pdfmark Reference Manual

Adobe Developer Technologies

Technical Note #5150

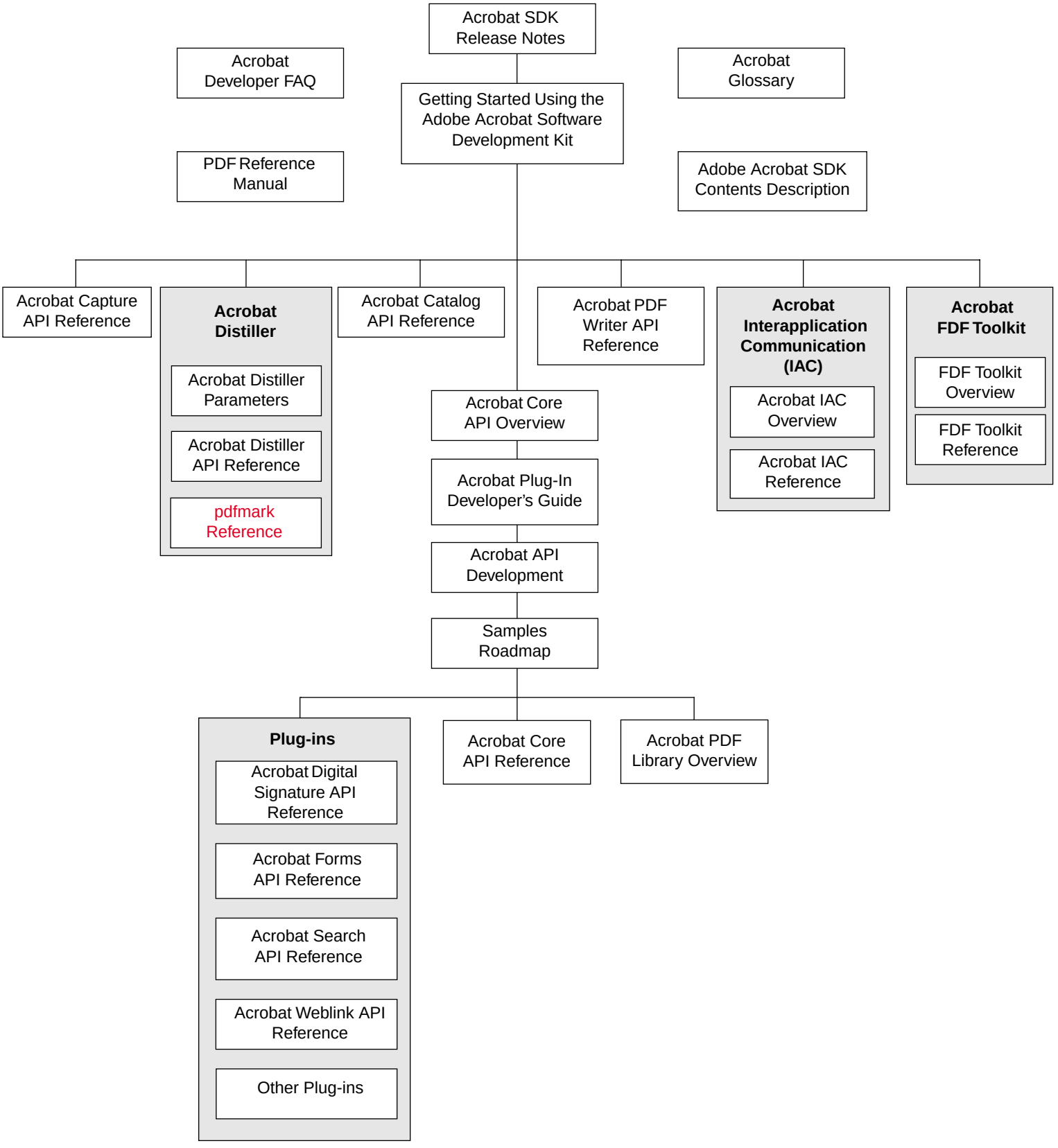
Revised: November 10, 1999

Version History

June 19, 1993	Tim Bienz	First version.
January 26, 1994	Tim Bienz	Updates and corrections.
5 December 1994	Tim Bienz	2.0 revisions.
7 April 1995	Tim Bienz	Reorganized.
7 July 1995	Tim Bienz	2.1 revisions.
24 October 1996	Gary Staas	3.0 revisions.
14 February 1997	Gary Staas	Minor revisions.
4 June 1997	Gary Staas	Add headers for examples.
22 July 1997	Gary Staas	Add link named action example.
5 November 1997	Gary Staas	Add create base URI example.

Version History		
13 January 1999	Gary Staas	Update format. Add structure and supporting commands. Add alternate image sample.
22 October 1999	Denise Stone	Updated 4.05 version.

Documentation Roadmap



Contents

CHAPTER 1

Introduction

1.1 Introduction

The Acrobat® Distiller® application converts PostScript® language files into Portable Document Format (PDF) files. PDF files can include special features such as notes, links, bookmarks, articles, info dictionary entries, and page cropping. This information is not normally present in a PostScript language file and therefore cannot be incorporated into the PDF file by the distilling process.

Note: The terms note, link, and bookmark are used in this document in the same way as they are in the user interface of Acrobat and Reader. These correspond, respectively, to the text annotation, link annotation, and outline entry objects that appear in a PDF file (see the latest version in the Portable Document Format Reference Manual for a description of the PDF file format).

This document describes the syntax and use of the **pdfmark** operator, which is used in PostScript language files to represent PDF features. **pdfmark** is present in implementations of the PostScript Level 2 interpreter used in the Distiller application. This document describes the implementation of the **pdfmark** operator that is present in version 2.1 of the Distiller application. Use of **pdfmark** makes it possible for an Independent Software Vendor (ISV) already supporting the PostScript language to support PDF features without having to write PDF files directly.

Note: It is extremely important to understand how pages in a PDF document are numbered. The **pdfmark** *operator uses the page's sequence number. All pages in a document are*

numbered sequentially; the first page in a document is page 1. All page numbers mentioned in this note must be specified using this sequence number, not the page number as it appears on the printed page.

If you received this technical note without obtaining the entire Acrobat Software Development Kit (SDK), you can get the complete SDK by visiting:

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

1.2 pdfmark Operator

The **pdfmark** operator takes as its arguments a mark object (that is, a"[" character), a variable number of key–value pairs, and a name object. It does not return any values. Each instance of the **pdfmark** operator in a PostScript language file is referred to as a *marker*.

The general syntax of the **pdfmark** operator is:

```
[
...Various key–value pairs...
KIND pdfmark
```

KIND is a name specifying the kind of **pdfmark**.

To ensure that PostScript devices, such as printers, that do not implement the **pdfmark** operator can use files containing that operator, the following PostScript language code should be placed in the prolog of the PostScript language file. It makes each marker a no-op if the PostScript interpreter processing the file does not implement the **pdfmark** operator.

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

1.3 Private Data

Some markers can accept arbitrary key–value pairs, providing a way to put private data into PDF files. All keys must be name objects. Unless otherwise stated, values may be boolean, number, string, name, array, or dictionary objects. Array elements must be boolean, number, string, or name objects.

When specifying arbitrary key–value pairs, key names must contain a specific prefix to ensure that they do not collide with key names used by other developers. Contact Adobe's Developer Technologies group to obtain a prefix to be used by your company or organization.

Note: The private key names in this technical note use the prefix ADBE.

CHAPTER 2

Basic Syntax

This section describes the various forms of the **pdfmark** operator. In general, the key–value pairs used as operands for the **pdfmark** operator follow closely the key–value pairs that appear in the PDF file.

2.1 Annotations

Annotation markers are used to create notes, links, and custom annotations. Annotations are specified using the **pdfmark** operator in conjunction with the name ANN.

Note: Prior to version 2.1 of the Distiller application, annotation markers could only be used to create notes, and the Subtype key was not supported. Link markers were used to create links. It was not possible to create custom annotations.

2.1.1 Notes

The syntax for creating a note is:

```
[/Contents string
/Rect [llx lly urx ury]
/SrcPg pagenum
/Open boolean
/Color array
/Title string
/ModDate datestring
/Subtype string
/ANN pdfmark
```

Table 1 *Note Attributes*

Key	*Type*	*Semantics*
Contents	string	(*Required*) Contains the note's text string. The maximum length of the **Contents** string is 65,535 characters. The encoding and character set used is the PDFDocEncoding (described in Appendix C in the *Portable Document Format Reference Manual)* or Unicode. If Unicode, the string must begin with <FEFF>.
Rect	array	(*Required*) An array of four numbers [*xll*, *yll*, *xur*, *yur*] specifying the lower-left x, lower-left y, upper-right *x*, and upper-right *y* coordinates—in user space—of the rectangle defining the open note window.
SrcPg	integer	(*Optional*) The sequence number of the page on which the note appears. The first page in a document is page 1, not 0. If the **SrcPg** key is present, the note marker may be placed anywhere in the PostScript language file. If omitted, the marker must occur within the PostScript language description for the page on which the note is to appear.
Open	boolean	(*Optional*) If *true*, the note is open, that is, the text is visible. If *false*, the note is closed, that is, displayed as an icon. If omitted, the note has no **Open** key in the PDF file, and the note is closed.
Color	array	(*Optional*) The background color of a note icon, the title bar color of an open active note's window, and the window frame color of an inactive open note's window. The value is an array containing three numbers, each of which must be between 0 and 1, inclusive, specifying a color in the DeviceRGB color space. See Section 7.11 in the *Portable Document Format Reference Manual* for a description of this color space. If omitted, a default color is used.

Table 1 *Note Attributes*

Key	*Type*	*Semantics*
Title	string	(*Optional*) The note's title. The encoding and character set used is either PDFDocEncoding (as described in Appendix C in the *Portable Document Format Reference Manual*) or Unicode. If Unicode, the string must begin with <FEFF>. For example, the Unicode string for (ABC) is <FEFF004100420043>. **Title** has a maximum length of 255 PDFDocEncoding characters or 126 Unicode values, although a practical limit of 32 characters is advised so that it can be read easily in the Acrobat viewer.
ModDate	string	(*Optional*) The date and time the note was last modified. It must be of the form: (*D:YYYYMMDDHHmmSSOHH'mm'*) "*D:*" is an optional prefix. *YYYY* is the year. All fields after the year are optional. *MM* is the month (01-12), *DD* is the day (01-31), *HH* is the hour (00-23), *mm* are the minutes (00-59), and *SS* are the seconds (00-59). The remainder of the string defines the relation of local time to GMT. *O* is either + for a positive difference (local time is later than GMT) or – for a negative difference. *HH'* is the absolute value of the offset from GMT in hours, and *mm'* is the absolute value of the offset in minutes. If no GMT information is specified, the relation between the specified time and GMT is considered unknown. Regardless of whether or not GMT information is specified, the remainder of the string should specify the local time.
Subtype	name	*(Optional, supported in version 2.1 and higher of the Distiller application)* The annotation' s PDF subtype. Must either be omitted or **/Text**.

In addition to the keys listed in Table 1, "Note Attributes," notes may also specify arbitrary key–value pairs.

Example 1 Notes

```
% Simple Note
[ /Rect [ 75 586 456 663 ]
/Contents (This is an example of a note.)
/ANN pdfmark

% Fancy Note
[/Rect [ 75 425 350 563 ]
/Open true
/Title (John Doe)
/Contents (This is an example of a note. \nHere is some text
after a forced line break.

% This is another way to do line breaks.)
/Color [1 0 0]
/Border [0 0 1]
/ANN pdfmark

% Private data in Note
[/Contents (My unimaginative contents)
/Rect [ 400 550 500 650 ]
/Open false
/Title (My Boring Title)

% The following is private data. Keys within the private
% dictionary do not need to use the % organization's prefix
% because the dictionary encapsulates them.
/ADBETest_MyInfo <<
/Routing [ (Me) (You) ]
/Test_Privileges << /Me /All /You /ReadOnly >>
>>
/ADBETest_PrivFlags 42
/ANN pdfmark
```

2.1.2 Links

Note: Prior to version 2.1 of the Distiller application, links could only be specified by using the **pdfmark** *operator in conjunction with the name LNK, and without the Subtype key. This form is supported in version 2.1 for backward compatibility, but should no longer be used.*

The syntax for creating a link is:

```
[/Rect [llx lly urx ury]
/Border [bx by c [d]]
/SrcPg pagenum
/Color array

/Subtype /Link
…Action-specifying key–value pairs…
/ANN pdfmark
```

Table 2 *Link Attributes*

Key	*Type*	*Semantics*
Rect	array	(*Required*) An array of four numbers [*xll*, *yll*, *xur*, *yur*] specifying the lower-left *x*, lower-left *y*, upper-right *x*, and upper-right *y* coordinates—in user space—of the rectangle defining the link button.
Border	array	(*Optional*) The link's border properties. **Border** is an array containing three numbers and, optionally, an array. All elements are specified in user space coordinates. If **Border** is of the form [bx by c], the numbers specify the horizontal corner radius (bx), the vertical corner radius (by), and the width (c) of the link's border. The link has a solid border. If it is of the form [bx by c [d]], the fourth element (d) is a dash array that specifies the lengths of dashes and gaps in the link's border. The default value for **Border** is [0 0 1].
SrcPg	integer	(*Optional*) The sequence number of the page on which the link is to appear. The first page in a document is page 1, not 0. If the **SrcPg** key is present, the link marker may be placed anywhere in the PostScript language file. A link marker that does not contain this key must occur within the PostScript language description for the link's source page.

Table 2 *Link Attributes*

Key	*Type*	*Semantics*
Color	array	(*Optional*) The link's border color. The value is an array containing three numbers, each of which must be between 0 and 1, inclusive, specifying a color in the DeviceRGB color space. See Section 7.10 in the *Portable Document Format Reference Manual* for a description of this color space. If omitted, a default color is used.
Subtype	name	*(Required)* The annotation' s PDF subtype. Must be **/Link**.

In addition to the keys listed in Table 2, "Link Attributes," a link must contain key–value pairs that specify an action such as traversing a link (see "Specifying Destinations and Actions" for more information), and may also specify arbitrary key–value pairs.

Example 2 Links

```
% Simple link (old style, compatible with all Distiller
% application versions)
[/Rect [ 70 650 210 675 ]
/Page 3
/View [ /XYZ -5 797 1.5]
/LNK pdfmark

% Fancy link
[/Rect [ 70 550 210 575 ]
/Border [ 0 0 2 [ 3 ] ]
/Color [0 1 0]
/Page /Next
/View [ /XYZ -5 797 1.5 ]
/Subtype /Link
/ANN pdfmark

% Link
[/Rect [ 70 650 210 675 ]
/Border [ 16 16 1 ]
/Color [1 0 0]
/Page 1
/View [ /FitH 5]
/Subtype /Link
/ANN pdfmark
```

```
% Link to a named destination
[/Rect [ 70 650 210 675 ]
/Border [ 16 16 1 [ 3 10 ] ]
/Color [ 0 .7 1 ]
/Dest /MyNamedDest
/Subtype /Link
/ANN pdfmark

% Link that launches another file
[/Rect [ 70 600 210 625 ]
/Border [ 16 16 1 ]
/Color [0 0 1]
/Action /Launch
/File (test.doc)
/Subtype /Link
/ANN pdfmark

% Custom Link action (URI link for the Acrobat WebLink plug-
% in)
[/Rect [ 50 425 295 445 ]
/Action << /Subtype /URI /URI
(http://www.adobe.com) >>
/Border [ 0 0 2 ]
/Color [ .7 0 0 ]
/Subtype /Link
/ANN pdfmark
```

2.1.3 Custom Annotations

Custom annotation support allows the creation of arbitrary annotation types, such as video, sound, or graphics.

