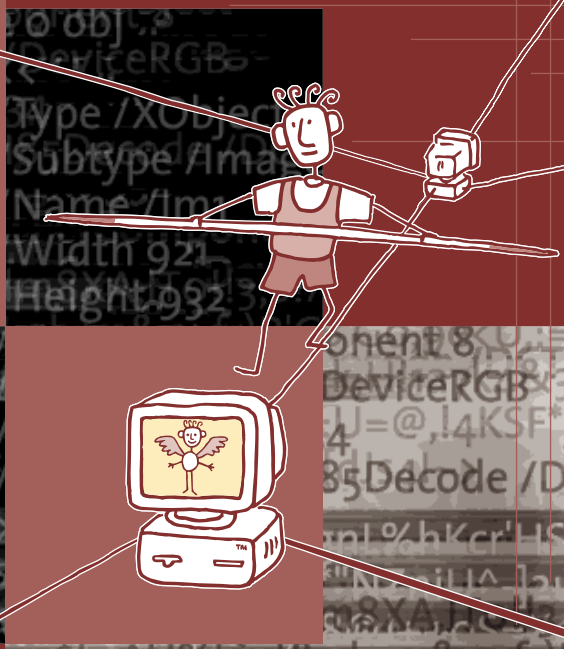
Thomas Merz

Web Publishing with Acrobat/PDF



with CD-ROM

Springer

Thomas Merz
Tal 40
80331 München
Germany

Title of the Original German Edition:
Mit Acrobat ins World Wide Web
© Thomas Merz Verlag 1997

CIP-Data applied for

Die Deutsche Bibliothek – CIP-Einheitsaufnahme
Web Publishing with Acrobat, PDF / Thomas Merz. – Berlin ; Heidelberg ; New York ; Barcelona ; Budapest ; Hong Kong ; London ; Milan ; Paris ; Santa Clara ; Singapore ; Tokyo : Springer.
Einheitssacht.: Mit Acrobat ins World: DM 69.00

Buch. – 1998
CD-ROM. – 1998

ISBN 3-540-63762-1 Springer-Verlag Berlin Heidelberg New York

This work consists of a printed book and a CD-ROM packaged with the book, and is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag or the author make no warranty of representation, either express or implied with respect to this CD-ROM or book, including their quality, merchantibility, or fitness for a particular pupose. In no event will Springer-Verlag or the author be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the CD-ROM or book, even if Springer-Verlag or the author have been advised of the possibility of such damages.

© Springer-Verlag Berlin Heidelberg 1998
Printed in Germany

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Camera ready by author
Cover design and illustrations: Alessio Leonardi, leonardi.wollein, Berlin
Translated from German by Richard Hunt, Tadcaster, UK, and the author
SPIN 10657710 Printed on acid-free paper 33/3142 – 5 4 3 2 1 0

The Websurfer's Point of View

1 HTML and PDF 2

1.1 What is HTML? *2*
1.2 What is PDF? *4*
1.3 Comparison of HTML and PDF *6*

2 PDF in the Browser 9

2.1 Web Browsers and Acrobat *9*
2.2 Optimized PDF and Page-at-a-Time Download *14*
2.3 Using PDFs in the Browser *17*
2.4 Conversion from HTML to PDF *19*
2.5 Conversion from PDF to HTML *20*

The Publisher's Point of View

3 Planning PDF Documents 26

3.1 PDF or HTML? *26*
3.2 PDF Hypertext Features *27*

4 Creating PDF Files 35

4.1 Several Paths to PDF *35*
4.2 Distiller Settings *39*
4.3 Optimized PDF Files *46*
4.4 Full Text Retrieval *48*
4.5 Encrypting PDF Files *51*
4.6 Acrobat Plugins *53*
4.7 Testing PDF Files *55*

5 PDF Support in Applications 57

5.1 PDF-savvy Applications *57*
5.2 Adobe FrameMaker *59*
5.3 Adobe PageMaker *65*
5.4 QuarkXPress *67*
5.5 Microsoft Word *68*
5.6 TEX *70*
5.7 Graphics Programs *71*

6 pdfmark Primer 75

6.1 Overview *75*
6.2 Preliminaries *77*
6.3 Application-specific Embedding Tricks *86*
6.4 Basic pdfmark Functions *89*
6.5 Destinations and Actions *102*
6.6 Additional Tips for Distilling *117*

7 PDF Forms 119

7.1 Form Features *119*
7.2 Form Fields *122*
7.3 Tips on Creating Form Fields *126*
7.4 PDF Forms on the World Wide Web *130*
7.5 Personal Field Names (PFN) *133*
7.6 Acrobat Forms *136*

8 PDF in HTML Pages 145

8.1 Navigator and Internet Explorer *145*
8.2 Embedding PDF in HTML *151*
8.3 VBScript Programming for PDFs *156*
8.4 Common HTML Code for all Browsers *160*
8.5 HTML Authoring Tools *163*
8.6 Navigation *168*

The Webmaster's Point of View

9 PDF on the Web Server 176

9.1 MIME Types and PDF Icons *176*
9.2 The Byterange Protocol *179*
9.3 PDF on SSL Servers *185*

10 Form Data Processing 187

10.1 FDF and the FDF Toolkit *187*
10.2 FDF and the CGI Interface *191*
10.3 FDF and Active Server Pages *194*
10.4 Software for PDF Forms *197*

11 Full Text Retrieval and Search Engines 201

11.1 Motivation *201*
11.2 Microsoft Index Server and PDF IFilter *203*

11.3 The Highlight File Format *211*

12 Dynamic PDF 213

12.1 Dynamic Web Pages *213*
12.2 Dynamic PDF Generation *214*
12.3 The PDFlib C Library *216*

A Contents of the CD-ROM 225

B PDF-related Web Resources 227

Index *229*

6 pdfmark Primer

6.1 Overview

This chapter is devoted to pdfmark programming. The pdfmark operator is a PostScript extension which is only implemented in Acrobat Distiller (as opposed to PostScript printers). Using this operator, many non.layout-related features of a PDF file can be defined in the original document or in the corresponding PostScript code. Why bother with pdfmarks since you can implement these features in Acrobat Exchange? Contrary to adding hypertext features manually in Exchange, the pdfmark method has a big advantage in that you don't have to redo all links and other special effects when document changes require generating a new PDF version. Instead, the hypertext features are automatically generated when distilling the PostScript file. It is very important to know that pdfmark instructions are processed in Acrobat Distiller only, but not in PDF Writer.

PostScript programming basics are quite helpful when you're working with pdfmark. In this chapter, however, I'll try to explain how to explore the power of pdfmark applications without any programming experience. Although this chapter is filled with lots of gory details – mostly stuffed into tables – you should be able to deal with many applications by simply using or adapting one of the examples. Unless to wish to make use of the more advanced features or additional options, you can get by without looking into the tables or the accompanying descriptions.

This chapter is based on Adobe's *pdfmark Reference Manual* which can be found on the Acrobat CD-ROM. However, many features and details can only be understood by additionally delving into the *Portable Document Format Reference Manual* (to be found on Adobe's Web server). If you consider yourself to be a serious Acrobat user, I definitely recommend reading (or at least glancing over) these manuals. There you will find additional details which I do not cover here.

This chapter isn't meant to replace the Adobe manuals (in fact, it surely can't). Instead, I'll try to tame the very technical contents by introducing many directly usable examples and also present additional information not documented in Adobe's manuals. And there are lots of undocumented features! While doing research for this chapter, I not only discovered undocumented pdfmark instructions (e.g., for executing Acrobat menu functions), but also PDF code generated by Acrobat software but not covered in the reference manual (e.g., assigning an index file to a PDF document). Looking at it the other way round, there are PDF features which are documented in the reference but which are nevertheless inaccessible with Acrobat software. The only way of using these features is by generating the re-

Table 6.1. Overview of all pdfmark functions described in this chapter

Page	Function
90	Notes
90	Links
91	Bookmarks
92	Article threads
92	Named destinations
93	PostScript instructions
94	Page cropping
94	General document information
95	Viewer properties
96	Page transitions
97	Viewer preferences
98	Encapsulating graphics
100	Page open actions
100	Creating form fields
99	Attaching an index file
104	Linking to a page in the same document
105	Link to another PDF document
106	Launching another document or application
107	Linking to a named destination
107	Linking to a file on the WWW
108	Defining a document's base URL
108	Inserting JavaScript instructions
110	Linking to an article
110	Playing sound and video files
112	Hiding and showing fields
112	Submitting a form to a URL
113	Resetting a form to its default values
113	Importing form data from a file
114	Executing menu items

spective pdfmark operators (e.g., named destinations; these are link targets labeled with a symbolic name, see Section 8.6). But you should take care when using such features – undocumented features usually don't warrant any support by the manufacturer of the software.

So you ask yourself why an author or editor should bother with technical details such as pdfmark programming? You're definitely on the right track. In my not so humble opinion, pdfmark only serves as a temporary kludge as long as application software is unable to generate the necessary pdfmarks automatically. In Chapter 5 you can find an overview of the current status of pdfmark support in several important application programs.

Function overview. Using pdfmarks, you can define a wealth of Acrobat features in the PostScript code. This chapter presents many functions along with working examples and extensive explanations. To avoid your getting lost in a plethora of samples, Table 6.1 gives an overview of all pdfmark features covered in this chapter.

6.2 Preliminaries

Embedding pdfmarks in the PostScript code. Before delving into pdfmark descriptions I'd like to present several means of including pdfmark statements in a document's PostScript code. The techniques covered in this chapter do not relate to certain application software. Chapter 5 has already covered automatic pdfmark generation in application programs; Section 6.3 talks about application-specific methods for embedding pdfmarks in the PostScript stream.

As you can see in Figure 6.1, several components are involved in creating the PostScript code. The exact number and kind of these components not only varies with the operating system in use, but also with the kind of application software.[1] The location at which to embed pdfmarks in the PostScript generating chain depends on the scope of the feature to be realized by pdfmarks. Consider the following examples:

- A URL link is expected to show up on a certain page only.
- Automatically attaching an index file may affect a certain file or several files in a group.
- It may be desirable for all files created with a certain program to contain document info fields with appropriate contents.
- The name of the creator (the person, not the program) may be inserted in the document info fields of all generated files, independently of the program used to create the file.

The following pages present important ways of embedding pdfmarks along with some examples. The remaining sections of this chapter will explain in more detail how these examples work. For now we will only consider how to embed pdfmark statements. The embedding techniques presented here require varying degrees of experience on the user's part.

Protecting your printout. I already mentioned the fact that the pdfmark operator is only implemented in Acrobat Distiller and not in any type of PostScript printer. For this reason, using pdfmarks implies getting PostScript errors when you try to distill and print the very same PostScript files (which is normally the case). This problem can be solved with a couple of PostScript statements which don't have any effect in Distiller but cancel the

1. You can find much more information on these topics in my book "PostScript & Acrobat/PDF – Applications, Troubleshooting, and Cross-Platform Publishing" (Springer-Verlag 1997).

pdfmark operator when printing the document. To achieve this effect I recommend placing the following line of PostScript code in front of any pdfmark operator, or – better still – in modified PostScript prolog:

```
/pdfmark where {pop} {userdict /pdfmark /cleartomark load put} ifelse
```

To achieve the protection it suffices to include these statements once at the beginning of the PostScript code – this cancels all forthcoming pdfmarks when printing the document.

