents, it isn't possible to manually build an example (as we have in other chapters), but we can use *pdftk* to process our standard *hello.pdf* file into an encrypted one, *encypted.pdf*:

```
pdftk hello.pdf output encrypted.pdf encrypt_40bit owner_pw fred
```

This creates the output file *encrypted.pdf* using the 40-bit RC4 method with an owner password of "`fred`". The *owner password* is the master password for the file. Someone who has it can do anything with the file, including re-encrypting it or changing the security settings. The *user password* allows ...

Chapter 9. Working with Pdftk

Pdftk is a multiplatform command-line tool built on the *iText* library (which is described in iText for Java and C#). It has facilities for merging, splitting, and stamping documents, and for setting and reading metadata.

OBTAINING PDFTK

Pdftk is an open source program, licensed under the GPL. Binary packages for Microsoft Windows and Mac OS X, and source code for all platforms can be found at PDF Labs.

The creator of *pdftk*, Sid Steward, is also the author of O'Reilly's *PDF Hacks*—a collection of tools and tips for working with PDF.

Command Line Syntax

Pdftk has a somewhat unusual command-line interface, where elements often have to appear in a particular order. We can split them into four groups, in the order they are specified:

1. The input file or files, and possible input passwords.
2. The *operation* and any arguments it requires.
3. The output and any output passwords and permissions.
4. Sundry output and other options.

The full details can be found in the manual for *pdftk*—in this chapter, we give only the subset needed for our examples.

Merging Documents

To merge documents, we use the `cat` operation. This is the default operation, so we don't actually need to specify the `cat` keyword. For example, to merge the pages of three files into one, in order, we need:

```
pdftk file1.pdf file1.pdf file3.pdf output output.pdf
```

This writes a new file to *output.pdf* containing all the pages of *file1.pdf*, *file2.pdf*, and *file3.pdf*, in order. The output file may not be the same as any of the input ...

Chapter 10. PDF Software and Documentation

In this chapter we list and describe software for viewing, converting, editing, and programming with PDF files. We consider both open source software, and zero-cost commercial software where it is provided by Adobe or operating system manufacturers. There is a large variety of commercial software from third parties, which we do not discuss here.

We also list sources of further documentation and information.

PDF Viewers

The job of a PDF viewer is to:

- Display the graphical and textual content of the document.
- Allow the user to interact with the document using bookmarks and hyperlinks.
- Enable searching of the textual content, and extraction of text via cut and paste.

Not every viewer has all of these features. Due to the huge complexity of the PDF format and the formats it encapsulates (for example, fonts and images), performance can vary significantly—especially on files using more modern PDF features.

Adobe Reader

Adobe Reader is Adobe's own, free PDF viewer and the only one guaranteed to support most if not all of the features of PDF. It comes with a PDF plug-in for common web browsers, and is available for Microsoft Windows, Mac OS X, Linux, Solaris, Android, and iOS. It allows forms to be filled in and submitted electronically.

Adobe Reader can be found at Adobe's website.

Preview

Many Mac OS X users prefer the fast, simple PDF viewer Preview, provided with the operating system. It launches more quickly, and is smoother in use than Adobe Reader, ...