The syntax for creating a custom annotation is:

```
[/Subtype string
…Appropriate key–value pairs…
/ANN pdfmark
```

Table 3 *Custom Annotation Attributes*

Key	*Type*	*Semantics*
Subtype	name	*(Required)* The annotation's PDF subtype. See Section 6.6 in the *Portable Document Format Reference Manual* for more information.

In addition to the key–value pair listed in Table 3, "Custom Annotation Attributes," custom annotations may contain arbitrary key–value pairs, which are interpreted as they would be if a link marker was used (except that the **/Border** attribute is not recognized).

Example 3 Custom Annotations

```
% Custom annotation type.
% The Acrobat viewers do not know how to interpret this
% annotation type.
% So this appears with an unknown annotation icon unless an
% installed plug-in can draw it.
[/Rect [ 400 435 500 535 ]
/Subtype /ADBETest_DummyType
/ADBETest_F8Array [ 0 1 1 2 3 5 8 13 ]
/ANN pdfmark
```

2.2 Bookmarks

A bookmark, known as an outline entry in PDF, is specified by using the **pdfmark** operator in conjunction with the name OUT.

The syntax for a bookmark marker is:

```
[/Title string
/Count int
…Action-specifying key–value pairs…
/OUT pdfmark
```

Table 4 *Bookmark Attributes*

Key	*Type*	*Semantics*
Title	string	(*Required*) The bookmark's text. The encoding and character set used is either PDFDocEncoding (as described in Appendix C in the *Portable Document Format Reference Manual*) or Unicode. If Unicode, the string must begin with <FEFF>. For example, the Unicode string for (ABC) is <FEFF004100420043>. **Title** has a maximum length of 255 PDFDocEncoding characters or 126 Unicode values, although a practical limit of 32 characters is advised so that it can be read easily in the Acrobat viewer.
Count	integer	(*Required if the bookmark has subordinate bookmarks*) Specifies the number and visual appearance of subordinate bookmarks, using the following rules: 1. Bookmark markers must appear in sequential order in the PostScript language file. 2. A bookmark with no subordinate bookmarks must omit the **Count** key. 3. A bookmark with subordinate bookmarks (for example, a chapter bookmark with several subordinate section headings) must include the **Count** key. The absolute value of this key is the number of bookmarks immediately subordinate—that is, excluding subordinates of subordinates. If the value is positive, the parent bookmark is open; if negative, the parent bookmark is closed. An open bookmark shows its subordinates, while a closed bookmark does not.

In addition to the keys listed in Table 4, "Bookmark Attributes," a bookmark must contain key–value pairs that specify an action. See "Specifying Destinations and Actions" for more information.

Bookmark markers may occur anywhere in the PostScript language file subject only to the ordering constraints imposed by the **Count** key.

Example 4 Bookmarks

```
% Bookmarks
[/Count 2 /Page 1 /View [/XYZ 44 730 1.0] /Title (Open
Actions) /OUT pdfmark
[/Action /Launch /File (test.doc) /Title (Open test.doc) /OUT
pdfmark
[/Action /GoToR /File (test.pdf) /Page 2 /View [/FitR 30 648
209 761]
/Title (Open test.pdf on page 2) /OUT pdfmark
[/Count 2 /Page 2 /View [/XYZ 44 730 1.0] /Title (Fixed Zoom)
/OUT pdfmark
[/Page 2 /View [/XYZ 44 730 2.0] /Title (200% Magnification) /
OUT pdfmark
[/Count 1 /Page 2 /View [/XYZ 44 730 4.0] /Title (400%
Magnification) /OUT pdfmark
[/Page 2 /View [/XYZ 44 730 5.23] /Title (523% Magnification)
/OUT pdfmark
[/Count 3 /Page 1 /View [/XYZ 44 730 1.0] /Title (Table of
Contents #1) /OUT pdfmark
[/Page 1 /View [/XYZ 44 730 1.0] /Title (Page 1 - 100%) /OUT
pdfmark
[/Page 2 /View [/XYZ 44 730 2.25] /Title (Page 2 - 225%) /OUT
pdfmark
[/Page 3 /View [/Fit] /Title (Page 3 - Fit Page) /OUT pdfmark
[/Count -3 /Page 1 /View [/XYZ 44 730 1.0] /Title (Table of
Contents #2) /OUT pdfmark
[/Page 1 /View [/XYZ null null 0] /Title (Page 1 - Inherit) /
OUT pdfmark
[/Page 2 /View [/XYZ null null 0] /Title (Page 2 - Inherit) /
OUT pdfmark
[/Page 3 /View [/XYZ null null 0] /Title (Page 3 - Inherit) /
OUT pdfmark
[/Count 1 /Page 0 /Title (Articles) /OUT pdfmark
[/Action /Article /Dest (Now is the Time) /Title (Now is the
Time) /OUT pdfmark
%Bookmark with a URI as an action
[/Count 0 /Title (Adobe's Home page)
/Action << /Subtype /URI /URI (http://www.adobe.com)>> /OUT
pdfmark
```

2.3 Articles

Articles consist of a title and a list of rectangular areas called *beads*. Each bead is specified by using the **pdfmark** operator in conjunction with the name ARTICLE. Beads are added to the article in the order that they are encountered in the PostScript language file.

The syntax for a bead marker is:

```
[/Title string
/Rect [llx lly urx ury]
/Page pagenum
/ARTICLE pdfmark
```

Table 5 *Article Bead Attributes*

Key	*Type*	*Semantics*
Title	string	(*Required*) The title of the article to which a bead belongs. The encoding and character set used is either PDFDocEncoding (as described in Appendix C in the *Portable Document Format Reference Manual*) or Unicode. If Unicode, the string must begin with <FEFF>. For example, the Unicode string for (ABC) is <FEFF004100420043>. **Title** has a maximum length of 255 PDFDocEncoding characters or 126 Unicode values, although a practical limit of 32 characters is advised so that it can be read easily in the Acrobat viewer.
Rect	array	(*Required*) An array of four numbers [*xll*, *yll*, *xur*, *yur*] specifying the lower-left x, lower-left y, upper-right x, and upper-right y coordinates—in user space—of the rectangle defining the bead.
Page	integer	(*Optional*) The sequence number of the page on which the bead is located. A bead marker that contains the optional **Page** key may be placed anywhere in the PostScript language file. A bead marker that does not contain this key must occur within the PostScript language description for the page on which the article bead is to appear.

In addition to the keys listed in Table 5, “Article Bead Attributes,” the first bead in an article may also specify arbitrary key–value pairs. Suggested keys are **Subject**, **Author**, and **Keywords**.

Note: Articles do not support dictionaries as values in arbitrary key–value pairs.

Example 5 Article

```
% Article containing two beads
 [/Title (Now is the Time)
/Author (John Doe)
/Subject (Coming to the aid of your country)
/Keywords (Time, Country, Aid)
/Rect [ 225 500 535 705 ]
/Page 2
/ARTICLE pdfmark
[/Title (Now is the Time)
/Rect [ 225 500 535 705 ]
/Page 3
/ARTICLE pdfmark
```

2.4 Named Destinations

The destination of a bookmark or link may be specified as a name instead of as a page number and view. When the Acrobat viewer encounters such a *named destination*, it looks through the PDF file’s table of named destinations to determine the page and view corresponding to the name. The advantage of using named destinations, particularly for cross-document links, is that it allows the document containing a link’s destination to be revised asynchronously from the document containing the link’s source, eliminating concern over whether link destinations moved to different pages. A named destination is specified by using the **pdfmark** operator in conjunction with the name DEST.

The syntax for a named destination marker is:

```
[/Dest name
/Page pagenum
/View destination
/DEST pdfmark
```

Table 6 *Named Destination Attributes*

Key	*Type*	*Semantics*
Dest	name	(*Required*) The destination's name.
Page	integer	(*Optional*) The sequence number of the destination page. If present, the named destination marker may be placed anywhere in the PostScript language file. If omitted, the marker must occur within the PostScript language description for the destination page.
View	array	(*Optional*) The view to display on the destination page. If omitted, defaults to a null destination (lower left corner of the page at a zoom of 100%). See "View Destinations" for information on specifying a view destination.

In addition to the keys listed in Table 6, "Named Destination Attributes," named destinations may also specify arbitrary key–value pairs.

In Acrobat 3.0, named destinations may be appended to URLs, following a "#" character, as in *http://www.adobe.com/test.pdf#nameddest=name*. The Acrobat viewer displays the part of the PDF file specified in the named destination.

Note: The 2.0 and 2.1 versions of the Acrobat products supported a maximum of approximately 4000 named destinations in a PDF file. The 3.0 version removed this limit.

Example 6 Named Destination

```
[ /Dest /MyNamedDest
/Page 1
/View [ /FitH 5 ]
/DEST pdfmark
```

2.5 Pass-through PostScript Language Commands

Blocks of PostScript language code can be specified by using the **pdfmark** operator in conjunction with the name PS. Any PostScript language code specified in this manner is copied directly into the PDF file without being distilled, and is ignored when the PDF file is viewed using Acrobat or Reader. It is only used when the PDF file is printed to a PostScript printer.

Note: Pass-through PostScript language code should be used only when PDF does not provide another way to achieve the same result.

The syntax for specifying a block of pass-through PostScript language code is:

```
[/DataSource string or file
/Level1 string or file
/PS pdfmark
```

Table 7 *Pass-through PostScript Language Command Attributes*

Key	*Type*	*Semantics*
DataSource	string or file	(*Required*) The PostScript language commands to copy into the PDF file. See the discussion of the **file** operator in the *PostScript Language Reference Manual, Third Edition* for information on specifying files.
Level1	string or file	(*Optional*) PostScript Level 1 language code that is used instead of the value of the **DataSource** key when printing to a printer that supports only PostScript Level 1.

This marker must be placed in the PostScript language program at the point where the block of code is to be executed during printing.

Example 7 Pass-through PostScript Language Code

```
% Note that this is not an appropriate use of the PS marker,
% since PDF supports commands to draw lines
[/DataSource (0 0 moveto 100 700 lineto stroke)
/PS pdfmark
```

2.6 Page Cropping

Page cropping is specified by using the **pdfmark** operator in conjunction with the names PAGES and PAGE. PAGES specifies the default page cropping for all pages in a document, while PAGE specifies the page cropping only for the current page.

The syntax for specifying the default page cropping for a document is:

```
[/CropBox [llx lly urx ury]
/PAGES pdfmark
```

This marker can be placed anywhere in the PostScript language program, but it is recommended that it be placed at the beginning of the file, in the Document Setup section between the document structuring comments **%%BeginSetup** and **%%EndSetup**, before any marks are placed on the first page.

The syntax for specifying a non-default page cropping for a particular page in a document is:

```
[/CropBox [llx lly urx ury]
/PAGE pdfmark
```

This marker must be placed before the **showpage** operator for the page it is to affect. It is recommended that it be placed before any marks are made on the page. For example, this marker affects only the first page of a document if it is placed before any marks are made on the first page.

Table 8 *Page Cropping Attributes*

Key	*Type*	*Semantics*
CropBox	array	(Required) The location and size of the viewable area of the page. **CropBox** is an array of four numbers [*xll*, *yll*, *xur*, *yur*] specifying the lower-left x, lower-left y, upper-right x, and upper-right y coordinates—measured in *default* user space—of the rectangle defining the cropped page. The minimum allowed page size is 1×1 inch (72×72 units in the default user space coordinate system) and the maximum allowed page size is 45×45 inches (3240×3240 units in the default user space coordinate system).

Example 8 Page Cropping

```
% Crop all pages
[/CropBox [54 403 558 720] /PAGES pdfmark

% Crop this page
[/CropBox [0 0 612 792] /PAGE pdfmark
```

2.7 Info Dictionary

A document's Info dictionary contains key–value pairs that provide various pieces of information about the document. Info dictionary information is specified by using the **pdfmark** operator in conjunction with the name DOCINFO.

The syntax for specifying Info dictionary entries is:

```
[/Author string
/CreationDate string
/Creator string
/Producer string
/Title string
/Subject string
/Keywords string
/ModDate string
/DOCINFO pdfmark
```

Table 9 *Info Dictionary Attributes*

Key	*Type*	*Semantics*
Author	string	(*Optional*) The document's author.
CreationDate	string	(*Optional*) The date the document was created. See the description of the **ModDate** key for information on the string's format.
Creator	string	(*Optional*) If the document was converted to PDF from another form, the name of the application that originally created the document.
Producer	string	(*Optional*) The name of the application that converted the document from its native form to PDF.
Title	string	(*Optional*) The document's title.
Subject	string	(*Optional*) The document's subject.
Keywords	string	(*Optional*) Keywords relevant for this document. These are used primarily in cross-document searches.
ModDate	string	(*Optional*) The date and time the document was last modified. It should be of the form: (*D:YYYYMMDDHHmmSSOHH'mm'*) "*D:*" is an optional prefix. *YYYY* is the year. All fields after the year are optional. *MM* is the month (01-12), *DD* is the day (01-31), *HH* is the hour (00-23), *mm* are the minutes (00-59), and *SS* are the seconds (00-59). The remainder of the string defines the relation of local time to GMT. O is either + for a positive difference (local time is later than GMT) or – for a negative difference. *HH'* is the absolute value of the offset from GMT in hours, and *mm'* is the absolute value of the offset in minutes. If no GMT information is specified, the relation between the specified time and GMT is considered unknown. Regardless of whether or not GMT information is specified, the remainder of the string should specify the local time.

Info dictionary markers may occur anywhere in the PostScript language file.

In addition to the keys listed in Table 9, “Info Dictionary Attributes,” arbitrary keys can be specified. Values must be string objects.

Example 9 Info Dictionary Entries

```
[/Title (My Test Document)
/Author (John Doe)
/Subject (pdfmark 3.0)
/Keywords (pdfmark, example, test)
/Creator (Hand Programmed)
/ModificationDate (D:19940912205731)
/ADBETest_MyKey (My private information)
/DOCINFO pdfmark
```

2.8 Catalog Dictionary

A document’s Catalog dictionary contains key–value pairs that provide various pieces of information about the document. The following Catalog dictionary information can be set using the **pdfmark** operator:

- The action that occurs when the file is opened.
- The way the document is displayed when it is opened. The choices allow the display of only the document, the document plus thumbnail images, the document plus bookmarks, or just the document in full screen mode.

Catalog dictionary information is specified by using the **pdfmark** operator in conjunction with the name DOCVIEW.