Another source of error is related to older printers equipped with a PostScript Level 1 interpreter: pdfmark sequences often make use of the << and >> operators which are only defined in Level 2. On Level 1 devices these two operators give rise to a syntax error – even if you followed the above advice and included the protection line! However, it's not very difficult to deal with this problem by adding a few more PostScript lines before using any pdfmark statement:

```
/pdfmark where {pop} {userdict /pdfmark /cleartomark load put} ifelse
/languagelevel where {pop languagelevel}{1} ifelse
2 lt {
    userdict (<<) cvn ([) cvn load put
    userdict (>>) cvn (]) cvn load put
} if
```

Including pdfmarks in the native document. Often it is more convenient to define pdfmark statements directly in your native DTP or word processor document. This requires application software which is capable of embedding user-defined PostScript code, or which at least offers some feature which may be (mis-) used for this purpose. Obviously, it's not sufficient to write the pdfmark statements in the document's text since the text is getting printed instead of being interpreted as PostScript code.

Two samples of nice embedding features are FrameMaker's PostScript frames and Microsoft Word's print fields. You may find similar functions in other programs, although these functions originally may have served a completely different purpose. According to the embedding technique, additional information for use in pdfmarks may be available, e.g., the enclosing text frame's coordinates which may be used for defining a link rectangle. You can find a more detailed explanation of embedding pdfmarks for several programs in Section 6.3.

Startup directory of Acrobat Distiller. Before it processes a document's actual page descriptions, Acrobat Distiller interprets all files found in its startup directory. This gives an easy way to activate or deactivate certain features by simply moving the respective PostScript file in or out of the startup directory. Note that startup files are only processed once at Distiller's launch time. For this reason, they generally remain active during Distiller's life time, independently of the number of files processed. Unfortu-

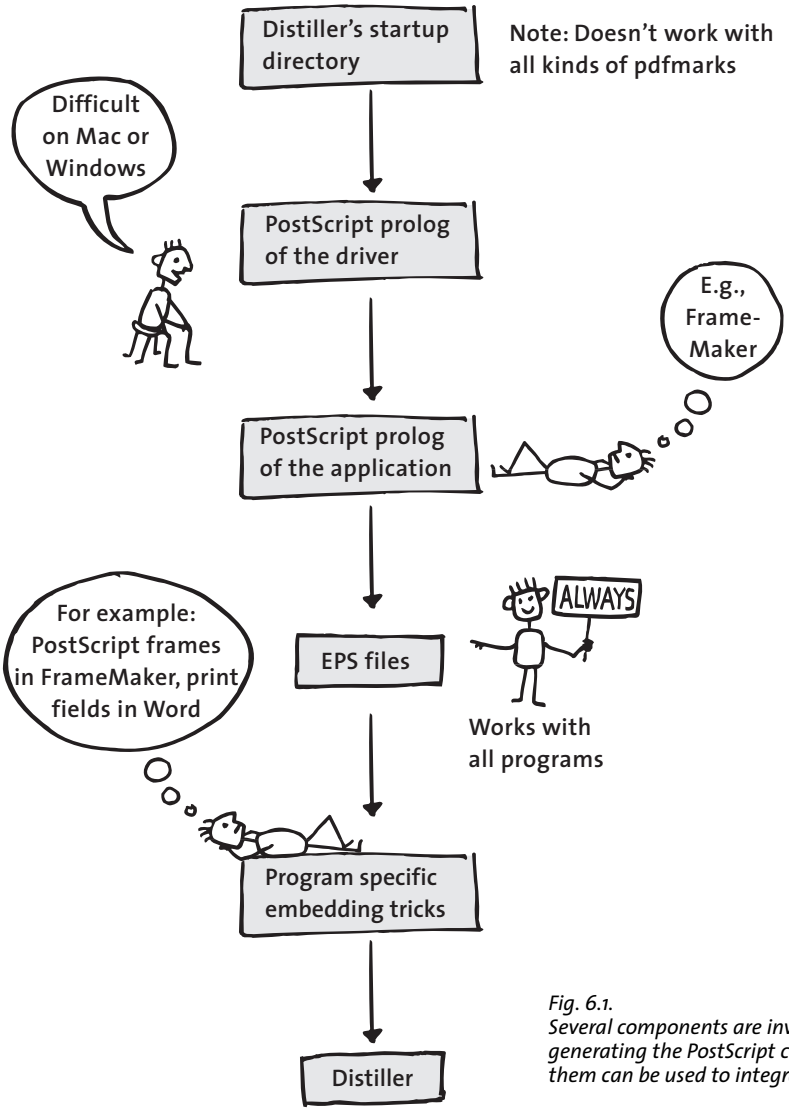


Fig. 6.1.
Several components are involved in generating the PostScript code. Some of them can be used to integrate pdfmarks.

nately, this also means that the document-related pdfmark stuff doesn't work. An example that works in the startup directory is the canceling of specific pdfmark variations as described on page 83.

To avoid confusion, I recommend displaying a descriptive message according to the following pattern:

```
(Article threads deactivated!\n) print flush
```

This way you can easily see which startup files are active when launching Distiller.

A note of warning: all files in Distiller's startup directory are loaded in an undocumented order. If you edit a file and your text editor leaves a backup file in this directory, both files will be loaded. If the old file is loaded after the new one, the old values will be used!

EPS files. Graphics files in the Encapsulated PostScript (EPS) format are supported within all major text, DTP, and graphics applications. By definition, they provide a means of embedding PostScript code in the document. EPS files therefore offer great opportunities for including pdfmark operators. Aside from being supported in all current programs, this method has the additional advantage of very easy handling via a program's "import" or "place". The disadvantage is an EPS file's rather limited scope: EPS files don't act globally, but are restricted in operation to one particular page. In some cases, however, it's possible to bypass this restriction. Another complication is that of the formal requirements for EPS files which have to be obeyed for pdfmarks too. Since EPS files don't know anything about their location on the page they can't use absolute coordinates within the page.

Sounds complicated? In many cases embedding pdfmarks in EPS is as easy as using the following template and simply adjusting the contained pdfmark operators:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 72 72
%%EndProlog
/pdfmark where {pop} {userdict /pdfmark /cleartomark load put} ifelse
[ {ThisPage} << /Trans << /S /Dissolve >> >> /PUT pdfmark
%%EOF
```

This dummy EPS file defines a size of 72 × 72 points (equalling 1 inch = 2.54 cm) within the "%%BoundingBox" comment. This is the EPS graphic's size in the document. The graphic can be arbitrarily positioned on the page. Since the EPS doesn't produce any printout, it doesn't matter where it is placed on the page. Generally, you can place it somewhere near the edge of the page in order not to disturb the layout.

Note that the EPS sample above already includes the additional line of PostScript code for eliminating PostScript errors when printing the document.

You can further reduce the size of the graphic in order to make it appear less prominent. The following line:

```
%%BoundingBox: 0 0 1 1
```

defines a size of 1 × 1 points, making the graphic nearly disappear from the screen. However, this small size makes it close to impossible to click the graphic with the mouse if you want to move or delete it.

pdfmark operators relating to a rectangular area – for example links, notes, or form fields – should use the BoundingBox's coordinates as values in the /Rect array. This simplifies changing the link's size by simply changing the size of the embedded EPS.

There is, however, a small glitch related to pdfmarks in EPS "graphics": you won't be able to see any screen preview of the EPS. But what should a hypertext function's preview look like anyway?

PostScript prolog of operating system or application. The PostScript prolog is a set of PostScript procedures loaded ahead of the actual page descriptions by the operating system, printer driver, or application program. The prolog is required for successfully printing the document and may also be modified to include pdfmark operators.

Firstly, you have to check whether the driver's or program's prolog is accessible at all or hidden deeply in the driver, only surfacing in the final PostScript output. Note some samples:

- In addition to the printer driver's prolog, FrameMaker for Windows uses another small prolog which can be found in the program's init directory. In Unix – which doesn't have any system-wide printer drivers – FrameMaker uses the prolog in the file *ps_prolog*.
- As an optimization, some PostScript drivers allow generating the prolog separately, and later create the print data without any prolog. The prolog must of course be available when printing the document. For PDF conversion, it may be possible to adjust the prolog.

When patching prolog files, note that pdfmark operators relate to all documents being distilled with the manipulated prolog. This is not always what you want.

Windows NT separator pages. Windows NT offers a hook for including custom instructions in the printer output stream. This feature is called separator pages because its most prominent use is to print exactly that: informative sheets between individual print jobs. In order to use separator pages for pdfmarks, we have to adhere to the syntax of the separator pages interpreter – the contents of a separator file are not simply copied to the output stream but may also contain certain variables. This means you have to insert a couple of special characters into your pdfmark code before installing it as a separator page.

As explained above, we include the printout protecting code (only to be sure) and the familiar "%!PS-Adobe-3.0" line (because our code will be sent ahead of all other PostScript instructions). For example, to define general document information in a separator page, create a file named *docinfo.sep* which contains the following lines:

```
@
@L%!PS-Adobe-3.0
@L/pdfmark where {pop} {userdict /pdfmark /cleartomark load put} ifelse
@L[ /Title (User Manual)
@L  /Author (Michael Heinzel)
@L  /Subject (Adjusting the electronics of MH-screen)
@L  /Keywords (screen display MH-screen)
@L  /Creator (DocMaker 1.0)
@L  /ModDate (D:19980110205731)
@L/DOCINFO pdfmark
```

The "@" character in the first line defines the special character used in this file, "@L" instructs the driver to include the entire line in the PostScript output without changing it. To make use of this separator page, select "Start", "Settings", "Printers", right-click your PostScript printer driver, choose "Properties", and click the "General" tab. In this menu, click "Separator Page..." and browse to locate the file defined above. (NT's default separator pages live in \winnt\system32*.sep.) The selected separator page is included in your PostScript output as long as you don't deselect it in the printer driver settings.

Note that PostScript drivers for Hewlett-Packard printers include some PJL code at the start of the output stream which does not contain PostScript. In order to avoid problems with Distiller, I recommend selecting a non-HP printer.

PPD files. Most current PostScript drivers are configured via PPD (PostScript printer description) files. The drivers read the PPD file in order to find out a device's features, and the PostScript code to activate these features. Therefore, such drivers may be instructed to include pdfmarks via the PPD file. Since a PPD may also configure the driver's user interface, PPDs may be used for constructing a convenient method for including pdfmarks. However, manipulating PPD files is an error-prone process which may result in damaged PostScript output or the driver not working any more. For this reason I only mention this method for specialists without exploring further details.

Post-processing the PostScript code after generation. In some cases it's reasonable to completely generate the PostScript files and do some processing afterwards. Automated text processing tools like Unix's sed and awk or the Perl programming language are quite useful for this task. Note that most text processing tools can't deal with binary data. For this reason, the PostScript data have to be generated in ASCII format to allow post-processing.

The method isn't suited for all pdfmark applications and may require a good deal of work for implementing it. As an example, let's define an article thread for which the columns (beads) have identical widths on each page.

This task may be achieved by inserting the following line at the beginning of each page's PostScript code:

```
[ /Title (A) /Rect [ 100 100 500 700 ] /ARTICLE pdfmark
```

Since the pages in PostScript files are separated by "%%Page:" comments (at least in DSC conforming PostScript files), it's easy to accomplish this. The following sed script implements the task on Unix systems (it requires the "%%EndPageSetup" comment to be present):

```
/%%EndPageSetup/a\
[ /Title (Main text) /Rect [100 100 500 800 ] /ARTICLE pdfmark
```

Canceling certain pdfmark operators. In some cases it may be useful to cancel certain pdfmarks which are automatically generated by an application, but which are not wanted. For example, FrameMaker (up to version 5.1.x) generates pdfmarks for bookmarks, links, and article threads. It's impossible to selectively activate or deactivate these features – you get either all or none. If you want to get rid of automatically generated article threads without sacrificing links and bookmarks, you can use the following code to cancel a single kind of pdfmarks (in this case the article feature):