The syntax for specifying Catalog dictionary entries is:

```
[/PageMode name
…Action-specifying key–value pairs…
/DOCVIEW pdfmark
```

Table 10 *Catalog Attributes*

Key	*Type*	*Semantics*
PageMode	name	(*Optional*) Specifies how the document is displayed when it is opened. Must be one of the following: *UseNone* — Open the document, displaying neither bookmarks nor thumbnail images. *UseOutlines* — Open the document and display bookmarks. *UseThumbs* — Open the document and display thumbnail images. *FullScreen* — Open the document in full screen mode. The default value is *UseNone*.

In addition to the keys listed in Table 10, "Catalog Attributes," a Catalog that contains an open action must have additional key–value pairs specifying the action. See "Specifying Destinations and Actions" for information.

Catalog dictionary markers may occur anywhere in the PostScript language file.

Example 10 Catalog Dictionary Entries

```
[/PageMode /UseOutlines
/Page 2 /View [/XYZ null null null]
/DOCVIEW pdfmark
```

CHAPTER 3

Cos Objects

Cos objects are the building blocks of PDF files. Version 3.0 of the Acrobat Distiller application adds the capability of defining composite (array, dictionary, and stream) Cos objects directly. A PostScript language program can create Cos objects, name them, and create indirect references to them in other objects by using these names. Cos objects are created using the **pdfmark** operator in conjunction with the name OBJ.

3.1 Naming Objects with _objdef

The syntax for specifying a Cos object is:

```
[/_objdef {OBJNAME}
/type name
/OBJ pdfmark
```

Table 11 *Cos Object Attributes*

Key	*Type*	*Semantics*
_**objdef**	special	(*Required*) The name of the Cos object. Must be enclosed by curly braces, as in {myobjectname}. *Note A Cos object name is not a standard name object, that is, it does not start with a slash "/".*
type	name	(*Required*) The type of Cos object. Must be one of the following: array — Create an array. dict — Create a dictionary. stream — Create a stream.

The */_objdef* {OBJNAME} key–value pair can also be added to the following **pdfmark** commands:

- ANN — annotation
- BP — encapsulated graphic
- DEST — destination
- LNK — link
- NI — encapsulated image
- PS — embedded PostScript
- StPNE — structure element

Note: The name given by _objdef exists only during the distillation process and has no relationship to any identifier created in the output PDF file.

All the Cos objects created with these **pdfmark** commands are dictionaries. This allows names to be given to other objects created by **pdfmark** operators, such as annotations. Simply prefix the object's **pdfmark** definition with a /_objdef {OBJNAME} key–value pair:

```
[/_objdef {OBJNAME}
...pdfmark operator that creates object...
```

Example 11 shows how a text annotation can be created and named for later reference.

Example 11 Naming a Text Annotation

```
[/_objdef {TextAnnot} /Contents (Added text annot)
/Rect [200 200 300 300] /Subtype /Text /ANN pdfmark
```

Cos objects created with an OBJ **pdfmark** can be used to define other Cos objects. An OBJNAME in curly braces can be passed to a **pdfmark** operator as the value in a key–value pair or as an element in an array. In this case, the Distiller program places an indirect reference to that object in the PDF file.

Example 12 contains a **pdfmark** to create a text annotation on the current page with extra keys in the annotation dictionary, /MyPrivateAnnotArrayData and /MyPrivateAnnotDictData, with values that are indirect references to the array and dictionary objects created by the previous **pdfmark** entries.

Example 12 pdfmarks Referencing Cos Objects

```
[/_objdef {myarray} /type /array /OBJ pdfmark
[/_objdef {mydict} /type /dict /OBJ pdfmark
[
/MyPrivateAnnotArrayData {myarray}
/MyPrivateAnnotDictData {mydict}
/SubType /Text
/Rect [500 500 550 550]
/Contents (Here is a text annotation)
/ANN pdfmark
```

Note: Names defined by _objdef are in the namespace governed by the stack operators NamespacePush and NamespacePop, defined in “Namespace Commands”*.*

3.2 Implicitly Named Objects

In addition to named Cos objects created by the OBJ **pdfmark**, there are several implicitly named objects:

- {Catalog} — the PDF file’s Catalog dictionary
- {DocInfo} — the PDF file’s Info dictionary
- {PageN} — the dictionary for page N (where N is a positive integer)
- {ThisPage} — the dictionary for the current page being processed in the PostScript stream
- {PrevPage} — the dictionary for the page before the current page
- {NextPage} — the dictionary for the page after the current page

To put information in composite objects created with OBJ, use the PUT, PUTINTERVAL, or CLOSE **pdfmark** names.

3.3 PUT

The **pdfmark** operator in conjunction with the name PUT allows information to be added to Cos objects created with the OBJ **pdfmark** operator.

The syntax for PUT has several forms, depending on the composite object added to:

```
[{ARRAYNAME} index value /PUT pdfmark
[{DICTNAME} <<key1 value1...>> /PUT pdfmark
[{STREAMNAME} string /PUT pdfmark
[{STREAMNAME} file /PUT pdfmark
```

The object name is an implicitly defined name, such as {Catalog} or {Page33}, or was defined previously in either an OBJ **pdfmark** or /_objdef {OBJNAME} key–value pair in another **pdfmark**.

For array Cos objects, PUT inserts the value argument at the location index. Indices start at 0, and the array grows automatically to hold the largest index specified. Unspecified entries are made *NULL* objects.

For dictionary Cos objects, PUT adds the key–value pairs specified as arguments.

For stream Cos objects, PUT concatenates the data provided to the stream object. The source of stream data may be either a string or a file. For a file source, file is the PostScript file entity, defined by the PostScript **file** operator. Stream data is always compressed using a lossless method (either LZW or ZIP, depending on the compatibility level set for the Distiller program).

3.4 PUTINTERVAL

The **pdfmark** operator in conjunction with the name PUTINTERVAL adds multiple entries to an array object, starting at an index.

The syntax for PUTINTERVAL is:

```
[ARRAYNAME} index [value1 ... valuen] /PUTINTERVAL pdfmark
```

The array is resized if necessary to hold the objects added.

3.5 CLOSE

The **pdfmark** operator in conjunction with the name CLOSE closes a stream object created by **pdfmark**.

The syntax for CLOSE is:

```
[{STREAMNAME} /CLOSE pdfmark
```

The named stream object is closed and written to the PDF file. The name is still valid and may be referenced by other objects, but it can no longer be written to. When the Distiller application completes writing a PDF file, any open streams are closed and written automatically.

Example 13 shows how to create and reference Cos objects with **pdfmark**.

Example 13 Creating Cos Objects with pdfmark

```
% Create composite objects
[/_objdef {myarrayname} /type array /OBJ pdfmark
[/_objdef {mydictname} /type dict /OBJ pdfmark
[/_objdef {mystreamname} /type stream /OBJ pdfmark

% Add values to objects
% insert 132 at location 0
[{myarrayname} 0 132 /PUT pdfmark

% insert key–value pair into dictionary
[{mydictname} << /TheKey 366 >> /PUT pdfmark
% insert string into stream object
[{mystreamname} (any string) /PUT pdfmark

% Use predefined named objects
% insert key–value pair into Catalog
[{Catalog} << /Answer 42 >> /PUT pdfmark

% insert key–value pair into Page 25’s dictionary
[{Page25} << /SpecialKey (special string) >> /PUT pdfmark

% insert key–value pair into the current page’s dictionary
[{ThisPage} << /NewKey (new string) >> /PUT pdfmark

% Create an annotation with a name and add to it
% create text annotation
[/_objdef {MikesAnnot} /Contents (a simple text annot)
/Rect [100 100 200 200] /Subtype /Text /ANN pdfmark
```

```
% add another key to this text annotation
[{MikesAnnot} << /AnotherKey (another string value) >> /PUT
pdfmark
```

Note: A PostScript language program can make an object reference {foo} before defining the object {foo}. If {foo} is never defined, it is left as an unresolved reference in the xref table. Hence any consumer of such a PDF file must be able to handle unresolved references.

CHAPTER 4

Logical Structure

In Version 1.3, PDF files can contain *structure trees* giving a logical structure to the information in a document. This section defines the structure suite used in conjunction with the **pdfmark** operator that can be used to specify logical structure within PDF files.

The facilities for describing logical structure in PDF are described in Section 6.17 in the *Portable Document Format Reference Manual*, *Version 1.3*. The uses of **pdfmark** specified here correspond closely to the constructs presented there. You should be familiar with these facilities for logical structure.

The structure suite has an *implicit parent stack* of elements whose top item is the parent for any created element and content. The structure of the tree is thus determined by pushing elements on this stack and adding children.

Example 14 gives a flavor of the structure suite. The example shows an entire structure tree, consisting of one section containing two paragraphs. It illustrates both how to create the tree structure and how the structure is related to the page content of the PDF file. Example 15 shows the parts of the output PDF file that result from the PostScript language code.

Example 14 A Simple Structure

```
% One section with two paragraphs, all on one page.
% On the first page:

% Start a section with the unnamed Structure Tree as parent.
% Push the Section element onto the implicit parent stack as
% current
```

```
% implicit parent.
[/Subtype /Section
/StPNE pdfmark

% Start a paragraph with the Section as implicitly-specified
% parent.
% Push the Paragraph element on top of the implicit parent
% stack as the current implicit parent.
[/Subtype /P
/StPNE
pdfmark

% Begin the marked content holding the text of the
% first paragraph. It is implicitly added to the Paragraph
% element.
[/StBMC pdfmark
% [PostScript code for the contents of the first paragraph
% goes % here.]

% End the marked content holding the text of the first
% paragraph.
[/EMC pdfmark

% Pop the Paragraph element off the implicit parent stack.
% This exposes the Section element as implicit parent again.
[/StPop pdfmark

% And now for the second paragraph:
[/Subtype /P
/StPNE
pdfmark

[/StBDC pdfmark
% [PostScript code for the contents of the second paragraph
% goes % here.]
[/EMC pdfmark

% We're being tidy by popping both the second Paragraph
% element and the Section element off the stack. We could have
% left everything hanging at the end of the document, or used
% [/StPopAll pdfmark.
[/StPop pdfmark
[/StPop pdfmark
```

Example 15 PDF Output Resulting from Code in Example 14

```
% [This example is for illustration only. The PDF code
% actually produced by Adobe Acrobat Distiller application
% may differ.]
% In the Catalog dictionary, under the key StructTreeRoot,
% the % following dictionary is entered as object 3 0:3 0 obj
<</Type /StructTreeRoot
% The Section element is the only kid.
/K [4 0 R]
/ParentTree 100 0 R
>> endobj

% The number tree that locates structure parents of marked
% content.
100 0 obj<<
/Nums [0 101 0 R]
>> endobj

% Structure parents for page 1.
101 0 obj[5 0 R 6 0 R] endobj
% End of parent tree objects.
% As object 4 0, the following dictionary representing the
% Section element:
4 0 obj
<</Type /StructElement
/S /Section
% Parent link, refers back to the dictionary representing the
% Structure Tree Root.
/P 3 0 R
% The Section element has two Paragraph elements as kids.
/K [5 0 R 6 0 R]
>> endobj

% Object 5 0, the first Paragraph element
5 0 obj
<</Type /StructElement
/S /P
/P 4 0 R
% Page in whose content stream integer Marked Content ID’s
% denote Kids
/Pg 10 0 R
/K [0]
>> endobj

% Object 6 0, the second Paragraph element
```

```
6 0 obj
<</Type /StructElement
/S /P
/P 4 0 R
% Page in whose content stream integer Marked Content ID’s
% denote Kids
/Pg 10 0 R
/K [1]
>> endobj

% Object 10 0, the Page object for the page on which both
% paragraphs are marked. Only the relevant entries in the
% dictionary are shown.
% The Resources dictionary of the Contents stream of the page.
<</StructParents 0
>>
% Inside the Contents stream of the page.
/P <</MCID 0>> BDC
% [Paragraph 1 content marking goes here.]
EMC
/P <</MCID 1>> BDC
% [Paragraph 2 content marking goes here]
EMC
```

4.1 Structure Operators

This section lists the names used in conjunction with the **pdfmark** operator that make up the structure suite.

- “StRoleMap” adds entries to the role map.
- “StClassMap” adds entries to the class map.
- “StPNE” creates a new structure element.
- “StBookmarkRoot” creates a root bookmark for a structure bookmark tree.
- “StPush” pushes an existing element onto the implicit parent stack.
- “StPop” pops an element off the implicit parent stack.
- “StPopAll” completely empties the implicit parent stack.
- “StBMC” indicates the beginning of marked content.

- “StBDC” indicates the beginning of marked content with a dictionary.
- “EMC” delimits the end of marked content.
- “StOBJ” adds an existing PDF object as part of an element’s content.
- “StAttr” enables the attachment of attribute objects to elements.
- “StStore” saves the current state of the implicit parent stack.
- “StRetrieve” restores the implicit parent stack from a saved state.

Most of these names are directly related to the features of the logical structure PDF facility, but some only manipulate the state of the PDF creation process, without corresponding to any particular output. The descriptions are grouped functionally by the logical structure feature or process state that they control.

Section 4.2 through Section 4.9 fill in the details of the structure suite. “Structure Examples” gives a variety of usage examples.

4.2 Structure Tree Root

The Acrobat Distiller application automatically creates a new Structure Tree Root the first time it creates a new element with “StPNE”.

The following two names (StRoleMap and StClassMap) can be used in conjunction with **pdfmark** to add information to the Structure Tree Root.

4.2.1 StRoleMap

StRoleMap adds entries to the role map of the Structure Tree Root. If the Structure Tree Root does not already exist, it is created. The dictionary entries added are provided as key–value pairs with StRoleMap. A role map dictionary is created for the Structure Tree Root if one

does not already exist. A given key–value pair always modifies the role map, even if the key is already in the dictionary, even for a single invocation of StRoleMap.

The syntax for adding entries to a role map is:

```
[/<new element subtype name>
/<standard structural subtype name>
...
/<new element subtype name>
/<standard structural subtype name>
/StRoleMap pdfmark
```

4.2.2 StClassMap

StClassMap behaves like StRoleMap, except that it adds entries to the class map of the Structure Tree Root, rather than the role map.

The syntax for adding entries to a class map is:

```
[/<class name> {<attribute object name>}
...
/<class name> {<attribute object name>}
/StClassMap pdfmark
```

The <attribute object name> can be any name defined with _objdef, but must represent either a dictionary or stream PDF object. You can use _objdef to associate an object with a name in a variety of ways, as noted in "Naming Objects with _objdef" and "Implicitly Named Objects".

4.3 Elements

The structure suite provides several commands concerned with creating elements and linking them into Structure Trees. All element creation operators share the common set of attributes in Table 12, "Common Element Attributes".

Table 12 *Common Element Attributes*

Key	*Type*	*Semantics*
Subtype	name	(*Required*) The element type, such as Link or Section.
Title	string	(*Optional*) A human-readable name for the particular element.
Alt	string	(*Optional*) An alternate representation of the element's contents as human-readable text.
ID	string	(*Optional*) A unique identifier for the element. The identifier must be unique within the document in which the element occurs. It is an error to specify an element with the same ID as an existing element in the same tree.
Class	name	(*Optional*) The class name to be associated with the element.

4.4 Specifying Parents and Containers

pdfmarks that create elements or element content require specification of the parent (in the case of elements) or the containing element (in the case of element content). Many applications of the structure suite are expected to have an element structure conforming to the physical structure within the document. Since nesting structures are easily processed in terms of a runtime stack, the structure suite maintains an *implicit parent stack* whose top item serves as the current implicit parent for element and content creation operators. The items on the stack can be either elements or the Structure Tree Root.

pdfmark operators that create an element use the element on the top of the processing application's implicit parent stack as the new element's parent. If the processing application's implicit parent stack is empty, the document's Structure Tree Root is made the parent; the Structure Tree Root is created if it does not already exist.

Some operators specify an element but cannot accept the Structure Tree Root as the implicit argument. These commands are in error if the implicit parent stack is empty

when they are encountered or if the top item on the stack is the Structure Tree Root rather than an element. These cases are noted in the command descriptions.

4.5 Element Operators

4.5.1 StPNE

StPNE ("Push New Element") creates a new element with the element on the top of the implicit parent stack as its parent. If the implicit parent stack is empty, the Structure Tree Root is pushed onto the stack and used as the parent. If there is no Structure Tree Root, one is created, pushed onto the stack, and immediately used as the parent.

The syntax for a creating a new element is:

```
[/Subtype name
/_objdef {OBJNAME}
/Title string
/Alt string
/ID string
/Class name
/At integer
/Bookmark dictionary
/StPNE pdfmark
```

The dictionary in an invocation of StPNE may contain the keys given in Table 12, "Common Element Attributes", Table 13, "Specifying Position Within a Container", and Table 14, "Specifying a bookmark".

The keys in Table 12 specify element attributes.

A new element is added to its parent at the index specified with the **At** key in Table 13. The newly-created element is pushed onto the implicit parent stack.

To allow subsequent **pdfmark** operators to refer to the element being created, StPNE may also take the key _objdef, defined in "Naming Objects with _objdef". Once an element is named, it can be referenced with the **E** key, described in Table 17, "Referring to an Existing Element or Structure Tree Root".

A bookmark can be automatically generated for an element using the **Bookmark** key in Table 14, “Specifying a bookmark”. The key’s value is a bookmark dictionary, which may contain most of the keys of the OUT **pdfmark**. The bookmark dictionary contains the **Title** and **Open** keys described in Table 15, “Bookmark dictionary”, which set the bookmark’s title and open state.

The bookmark dictionary may also contain the key-value pairs that specify an action in Table 22, “Action Types”. (See “Specifying Destinations and Actions” for more information on actions.) If an action is specified, the element is added to the structure bookmark subtree unconditionally. If none of these action keys are present, the bookmark’s action is to go to either the first page where a marked content is a kid of this element or a kid in one of its descendant elements.

If the **Bookmark** key is present (even with an empty dictionary), this element is added to the Structured Bookmark subtree. Example 16 defines a bookmark for an element.

Example 16 A Bookmark for a Structural Element

```
[ …other /StPNE key-value pairs…
/Bookmark
<<
/Title (an element in my structure)
/Open true
>>
/StPNE pdfmark
```

Table 13 *Specifying Position Within a Container*

Key	*Type*	*Semantics*
At	integer	(*Optional*) Index at which to insert this item within its parent. If omitted, the child item is added as the *last* child of its parent, retaining all existing items in their original positions. If less than or equal to zero, the new item becomes the *first* child of its parent. If greater than or equal to the current number of children of the intended parent, the new item becomes the *last* element of its parent. If the index is any other number, the item is inserted at that index within the container, and all items that had indices greater than or equal to the given index are shifted to the position with index one greater. An item may be an element, marked content, or a PDF object.

Table 14 *Specifying a bookmark*

Key	*Type*	*Semantics*
Bookmark	dictionary	(*Optional*) Specifies a bookmark that is generated for this structural element. Table 15, "Bookmark dictionary" describes this dictionary.

Table 15 *Bookmark dictionary*

Key	*Type*	*Semantics*
Title	string	(*Optional*) Bookmark title. The encoding and character set used is either PDFDocEncoding (as described in Appendix C in the *Portable Document Format Reference Manual*) or Unicode. If Unicode, the string must begin with <FEFF>. For example, the Unicode string for (ABC) is <FEFF004100420043>. **Title** has a maximum length of 255 PDFDocEncoding characters or 126 Unicode values, although a practical limit of 32 characters is advised so that it can be read easily in the Acrobat viewer. If this key is absent, the title is the title of the element or SubType.

Key	*Type*	*Semantics*
Open	boolean	(*Optional*) If *true*, the bookmark is open, that is, its children are visible. If *false*, the bookmark is closed. If this key is absent, the bookmark is closed.

4.5.2 StBookmarkRoot

StBookmarkRoot creates the root bookmark for structure bookmarks added by a StPNE with a **Bookmark** key. It contains the **Title** and **Open** keys in Table 16, “Specifying a bookmark tree root”. It may also contain the action keys in Table 22, “Action Types”; if none of these keys are present, the bookmark root has no action associated with it. An operator with StBookmarkRoot *must* appear before any StPNE with a **Bookmark** key, otherwise the default (“Untitled”, closed, no action) is used for the structured bookmark subtree.

Example 17 shows a StBookmarkRoot **pdfmark**.

Example 17 Structure Bookmark Tree Root

```
[ /Title (My structure tree)
/Open true
/StBookmarkRoot pdfmark
```

Table 16 *Specifying a bookmark tree root*

Key	*Type*	*Semantics*
Title	string	(*Optional*) Bookmark title. The encoding and character set used is either PDFDocEncoding (as described in Appendix C in the *Portable Document Format Reference Manual*) or Unicode. If Unicode, the string must begin with <FEFF>. For example, the Unicode string for (ABC) is <FEFF004100420043>. **Title** has a maximum length of 255 PDFDocEncoding characters or 126 Unicode values, although a practical limit of 32 characters is advised so that it can be read easily in the Acrobat viewer. If this key is absent, the title is “Untitled”.

Key	*Type*	*Semantics*
Open	boolean	(*Optional*) If *true*, the bookmark is open, that is, its children are visible. If *false*, its children are unseen. If this key is absent, the bookmark is closed.

4.5.3 StPush

StPush pushes an existing element onto the implicit parent stack. It takes the **E** key described in Table 17, “Referring to an Existing Element or Structure Tree Root”. It is an error for StPush to specify an object that is not an element created by a previous StPNE.

The syntax for pushing an element is:

```
[/E {OBJNAME}
/StPush pdfmark
```

Table 17 *Referring to an Existing Element or Structure Tree Root*

Key	*Type*	*Semantics*
E (Element)	obj name	(*Optional*) Specifies an existing element, given as an object name of the special form {*name*} used to refer to Cos objects. *Note.* If the **E** key is omitted, the Structure Tree Root of the document is specified. The Structure Tree Root is created if it does not already exist.

4.5.4 StPop

StPop pops the implicit parent stack, removing the element formerly at the top of the stack. It has no keys in its dictionary. It is an error for StPop to be encountered when the implicit parent stack is empty.

The syntax for pop element is:

```
[/StPop pdfmark
```

4.5.5 StPopAll

StPopAll completely empties the implicit parent stack. It has no keys in its dictionary.

The syntax for a pop all is:

```
[/StPopAll pdfmark
```

4.6 Specifying Element Content

Elements may have two kinds of document content: marked content (*MC*) and references to PDF objects (*OBJR*s).

To specify marked content, use StBDC and StBMC to indicate the beginning of marked content; EMC delimits the end of marked content. These operators combine the creation of the marked content region in the PDF content stream with the creation of marked content and its placement within the **Kids** list of a specified element or Structure Tree Root.

It is possible to nest marked content as specified by these operators. This nesting has *nothing* to do with the implicit element nesting maintained by the implicit parent stack; nested marked content may belong to elements in different branches of a Structure Tree. See “Nesting Structure Elements”.

4.6.1 StBMC

StBMC marks the beginning of a sequence of marked content objects. The MC created is contained by the element on top of the implicit parent stack. StBMC takes the **P** (**Properties**) and **T** (**Tag**) keys specified in Table 18, “Specifying Tags and Property List Entries for Marked Content”. The marked content is added to its containing element at the position optionally specified by the **At** key in Table 13, “Specifying Position Within a Container”. It is an error if the implicit parent stack is empty when StBMC is encountered.

The syntax for beginning page content is:

```
[/T <tag>
/StBMC pdfmark
```

4.6.2 StBDC

StBDC marks the beginning of a sequence of page content objects with an associated property list, given by a dictionary. StBDC behaves just like StBMC, with the addition of a property list. The MC created is contained by the element on top of the implicit parent stack. StBDC takes the **P** (**Properties**) and **T** (**Tag**) keys specified in Table 18, “Specifying Tags and Property List Entries for Marked Content”. The marked content is added to its containing element at the position optionally specified by the **At** key in Table 13, “Specifying Position Within a Container”. It is an error if the implicit parent stack is empty when StBDC is encountered.

The syntax for beginning page content with a dictionary is:

```
[/T <tag>
/P <properties dictionary>
/StBDC pdfmark
```

Table 18 *Specifying Tags and Property List Entries for Marked Content*

Key	*Type*	*Semantics*
P (Properties)	dictionary	*(Optional)* Key–value pairs that are entered into the properties dictionary of the marked content being created. If this key is omitted, no properties other than those required by the implementation of logical structure in PDF are entered into the properties dictionary. This key is supported only with StBDC.
T (Tag)	name	(*Optional*) The tag to be given to the marked content being created. If this key is omitted, the subtype of the containing element is used.

4.6.3 EMC

The end of a marked sequence of page content operators is signalled by EMC. The syntax for an end of page content is:

```
[ /EMC pdfmark
```

4.6.4 StOBJ

StOBJ adds an existing PDF Object to the content of the element on top of the implicit parent stack. It is an error if the implicit parent stack is empty when StOBJ is encountered. StOBJ identifies the object to be added with the **Obj** key in Table 19, “Referring to an Arbitrary PDF Object”, and it sets the new content’s index within the containing element to the position specified by the **At** key in Table 13, “Specifying Position Within a Container”.

The syntax for adding a PDF Object to an element’s content is:

```
[/Obj {OBJNAME}
/At integer
/StOBJ pdfmark
```

To specify a PDF object as content of an element, use the Cos object reference mechanism. The PostScript file creates a Cos object with a given name (as specified in “Naming Objects with _objdef”) then uses that name with StOBJ to create an Object Reference (*OBJR*). Only dictionary and stream PDF objects may be used for OBJRs.

Note that _objdef associates an object with a name in a variety of ways, as noted in “Naming Objects with _objdef” and “Implicitly Named Objects”. A name defined in this way can be used with any StOBJ, as long as it represents a dictionary or stream PDF object.

Table 19 *Referring to an Arbitrary PDF Object*

Key	*Type*	*Semantics*
Obj	obj name	(*Required*) The object to be added as data to the specified element, given as an object name of the special form {*name*} used to refer to Cos objects. It is an error to specify an unassigned object name.

4.7 Nesting Structure Elements

You nest structure *elements* by using StPNE and StPop—not by nesting StBMC/BDC and EMC. In other words, the nesting is in the *tree structure of elements*—not in the marked content regions, which should be disjointed. Nesting StBMC/BDC and EMC results in the marked content itself being nested—not the corresponding elements in the structure tree.

4.8 Attribute Objects

Attribute Objects (*AO*s) can be inserted either under the **Attributes** key in an element or in the **ClassMap** of a Structure Tree Root. StAttr attaches attribute objects to *elements*. StClassMap enters attribute objects in the ***ClassMap*** of the Structure Tree Root.

4.8.1 StAttr

StAttr creates a new attribute object and adds it to the element on top of the implicit parent stack. The AO's contained object is specified via an object reference {name}. The StAttr's dictionary contains the **Obj** key specified in Table 19, "Referring to an Arbitrary PDF Object". It is an error if the implicit parent stack is empty when StAttr is encountered.

The syntax to create a new attribute object is:

```
[/Obj {OBJNAME}
/StAttr pdfmark
```

The *object name* must refer to a PDF dictionary or stream.

4.9 Storing and Retrieving the Implicit Parent Stack

Using operators that specify a parent implicitly depends on the ability to mimic a tree's structure by nesting the structure within the document. However, it is not always possible to maintain this representation across page boundaries. For example, a paragraph may be represented by regions on more than one page, or it may be interrupted by other page content. To allow originating applications some flexibility in their page output without compromising the convenience of specifying tree structure, the structure suite provides a way of storing and later retrieving the tree's context.

The names under which implicit parent stacks are stored and retrieved are in a *separate* namespace from other names managed by the Distiller. The namespace stack commands NamespacePush and NamespacePop defined in "Namespace Commands" affect this namespace as well as the namespace for Cos object names.

4.9.1 StStore

StStore saves the current state of the implicit parent stack under a name in a namespace used internally by the Distiller for this purpose alone. StStore does *not* change the implicit parent stack. Table 20, "Referring to Saved Implicit Parent Stack State" specifies the **StoreName** key for the name associated with the saved implicit parent stack state. Storing an implicit parent stack state under a previously used name completely replaces the implicit parent stack state already stored under that name.

The syntax for a saving the current implicit parent stack state is:

```
[/StoreName name
/StStore pdfmark
```

Note: Names defined by StStore are in the namespace governed by the stack operators NamespacePush and NamespacePop, defined in "Namespace Commands"*.*

4.9.2 StRetrieve

StRetrieve restores the state of the implicit parent stack from a saved state in the Distiller. The previous state of the implicit parent stack is overwritten by the restored state. The **StoreName** key in Table 20, “Referring to Saved Implicit Parent Stack State” indicates the name of the implicit parent stack state to be retrieved. It is an error to try to retrieve a nonexistent state, that is, to use a name that was not associated with a stack state by a previous StStore**.**

The syntax for a restoring the current state is:

```
[/StoreName name
/StRetrieve pdfmark
```

Table 20 *Referring to Saved Implicit Parent Stack State*

Key	*Type*	*Semantics*
StoreName	name	(*Required*) The name under which an implicit parent stack state is to be stored or retrieved.

4.10 EPS Considerations

Encapsulated PostScript (EPS) is a special form of PostScript used for embedding graphics created in one application in a document created in another application. Applications can create EPS files containing structure elements without knowing anything about the environment into which the EPS file is to be embedded, which complicates the processing of a structure inside embedded EPS. The logical structure design here allows structure within an embedded EPS to be connected to the structure of the surrounding file by way of the implicit parent stack, while insulating the namespace of the containing file from accidents due to naming coincidences in embedded EPS files.

It is strongly recommended that applications embedding EPS files wrap the embedded PostScript between NamespacePush and NamespacePop to insulate the overall PostScript document from the consequences of multiply-defined object names.

CHAPTER 5

Support Commands

This section treats several commands that support other features, such as Cos object naming and logical structure.

5.1 Namespace Commands

The Distiller application maintains a pushdown stack of namespaces for Cos object names and implicit parent stack names. The only Cos object names visible at a given point in processing a PostScript file are those in the namespace on top of the stack. The stack starts with a default outermost namespace on top; this namespace cannot be popped off the stack.

The implicitly-named objects described in "Implicitly Named Objects" are always visible. They are not subject to namespace pushing and popping.