```
/pdfmark where { pop
    /_origpdfmark /pdfmark load def
    /pdfmark {
        dup /ARTICLE eq {
            cleartomark
        }{
            _origpdfmark
        } ifelse
    } bind def
} if
```

Data types for pdfmarks. Since pdfmarks are part of the PostScript code they share data types and syntax with the page description language. In order to spare you reading PostScript programming books, I'll try to briefly explain the most important data types used in pdfmark instructions. You can separate instructions with arbitrary numbers of spaces, tabs, or line-end characters. Note that case is significant in PostScript and pdfmark programming.

Concerning *integers* and *floating point numbers* there's nothing more to note except that they work as expected. *Boolean values* can take on the values "true" or "false". An *array* is an arbitrary long collection of (possibly different) data types delimited by square brackets:

```
[ /XYZ null null null ]
```

A *name* (don't confuse names with the strings presented below) is an identifier for a function or a parameter, always starting with a slash "/". Names can be up to 127 characters long (including the leading slash). Names

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017
1	□	□	□	□	□	□	□	□	˘	ˇ	ˆ	˙	˝	˛	˚	˜
	020	021	022	023	024	025	026	027	030	031	032	033	034	035	036	037
2		!	"	#	$	%	&	'	(	)	*	+	,	–	.	/
	040	041	042	043	044	045	046	047	050	051	052	053	054	055	056	057
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	060	061	062	063	064	065	066	067	070	071	072	073	074	075	076	077
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	100	101	102	103	104	105	106	107	110	111	112	113	114	115	116	117
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	120	121	122	123	124	125	126	127	130	131	132	133	134	135	136	137
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	140	141	142	143	144	145	146	147	150	151	152	153	154	155	156	157
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	□
	160	161	162	163	164	165	166	167	170	171	172	173	174	175	176	177
8	•	†	‡	…	—	–	ƒ	⁄	‹	›	−	‰	„	“	”	‘
	200	201	202	203	204	205	206	207	210	211	212	213	214	215	216	217
9	’	‚	™	ﬁ	ﬂ	Ł	Œ	Š	Ÿ	Ž	ı	ł	œ	š	ž	□
	220	221	222	223	224	225	226	227	230	231	232	233	234	235	236	237
A	□	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	□	®	¯
	240	241	242	243	244	245	246	247	250	251	252	253	254	255	256	257
B	°	±	2	3	´	µ	¶	·	¸	1	º	»	¼	½	¾	¿
	260	261	262	263	264	265	266	267	270	271	272	273	274	275	276	277
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
	300	301	302	303	304	305	306	307	310	311	312	313	314	315	316	317
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
	320	321	322	323	324	325	326	327	330	331	332	333	334	335	336	337
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
	340	341	342	343	344	345	346	347	350	351	352	353	354	355	356	357
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ
	360	361	362	363	364	365	366	367	370	371	372	373	374	375	376	377

Fig. 6.2. The PDFDocEncoding character set for PDF bookmarks and notes also has to be used in pdfmark strings. The octal codes are shown below each character.

mustn't contain one of the characters %, (,), <, >, [,], {, }, /, or #. Neither must they contain any special characters.

Strings are text in parentheses. Parentheses themselves, line-end characters, or backslashes inside strings must be "escaped" with a leading backslash:

```
(This is a string with two \(2\) parens.)
```

Strings have to obey a certain character set. Unlike the actual page contents, PDF internally uses a fixed character set called PDFDocEncoding (see Figure 6.2) which is different from the Mac or Windows character sets. PDF bookmarks and notes use this character set. Therefore, strings used in pdfmark instructions must also use PDFDocEncoding. Since this character set shares many (though not all) character positions with the Windows character set, you can use many Windows special characters within pdfmarks since their Windows code equals their PDFDocEncoding code.

On the Mac, however, octal notation has to be used for special characters, i.e., the octal value of the character after a leading backslash. Here are several examples:

```
™=\222, ‰=\213, •=\200, «=\253, »=\273, ©=\251, ®=\256
```

You can determine the codes of other characters by looking at the table in Figure 6.2. There, you can also find out whether a certain character is supported in PDFDocEncoding at all. For example, the French word "Télégramme" as a PDF string looks like this:

```
(T\351l\351gramme)
```

Colors in pdfmarks are defined as RGB triples in an array. Translating to plain talk, this means three numbers in the range 0 to 1 for the red, green, and blue components. The following array defines 100 percent blue:

```
[ 0 0 1 ]
```

Dictionaries are data structures containing an arbitrary number of key/value pairs (note that /URI is used as a value in the first pair, and as a key in the third pair):

```
<< /Subtype /URI /IsMap true /URI (http://www.ifconnection.de/~tm) >>
```

Dictionaries are delimited by two angle brackets ("less than" and "greater than" characters – not to be confused with French quotes "«" and "»") on either side. Dictionaries may be nested, requiring the appropriate number of angle brackets. The order of pairs in a dictionary isn't significant. However, the ordering key/value must be strictly obeyed for each pair.

Finally, *comments* may contain remarks on the sometimes otherwise incomprehensible pdfmark instructions. Comments are introduced with a percent character and continue until the end of the line:

```
% The following code defines a bookmark:
[   /Page 1
    /View [/XYZ 44 730 1.0]
    /Title (Start)
/OUT pdfmark
```

In order to keep track of your pdfmark tricks I recommend using comments for all pdfmark instructions. Since all examples in this chapter are explained in the text, however, I will do without comments.

Coordinate system. Many pdfmark instructions involve geometrical coordinates, especially when you define rectangles for a link's active area. pdfmarks use the PostScript coordinate system which has the origin in the lower left corner, the first coordinate increasing to the right, and the second coordinate to the top. The unit of measurement is the well-known DTP point, defined as follows:

```
1 point = 1/72 inch = 25.4/72 mm = 0.3528 mm
```

For example, a U.S. letter page measures 612 x 792 points.

6.3 Application-specific Embedding Tricks

The method described above for embedding pdfmark instructions in EPS files works in all programs which support EPS embedding. In addition, there are some program-specific options for embedding pdfmarks in the native application document which I'd like to present in this chapter.

Adobe FrameMaker. You may already have wondered about the somehow strange "PostScript Code" option in a text frame's properties dialog. In FrameMaker's infancy, this was meant to include graphical tricks to be manually programmed in PostScript (e.g., rotated text – considered highly innovative at that time). Today – since every man and his dog is able to do sexy text and graphics effects (FrameMaker supports rotated text by default) – PostScript frames are rarely used.

Including pdfmarks for preparing PDF conversion puts the PostScript frames back to work in our days. PostScript frames are a simple and convenient way of defining PDF effects in the native Frame document. The advantage is that you can see and edit the pdfmark code in your documents (contrary to the EPS technique). Additionally, FrameMaker makes available the frame's coordinates which may be used for defining a rectangular area for a link or another type of active area, e.g., a form field.

The following sections of this chapter contain tons of pdfmark samples. Pick the appropriate code for your particular PDF feature of interest and type it into a newly created text frame. To make life easier and prevent typos, you may want to open the PDF file of this chapter from the accompanying CD-ROM and cut-and-paste the samples. Take care not to hyphenate or otherwise alienate the pdfmark code. I found it convenient to define a paragraph style called "pdfmark" which uses a small font size and deactivates hyphenation. Now select your text frame with the arrow pointer and click "Graphics", "Object Properties...". This brings up the dialog box shown in Figure 6.3 where you activate the "PostScript Code" check box. The option results in the text being passed through as PostScript instructions instead of being printed as text. Note that you have to de-select the "PostScript Code" option before you can again edit the frame's contents. It's also possible to position PostScript frames on a master page. This is particularly useful, for example, for defining page transitions for PDF presentations.

The PostScript frame's size depends on whether or not the respective PDF feature needs geometrical layout information. For example, document info fields don't relate to a specific location on any page but to the whole document. In this case, place a frame with code similar to the following snippet somewhere near one of the page edges (so that it doesn't disturb your on-screen layout when working on the document):

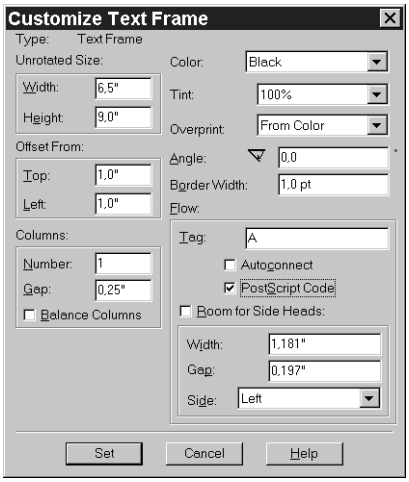


Fig. 6.3.
Defining a PostScript frame in FrameMaker

```
pop pop pop pop
/pdfmark where {pop} {userdict /pdfmark /cleartomark load put} ifelse
[   /Title (Installation Instructions)
    /Author (Thomas Merz)
    /Subject (Preparing and Starting the Engine)
    /Keywords (Introduction Manual Troubleshooting)
/DOCINFO pdfmark
```

Since the four coordinates of the PostScript frame which are supplied by FrameMaker are not needed here, we get rid of them via pop instructions at the beginning of the code.

For links, form fields, and other PDF features the situation is quite different since we need a rectangular area for defining their size. pdfmarks specify this area by means of the /Rect parameter. The following sample defines a URL link which has the size of the PostScript frame as its active area. The PostScript gobbledygook after /Rect is necessary to adapt Frame coordinates to the notation required for pdfmarks:[1]

```
/pdfmark where {pop} {userdict /pdfmark /cleartomark load put} ifelse
[   /Rect [ 7 -4 roll 4 -2 roll pop pop 0 0 ]
    /Action << /Subtype /URI /URI (http://www.ifconnection.de/~tm) >>
    /Subtype /Link
/ANN pdfmark
```

If you want to use pdfmark instructions which make use of /Rect, place /Rect as first parameter after the opening bracket (parameter ordering isn't relevant for pdfmarks) and use the /Rect line shown above instead of the

1. Note to readers fluent in PostScript: FrameMaker resets the coordinate system's origin to the lower left corner of the frame, and pushes the offset of this corner from the page origin on the stack, as well as width and height of the frame.

one with four explicit numbers as used in the other examples in this chapter. If /Rect is missing from the pdfmark description, enter "pop" four times at the beginning of your pdfmark code to delete the FrameMaker coordinates from the PostScript stack.

Note that it's quite useful to include FrameMaker variables inside pdfmark instructions. This is very efficient for repeating commands, such as the author field in the document info. Cross-references used for placing a chapter heading inside an info field also work nicely.

Section 5.2 explains FrameMaker's integrated PDF support which spares you pdfmark programming in many cases.

Microsoft Word. Word too has some means for incorporating your own pdfmark code in a document. Firstly, use "Tools", "Options...", "View" to activate screen display of field codes. This makes life easier in the following steps. Using "Insert", "Field...", "Field name: Print" you can insert pdfmark instructions (surrounded by double quotes) in a dialog box (see Figure 6.4). The following code in a print field defines PDF document information:

```
{print "[ /Title (Manual) /Author (Thomas Merz) /DOCINFO pdfmark "}
```

You will need the coordinates of the position on the page or the size of the paragraph containing the print field in order to use pdfmark instructions which make use of rectangles. The "\p" option tells Word to define some variables which can be useful for defining a link's active area. A complete list of these variables can be found in Word's online help. The most important variable is "wp$box" because it's perfectly suited for defining a rectangle. This variable supplies a PostScript definition of a rectangular shape around the current paragraph. You can use it in a pdfmark instruction as follows:

```
{print \p para "[ /Rect [ wp$box pathbbox ] /Page 3 /Subtype /Link /ANN
pdfmark "}
```

Fig. 6.4. pdfmark code in a Microsoft Word print field

The group instruction "para" results in Word's PostScript relating to the current paragraph. The code above transforms the paragraph containing the print field in a PDF link with a jump to page 3. Remember to include the additional line with the print protection code if you want to print and distill the resulting PostScript file (see Section 6.2).

Finally, here's an example for defining a URL link inside a Word print field:

```
{print \p para "[ /Rect [ wp$box pathbbox ] /Action << /Subtype /URI
/URI (http://www.ifconnection.de/~tm) >> /Subtype /Link /ANN pdfmark "}
```

This code transforms the current paragraph into a Weblink's active area.

Section 5.5 informs about more comfortable ways of generating hypertext-enhanced PDF from Word documents.

TEX. The powerful TEX typesetting system also has a mechanism for including user-defined printer instructions into the output stream. The \special instruction can be used to embed PostScript code. It is also suited for pdfmark programming. The following example defines PDF document information within a (device-driver-dependent) \special sequence:

```
\special{ps::
[   /Title (User Manual)
    /Author (Michael Heinzel)
    /Subject (Adjusting the electronics of MH-screen)
    /Keywords (screen display MH-screen)
/DOCINFO pdfmark
}%
```

The macro package hyperref can automatically insert pdfmarks for hypertext links and for certain document information. You can find more information on creating PDF from a TEX source in Section 5.6.

6.4 Basic pdfmark Functions

Now that we've dealt with the necessary preliminaries, let's take a closer look at the specific instructions. This section and the next one contain a list of all pdfmark instructions, including descriptions of the PDF features they achieve. Since there is a wealth of link and action related options which are used in many pdfmark operators, these options are discussed separately in the next section.

The information presented in this and the next section relates to Distiller 3.0 and higher. Older versions support a subset of these pdfmarks.

There's a common scheme for all descriptions: Following a heading which briefly identifies the topic, a functional description is given along with usage samples. These samples illustrate important applications of the operator. The tables given last in the descriptions contain more detailed information concerning additional variations or options. Note that the sam-

ples only contain the "pure" pdfmark code. Don't forget to include additional instructions which may be necessary depending on the software in use, e.g., for protecting against printing errors, or including the pdfmarks in the program (like coordinate transformations in a FrameMaker PostScript frame).

Keys in the tables which must be present are labeled as such, others are optional. For most optional keys the default value is shown which is used when the key is missing.

Notes. The /ANN pdfmark instruction generations PDF annotations. These include notes, links, and special goodies such as sound and video, Acrobat menu items, and Weblinks (URLs). The /Subtype key defines the particular kind of annotation to create. If /Subtype is missing, a PDF note is generated.
The following code creates an open note with red window border:

```
[   /Rect [ 75 400 175 550 ]
    /Open true
    /Title (Important note)
    /Contents (This document is a preliminary version!)
    /Color [1 0 0]
/ANN pdfmark
```

Table 6.2 contains all keys for notes which may be used with /ANN if /Subtype is missing or has a value of /Text. Along with other /Subtype values, /ANN is used for many other functions too which are explained in the rest this chapter.

Table 6.2. Keys for notes with /ANN, if the /Subtype is /Text or is missing altogether

Key	***Explanation***
/Contents[1]	*The note's text string*
/Rect[1]	*Array of four numbers specifying the note's rectangle*
/SrcPg	*Number of the page on which the note appears. If this key is missing, the note appears on the current page.*
/Open	*If "true", the note is open and the text visible. If "false", only an icon is displayed for the note. If the key is missing, the note will be closed.*
/Color	*Array of three RGB values defining the color of the note's icon or frame.*
/Title	*The note's title*
/ModDate	*Date and time the note was last modified.*
/Subtype	*For notes always has a value of /Text (this is the default too).*

1. This key is required.

Links. Early Distiller versions used the /LNK key for defining links. In Distiller 3.0, this is accomplished with /ANN and a /Subtype value of /Link.

The following code creates a link inside the given rectangle with a blue border. The link jumps to the next page and retains the viewing parameters (zoom factor):

```
[   /Rect [ 70 550 210 575 ]
    /Border [ 0 0 1 ]
    /Color [0 0 1]
    /Page /Next
    /View [ /XYZ null null null ]
    /Subtype /Link
/ANN pdfmark
```

The following code creates a link inside the given rectangle with a red border. The link jumps to the document named *chapter02.doc*:

```
[   /Rect [ 70 600 210 625 ]
    /Color [1 0 0]
    /Action /Launch
    /File (chapter02.doc)
    /Subtype /Link
/ANN pdfmark
```

I'd like to restrict myself to describing the actual link definition here. Many ways of selecting a link destination or action to be triggered when the link is clicked on are described in Section 6.5 (along with many samples).

With the exception of destinations and actions, Table 6.3 lists all keys for /ANN, when /Subtype has a value of /Link.

Table 6.3. Keys for links with /ANN and the /Subtype value /Link

Key	***Explanation***
Destination or action for this link (see Section 6.5)[1]	
/Rect[1]	*Array of four numbers defining the link's active area*
/Subtype[1]	*For links always /Link*
/Border	*Array defining the link rectangle's appearance (line width and line dash). The array [0 0 0] means no border at all.*
/SrcPg	*Number of the page on which the link is to appear. If the key is missing, the link appears on the current page.*
/Color	*Array of three RGB values defining the link rectangle's color.*

1. This key is required.

Bookmarks. Bookmarks (outline entries) contain text which is related to a particular location in the document or an action. This means that the information in Section 6.5 applies to bookmarks too.

The following code creates a bookmark for jumping to page 1:

```
[ /Page 1 /View [/XYZ 44 730 1.0] /Title (Start) /OUT pdfmark
```

The following code creates a bookmark with the title "Introduction" which jumps to the article thread labeled "A" if clicked on:

```
[ /Action /Article /Dest (A) /Title (Introduction) /OUT pdfmark
```

The following code creates a bookmark with a URL link:

```
[   /Title (Home page)
    /Action << /Subtype /URI /URI (http://www.ifconnection.de/~tm) >>
/OUT pdfmark
```

With the exception of destinations and actions, Table 6.4 lists all keys for /OUT. For nested bookmarks, you first have to define the higher-level bookmarks with the correct /Count value, and then the subordinate bookmarks.

Table 6.4. Keys for bookmarks with /OUT

Key	***Explanation***
Destination or action for this link (see Section 6.5)[1]	
/Title[1]	*The bookmark's text (a maximum of 32 characters is recommended)*
/Count	*If there are subordinate bookmarks, /Count specifies their number, otherwise this key is missing. If /Count is negative, the bookmark is closed, otherwise open.*

1. This key is required.

Article threads. A PDF article consist of several rectangular areas, called beads, which are placed on one or more pages. They are linked via their common title. Additionally an article thread can have some meta-information associated with it (keywords, for example) which you can view or edit in the "View", "Articles...", "Info..." menu of Acrobat Exchange.

The following code defines a rectangle for an article bead with the title "Introduction" and additional meta-information:

```
[   /Title (Introduction)
    /Author (Thomas Merz)
    /Subject (Brief Overview of Engine Maintenance)
    /Keywords (Maintenance Overview)
    /Rect [ 225 500 535 705 ]
/ARTICLE pdfmark
```

More rectangles for the same article may follow this first definition. They are linked to the other parts by a common /Title key. Subsequent rectangles don't have /Subject, /Author, and /Keywords keys. Table 6.5 lists all keys for /ARTICLE.

Named destinations. PDF link targets can not only be defined in a layout-oriented manner (i.e., by specifying a page number and geometric coordinates), but also symbolically. A location in a document may be assigned a symbolic name. This name may later be used as a link target. A target with a

Table 6.5. Values for article beads with /ARTICLE

Key	Explanation
/Title[1]	The title of the article
/Rect[1]	Array of four numbers specifying the current bead of the article
/Page	Number of the page on which the bead will be defined
/Subject	The article's subject
/Author	The article's author
/Keywords	The article's keywords

1. This key is required.

symbolic name is called a "named destination". This technique has the advantage that the target location may change without invalidating the link. For this reason programs which automatically generate pdfmarks use symbolic names for their link targets. These names are visible in Acrobat Exchange too; however, you cannot create named destinations in Exchange, nor change existing names or their locations. More information on named destinations can be found in Section 8.6.

The following code defines a link target with a symbolic name of "chapter01" which is located on the current page. Jumping to this target doesn't change the zoom factor:

```
[ /Dest /intro /View [ /XYZ null null null ] /DEST pdfmark
```

Table 6.6 contains all keys for /DEST.

Table 6.6. Values for named destinations with /DEST

Key	Explanation
/Dest[1]	The destination's symbolic name
/Page	The target page's number. If this key is missing, the named destination is defined on the current page.
/View	Viewing parameters for the link target

1. This key is required.

PostScript instructions. It is well known that PostScript and PDF are very closely related to each other. However, there are some PostScript features and tricks which are not possible in PDF, and which are lost when converting from PostScript to PDF and back to PostScript.

In order to solve this problem you can use pdfmarks to store PostScript instructions in a PDF file. Acrobat ignores the PostScript when rendering the file on screen, but when printing to a PostScript device the included instructions are embedded in the print data.

Probably a more practical use of this feature than exploring somehow obscure PostScript features in PDF files is a "don't display, but print" effect:

If certain objects on the page are to appear in the printed version only but shouldn't be displayed on screen, PostScript pass-throughs might be used.

```
[ /DataSource (100 100 50 0 360 arc fill) /PS pdfmark
```

Table 6.7 contains all keys for /PS.

Table 6.7. Values for "pass-through" PostScript instructions with /PS

Key	Explanation
/DataSource[1]	String or file containing the PostScript code
/Level1	Alternative PostScript code for printing to a PostScript Level 1 device

1. This key is required.

Page cropping. Using pdfmarks you can define the size of one or more pages in a PDF document. Distiller crops the page according to the pdfmark values, independent of the page size defined in the printer-relevant PostScript instructions. These values can be found in Exchange's status line at the bottom of the window.

The following code crops all pages of the document to letter size. The line should be inserted at the beginning of the PostScript file (at the end of the prolog):

```
[ /CropBox [0 0 612 792] /PAGES pdfmark
```

Using /PAGE instead of /PAGES crops only the current page. Again, this code should be inserted at the beginning of the PostScript page description.

The following code defines a cropping rectangle for the current page:

```
[ /CropBox [54 403 558 720] /PAGE pdfmark
```

Table 6.8 lists the key for /PAGE and /PAGES.

Table 6.8. Key for cropping pages with /PAGE or /PAGES

Key	Explanation
/CropBox	Array of four numbers specifying location and size of the visible page area. The page size may vary from 1 to 45 inches (72 to 3240 points).

General document information. Using "File", "Document Info", "General..." you can view or change a PDF's general document information fields. According to the documentation, these fields can be defined with pdfmark instructions of type /DOCINFO at an arbitrary location within the PostScript code. However, I found /DOCINFO only to work reliably if placed on the first document page.

The following code defines the document information fields of a PDF file:

```
[   /Title (User Manual)
    /Author (Michael Heinzel)
    /Subject (Adjusting the electronics of MH-screen)
    /Keywords (screen display MH-screen)
    /Creator (DocMaker 1.0)
    /ModDate (D:19980210205731)
/DOCINFO pdfmark
```

Table 6.9 lists the keys for /DOCINFO. Additionally, custom key names can be inserted. Although they can't be viewed in Acrobat, custom fields are quite useful in index queries (you can find more details about custom info field names in Section 4.4). In order to define a custom field called "Department", for example, add the following line to the pdfmark code above:

```
/Department (Marketing)
```

If such a custom info field exists in all documents, you can restrict an index query to all files created in a specific department.

Note that Distiller attempts to extract some information from the DSC comments at the beginning of the PostScript file if there are no /DOCINFO pdfmarks present. These comments are listed in parentheses in the table.

Table 6.9. Keys for document info fields using /DOCVIEW

Key	***Explanation***
/Author	*Author of the document (%%For)*
/Creation-Date	*Document creation date and time*
/Creator	*Name of the program used to create the original document (%%Creator)*
/Producer	*Name of the program used to convert the original document to PDF*
/Title	*Document title (%%Title)*
/Subject	*Subject of the document contents*
/Keywords	*Keywords for the document*
/ModDate	*Date and time of last document change*

Viewer properties. A PDF file may specify the Acrobat viewer's behavior. This includes bookmark or thumbnail display as well as full-screen mode. In Acrobat Exchange, you can edit these settings via "File", "Document Info", "Open...".

The following code results in Acrobat opening the document at page 3 with thumbnail display enabled:

```
[ /PageMode /UseThumbs /Page 3 /DOCVIEW pdfmark
```

For a screen presentation it may be useful to open the document in full-screen mode:

```
[ /PageMode /FullScreen /DOCVIEW pdfmark
```

Table 6.10 lists all keys for /DOCVIEW.

Table 6.10. Keys for viewer properties using /DOCVIEW

Key	Explanation
Document open action (see Section 6.5)	
/PageMode	/UseNone (default): The document is displayed without bookmarks and thumbnails. /UseOutlines: The document is displayed with bookmarks. /UseThumbs: The document is displayed with thumbnails. /FullScreen: The document is displayed in full-screen mode.

Page transitions. In Acrobat you can choose among several page transitions in order to make the process of replacing a page with the next more attractive. Unfortunately, these effects cannot be set in Acrobat Exchange. Although you can choose a page transition in "File", "Preferences", "Full Screen", this setting relates to the whole document, not to individual pages in the file. Similarly to named destinations, individual page transitions are a PDF feature not supported in Acrobat's user interface. Even the pdfmark reference manual remains silent about this topic – you have to take a look at the PDF specification.

The following code specifies a mosaic-like transition from the old page to the new page. The old page contents "dissolve" to reveal the new page:

```
[ {ThisPage} << /Trans << /S /Dissolve >> >> /PUT pdfmark
```

The following code specifies a wiping effect which generates the new page by wiping over the old page from left to right:

```
[ {ThisPage} << /Trans << /S /Wipe /Di 180 >> >> /PUT pdfmark
```

{ThisPage} is a symbolic name for the current page. This entry always has to be included exactly as shown. The page transitions are always activated when opening the page, irrespective of the previous page. Therefore it doesn't matter whether the page is displayed through manual navigation, by page number, or a link.
Table 6.11 lists all keys for page transitions supported in Acrobat 3.0.

Table 6.11. Keys for page transitions with /PUT

Key	Explanation
/Split	Two lines sweep across the screen to reveal the new page similar to opening a curtain.
/Blinds	Similar to /Split, but with several lines resembling "venetian blinds"
/Box	A box enlarges from the center of the old page to reveal the new one.
/Wipe	A single line "wipes" across the old page to reveal the new one.
/Dissolve	The old page "dissolves" to reveal the new one.

Table 6.11. Keys for page transitions with /PUT (cont.)

Key	Explanation
/Glitter	Similar to /Dissolve, except the effect sweeps from one edge to another.
/R (Replace)	The old page is simply replaced with the new one without any special effect. This is the default.

For some of the transitions additional parameters may be specified. The following code results in a split effect with the lines moving horizontally (/H) from the inner parts of the page to the outer parts (/O). The duration of the effect is two seconds (/D):

```
[ {ThisPage} << /Trans << /S /Split /D 2 /Dm /H /M /O >> >> /PUT pdfmark
```

Table 6.12 lists all supported parameters for /Trans, along with the kind of transition on which the parameters may be applied.

Table 6.12. Additional parameters for page transitions with /Trans

Key	Explanation
/D	Duration of the transition effect in seconds (applies to all effects)
/Di (Direction)	Direction of the movement (multiples of 90° only). Values increase in a counterclockwise fashion, 0° points to the right (for /Wipe and /Glitter).
/Dm (Dimension)	Possible values are /H or /V for a horizontal or vertical effect, respectively (for /Split and /Blinds).
/M (Motion)	Specifies whether the effect is performed from the center out or the edges in. Possible values are /I for in and /O for out (for /Split and /Box).

Generally, a page transition will be defined for the current page. However, it's also possible to define transitions for another page. Table 6.13 lists all keys which can be used with /PUT. Note that only direct page specification may be used for page transitions. {Catalog} and {DocInfo} are reserved for other /PUT applications.

Table 6.13. Selecting pages with /PUT

Key	Explanation
{Catalog}	(reserved for other /PUT applications)
{DocInfo}	(reserved for other /PUT applications)
{PageN}	Page N (replace N with a page number)
{ThisPage}	Current page
{PrevPage}	Previous page
{NextPage}	Next page

Viewer preferences. A PDF document may specify several Viewer preferences which apply when opening the file in Acrobat Reader or Exchange.

You can edit these settings in Exchange with "File", "Document Info", "Open..."

The following code instructs the viewer to hide the toolbar when opening the document:

```
[ {Catalog} << /ViewerPreferences << /HideToolbar true >> >> /PUT pdfmark
```

You can also specify the page layout mode in the document. The following code makes Acrobat open the file in two-column layout (two pages are displayed side-by-side):

```
[ {Catalog} << /PageLayout /TwoColumnRight >> /PUT pdfmark
```

Note that the syntax is different to the preceding example. The /PageLayout description in the PDF specification doesn't match the implementation in Acrobat.

Table 6.14 lists all entries for the /ViewerPreferences dictionary. The default settings are given in parentheses. These apply when the respective entry is missing. Except for the last two entries, all keys are populated with Boolean values.

Table 6.14. Additional parameters for /ViewerPreferences

Key	***Explanation***
/HideToolbar	*Hide toolbar (false).*
/Hide-Menubar	*Hide menu bar (false).*
/Hide-WindowUI	*Hide other user interface elements (false).*
/FitWindow	*Adjust window size to the size of the first page (false).*
/Center-Window	*Place window in the middle of the screen (false).*
/PageLayout[1]	*Specify page layout. Possible values are:* */SinglePage: Display single pages* */OneColumn: Display pages in columns* */TwoColumnLeft: Display pages in two-column layout, starting with a left page* */TwoColumnRight: Display pages in two-column layout, starting with a right page*
/NonFull-Screen-PageMode	*Specifies how to display the document when exiting full-screen mode. Except /FullScreen the same values are allowed as for the /PageMode key in a /DOCVIEW instruction (see Table 6.10).*

1. /PageLayout isn't documented correctly in the specification and is used differently than the other parameters (see example).

Encapsulating graphics. Acrobat's optimization function compresses PDF data and rearranges the objects for page-at-a-time download from a Web server. There is another optimizarion that is not as well known even though

it further decreases file size in certain situations. Exchange checks for images in the document which are repeatedly used on multiple pages, for example a logo on each page header. If such an image is found, its PDF data is included only once in the file. Other pages reference the image data already included for another page. If an image is used several times in a document, this feature dramatically reduces the overall file size.

Similar optimization can be achieved by means of pdfmarks, although more programming is needed which requires a good command of the PostScript language. For this reason, I'd like to restrict myself to explaining the basic principle without working out a full-blown example.

Let's suppose we have an EPS file with the image to be included in the document. /BP (BeginPicture) and /EP (EndPicture) instructions surround the image's PostScript code. The image is assigned an arbitrary symbolic name in /BP. Using the /SP (ShowPicture) pdfmark operator along with the symbolic name, the image can be referred to on arbitrary pages without repeating the actual PostScript instructions:

```
[ /BBox [100 100 400 600] /_objdef {company_logo} /BP pdfmark
...the company logo's PostScript instructions...
[ /EP pdfmark
```

Use the following code to reuse the image on another page in its original size and on the original position:

```
[ {company_logo} /SP pdfmark
```

If you want to change the image's location or size, you have to transform the PostScript coordinate system with appropriate PostScript language commands before issuing the /SP instruction.

Table 6.15 lists the keys for /BP. The /EP instruction doesn't need any additional parameters. /SP only needs the symbolic name defined in /BP.

Table 6.15. Keys for embedding EPS graphics with /SP

Key	*Explanation*
/BBox	*Array of four numbers defining the graphic's bounding box*
/_objdef	*The graphic's symbolic name in curly braces*

Attaching an index file. Using "File", "Document Info", "Index..." in Exchange you can attach an index file to a PDF file. This index is activated without any further user intervention when the document is opened.

The PDF specification doesn't mention the instructions necessary to define the index's name in the PDF file. However, it's easy to find out the instructions by manually attaching an index in Exchange, and analyzing the resulting PDF file. As it turns out, you can achieve the same effect with pdfmarks.

The following code attaches an index called *cms.pdx* to the PDF file. Carefully take into account the many angle brackets in order to avoid error messages from Distiller.

```
[   {Catalog} << /Search << /Indexes
        [ << /Name /PDX /Index (cms.pdx) >> ]
    >> >>
/PUT pdfmark
```

Page open actions. Using "Document", "Set Page Action..." in Exchange you can define actions which automatically take place when opening the page. This effect can also be achieved with pdfmarks. You can find an example in the next section which describes actions. Note the difference between "page action" and "page transition": page transitions define how the previously displayed page is replaced with the new page.

Creating form fields. Exchange's form tool offers plenty of features for defining PDF form fields. However, defining fields manually is a time-consuming and labor-intensive process. This effort can be reduced by using pdfmarks. Although not all form features are supported, creating fields and defining some of their features is indeed possible with pdfmarks. To facilitate changing the size and position of the fields, I recommend the EPS technique for including the form pdfmarks in the PostScript code. Detailed descriptions of the various field types can be found in Section 7.2.

Since form fields have many attributes, and not all field properties can be defined with pdfmarks, I'd like to restrict myself to some examples for defining field types. More detailed information can be found in the *PDF Reference Manual*. With the code shown below you can at least prepare and exactly position the form fields in the original file. Setting the field attributes is best done in Acrobat Exchange.

Setting font and other field attributes requires so-called widget definitions. These are additional pdfmark instructions which must precede the following examples (on the first page of the document). You can find an EPS file called *afrmdict.eps* in the PFN directory on the Acrobat CD-ROM. This EPS file contains the necessary widget definitions.

The following code defines a text field with a border width of one point. The default value is "Thomas Merz", the field's size is specified in /Rect:

```
[   /T (text input field)   % title
    /Subtype /Widget
    /FT /Tx                 % field type text box
    /DV (Thomas Merz)       % default value
    /Rect [ 0 0 216 18]
    /F 4                    % field is printable
    /BS << /S /S /W 1 >>    % border style solid, width = 1
    /MK <<
    /BC [ 1 0 0 ]           % border color red
```

```
    /BG [ 1 1 1 ] >>        % background color white
/ANN pdfmark
```

The following code defines a list box with two list elements called "element1" and "element2". When activated, the elements export the values "e1" and "e2", respectively. "e1" is the default value:

```
[   /T (list box)
    /Subtype /Widget
    /FT /Ch                 % field type choice: list box
    /Rect [ 0 0 216 18]
    /F 4
    /DV (e1)                % default value
    /DA (/Helv 12 Tf 0 g)
    /Opt [ [ (e1)(element1)] [ (e2)(element2)] ]
/ANN pdfmark
```

Including a suitable /Ff flag value in the preceding example results in a combo box. This means the list elements are editable by the form user:

```
[   /T (combo box)
    /Subtype /Widget
    /FT /Ch                 % field type choice: combo box
    /Rect [ 0 0 216 180]
    /F 4
    /Ff 393216              % special flag signals combo box
    /DV (e1)                % default value
    /DA (/Helv 12 Tf 0 g)
    /Opt [ [ (e1)(element1)] [ (e2)(element2)] ]
/ANN pdfmark
```

Next, let's create a check box:

```
[   /T (check box)
    /Subtype /Widget
    /FT /Btn                % field type button: check box
    /Rect [ 0 0 216 18]
    /F 4
    /BS << /S /S /W 1 >>
    /MK << /BC [ 1 0 0 ] /BG [ 1 1 1 ] >>
/ANN pdfmark
```

Finally, a push button:

```
[   /T (button)
    /Subtype /Widget
    /FT /Btn                % field type button: check box
    /Rect [ 0 0 216 18]
    /F 4
    /Ff 65540               % special flag signals push button
    /BS << /S /S /W 1 >>
    /MK << /BC [ 1 0 0 ] /BG [ 1 1 1 ] >>
    /DA (/Helv 12 Tf 0 g)
/ANN pdfmark
```

Since I can't cover creating form fields with all options and variations, Table 6.16 only gives an overview of possible field types. These main field types are further refined using several values in the /Ff flag (see examples above). More information can be found in the "PDF Reference Manual".

Table 6.16. Possible form field types (/FT key) with /ANN and /Widget subtypes

Key	Explanation
/Tx	Text field
/Ch	(choice) List box
/Btn	(button) Check box, radio button, or push button.

6.5 Destinations and Actions

Destinations and actions are used in three areas, all of which can be set up in an interactive manner in Acrobat Exchange or with pdfmarks:

- Links are created with the /ANN pdfmark and a /Subtype of /Link. In Exchange simply use the link tool to define a rectangle and select the type of action from a pull-down menu (see Figure 6.5).
- Bookmarks are created with the /OUT pdfmark. In Exchange choose "Document", "New Bookmark", select the bookmark, "Edit", "Properties...", and choose the desired type of action.
- Page actions – actions which are performed when the page is opened – are defined with the /AA and /PUT pdfmark instructions. The corresponding menu sequence in Exchange is "Document", "Set Page Action...", "Add...", choose type of action.

Figure 6.5 shows the dialog box for creating a link.[1] It contains all link destinations and actions covered in this section.

Before delving into the many and somehow confusing variations for link destinations and page actions, I'd like to give an example for each of the three applications outlined above. We'll use an intradocument link, a URL link, and playing a sound file as actions.

The following code creates a link which jumps to page 5 of the document:

```
[   /Rect [ 70 550 210 575 ]
    /Page 5
    /View [ /XYZ null null null]
    /Subtype /Link
/ANN pdfmark
```

The following code creates a bookmark. Clicking this bookmark results in jumping to the specified URL:

1. *In case you wonder what the "JavaScript" entry is supposed to do – read Section 7.6.*

```
[   /Count 0
    /Title (Click here for home page)
    /Action << /Subtype /URI /URI (http://www.ifconnection.de/~tm) >>
/OUT pdfmark
```

The following code results in the sound file *melody.snd* being played as soon as the first page is opened (generally, when opening the document):

```
[   /Rect [0 0 0 0]
    /Subtype /Movie
    /Title (Greeting sound)
    /Movie << /F (melody.snd) >>
/ANN pdfmark
[   {Page1} << /AA << /O << /S /Movie
        /T (Greeting sound) /Operation /Play  >> >> >>
/PUT pdfmark
```

Note that the greeting sound is attached to an annotation, although a sound doesn't need an active area on the page. We use a size of 0 for this area to make sure it doesn't disturb anybody. The page action simply relates to the name of the annotation.

Overview. Following the three application examples, I'd like to give an overview of all link destinations and page actions. More detailed descriptions of all flavors can be found after the overview.

Table 6.17. Keys for links and actions

Key	*Explanation*
/View	*Array describing a document location which serves as a destination for a bookmark or link. See Table 6.19 for more details.*
/Page	*Describes a link destination along with /View. /Page specifies the number of the page (counting from 1). A value of 0 means no destination at all. For links and article threads, the values /Next and /Prev relate to the next or previous page. /Page is only required if the destination is not located on the current page.*
/Action	*Most general specification. See Table 6.18 for more details.*
/Dest	*Symbolic name of a named destination defined with /DEST. If used as a destination for an article, /Dest contains the title or number of the article (counting article numbers from 0).*

A closer look at Figure 6.5 clarifies the relationship of pdfmarks and Exchange's user interface elements: The "View" link type relates to the /View and /Page keys. All other types are implemented with the /Action key. Links to named destinations cannot be defined in Exchange at all but only with the /DEST pdfmark operator.

Table 6.18 lists all keys for /Action. The samples and tables on the following pages explain how to use them, and which subkeys are involved.

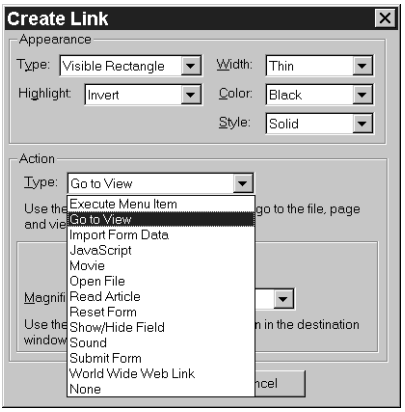


Fig. 6.5.
All of the actions which can be defined in Exchange using a link, a bookmark, or a page action can also be defined using pdfmarks.

Table 6.18. Direct and indirect keys for /Action

Key	***Explanation***
/GoTo	*Jump to a page in the current document. Requires the /Dest key or both the /Page and /View keys.*
/GoToR	*Jump to a page in another PDF document. Requires the /Dest key or both the /Page and /View keys, and the /File key.*
/Launch	*Launches a non-PDF document or an application program. Requires /File.*
/Article	*Link to an article in the current or another PDF document. Requires /Dest and additionally /File, if the article is contained in another PDF file.*
/URI[1]	*URL for linking to a document on the WWW*
/Sound[1]	*Play a sound file*
/Movie[1]	*Play a movie or sound file*
/SetState[1,2]	*Store viewing definitions for an annotation*
/Hide[1]	*Hide or show an annotation*
/Named[1]	*Execute one of Acrobat's menu functions*
/SubmitForm[1]	*Send form contents to a URL*
/ResetForm[1]	*Reset form contents to default values*
/ImportData[1]	*Import form fields from a file*
/JavaScript	*Insert JavaScript for the whole document or a form field*

1. These actions cannot be activated directly by a key, but must be defined indirectly via a dictionary entry for the /Action key. This is already taken into account in the following descriptions.
2. Acrobat Exchange doesn't offer any user interface for this feature.

Linking to a page in the same document. The following code defines a rectangular link area. Clicking on this area results in jumping to the next page:

```
[    /Rect [ 70 550 210 575 ]
```

```
    /Page /Next
    /View [ /XYZ -5 797 1.5]
    /Subtype /Link
/ANN pdfmark
```

Table 6.19 lists all keys for the /View array, as well as the corresponding values.

Table 6.19. Keys for the /View array

Key	Explanation
/Fit	Fit page to window size.
/FitB	Fit visible page contents to page width.
/FitH	top Fit width of the page to window size. "top" specifies the desired distance from the page origin to the upper edge of the window. If "top" has a value of -32768, Acrobat calculates the distance from the page origin to the upper edge of the window automatically.
/FitBH	top Fit visible page contents to window size. "top" specifies the desired distance from the page origin to the upper edge of the window.
/FitR	x_1 y_1 x_2 y_2 Fit the rectangle specified by the four numbers to the window.
/FitV	left Fit page height to window size. "left" specifies the desired distance from the page origin to the left edge of the window.
/FitBV	left Fit height of visible page contents to window size. "left" specifies the desired distance from the page origin to the left edge of the window.
/XYZ	left top zoom "left" and "top" specify the desired distance from the page origin to the upper left corner of the window. "zoom" specifies the zoom factor (1 means 100%). A value of "null" for one of the three numbers instructs Acrobat to retain the old value.

Link to another PDF document. Links can jump not only to a page in the current document, but also to a page in another PDF file.

The following code defines a link rectangle. Clicking on the link's active area jumps to a document with a file name of *chapter06.pdf*:

```
[   /Rect [ 70 600 210 625 ]
    /Action /GoToR
    /File (chapter06.pdf)
    /Subtype /Link
/ANN pdfmark
```

Table 6.20 lists all keys for /GoToR. The platform dependent keys can be used to specify platform-specific variations of the file name. These are only used on the respective platform and override the generic /File key.

Table 6.20. Keys for /GoToR

Key	Explanation
/File[1]	Path name of the PDF file (relative names are allowed)
/DOSFile	MS-DOS path name (overrides /File if present)
/MacFile	Mac path name (overrides /File if present)
/UnixFile	Unix path name (overrides /File if present)
/URI	URL of the PDF file (note: "URI" for "Universal Resource Identifier, not "URL")
/ID	ID number of the PDF file (rarely used)
/New-Window	If this key has a value of "true", Acrobat opens a new window for the file.

1. This key is required.

Launching another document or application. The next step in generalizing link destinations is a document in a format other than PDF, or another application program. Acrobat launches an external program and passes the file name given in the pdfmark link or bookmark.

The first example illustrates the main problem with such a construct – it is no longer portable. This means the link doesn't work on all operating system platforms. The names of the programs as well as the details of launching a program depend heavily on the underlying operating system (the user's, not yours!).

The following code defines a link. When activated, it starts the Windows Paintbrush program (without a document file name):

```
[   /Rect [ 70 600 210 625 ]
    /Border [ 16 16 1 ]
    /Action /Launch
    /File (pbrush.exe)
    /Subtype /Link
/ANN pdfmark
```

Table 6.21 lists all keys for /Launch.

Table 6.21. Keys for/Launch

Key	Explanation
/File[1]	Path name of the file or executable program (relative names are allowed). Under Windows 95 and NT, when /File points to a directory, Explorer is launched to explore this directory.
/DOSFile	MS-DOS path name (overrides /File if present)
/MacFile	Mac path name (overrides /File if present)
/UnixFile	Unix path name (overrides /File if present)
/URI	URL of the PDF file (note: "URI" for "Universal Resource Identifier, not "URL")
/Dir	For Windows: initial directory of the application

Table 6.21. Keys for/Launch (cont.)

Key	Explanation
/Op	(open) or (print). Only supported in Windows
/WinFile	File name of the document or application
/Params[2]	Parameters for a Windows application
/Unix	Parameters for a Unix application
/New-Window	If this key has a value of "true", Acrobat opens a new window for the file.

1. This key is required.
2. This key doesn't work in Distiller 3.01.

Linking to a named destination. Symbolic names for document locations defined with a /DEST instruction (named destinations) can also be used as destinations. As already explained, Acrobat Exchange doesn't offer any user interface for defining such named destinations. Although existing named destinations in a document are displayed, you cannot edit or create them in Exchange.

The following code defines a link to the destination with the symbolic name "chapter5" in the file *target.pdf*:

```
[   /Rect [ 70 650 210 675 ]
    /Color [ 0 0 1 ]
    /Dest /chapter5
    /File (target.pdf)
    /Subtype /Link
/ANN pdfmark
```

Linking to a file on the WWW. A bookmark or link may also reference a document (more generally, a resource) on the Internet. In this case, the target is defined by its URL (Universal Resource Locator). If Acrobat doesn't already display in a Web browser's window, the browser is launched. Note that URLs in PDF are always labeled URI (Universal Resource Identifier).

The following code defines a link rectangle. When activated, it jumps to a home page:

```
[   /Rect [ 50 425 295 445 ]
    /Action << /Subtype /URI /URI (http://www.ifconnection.de/~tm) >>
    /Subtype /Link
/ANN pdfmark
```

Note that the /IsMap key allows you to define an image map for interactive graphics. Clicking in the link's area can trigger different actions according to the mouse pointer's location inside the link graphic. More information on PDF image maps can be found in Section 8.6. Table 6.22 lists all keys for /URI.

Table 6.22. Keys for /URI

Key	Explanation
/Subtype[1]	For Internet links always /URI
/URI[1]	The target's URL in 7-bit ASCII encoding
/IsMap[2]	"true" if the mouse coordinates are to be appended to the URL (image map feature). Default is "false".

1. This key is required.
2. It's not possible to define image maps in Acrobat Exchange.

Defining a document's base URL. Like HTML, PDF supports the concept of a base URL and relative URLs. This is useful when a large document collection is to be moved to another server. If base URLs and relative links are used, not all links have to be adjusted but only the base URLs. Any PDF document can be assigned a base URL. In Acrobat Exchange this is achieved via "File", "Document Info", "Base URL...". Attaching a base URL can also be implemented with pdfmarks in the original file. Absolute URLs – WWW locations with complete server addresses – remain unchanged by such a base URL definition.

The following code defines the given URL as base URL for all relative Internet links in the document:

```
[ {Catalog} << /URI << /Base (http://www.ifconnection.de/~tm) >> >>
/PUT pdfmark
```

Inserting JavaScript instructions. Using Acrobat 3.01 and the Forms extension it's possible to insert JavaScript instructions in a document. These are executed when the user clicks on a link, opens a PDF file, or fills form fields. More information on JavaScript in PDF can be found in Section 7.6.

JavaScript can also be embedded using pdfmarks. The following code defines JavaScript code which is activated by clicking on a link:

```
[   /Rect [ 400 400 500 450 ]
    /Action << /Subtype /JavaScript
        /JS (console.show\(\)\r console.println\("Cuckoo!"\);) >>
    /Subtype /Link
/ANN pdfmark
```

As can be seen in the example, the JavaScript instructions are contained in a string. This means that several special characters, mainly (,), and \, must be "escaped" with a backslash character \. Line-end characters in the JavaScript are labeled with \r.

JavaScript instructions can be defined for use in PDF forms so that they are executed when filling out the fields. The following example defines a text field with additional JavaScript instructions which are executed for calculating field values, verifying and formatting the field input, and checking the keyboard input for valid characters.

```
[   /Rect [ 400 400 600 450 ] /T (text input) /Subtype /Widget
    /FT /Tx /F 4 /Ff 65540 /BS << /S /S /W 1 >>
    /AA <<
    /C << /S /JavaScript /JS (AFSimple_Calculate\("SUM", "F1, F2, F3"\);) >>
    /V << /S /JavaScript /JS (AFRange_Validate\(true, 0, true, 1000\);) >>
    /F << /S /JavaScript /JS
        (event.value=util.printx\("999",\revent.value\);) >>
    /K << /S /JavaScript /JS (var valid = new String\("-0123456789"\);\rif
        \(valid.indexOf\(event.key\) == -1\)\r{\r\tsys.beep\(1\);\revent.rc
        = false;\r}) >>
    >>
    /MK << /BC [ 1 0 0 ] /BG [ 1 1 1 ] >>
/ANN pdfmark
```

Finally, it's possible to embed JavaScript instructions which are executed when the document is opened. The script is assigned the symbolic name "TMScript" and stored in the "Catalog" data structure of the PDF document:

```
[ /_objdef {TMScript} /type /dict /OBJ pdfmark
[   {TMScript} << /JavaScript << /Names [ (Cuckoo)  << /S /JavaScript
        /JS (console.show\(\)\rconsole.println\("Cuckoo!"\);) >> ]
    >> >>
/PUT pdfmark
[ {Catalog} << /Names {TMScript} >> /PUT pdfmark
```

To insert multiple scripts, adjust the example above according to the following scheme:

```
[   {TMScript} << /JavaScript << /Names [
        (script1)  << /S /JavaScript /JS (...) >>
        (script2)  << /S /JavaScript /JS (...) >>
    ] >> >>
/PUT pdfmark
```

The names of the JavaScripts are stored in the "names tree" of the PDF document which also holds destination names. If the document contains named destinations, the names of the JavaScripts must be inserted in the existing names tree. Since this is not possible with pdfmarks, the above technique does not work for PDF file containing named destinations.

More details on JavaScript programming in PDF, especially on additional objects and methods, can be found in Section 7.6 of this book, and in the Acrobat Forms documentation. Table 6.23 lists the keys for several kinds of JavaScript instructions which can be bound to form fields.

Table 6.23. Keys for JavaScript instructions with /AA

Key	***Explanation***
/K	*(keystroke) JavaScript for checking the validity of the keyboard input*
/F	*(format) JavaScript for formatting the input*
/V	*(validate) JavaScript for checking the field values when the field is exited*
/C	*(calculate) JavaScript for calculating field values when the field is exited*

Linking to an article. When jumping to an article, the first bead of the article is displayed. The article thread may be identified by title or number.

The following code defines a bookmarks labeled "Lead story" which jumps to the article thread of the same name:

```
[ /Dest (Lead story) /Title (Lead story) /Action /Article /OUT pdfmark
```

When jumping to an article, all /GoToR keys (Table 6.20) are allowed. /File is necessary only when the article is located in another PDF file. Additionally, the key in Table 6.24 is required.

Table 6.24. Additional key for /Article

Key	*Explanation*
/Dest[1]	*Article target. A string containing the article's title. Alternatively, a number may be specified which is the number of the article in the document (the first article in the document is numbered 0).*

1. This key is required.

Playing sound and video files. Given suitable hardware and software (sound card, QuickTime) Acrobat is capable of playing sound and video files. External sound files and video clips both are stored with the /Movie key in the PDF. Strictly speaking, there's another key called /Sound for directly embedding the sound data in the PDF file. In practice, however, both types of data are treated as external data of type /Movie. Actually, embedding the movie or sound data in the PDF file isn't possible, only linking.

To be sure to include movies successfully, note that not all file formats are supported on all platforms. A list of supported sound and video formats can be found in the Acrobat documentation. Since videos can only be scaled proportionally, the movie rectangle must be created proportional to the movie's edges.

The following code plays the film from the external file *vacation.mov* in the specified rectangle:

```
[   /Rect [ 216 503 361 612 ]
    /Type /Annot
    /Subtype /Movie
    /Movie << /F (vacation.mov) >>
/ANN pdfmark
```

The following code plays the sound file "melody.snd" when the rectangle's active area is clicked:

```
[   /Rect [ 216 503 361 612 ]
    /Type /Annot
    /Subtype /Movie
    /T (My composition)
    /Movie << /F (melody.snd) >>
/ANN pdfmark
```

Table 6.25 lists the keys for /Movie.

Table 6.25. Keys for /Movie

Key	Explanation
/Subtype[1]	For videos and external sounds always /Movie
/A	false: Play the movie when clicked true: Play the movie with the default activation values (this is the default for /A) Alternatively, a dictionary may be used as value for the /A key. See Table 6.26 for details.
/Movie[1]	Dictionary describing the movie. It may contain the following keys: /F: Name of the movie file /Aspect: Array of two values describing the movie's size in pixels /Rotate: Number of degrees the movie has to be rotated clockwise (multiples of 90° only, 0° is top) /Poster: Boolean which indicates whether or not to display the poster from the movie file
/Operation	/Play: start playing the movie (default value) /Stop: stop playing the movie /Pause: pause playing the movie /Resume: resume playing a paused movie
/T (Title)	The movie's title

1. This key is required.

The information in the /A dictionary describes the dynamics of playing the movie. Table 6.26 lists all keys for the /A dictionary. All keys are optional (details on the keys not described in the table can be found in the PDF specification).

Table 6.26. Keys for the optional /A dictionary in a /Movie instruction

Key	Explanation
/Show-Controls	Boolean indicating whether or not a movie controller bar is shown when the movie is displayed
/Mode	/Once: Show the movie once and stop /Open: Show movie and leave controller open /Repeat: Repeat movie until stopped by user /Palindrome: play back and forth until stopped by user
/Syn-chronous	If true, the user must wait for the movie to be finished before further Acrobat interaction is allowed
/Start	Starting time of the movie segment
/Duration	Duration of the movie segment
/Rate	Initial relative speed of the movie
/Volume	Volume setting for the movie

Table 6.26. Keys for the optional /A dictionary in a /Movie instruction

Key	Explanation
/FWScale	For floating window movies, the magnification at which to play the movie
/FWPosition	For floating window movies, the screen position at which to play the movie

Hiding and showing fields. Hiding and showing form fields is quite a useful feature for implementing context-sensitive help and similar interactive elements. The field to be hidden or shown is identified by its name.

When the link generated by the following code is activated, the contents of the field called "help" are displayed if the field was hidden before:

```
[   /Rect [ 50 425 295 445 ]
    /Action << /Subtype /Hide
        /T (help)
        /H true >>
    /Border [ 1 1 1 ]
    /Subtype /Link
/ANN pdfmark
```

Table 6.27 lists all keys for /Hide.

Table 6.27. Keys for /Hide

Key	Explanation
/Subtype[1]	For showing/hiding fields always /Hide
/T[1]	Name of a form field or array of several field names to be hidden or shown
/H (Hide)	"true" means hide, "false" means show the field. The default is "true".

1. This key is required.

Submitting a form to a URL. The ability to submit a form's contents to a certain URL is the cornerstone of PDF form usage on the Web. The form's contents can be sent in either FDF (Forms Data Format) or HTML to the server.

The following code submits the form data in FDF format to the specified CGI script on the server:

```
[   /Rect [ 50 425 295 445 ]
    /Action << /Subtype /SubmitForm
        /F (http://www.ifconnection.de/~tm/cgi-bin/order.pl#FDF) >>
    /Flags 0
    /Subtype /Link
    /Border [ 1 1 1 ]
/ANN pdfmark
```

Table 6.28 lists all keys for /SubmitForm.

Table 6.28. Keys for /SubmitForm

Key	Explanation
/Subtype[1]	For submitting form data always /SubmitForm
/F (File)[1]	URL of the Web server script for processing the data
/Fields	(used internally)
/Flags	Integer for specifying FDF or HTML as submission format. Flags = 0 means FDF, Flags = 4 means HTML. Default is 0 (FDF).

1. This key is required.

Resetting a form to its default values. A button for resetting a form to its default values makes it easy for the user to start over with a form. Resetting means all fields with predefined default values are set to this value, and fields without any default value remain empty.

The following code resets all fields of the document to their default values:

```
[   /Rect [ 50 425 295 445 ]
    /Action << /Subtype /ResetForm >>
    /Subtype /Link
    /Border [ 1 1 1 ]
/ANN pdfmark
```

Table 6.29 contains all fields for /ResetForm.

Table 6.29. Keys for /ResetForm

Key	Explanation
/Subtype[1]	For resetting form fields always /ResetForm
/Fields	(used internally)
/Flags	(used internally)

1. This key is required.

Importing form data from a file. Importing form contents from a file facilitates filling multiple similar forms, especially when the forms adhere to the PFN standard (Personal Field Names; a description of the PFN standard can be found in Section 7.5). The creator of a form may further facilitate importing form data from a file by supplying an import push button.

The following code defines a rectangle for importing all form fields from a FDF file called *myprof.fdf*:

```
[   /Rect [ 50 425 295 445 ]
    /Action << /Subtype /ImportData
    /F (myprof.fdf) >>
    /Subtype /Link
    /Border [ 1 1 1 ]
/ANN pdfmark
```

Table 6.30 lists all keys for /ImportData.

Table 6.30. Keys for /ImportData

Key	Explanation
/Subtype[1]	For importing form data always /ImportData
/F (File)[1]	Name of the file from which to import the form data

1. This key is required.

Executing menu items. Although the PDF specification "officially" only supports the menu items "Next Page", "Previous Page", "First Page", and "Last Page", actions in a PDF file may execute all menu items of Acrobat Reader or Exchange. In order to determine the corresponding PDF code and

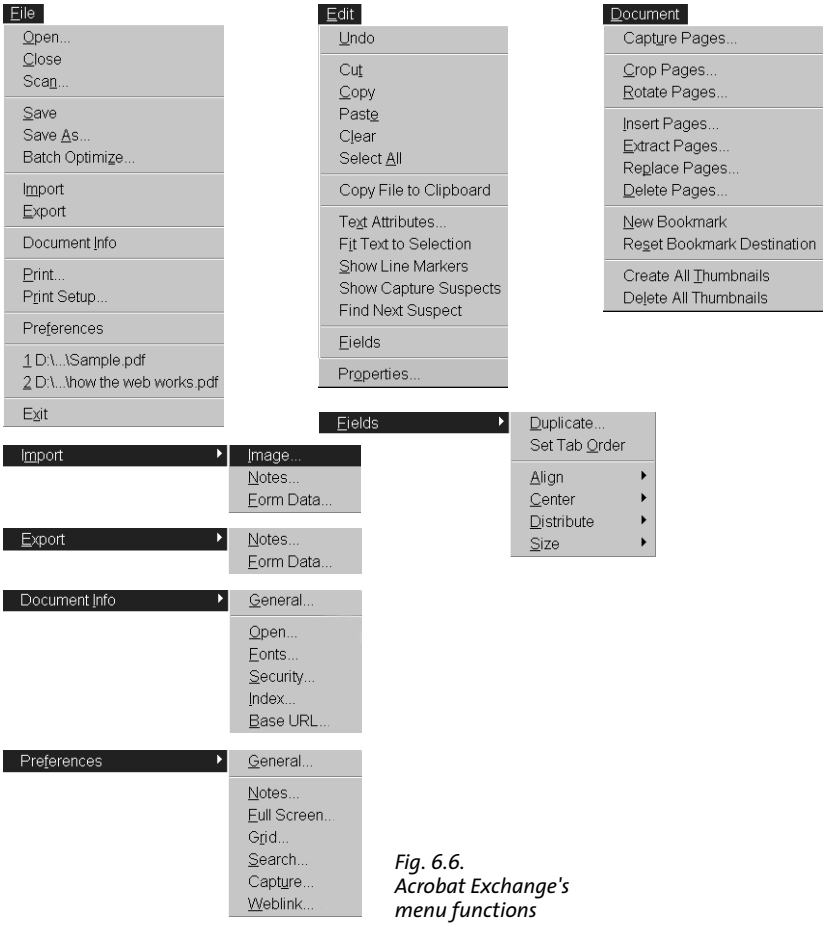


Fig. 6.6. Acrobat Exchange's menu functions

the pdfmarks for these undocumented menu items, I manually created actions for all menu items in Exchange. Unlike the specification, the resulting PDF file reveals all the required keys.

Note that not all menu items are available in Acrobat Reader. Items which are only available in Exchange are marked in the table. Another remark relates to subordinate menus: pdfmark cannot predefine the settings in higher-order menus. For example, although you can launch the print menu, you cannot define the print menu's settings (page range, etc.). The user must adjust these settings manually and confirm the menu before the print action is launched.

The following code creates a rectangle which, when clicked, closes the current PDF document:

```
[   /Rect [ 50 500 150 600]
    /Action << /Subtype /Named /N /Close >>
    /Subtype /Link
/ANN pdfmark
```

Table 6.31 lists all keys for activating Acrobat Exchange's menu items when used as /N value for the /Subtype /Named. Since the key names correspond to the menu items and are given in the same order as they appear on screen, the table doesn't need any additional explanation for the key names.

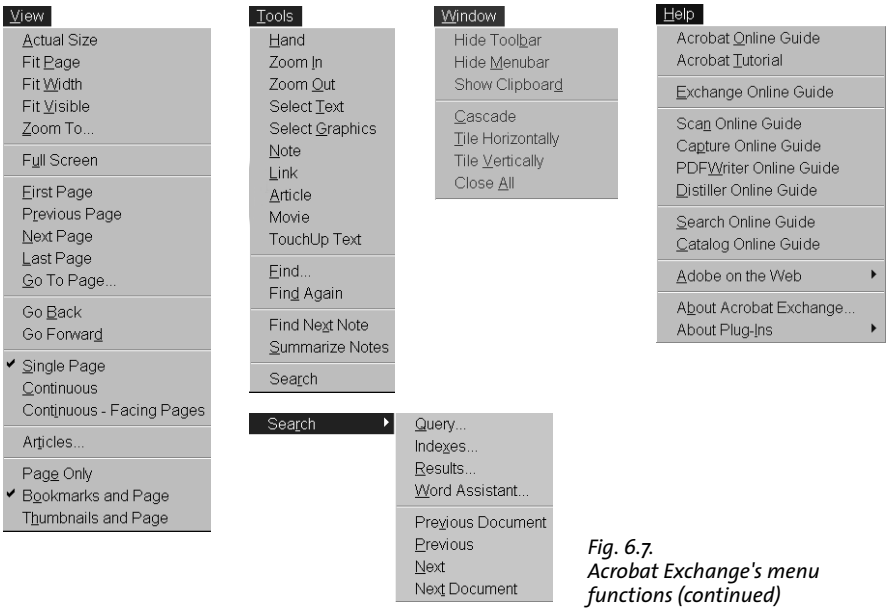


Fig. 6.7.
Acrobat Exchange's menu functions (continued)

Table 6.31. Keys for /N for executing menu functions with /Named

Acrobat menu	***Keys for the menu function***
File	*/Open, /Close, /Scan[1], /Save[1], /SaveAs[1], /Optimizer:SaveAsOpt[1], /Print, /PageSetup, /Quit*
File → Import	*/ImportImage[1], /ImportNotes[1], /AcroForm:ImportFDF[1]*
File → Export[1]	*/ExportNotes[1], /AcroForm:ExportFDF[1]*
File → Document Info	*/GeneralInfo, /OpenInfo, /FontsInfo, /SecurityInfo, /Weblink:Base[1], /AutoIndex:DocInfo[1]*
File → Preferences	*/GeneralPrefs, /NotePrefs, /FullScreenPrefs, /Weblink:Prefs, /Acro-Search:Preferences (Windows) or /AcroSearch: Prefs (Mac)[2], /Cpt:Capture[1]*
Edit	*/Undo, /Cut, /Copy, /Paste, /Clear, /SelectAll, /Ole:CopyFile[1,3], /TouchUp: TextAttributes...[1], /TouchUp:FitTextToSelection[1], /Touch-Up:ShowLineMarkers[1], /TouchUp:ShowCaptureSuspects[1], /TouchUp:FindSuspect[1], /Properties*
Edit → Fields[1]	*/AcroForm:Duplicate[1], /AcroForm:TabOrder[1]*
Document[1]	*/Cpt:CapturePages[1], /AcroForm:Actions[1], /CropPages[1], /Rotate-Pages[1], /InsertPages[1], /ExtractPages[1], /ReplacePages[1], /Delete-Pages[1], /NewBookmark[1], /SetBookmarkDest[1], /CreateAllThumbs[1], /DeleteAllThumbs[1]*
View	*/ActualSize, /FitVisible, /FitWidth, /FitPage, /ZoomTo, /FullScreen, /FirstPage, /PrevPage, /NextPage, /LastPage, /GoToPage, /GoBack, /GoForward, /SinglePage, /OneColumn, /TwoColumns, /Article-Threads, /PageOnly, /ShowBookmarks, /ShowThumbs*
Tools	*/Hand, /ZoomIn, /ZoomOut, /SelectText, /SelectGraphics, /Note[1], /Link[1], /Thread[1], /AcroForm:Tool[1], /Acro_Movie:MoviePlayer[1], /TouchUp:TextTool[1], /Find, /FindAgain, /FindNextNote, /Create-NotesFile[1]*
Tools → Search	*/AcroSrch:Query, /AcroSrch:Indexes, /AcroSrch:Results, /AcroSrch: Assist, /AcroSrch:PrevDoc, /AcroSrch:PrevHit, /AcroSrch:NextHit, /AcroSrch:NextDoc*
Window	*/ShowHideToolBar, /ShowHideMenuBar, /ShowHideClipboard, /Cascade, /TileHorizontal, /TileVertical, /CloseAll*
Help	*/HelpUserGuide[1], /HelpTutorial[1], /HelpExchange[1], /HelpScan[1], /HelpCapture[1], /HelpPDFWriter[1], /HelpDistiller[1], /HelpSearch[1], /HelpCatalog[1], /HelpReader[4], /Weblink:Home*
Help (Windows) or Apple menu (Mac)	*/About*

1. *Only available in Acrobat Exchange.*
2. *Unfortunately, this entry is platform dependent – even if you create the button in Exchange. This means that different buttons for the two platforms are required!*
3. *Only available in Windows or on the Mac with OLE support installed.*
4. *Only available in Acrobat Reader.*

6.6 Additional Tips for Distilling

Distiller parameters. Distiller's job options can be controlled from within the PostScript code. This provides a means to specify certain options within a PostScript file without having to rely on appropriate menu settings. As an additional plus, this makes it possible to change Distiller settings during processing of a single file. For example, according to the image contents one could deploy different compression schemes for certain images (e.g., gray level images), while the regular Distiller options apply to all other images in the file.

Distiller options can be queried and set using the "currentdistillerparams" and "setdistillerparams" PostScript operators. A complete description of these operators can be found under "Acrobat Distiller Parameters" in the Acrobat documentation.

Distilling multiple PostScript files to a single PDF. Sometimes it is necessary or convenient to distill multiple PostScript files to a single PDF document. For example, the whole document may consist of several chapters in individual files, or you want to include your own pdfmark file. Since fonts are embedded only once, combining multiple files decreases the overall file size. Detailed instructions of how this works can be found in Acrobat Distillers's *Xtras* folder.

Automatically creating named destinations. In many cases named destinations are created by application programs which generate the symbolic name by using the content, e.g. the text in a heading. In order to jump to a specific page of a PDF document via a Web link, it may be useful to assign a symbolic name to each page. Named destinations are covered in more detail in Section 8.6.

By making use of a PostScript Level 2 feature, Distiller may be instructed to automatically create a symbolic name for each page. Phil Smith was the first to implement this idea. Copy the file *namedest.ps* from the accompanying CD-ROM to Distiller's startup directory. This file creates a named destination on each page. The names are "Page1", "Page2", etc. These symbolic names may be used as a fragment identifiers after the "#" character:

```
http://www.ifconnection.de/~tm/manual.pdf#Page41
```

Note that the page numbers refer to physical page numbers (starting with page 1 in the document) and not to whatever page number may be typeset in the footer or on another location on the page.