To guarantee that embedded EPS files cannot interfere with structure element processing of the containing file (see "EPS Considerations"), this namespace contains:

- Names for Cos objects, defined by the ***/_objdef*** key.
- Names for stored implicit parent stacks, defined by "StStore".
- Names for images, defined by "NI".

5.1.1 NamespacePush

NamespacePush causes a new, empty namespace to be pushed onto the namespace stack and causes all other namespaces to be hidden. It takes no parameters.

The syntax for pushing a namespace is:

```
[ /NamespacePush pdfmark
```

5.1.2 NamespacePop

NamespacePop pops the topmost namespace from the stack, never to be accessed again. The next lower namespace on the stack becomes the current namespace for Cos object definitions. It is an error if NamespacePop is encountered when the outermost namespace is the only occupant of the stack. NamespacePop takes no parameters.

The syntax for popping a namespace is:

```
[ /NamespacePop pdfmark
```

There is no way to save and restore namespaces.

CHAPTER 6

Naming Graphics and Images

6.1 Naming Graphics with BP and EP

Version 3.0 of the Acrobat Distiller application allows a PostScript language program to specify that a given set of graphical operations should be encapsulated and treated as a single object. **pdfmark** operators using the names BP (Begin Picture) and EP (End Picture) enclose a set of graphic operations; the **pdfmark** operator with the name SP (Show Picture) indicates where to insert the object, which may be inserted in more than one place.

The syntax for the graphics encapsulation commands is:

```
[/BBox [llx lly urx ury] /_objdef {OBJNAME} /BP pdfmark
[/EP pdfmark
[{OBJNAME} /SP pdfmark
```

Table 21 *Begin Picture Attributes*

Key	*Type*	*Semantics*
BBox	array	(*Required*) An array of four numbers [*xll*, *yll*, *xur*, *yur*] specifying the lower-left x, lower-left *y*, upper-right *x*, and upper-right *y* coordinates—in user space—of the rectangle defining the graphic's bounding box.

When the Distiller program sees a **pdfmark** with the name BP, it forks the distillation from the current context and distills subsequent graphics into a PDF Form object. When it encounters a **pdfmark** with the name EP, the Distiller application finishes the Form object, and distillation continues in the original context. The **pdfmark** named SP tells the Distiller program to insert a use of the picture in

the current context—in the same manner as if it were a cached PostScript form painted with the **execform** PostScript language operator. It includes the picture in the current context (page, form, and so forth) using the CTM to position the graphic.

The /_objdef {OBJNAME} key–value pair in the BP **pdfmark** names the picture OBJNAME. Any subsequent **pdfmark** can refer to this object.

Note: Graphics names are in the namespace governed by NamespacePush and NamespacePop, defined in “Namespace Commands”*.*

pdfmark operators with the names BP and EP can be nested.

The picture built using the BP and EP names need not be added to a page using an SP **pdfmark**. It can be inserted into other objects by referring to it by name in another **pdfmark**.

Defining the **pdfmark** operator so that a PostScript interpreter ignores any text between a mark and a **pdfmark** does *not* nullify processing any PostScript operators between the BP and EP **pdfmark**s. To avoid printing anything between the BP and EP **pdfmark**s, use a construct like the one shown in Example 18.

Example 18 Ignoring Text Between BP and EP pdfmarks

```
% Set __pdfMark__ true if pdfmark is already defined
%%BeginPDFMarkPrefix
/pdfmark where {pop
/__pdfMark__ true def
}{
/pdfmark {cleartomark} def
/__pdfMark__ false def
} ifelse
% Use __pdfMark__ to avoid printing text between BP and EP
[/BBox [0 0 100 100] /_objdef {Check} /BP pdfmark
__pdfMark__ {
0 0 1 setrgbcolor /ZapfDingbats 119 selectfont 0 7 moveto (4)
show
} if
[/EP pdfmark
```

The PostScript language sample in Example 19 draws a gray rectangle, then builds a picture enclosed by the BP and EP **pdfmark** names. (The picture is simply an X.) It shows the picture in three places on the page using the SP **pdfmark**, then draws another gray rectangle.

Example 19 Using the Graphics Encapsulation pdfmark Names

```
% draw a gray rectangle
0.5 setgray
0 0 100 100 rectfill

% create a picture
[/BBox [0 0 100 100] /_objdef {MyPicture} /BP pdfmark
0 setgray
0 0 moveto 100 100 lineto stroke
100 0 moveto 0 100 lineto stroke
[/EP pdfmark

% make the picture appear on the page
[{MyPicture} /SP pdfmark

% make the picture appear in another place on the page
% gsave
200 200 translate
[{MyPicture} /SP pdfmark
grestore

% make the picture appear in another place on the page
% at a different size
gsave
100 400 translate
.5 .5 scale
[{MyPicture} /SP pdfmark
grestore

% draw another gray rectangle
0.5 setgray
512 692 100 100 rectfill showpage
```

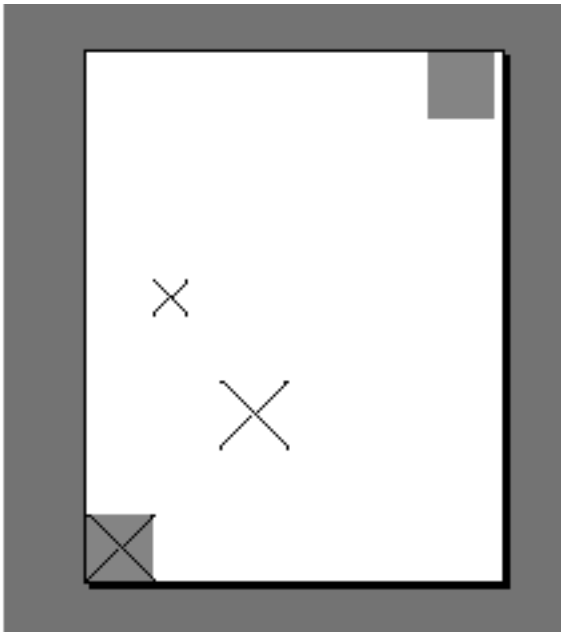
The resulting page stream in the PDF file contains the following:

```
0.5 g
0 0 100 100 re f
q 1 0 0 1 0 0 cm /Fm1 Do Q
q 1 0 0 1 200 200 cm /Fm1 Do Q
q 0.5 0 0 0.5 100 400 cm /Fm1 Do Q
512 692 100 100 re f
```

The graphics between the BP and the EP **pdfmark**s have been saved in a Form object, which has this stream:

```
0 g
0 0 m
100 100 l
100 0 m
0 100 l
S
```

The resulting page looks like this:



6.2 Naming Images with the NI Command

To allow a PostScript image to be referenced the same way a Cos object is referenced, the NI command gives a name to an image. Once named, the name can be used to identify the image. For example, this allows an image to be included in PDF logical structure via "StOBJ", so that it can be included later in element content. The example "Using OBJ and PUT pdfmarks to Create an Alternate Image" shows using NI with an alternate image.

NI defines a name for an image to be found subsequently in the PostScript source file. It takes the standard **_objdef** key to name the image within Distiller. The image does not need to immediately follow the NI. Images are assigned the name given in the most recent invocation of NI not yet paired with an image.

The syntax for defining an image name is:

```
[/_objdef {OBJNAME}
/NI pdfmark
```

Another way of understanding the rule for associating names with images is that Distiller maintains a stack of names pushed by NI and popped by the occurrence of an image. It is *not* an error to encounter an image when this stack is empty: it merely receives no name.

Note: *Image names are in the namespace governed by NamespacePush and NamespacePop, defined in "Namespace Commands".*

CHAPTER 7

Specifying Destinations and Actions

PDF supports three general ways to specify what happens when a user opens a file, clicks on a link, or clicks on a bookmark: *view destinations*, *actions*, and *named destinations*.

View destinations specify another location in the same file.

They are generalizations of the concept of a view destination. Four types of actions are supported by version 2.0 and later of the Acrobat viewers. They are:

- *GoTo* — Same capabilities as view destinations.
- *GoToR* — Specifies a location in another PDF file.
- *Launch* — Launches an arbitrary application or document.
- *Article* — Begins reading a specified article.

Named destinations, described in "Named Destinations", can also be used.

Bookmarks, links, and the Catalog dictionary of documents that have an open action must contain exactly one of the keys listed in Table 22, "Action Types". Additional key–value pairs are required, depending on which of the three specification schemes is used. These are described in the following paragraphs.

Table 22 *Action Types*

Key	*Type*	*Semantics*
Action	name or dictionary	Specifies the action type. Must be either a predefined name or an action dictionary. If a name, must be one of the following: *GoTo* — Jumps to a specified page and zoom factor within the current document. Requires the **Dest** key, or both the **Page** and **View** keys. See "Goto Actions". *GoToR* — Opens another PDF document at a specified page and zoom factor. Requires the **Dest** key, or both the **Page** and **View** keys, plus file-specifier keys. See "GotoR Actions". *Launch* — Launches a document or an application. Requires file-specifier keys. See "Launch Actions". *Article* — Jumps to an article, either in the current document or another PDF document. Requires the **Dest** key. In addition, requires file-specifier keys if the article is in a different PDF file. See "Article Actions". The file-specifier keys **File**, **MacFile**, **DOSFile**, **UnixFile**, **URI**, and **ID** define an external file. At least one of these keys must be defined for actions that require a file-specifier key. If the value of **Action** is a dictionary, it specifies a custom action. Custom action dictionaries, which are supported by version 2.1 and higher of the Distiller application, are generally used to specify actions that are handled by action handler plug-ins for Acrobat. See "Custom Actions". The default value for **Action** is GoTo.
Dest	name, integer, or string	Specifies an Article action's destination, or a named destination for any action. If it is a named destination, the value must be a name object that matches the name of a destination defined with the **DEST** pdfmark. For an Article destination, the value may be an integer that specifies the article's index in the document (the first article in a document has an index of 0), or a string that matches the article's Title. See "Named Destinations".

Table 22 *Action Types*

Key	*Type*	*Semantics*
View	array	Specifies a link or bookmark's destination on a page, and the fit type. "View Destinations" describes the **View** array's elements.

7.1 View Destinations

The value of the **View** key is an array that specifies a page number, a location on the page, and a fit type. The location is either a rectangle, a point, or an *x*– or *y*–coordinate, depending on the fit type. View destinations should be used when compatibility with version 1.0 Acrobat products is important. *GoTo* actions should be used to obtain the same functionality when this compatibility is not important.

Key	*Type*	*Semantics*
Page	integer or name	The destination page. If an integer value is specified, it must be the sequence number of the page within the PDF file. The first page in a file is page 1, not page 0. For links and articles, the name objects **Next** and **Prev** are also valid page destination values. If the destination page of a link is the same page, the **Page** key should be omitted. If the value of the **Page** key is 0, the bookmark or link has a *NULL* destination.

Destinations, which are the value of the **View** key, are specified as an array containing one of the fit type names from Table 23, “Fit Type Names and Parameters”, followed by any required parameters.

Table 23 *Fit Type Names and Parameters*

Name	*Parameters and semantics*
Fit	*No parameters.* Fit the page to the window. This is a shortcut for specifying **FitR** with the rectangle being the crop box for the page.
FitB	*No parameters.* Fit the bounding box of the page contents to the window.
FitH	*top* Fit the width of the page to the window. *top* specifies the distance in default user space from the page origin to the top of the window. This is a shortcut for specifying **FitR** with the rectangle having the width of the page, and both *y*-coordinates equal to *top*.
FitBH	*top* Fit the width of the bounding box of the page contents to the window. *top* specifies the distance in default user space from the page origin to the top of the window.
FitR	*x1 y1 x2 y2* Fit the rectangle specified by the parameters (in default user space) to the window.
FitV	*left* Fit the height of the page to the window. *left* specifies the distance in default user space from the page origin to the left edge of the window. This is a shortcut for specifying **FitR** with the rectangle having the height of the page, and both *x*-coordinates equal to *left*.
FitBV	*left* Fit the height of the bounding box of the page contents to the window. *left* specifies the distance in default user space from the page origin to the left edge of the window.

Table 23 *Fit Type Names and Parameters*

Name	*Parameters and semantics*
XYZ	*left top zoom* *left* and *top* specify the distance in default user space from the origin of the page to the top-left corner of the window. *zoom* specifies the zoom factor, with 1 being 100% magnification. If *left*, *top* or *zoom* is *NULL*, the current value of that parameter is retained. For example, specifying a view destination of /View [/XYZ *NULL NULL NULL*] goes to the specified page and retain the same horizontal and vertical offset and zoom as the current page. A zoom of 0 has the same meaning as a zoom of *NULL*.

The zoom factors for the horizontal and vertical directions are identical; there are not separate zoom factors for the two directions. As a result, more of the page may be shown than specified by the destination. For example, when using **FitR**, portions of the page outside the destination rectangle appear in the window, unless the window happens to have the same aspect ratio (height-to-width ratio) as the destination rectangle.

A common destination is "upper left corner of the specified page, with a zoom factor of 1." This can be obtained using the **XYZ** destination form, with a *left* of – 4 and a *top* equal to the top of the **CropBox** (or the page size if no **CropBox** was specified) plus 4. The offset of 4 is used to slightly move the page corner from the corner of the window, to provide a visual cue that the corner of the page is being shown.

Note: In version 1.0 of Acrobat and Reader for the Microsoft Windows environment, the position of a document within a window is not completely arbitrary. As a result, the offset displayed may not be exactly the offset of 4 that was specified.

Example 20 View Destination

```
[/Rect [ 70 650 210 675 ]
/Page 3
/View [ /XYZ -5 797 1.5]
/LNK pdfmark
```

7.2 Goto Actions

GoTo actions specify the same information as a view destination. They exist primarily to bring the version 1.0-style view destination into the same model as the action types in the version 2.0 and later Acrobat products.

7.3 GotoR Actions

GoToR actions specify a location in another PDF file. They contain the same information as a *GoTo* action, plus a file name.

Key	*Type*	*Semantics*
File	string	(*Required*) The device-independent pathname of the PDF file. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
DOSFile	string	(*Optional*) The MS-DOS pathname (in the PDF pathname format), of the PDF file. Acrobat viewer applications on Windows and DOS computers ignore the **File** key if the **DOSFile** key is present. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
MacFile	string	(*Optional*) The Mac OS filename (in the PDF pathname format) of the PDF file. Acrobat viewer applications on Mac OS computers ignore the **File** key if the **MacFile** key is present. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
UnixFile	string	(*Optional*) The UNIX filename (in the PDF pathname format) of the PDF file. Acrobat viewer applications on UNIX computers ignore the **File** key if the **UnixFile** key is present. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.

Key	*Type*	*Semantics*
URI	string	(*Optional*) The URI of the PDF file. This string is a URI formatted as specified in RFC 1738 and must follow the character encoding requirements of that RFC. See Section 7.4.3 in the *Portable Document Format Reference Manual* for more information. Acrobat viewer applications ignore the **File** key if the **URI** key is present. Named destinations may be appended to URLs, following a "#" character, as in *http://www.adobe.com/test.pdf#nameddest=name*. The Acrobat viewer displays the part of the PDF file specified by the named destination.
ID	array	(*Optional*) An array of two strings specifying the PDF file ID. This key can be used to ensure the correct version of the destination file is found. If present, the destination PDF file's ID is compared with **ID**, and the user is warned if they are different. See Section 6.12 in the *Portable Document Format Reference Manual* for a description of the file ID.

Example 21 GoToR Action

```
[/Action /GoToR /File (test.pdf) /Page 2
/View [/FitR 30 648 209 761]
/Title (Open test.pdf on page 2)
/OUT pdfmark
```

7.4 Launch Actions

Launch actions launch an arbitrary application or document. On some platforms, options or filenames may be passed to the application that is launched. *Launch* actions are specified by an application or document name and, if necessary, the options and filename that are passed to the application that is launched.

Note: The Acrobat 2.0 or later viewer applications running under Windows use the Windows function ShellExecute() to launch an application specified using the Launch action. The keys **WinFile**, **Dir**, **Op**, *and* **Params** *correspond to the parameters of ShellExecute.*

Key	*Type*	*Semantics*
File	string	(*Required*) The device-independent pathname of the application or document to launch. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
DOSFile	string	(*Optional*) The MS-DOS pathname (in the PDF pathname format) of the application or document to launch. Acrobat viewer applications on Windows and DOS computers ignore the **File** key if the **DOSFile** key is present. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
MacFile	string	(*Optional*) The Mac OS filename (in the PDF pathname format) of the application or document to launch. Acrobat viewer applications on Mac OS computers ignore the **File** key if the **MacFile** key is present. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
UnixFile	string	(*Optional*) The UNIX filename (in the PDF pathname format) of the application or document to launch. Acrobat viewer applications on UNIX computers ignore the **File** key if the **UnixFile** key is present. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
URI	string	(*Optional*) The URI of the PDF file. This string is a URI formatted as specified in RFC 1738 and must follow the character encoding requirements of that RFC. See Section 7.4 in the *Portable Document Format Reference Manual* for more information. Acrobat viewer applications ignore the **File** key if the **URI** key is present. Named destinations may be appended to URLs, following a "#" character, as in *http://www.adobe.com/test.pdf#nameddest=name*. The Acrobat viewer displays the part of the PDF file specified by the named destination.

Key	*Type*	*Semantics*
Dir	string	(*Optional*) The default directory of a Windows application. See the description of **WinFile**.
Op	string	(*Optional*) The operation to perform. See the description of **WinFile**. The string must be open or print. The default is open. If **WinFile** specifies an application, not a document, this key is ignored and the application is launched. This key is used only under Windows.
WinFile	string	(*Optional*) The MS-DOS filename of the document or application to launch.
Params	string	(*Optional*) The parameters passed to a Windows application started with the Launch action. See the description of **WinFile**. If **WinFile** keys specifies an application, **Params** must not be present.
Unix	string	(Optional) The parameters passed to a UNIX application started with the Launch action. See the description of **UNIXFile**. *The format of this string is not yet defined.*

Example 22 Launch Action

```
[/Rect [ 70 600 210 625 ]
/Border [ 16 16 1 ]
/Color [0 0 1]
/Action /Launch
/File (test.doc)
/Subtype /Link
/ANN pdfmark
```

7.5 Article Actions

Article actions set the Acrobat viewer to article-reading mode, at the beginning of a specified article.

Key	*Type*	*Semantics*
Dest	integer or string	(*Required*) An Article action's destination. The value may be an integer that specifies the article's index in the document (the first article in a document has an index of 0), or a string that matches the article's Title.

Key	*Type*	*Semantics*
File	string	(*Required if the destination article is in another PDF file*) The device-independent pathname of the PDF file. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
DOSFile	string	(*Optional*) The MS-DOS pathname (in the PDF pathname format), of the destination PDF file. Acrobat viewer applications on Windows and DOS computers ignore the **File** key if the **DOSFile** key is present. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
MacFile	string	(*Optional*) The Mac OS filename (in the PDF pathname format), of the destination PDF file. Acrobat viewer applications on Mac OS computers ignore the **File** key if the **MacFile** key is present. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
UnixFile	string	(*Optional*) The UNIX filename (in the PDF pathname format), of the destination PDF file. Acrobat viewer applications on UNIX computers ignore the **File** key if the **UnixFile** key is present. See Section 7.4 in the *Portable Document Format Reference Manual* for a description of the pathname format.
URI	string	(*Optional*) The URI of the PDF file. This string is a URI formatted as specified in RFC 1738 and must follow the character encoding requirements of that RFC. See Section 7.4.3 in the *Portable Document Format Reference Manual* for more information. Acrobat viewer applications ignore the **File** key if the **URI** key is present. Named destinations may be appended to URLs, following a "#" character, as in *http://www.adobe.com/test.pdf#nameddest=name*. The Acrobat viewer displays the part of the PDF file specified by the named destination.

Key	*Type*	*Semantics*
ID	array	(*Optional*) An array of two strings specifying the file ID of the destination file. This key can be used to ensure the correct version of the destination file is found. If present, the destination PDF file's ID is compared with **ID**, and the user is warned if they are different. See Section 6.12 in the *Portable Document Format Reference Manual* for a description of the file ID.

Example 23 Article Action

```
[/Action /Article /Dest (Now is the Time)
/Title (Now is the Time)
/OUT pdfmark
```

7.6 Custom Actions

Custom actions allow creation of action types other than those provided by the predefined names.

Note: Custom action dictionaries are supported by version 2.1 and higher of the Distiller application.

Custom actions are specified by providing a dictionary containing all the key–value pairs that are to be placed into the action dictionary in the PDF file. This requires a detailed knowledge of an action's representation in PDF. See Section 6.8 in the *Portable Document Format Reference Manual* for a description of actions.

Note: Known action keys are filtered in the same way as they are for other action types, for example, Subtype is filtered to S, Dest is filtered to D, and File is filtered to F. See Section 6.8 in the Portable Document Format Reference Manual for a list of the abbreviated action key names used in PDF files.

One common use of custom actions is to create URI actions, which allow a PDF file to link to locations on the World-Wide Web (WWW). Example 24 shows a note marker containing a custom URI action.

Example 24 Link Containing a Custom URI Action

```
[/Rect [ 50 425 295 445 ]
/Action << /Subtype /URI /URI (http://www.adobe.com) >>
/Border [ 0 0 2 ]
/Color [ .7 0 0 ]
/Subtype /Link
/ANN pdfmark
```

7.7 Named Destinations

Named destinations can be used as the destination for any bookmark or link, or by the optional open action in a document's Catalog dictionary.

Key	*Type*	*Semantics*
Dest	name	The destination name. It must match the name of a destination defined with the *DEST* pdfmark.

Example 25 Named Destination

```
[/Rect [ 70 650 210 675 ]
/Border [ 16 16 1 [ 3 10 ] ]
/Color [ 0 .7 1 ]
/Dest /MyNamedDest
/Subtype /Link
/ANN pdfmark
```

CHAPTER 8

Examples

This section gives examples illustrating many uses of the **pdfmark** operator.

8.1 Define pdfmark So PostScript Interpreters Ignore pdfmarks

```
%!PS-Adobe-3.0
%%BeginProlog
/bd {bind def} bind def
/fsd {findfont exch scalefont def} bd
/sms {setfont moveto show} bd
/ms {moveto show} bd
/pdfmark where
{pop} {userdict /pdfmark /cleartomark load put} ifelse
%%EndProlog
%%BeginSetup
```

8.2 File Open Action

```
[/PageMode /UseOutlines
/Page 2 /View [/XYZ null null null]
/DOCVIEW pdfmark
```

8.3 Info Dictionary

```
[/Title (My Test Document)
/Author (John Doe)
/Subject (pdfmark 3.0)
/Keywords (pdfmark, example, test)
/Creator (Hand Programmed)
/ModificationDate (D:19940912205731)
/ADBETest_MyKey (My private information)
/DOCINFO pdfmark
```

8.4 Crop All Pages

```
[/CropBox [54 403 558 720]
/PAGES pdfmark
/DrawBorder
{58 407 moveto 554 407 lineto 554 716 lineto
58 716 lineto closepath stroke
} bd
/F1 10 /Helvetica fsd
/F2 10 /Helvetica-Oblique fsd
/F3 10 /Helvetica-Bold fsd
/F4 12 /Helvetica-Bold fsd
%%EndSetup
%%Page: 1 1
DrawBorder
(This is Page 1) 75 690 F4 sms
(Below is a closed, default note created using pdfmark:) 75
670 F1 sms
(Below is an open note with a custom color and label:) 75 570
F1 sms
(Below is a closed note) 400 670 F1 sms
(containing private data:) 400 655 F1 sms
(Below is a custom annotation.) 400 570 F1 sms
(It should appear as an unknown) 400 555 F1 sms
(annotation icon:) 400 540 F1 sms
```

8.5 Annotations

8.5.1 Simple Note

```
[ /Rect [ 75 586 456 663 ]
/Contents (This is an example of a note. You can type text
directly into a note or copy text from the clipboard.)
/ANN pdfmark
```

8.5.2 Fancy Note

```
[/Rect [ 75 425 350 563 ]
/Open true
/Title (John Doe)
/Contents (This is an example of a note. \nHere is some text
after a forced line break.
This is another way to do line breaks.)
/Color [1 0 0]
/Border [0 0 1]
/ANN
pdfmark
```

8.5.3 Private Data in Note

```
[/Contents (My unimaginative contents)
/Rect [ 400 550 500 650 ]
/Open false
/Title (My Boring Title)
% The following is private data. Keys within the private
% dictionary do not need to use the
% organization’s prefix because the dictionary encapsulates
% them.
/ADBETest_MyInfo <<
/Routing [ (Me) (You) ]
/Test_Privileges << /Me /All /You /ReadOnly >>
>>
/ADBETest_PrivFlags 42
/ANN pdfmark
```

8.5.4 Movie or Sound Annotation

```
[
/Type /Annot
/Subtype /Movie
/Rect [ 216 503 361 612 ]
/T (Title)
/F 1
% The specified file may be a movie or sound file
/Movie << /F (/Disk/moviefile) /Aspect [ 160 120 ] >>
/A << /ShowControls true >>
/Border [0 0 3]
/C [0 0 1]
/ANN pdfmark
```

8.5.5 Simple Link (Old style, compatible with all Distiller application versions)

```
[/Rect [ 70 650 210 675 ]
/Page 3
/View [ /XYZ -5 797 1.5 ]
/LNK pdfmark Fancy Link
[/Rect [ 70 550 210 575 ]
/Border [ 0 0 2 [ 3 ] ]
/Color [0 1 0]
/Page /Next
/View [ /XYZ -5 797 1.5]
/Subtype /Link
/ANN pdfmark showpage
%%Page: 3 3
```

8.5.6 Link that Launches Another File

```
[/Rect [ 70 600 210 625 ]
/Border [ 16 16 1 ]
/Color [0 0 1]
/Action /Launch
/File (test.doc)
/Subtype /Link
/ANN pdfmark
```

8.5.7 Custom Link Action (URI Link for the Acrobat WebLink Plug-in)

```
[/Rect [ 50 425 295 445 ]
/Action << /Subtype /URI /URI (http://www.adobe.com) >>
/Border [ 0 0 2 ]
/Color [ .7 0 0 ]
/Subtype /Link
/ANN pdfmark
% URI link with a named destination
[/Rect [ 50 425 295 445 ]
/Action << /Subtype /URI /URI (http://
www.adobe.com#YourDestination) >>
/Border [ 0 0 2 ]
/Color [ .7 0 0 ]
/Subtype /Link
/ANN pdfmark
```

8.5.8 Custom Link Action (Named Action)

```
% Link with a named action—executes a menu item
[ /Rect [ 50 425 295 445 ]
/Action << /Subtype /Named /N /GeneralInfo >>
/Border [ 0 0 2 ]
/Color [ .7 0 0 ]
/Subtype /Link
/ANN pdfmark
```

8.5.9 Custom Annotation Type

This appears with an unknown annotation icon in the Acrobat viewers, because they do not know how to interpret this annotation type.

```
[/Rect [ 400 435 500 535 ]
/Subtype /ADBETest_DummyType
/ADBETest_F8Array [ 0 1 1 2 3 5 8 13 ]
/ANN pdfmark showpage
%%Page: 2 2
DrawBorder
(This is Page 2) 75 690 F4 sms
(Click here to go to page 3.) 75 660 F2 sms
(Click here to go to page 3.) 75 560 F2 sms
```

8.5.10 Putting a File's Contents Into a Text Annotation.

See "Putting a File's Contents Into a Text Annotation".

8.6 Crop This Page

```
[/CropBox [0 0 612 792] /PAGE pdfmark
(This is Page 3) 75 690 F4 sms
(Click here to go to page 1.) 75 660 F2 sms
(Click here to open test.doc.) 75 610 F2 sms
```

8.7 Create Text for the Article "Now is the Time"

```
(Now is the Time \(Article\)) 230 690 F4 sms
(Now is the time for all good men to come to the aid of their
country.) 230 670 F1 sms
(Now is the time for all good men to come to the aid of their
country.) 230 655 ms
(Now is the time for all good men to come to the aid of their
country.) 230 640 ms
(Now is the time for all good men to come to the aid of their
country.) 230 625 ms
(Now is the time for all good men to come to the aid of their
country.) 230 610 ms
(Now is the time for all good men to come to the aid of their
country.) 230 595 ms
(Now is the time for all good men to come to the aid of their
country.) 230 580 ms
(Now is the time for all good men to come to the aid of their
country.) 230 565 ms
(Now is the time for all good men to come to the aid of their
country.) 230 550 ms
(Now is the time for all good men to come to the aid of their
country.) 230 535 ms
(Now is the time for all good men to come to the aid of their
country.) 230 520 ms
(Now is the time for all good men to come to the aid of their
country.) 230 505 ms
```

8.7.1 Continue Text for the Article “Now is the Time”

```
(Now is the Time continued...) 230 690 F2 sms
(Now is the time for all good men to come to the aid of their
country.) 230 670 F1 sms
(Now is the time for all good men to come to the aid of their
country.) 230 655 ms
(Now is the time for all good men to come to the aid of their
country.) 230 640 ms
(Now is the time for all good men to come to the aid of their
country.) 230 625 ms
(Now is the time for all good men to come to the aid of their
country.) 230 610 ms
(Now is the time for all good men to come to the aid of their
country.) 230 595 ms
(Now is the time for all good men to come to the aid of their
country.) 230 580 ms
(Now is the time for all good men to come to the aid of their
country.) 230 565 ms
(Now is the time for all good men to come to the aid of their
country.) 230 550 ms
(Now is the time for all good men to come to the aid of their
country.) 230 535 ms
(Now is the time for all good men to come to the aid of their
country.) 230 520 ms
(Now is the time for all good men to come to the aid of their
country.) 230 505 ms
(Click here to go to Adobe’s Home Page on the Web) 55 430 ms
```

8.8 Named Destination

```
[ /Dest /MyNamedDest
/Page 1
/View [ /FitH 5 ]
/DEST pdfmark
Link to a Named Destination
[/Rect [ 70 650 210 675 ]
/Border [ 16 16 1 [ 3 10 ] ]
/Color [ 0 .7 1 ]
/Dest /MyNamedDest
/Subtype /Link
/ANN pdfmark
```

8.9 Article Containing Two Beads

```
 [/Title (Now is the Time)
/Author (John Doe)
/Subject (Coming to the aid of your country)
/Keywords (Time, Country, Aid)
/Rect [ 225 500 535 705 ]
/Page 2
/ARTICLE pdfmark
[/Title (Now is the Time)
/Rect [ 225 500 535 705 ]
/Page 3
/ARTICLE pdfmark
```

8.10 Pass-through PostScript Language Code

```
[/DataSource (0 0 moveto 100 700 lineto stroke)
/PS pdfmark showpage
```

8.11 Bookmarks

```
[/Count 2 /Page 1 /View [/XYZ 44 730 1.0] /Title (Open
Actions) /OUT pdfmark
[/Action /Launch /File (test.doc) /Title (Open test.doc) /OUT
pdfmark
[/Action /GoToR /File (test.pdf) /Page 2 /View [/FitR 30 648
209 761]
/Title (Open test.pdf on page 2) /OUT pdfmark

[/Count 2 /Page 2 /View [/XYZ 44 730 1.0] /Title (Fixed Zoom)
/OUT pdfmark
[/Page 2 /View [/XYZ 44 730 2.0] /Title (200% Magnification) /
OUT pdfmark
[/Count 1 /Page 2 /View [/XYZ 44 730 4.0] /Title (400%
Magnification) /OUT pdfmark
[/Page 2 /View [/XYZ 44 730 5.23] /Title (523% Magnification)
/OUT pdfmark

[/Count 3 /Page 1 /View [/XYZ 44 730 1.0] /Title (Table of
Contents #1) /OUT pdfmark
[/Page 1 /View [/XYZ 44 730 1.0] /Title (Page 1 - 100%) /OUT
pdfmark
[/Page 2 /View [/XYZ 44 730 2.25] /Title (Page 2 - 225%) /OUT
pdfmark
```

```
[/Page 3 /View [/Fit] /Title (Page 3 - Fit Page) /OUT pdfmark

[/Count -3 /Page 1 /View [/XYZ 44 730 1.0] /Title (Table of
Contents #2) /OUT pdfmark
[/Page 1 /View [/XYZ null null 0] /Title (Page 1 - Inherit) /
OUT pdfmark
[/Page 2 /View [/XYZ null null 0] /Title (Page 2 - Inherit) /
OUT pdfmark
[/Page 3 /View [/XYZ null null 0] /Title (Page 3 - Inherit) /
OUT pdfmark

[/Count 1 /Page 0 /Title (Articles) /OUT pdfmark
[/Action /Article /Dest (Now is the Time) /Title (Now is the
Time) /OUT pdfmark
```

8.11.1 Bookmark with a URI as an Action

```
[/Count 0 /Title (Adobe’s Home page)
/Action << /Subtype /URI /URI (http://www.adobe.com)>> /OUT
pdfmark
%%EOF
```

8.12 Putting a File’s Contents Into a Text Annotation

```
% Put a file's contents into a text annotation.

/F (file's platform dependent path name) (r) file def
[/_objdef {mystream} /type /stream /OBJ pdfmark
[{mystream} F /PUT pdfmark
[
/MyPrivateAnnotmyStreamData {mystream}
/SubType /Text
/Rect [500 500 550 550]
/Contents (Here is a text annotation)
/ANN pdfmark
```

8.13 Using OBJ pdfmark to Add an Open Action to a PDF File

```
% Go to the 5th page of a document upon opening it.
% First and third lines can be reused.
% Second line specifies the GoTo action, which can be
% customized easily.
[ /_objdef {MyAction} /type /dict /OBJ pdfmark
[ {MyAction} << /S /GoTo /D [ {Page5} /FitH 770 ] >> /PUT
pdfmark
[ {Catalog} << /OpenAction {MyAction} >> /PUT pdfmark
```

8.14 Using OBJ pdfmark to Create a Base URI

```
% Create a dictionary object
  [ /_objdef {myURIdict} /type /dict /OBJ pdfmark
% Add a "Base" key-value pair to the dictionary we just
% created
  [{myURIdict} << /Base (http://www.adobe.com) >> /PUT
pdfmark
% Add our dictionary to the PDF file’s Catalog dictionary
  [{Catalog} << /URI {myURIdict} >> /PUT pdfmark
```

8.15 Using OBJ and PUT pdfmarks to Create an Alternate Image

This example shows how to create alternate images. In this case, we create an image that has one Alternate. The Alternate is stored on as a JPEG file on a web server, and is the default image used when printing.

```
%Give the next image a name, so we can add an Alternates array
% to it later
[/_objdef {myImage} /NI pdfmark
%Create the base image (just a 2x1 pixel grayscale image for
% this sample)
<<
   /Width 2
   /Height 1
   /ImageMatrix [1 0 0 1 0 0]
   /ImageType 1
   /Decode [0 1]
   /BitsPerComponent 8
```

```
   /DataSource (1Z)
>> image
%Create a stream for the Alternate Image
 [/_objdef {myPrintingImageStream} /type /stream /OBJ pdfmark
%Add the necessary key-value pairs to the stream dictionary
% to make it a valid image XObject.
%This particular image XObject uses the external streams
% capability of PDF to point to an image
%stored on an IIP server, retrieving it as a JPEG file.
%Since all stream data is stored on a web server, we don't
% explicitly add data to the stream.
%As a result, the stream ends up with a length of zero, which
% is OK for external streams.
[{myPrintingImageStream} << /Type /XObject /Subtype /Image /
Width 150 /Height 150
/FFilter /DCTDecode /ColorSpace /DeviceRGB /BitsPerComponent
8
/F << /FS /URL /F (http://www.mycompany.com/myfile.jpg) >>
 >> /PUT pdfmark
%Add an Alternates array to the base image
[{myImage} << /Alternates [ <</Image {myPrintingImageStream}
/DefaultForPrinting true >> ] >> /PUT pdfmark
```

There are two possibilities for alternate images:

- Alternate image data is outside of PDF file
- Alternate image data inside PDF file

The above sample shows only how to construct the first type. Note also that if the Alternate uses a different color space than the base image, it is possible that the PDF file may not contain the appropriate **ProcSet** references in the Resources dictionary to print the page to PostScript. For example, if the base image is grayscale and the Alternate is **DeviceRGB**, it is likely that the page's Resources contains only the **ImageB** procset (for grayscale images) and not the **ImageC** procset (for color images).

8.16 Using OBJ, PUT, BP, and EP pdfmarks to Create an Acrobat Form

This example illustrates how to define a form.

8.16.1 Define __pdfMark__ so Anything Between BP and EP pdfmarks Is Not Printed

```
% Set __pdfMark__ true if pdfmark is already defined
%%BeginPDFMarkPrefix
/pdfmark where {
pop
/__pdfMark__ true def
}{
/pdfmark {cleartomark} def
/__pdfMark__ false def
} ifelse
%%EndPDFMarkPrefix
```

8.16.2 Acrobat Form Definitions

Definition of common objects that are used by the widgets such as Fonts, Encoding arrays and Form XObjects for Button faces.

```
%%<<AcroForm Begin
[/BBox [0 0 100 100] /_objdef {Check} /BP pdfmark
__pdfMark__ {
0 0 1 setrgbcolor /ZapfDingbats 119 selectfont 0 7 moveto (4)
show
} if
[/EP pdfmark

[/BBox [0 0 100 100] /_objdef {Cross} /BP pdfmark
__pdfMark__ {
0 0 1 setrgbcolor /ZapfDingbats 119 selectfont 9.7 7.3 moveto
(8) show
} if
[/EP pdfmark

% Up/Down button appearances
[/BBox [0 0 200 100] /_objdef {Up} /BP pdfmark
__pdfMark__ {
0.3 setgray 0 0 200 100 rectfill 1 setgray 2 2 moveto 2 98
```

```
lineto 198 98 lineto
196 96 lineto 4 96 lineto 4 4 lineto fill 0.34 setgray 198 98
moveto 198 2 lineto
2 2 lineto 4 4 lineto 196 4 lineto 196 96 lineto fill
0 setgray 8 22.5 moveto 1 0 0 setrgbcolor /Helvetica 72
selectfont ( Up) show
} if
[/EP pdfmark

[/BBox [0 0 200 100] /_objdef {Down} /BP pdfmark
__pdfMark__ {
0.7 setgray 0 0 200 100 rectfill 1 setgray 2 2 moveto 2 98
lineto 198 98 lineto
196 96 lineto 4 96 lineto 4 4 lineto fill 0.34 setgray 198 98
moveto 198 2 lineto
2 2 lineto 4 4 lineto 196 4 lineto 196 96 lineto fill
0 setgray 8 22.5 moveto 0 0 1 setrgbcolor /Helvetica 72
selectfont (Down) show
} if
[/EP pdfmark
% Submit button appearances
[/BBox [0 0 250 100] /_objdef {Submit} /BP pdfmark
__pdfMark__ {
0.6 setgray 0 0 250 100 rectfill 1 setgray 2 2 moveto 2 98
lineto 248 98 lineto
246 96 lineto 4 96 lineto 4 4 lineto fill 0.34 setgray 248 98
moveto 248 2 lineto
2 2 lineto 4 4 lineto 246 4 lineto 246 96 lineto fill
/Helvetica 76 selectfont 0 setgray 8 22.5 moveto (Submit)
show
} if
[/EP pdfmark

[/BBox [0 0 250 100] /_objdef {SubmitP} /BP pdfmark
__pdfMark__ {
0.6 setgray 0 0 250 100 rectfill 0.34 setgray 2 2 moveto 2 98
lineto 248 98 lineto
246 96 lineto 4 96 lineto 4 4 lineto fill 1 setgray 248 98
moveto 248 2 lineto
2 2 lineto 4 4 lineto 246 4 lineto 246 96 lineto fill
/Helvetica 76 selectfont 0 setgray 10 20.5 moveto (Submit)
show
} if
[/EP pdfmark
```

8.16.3 Font Encoding Resource

```
[ /_objdef {pdfDocEncoding}
  /type /dict
/OBJ pdfmark

[ {pdfDocEncoding}
  <<
  /Type /Encoding
  /Differences [
24 /breve /caron /circumflex /dotaccent /hungarumlaut /ogonek
/ring
/tilde 39 /quotesingle 96 /grave 128 /bullet /dagger /
daggerdbl
/ellipsis /emdash /endash /florin /fraction /guilsinglleft /
guilsinglright
/minus /perthousand /quotedblbase /quotedblleft /
quotedblright /quoteleft
/quoteright /quotesinglbase /trademark /fi /fl /Lslash /OE /
Scaron
/Ydieresis /Zcaron /dotlessi /lslash /oe /scaron /zcaron 164
/currency
166 /brokenbar 168 /dieresis /copyright /ordfeminine 172 /
logicalnot
/.notdef /registered /macron /degree /plusminus /twosuperior
/threesuperior
/acute /mu 183 /periodcentered /cedilla /onesuperior /
ordmasculine
188 /onequarter /onehalf /threequarters 192 /Agrave /Aacute /
Acircumflex
/Atilde /Adieresis /Aring /AE /Ccedilla /Egrave /Eacute /
Ecircumflex
/Edieresis /Igrave /Iacute /Icircumflex /Idieresis /Eth /
Ntilde
/Ograve /Oacute /Ocircumflex /Otilde /Odieresis /multiply /
Oslash
/Ugrave /Uacute /Ucircumflex /Udieresis /Yacute /Thorn /
germandbls
/agrave /aacute /acircumflex /atilde /adieresis /aring /ae /
ccedilla
/egrave /eacute /ecircumflex /edieresis /igrave /iacute /
icircumflex
/idieresis /eth /ntilde /ograve /oacute /ocircumflex /otilde
/odieresis
/divide /oslash /ugrave /uacute /ucircumflex /udieresis /
```

```
yacute
/thorn /ydieresis
  ]
  >>
/PUT pdfmark
```

8.16.4 Font Dictionaries

```
[ /_objdef {Helv}
  /type /dict
/OBJ pdfmark

[ {Helv}
  <<
/Type /Font
/Subtype /Type1
/Name /Helv
/BaseFont /Helvetica
/Encoding {pdfDocEncoding}
  >>
/PUT pdfmark

[ /_objdef {TiIt}
  /type /dict
/OBJ pdfmark
[ {TiIt}
  <<
/Type /Font
/Subtype /Type1
/Name /TiIt
/BaseFont /Times-Italic
/Encoding {pdfDocEncoding}
  >>
/PUT pdfmark

[ /_objdef {TiRo}
  /type /dict
/OBJ pdfmark

[ {TiRo}
  <<
/Type /Font
/Subtype /Type1
/Name /TiRo
/BaseFont /Times-Roman
```

```
/Encoding {pdfDocEncoding}
  >>
/PUT pdfmark

[ /_objdef {ZaDb}
  /type /dict
/OBJ pdfmark

[ {ZaDb}
  <<
/Type /Font
/Subtype /Type1
/Name /ZaDb
/BaseFont /ZapfDingbats
  >>
/PUT pdfmark
```

8.17 Forms Examples

This section gives examples illustrating various uses of the Forms **pdfmark** suite.

The following examples are derived from the *sampler* example; see *sampler.pdf* and *sampler.ps* in the *Docs:OtherDoc* folder on the Acrobat 4.05 SDK CD.

8.17.1 PDFMarkPrefix

Discriminate between running in a printer, where the **pdfmark** operator is not defined, and on Distiller.

```
%%BeginPDFMarkPrefix
systemdict /currentdistillerparams known not {
  /str 256 string def
  {currentfile str readline not {exit} if
  (%%EndPDFMarkPrefix) eq {exit} if } loop
} if
```

8.17.2 Define the AcroForm Dictionary at the Catalog of the Document

Includes these required entries:

- Fields (the array from where all widgets in the form can be found)
- Default Appearance (DA)
- Default Resources (DR)
- NeedApperances boolean, set to *true* to indicate that when the document is opened, traverse all widgets to generate their display and add them to the Fields array

Also includes definition of common objects that are used by the widgets such as fonts, encoding arrays, and Form *XObjects* for button faces.

```
%%<<AcroForm Begin

[ /_objdef {pdfDocEncoding}
  /type /dict
/OBJ pdfmark

[ {pdfDocEncoding}
  <<
  /Type /Encoding
  /Differences [
     24 /breve /caron /circumflex /dotaccent /hungarumlaut /
     ogonek /ring /tilde 39 /quotesingle 96 /grave 128 /
     bullet /dagger /daggerdbl /ellipsis /emdash /endash /
     florin /fraction /guilsinglleft /guilsinglright /minus /
     perthousand /quotedblbase /quotedblleft /quotedblright /
     quoteleft /quoteright /quotesinglbase /trademark /fi /fl
     /Lslash /OE /Scaron /Ydieresis /Zcaron /dotlessi /lslash
     /oe /scaron /zcaron 164 /currency 166 /brokenbar 168 /
     dieresis /copyright /ordfeminine 172 /logicalnot /
     .notdef /registered /macron /degree /plusminus /
     twosuperior /threesuperior /acute /mu 183 /
     periodcentered /cedilla /onesuperior /ordmasculine 188 /
     onequarter /onehalf /threequarters 192 /Agrave /Aacute /
     Acircumflex /Atilde /Adieresis /Aring /AE /Ccedilla /
     Egrave /Eacute /Ecircumflex /Edieresis /Igrave /Iacute /
     Icircumflex /Idieresis /Eth /Ntilde /Ograve /Oacute /
     Ocircumflex /Otilde /Odieresis /multiply /Oslash /Ugrave
```

```
      /Uacute /Ucircumflex /Udieresis /Yacute /Thorn /
      germandbls /agrave /aacute /acircumflex /atilde /
      adieresis /aring /ae /ccedilla /egrave /eacute /
      ecircumflex /edieresis /igrave /iacute /icircumflex /
      idieresis /eth /ntilde /ograve /oacute /ocircumflex /
      otilde /odieresis /divide /oslash /ugrave /uacute /
      ucircumflex /udieresis /yacute /thorn /ydieresis
    ]
    >>
/PUT pdfmark

[ /_objdef {ZaDb}
  /type /dict
/OBJ pdfmark

[ {ZaDb}
  <<
/Type /Font
/Subtype /Type1
/Name /ZaDb
/BaseFont /ZapfDingbats
  >>
/PUT pdfmark

[ /_objdef {Helv}
  /type /dict
/OBJ pdfmark

[ {Helv}
  <<
/Type /Font
/Subtype /Type1
/Name /Helv
/BaseFont /Helvetica
/Encoding {pdfDocEncoding}
  >>
/PUT pdfmark

[ /_objdef {aform}
  /type /dict
/OBJ pdfmark

[ /_objdef {afields}
  /type /array
/OBJ pdfmark
```

```
[ {aform}
  <<
    /Fields {afields}
    /DR << /Font << /ZaDb {ZaDb} /Helv {Helv} >> >>
    /DA (/Helv 0 Tf 0 g)
    /NeedAppearances true
  >>
/PUT pdfmark

[ {Catalog}
  <<
    /AcroForm {aform}
  >>
/PUT pdfmark
%%>> %End AcroForm
```

8.17.3 Define the Widget Annotations, Which Are Also Field Dictionaries for this Form

The collection of all individual widget annotations.

It is possible to have multiple instances of these sections, maybe defining a single widget on each instance.

```
%%<<Widgets Begin
[
    /Subtype /Widget
    /Rect [216 647 361 684]
    /F 4
    /T (SL Text)
    /FT /Tx
    /DA (/Helv 14 Tf 0 0 1 rg)
    /V (5)
/AA <</K<<
    /S /JavaScript
    /JS (AFNumber_Keystroke\(2, 0, 0, 0, "$", true\);)
    >>
    /F<<
    /S /JavaScript
    /JS (AFNumber_Format\(2, 0, 0, 0, "$", true\);)
    >>
>>
/ANN pdfmark
```

```
[
    /Subtype /Widget
    /Rect [216 503 361 612]
    /F 4
    /T (Ping Result)
    /FT /Tx
    /DA (/Helv 0 Tf 0 0 1 rg)
    /Ff 4096
/ANN pdfmark

[
    /Subtype /Widget
    /Rect [216 432 252 468]
    /F 4
    /T (Check Box)
    /FT /Btn
    /DA (/ZaDb 0 Tf 0 g)
    /AS /Off
    /MK << /CA (4)>>
    /AP <<
        /N << /Oui null >>
    >>
/ANN pdfmark

[
    /Subtype /Widget
    /Rect [216 360 252 396]
    /F 4
    /T (Radio)
    /FT /Btn
    /DA (/ZaDb 0 Tf 0 g)
    /Ff 49152
    /AS /Off
    /MK << /CA (8)>>
    /AP <<
    /N << /V1 null >>
    >>
/ANN pdfmark

[
    /Subtype /Widget
    /Rect [ 261 360 297 396 ]
    /F 4
    /T (Radio)
    /FT /Btn
```

```
    /DA (/ZaDb 0 Tf 0 g)
    /Ff 49152
    /AS /Off
    /MK << /CA (8)>>
    /AP <<
        /N << /V2 null >>
>>
/ANN pdfmark

[
    /Subtype /Widget
    /Rect [ 306 360 342 396 ]
    /F 4
    /T (Radio)
    /FT /Btn
    /DA (/ZaDb 0 Tf 0 g)
    /Ff 49152
    /AS /Off
    /MK << /CA (8)>>
    /AP <<
        /N << /V3 null >>
>>
/ANN pdfmark

[
    /Subtype /Widget
    /Rect [ 351 360 387 396 ]
    /F 4
    /T (Radio)
    /FT /Btn
    /DA (/ZaDb 0 Tf 0 g)
    /Ff 49152
    /AS /Off
    /MK << /CA (8)>>
    /AP <<
        /N << /V4 null >>
>>
/ANN pdfmark

[
    /Subtype /Widget
    /Rect [216 287 361 324]
    /F 4
    /T (Pop Down)
    /FT /Ch
```

```
    /Ff 131072
    /Opt [ [(1)(First)] [(2)(Second)] [(3)(Third)]
    [(4)(Fourth)] [(5)(Fifth)]]
    /DV (5)
    /V (5)
    /DA (/TiIt 18 Tf 0 0 1 rg)
/ANN pdfmark

[
    /Subtype /Widget
    /Rect [216 215 361 252]
    /F 4
    /T (Combo)
    /FT /Ch
    /Ff 917504
    /Opt [ (Black)(Blue)(Green)(Pink)(Red)(White)]
    /DA (/TiRo 18 Tf 0 g )
    /V (Black)
    /DV (Black)
/ANN pdfmark

[
    /Subtype /Widget
    /Rect [216 107 253 180]
    /F 4
    /T (ListBox)
    /FT /Ch
    /DA (/Helv 10 Tf 1 0 0 rg)
    /Opt [(1)(2)(3)(4)(5)]
    /DV (3)
    /V (3)
/ANN pdfmark

%
% Example of how the /MK dictionary is used.
% Notice that the text will be shown upside-down (180 degree
rotation).
%
[
    /Subtype /Widget
    /Rect [ 430 110 570 150 ]
    /F 4
    /T (Clear)
    /FT /Btn
    /H /P
```

```
    /DA (/HeBo 18 Tf 0 0 1 rg)
    /Ff 65536
    /MK <<
        /BC [ 1 0 0 ]
        /BG [ 0.75 0.45 0.75 ]
        /CA (Clear)
        /AC (Done!)
        /R 180
    >>
    /BS <<
        /W 3
    /S /I
    >>
    /A <<
        /S /ResetForm
    >>
/ANN pdfmark
%%>> %End Widgets

%%EndPDFMarkPrefix
```

8.18 Structure Examples

This section gives examples illustrating various uses of the structure **pdfmark** suite.

8.18.1 Interrupted Structure

This example shows a paragraph that is graphically interrupted by a table. The originating application has chosen to write out the PostScript in graphical order, but logically the paragraph is one element and the table is another. To further complicate matters, the document contains a special element that is a list of tables.

```
% Start a ListOfTables element directly under the Structure
% Tree Root.  Give it an object name for later reference.
[/Subtype /ListOfTables
/_objdef {LOT}
/StPNE
pdfmark

% Pop it off the stack so that the next element becomes a
% child of the Structure Tree Root.
```

```
[/StPop pdfmark

% Start the page with the section on it.

% Start the section, also making it the default parent
% element.
[/Subtype /Section
 /StPNE
pdfmark

% Start the paragraph.
[/Subtype /P
/StPNE
pdfmark

% Here comes the portion of the paragraph before the table
[/StBDC pdfmark

% [code to write the first portion of the paragraph goes here]

[/EMC pdfmark

% Now we're interrupted by a table that doesn't belong to the
% paragraph.  Save the context as a conservative move because
% we don't want to worry about what the table code does to the
% implicit parent stack.
[/StoreName /S1
 /StStore
pdfmark

% The table is an element, and it contains cells as child
% elements.
[/E {LOT}
/StPush
pdfmark
[/Subtype /Table
 /StPNE
pdfmark

% ... code to draw the table and establish its logical
% substructure here ...

% Pop the table and the List of Tables off the implicit parent
% stack.
[/StPop pdfmark
```

```
[/StPop pdfmark

% Resume the paragraph.  It turns out that the table code was
% tidy, but it’s probably a good thing that we didn’t count on
% it.  Get the implicit parent stack back into a known state.
[/StoreName /S1
 /StRetrieve
pdfmark

[/StBDC pdfmark

% ... code to write the second portion of the paragraph ...

[/EMC pdfmark

% Pop the Paragraph and Section elements and the Structure
% Tree Root off the stack.
[/StPop pdfmark
[/StPop pdfmark
[/StPop pdfmark
```

8.18.2 Independence of Logical and Physical Structure

This shows that the logical structure and the physical nesting of marked content can have different tree structures. In this example there are again two Structure Trees. One is the usual hierarchical structure of the document; the other is a list of funny words that occur within the document. The words occur as nested marked content within the marked content forming the contents of a paragraph, but the words become the content of elements in a separate branch of the structure tree from the Paragraph elements.

```
% Set up a List element to hold the Funny Word List.
[/Subtype /List
/Title (Funny Words)
/_objdef {FWL}
/StPNE
pdfmark

[/StPop pdfmark

[/Subtype /Section
```

```
 /StPNE
pdfmark

[/Subtype /P
/StPNE
pdfmark

% Begin PostScript code for the paragraph
[/StBDC pdfmark
(John was thrilled to find some ) show
% Here’s an occurrence of a funny word coming up.
% Start an element for the funny word list...
[/E {FWL}
/StPush
pdfmark
[/Subtype /Word
/StPNE
pdfmark
% Fill that element with the funny word from the
% page content.  This content is still in the
% marked content within the paragraph element.
[/StBDC pdfmark
(puccoon) show
[/EMC pdfmark
% Pop the Word element off the implicit parent stack.
[/StPop pdfmark
% Resume paragraph content that’s not in the funny word
% (, not knowing that it could also be called )
% ... another funny word ...
[/E {FWL}
/StPush
pdfmark
[/Subtype /Word
/StPNE
pdfmark
[/StBDC pdfmark
(gromwell) show
[/EMC pdfmark
[/StPop pdfmark
(.) show
% Close off the marked content for the paragraph...
```

```
[/EMC pdfmark
% ...and tidy up the stack
[/StPop pdfmark
[/StPop pdfmark
[/StPop pdfmark
```

8.18.3 Page Break Within Logical Structure

This shows how to handle logical structure spanning more than one page. The example shows a logical paragraph spanning a page break.

```
%%Page: 1 1

% Begin a Paragraph element
[/Subtype /P
 /StPNE
pdfmark

[/StBDC pdfmark
% ... write the portion of the paragraph that’s on Page 1 ...
[/EMC pdfmark
showpage

%%Page: 2 2

% The Paragraph element is still on the top of the stack, so
% we can just add some more content to it implicitly.
[/StBDC pdfmark
% ... write the portion of the paragraph that’s on Page 2 ...
[/EMC pdfmark
```

8.18.4 Logical Structure Out-of-order in Physical Structure

This example shows how to build a logical structure whose elements appear in a different physical order in the document from their logical order. The example is based on a magazine in which an opinion piece starting on the last inside page is continued on an earlier page in the printing order.

```
%%Page 5 5
[/Subtype /Section
 /ID (ID string)
 /StPNE
pdfmark

% This Paragraph element is actually a later paragraph within
% the Section element than the Paragraph element that appears
% on the next page.
[/Subtype /P
 % No /At key, so defaults to being inserted as last child of
% its parent.
 /StPNE
pdfmark

[/StBDC pdfmark
% ... draw the paragraph...
[/EMC pdfmark

% ... the rest of the page ...

showpage

% Pop the Paragraph element off the stack
[/StPop pdfmark
%%Page 6 6

[/Subtype /P
 % Insert as first child of parent.
/At 0
 /StPNE
pdfmark

[/StBDC pdfmark
% ... draw the paragraph...
```

```
[/EMC pdfmark

% Pop the Paragraph and Section elements off the stack
[/StPop pdfmark
[/StPop pdfmark
```

APPENDIX A

Changes Since Earlier Versions

Changes since 5 November 1997 version

- Added structure operator information and related additions in new sections:
  “Logical Structure”
  “Support Commands”
  “Naming Images with the NI Command”
- Added Alternate image sample code:
  “Using OBJ and PUT pdfmarks to Create an Alternate Image”

Changes since 22 July 1997 version

- Added creating a base URI example.

Changes since 4 June 1997 version

- Added link named action example.

Changes since 14 February 1997 version

- Added headers for pdfmark examples.

Changes since 07 July 1995 version

- Added changes for Acrobat Distiller 3.0:
  - Specifying Cos objects.
  - Encapsulating graphics.
  - URI as a file specification.

- Added example showing form definition for Acrobat forms.
- Added example defining movie and sound annotations.
- Changed headings in large example at end to delineate examples better.
- General updates, such as removing reference to obsolete documentation.

Changes since 07 April 1995 version

- Updated contents for version 2.1 of the Distiller application:
  - Updated version number in Section 1.
  - Updated list of version numbers with ~4k limit on the number of named destinations (Section 3.4).
  - Added “dictionary” to the enumeration of value types allowed in arbitrary key–value pairs (Section 1).
  - Added **UnixFile** key (Sections 4.3, 4.4, and 4.5).
  - Added **Unix** key (Section 4.4).
  - Added **Subtype** key (Tables 1, 2, and 3).
  - Rewrote annotation section to show how to create links and custom annotations using the annotation marker (Section 3.1).
  - Updated almost all link examples (Example 2) to use the new preferred method of link creation via annotation markers. Left one example in the old style, using a link marker.
  - Added custom annotation example (Example 3).
  - Added “private data” Note example (Example 1).

  - Added URI link example (Example 2).
  - Added description of custom actions (Table 13 and Section 4.6).
  - Added examples of custom URI actions (Examples 4, and 20).

- Added a number of index entries.
- Corrected improper capitalization of **DataSource** in pass-through PostScript language code (Table 7).
- Added cross-reference to Tech Note #5151 for determining the Distiller application's version number programmatically (Section 2).
- Corrected Info dictionary example to use a prefix on the private keys (Example 9).
- Corrected the specification of a note's and link's color (Table 1 and Table 2). The text previously stated that the color could be specified in the *DeviceGray*, *DeviceRGB*, or *DeviceCMYK* color spaces. In fact, it must be in *DeviceRGB*.

Changes since 05 December 1994 version

- Completely reorganized the document to make it easier to understand and use.
- Removed several spurious key–value pairs (for example, removed **Title** and **ModDate** from links).
- Modified example to show a named destination.

Changes since January 26, 1994 version

- Complete update for version 2.0 of the Acrobat products. New features include articles, actions, named destinations, Info and Catalog dictionary entries, and pass-through PostScript language code.
- Replaced examples with a larger, single example.

Changes since June 19, 1993 version

- Corrected the description of the **Count** key for bookmarks to state that the value of this key specifies the number of immediate subordinate bookmarks, not the total number of subordinate bookmarks at all levels.
- Example code for Figure 6 — Updated the value of the **Count** key to be consistent with the correct definition of the **Count** key.