pdfmark Reference Manual

Technical Note #5150

Version : Acrobat 5.0

ADOBE SYSTEMS INCORPORATED

Corporate Headquarters

345 Park Avenue

San Jose, CA 95110-2704

(408) 536-6000

February 1, 2001

Contents

Preface

The Acrobat® Distiller® application converts PostScript® language files into Portable Document Format (PDF) files. However, PDF has several features, such as annotations, bookmarks, articles, and forms, that have no counterpart in PostScript.

The **pdfmark** operator is used to represent PDF features in PostScript language files. When PostScript files containing **pdfmark**s are processed by Acrobat Distiller, the corresponding PDF is generated.

The use of **pdfmark** allows an independent software vendor (ISV) already supporting the PostScript language to support PDF features without having to write PDF files directly.

Purpose

This document describes the syntax and use of the **pdfmark** operator, and contains examples of many of the features that can be implemented using **pdfmark**.

Audience

This document is intended primarily for users who want to include **pdfmark** commands in PostScript code, in order to generate PDF features. These commands can be incorporated manually into PostScript files. In addition, some applications that generate PostScript have special methods of allowing users to add **pdfmark**s.

Contents

Chapter 1, "Introduction," introduces the **pdfmark** operator.

Chapter 2, "Basic pdfmark Types," describes the basic forms of the **pdfmark** operator.

Chapter 3, "Specifying Actions and Destinations," goes into detail about how to specify actions and destinations.

Chapter 4, "Cos Objects," describes how to create and use Cos objects, allowing finer control of PDF features.

Chapter 5, "Logical Structure," describes how to implement logical structure in PDF.

Chapter 6, "Examples," gives several examples of using **pdfmark**.

Other Useful Documentation

Making full use of **pdfmark** requires knowledge of the PDF file format. The latest version of the *PDF Reference (Version 1.3, 2nd Edition)* describes the PDF file format in detail. If you do not have this document, it can be obtained at

http://partners.adobe.com/asn/developer/technotes.html#acrobat-pdf

In addition, the Acrobat Software Development Kit (SDK) consists of documentation, headers and sample code that enable software developers to access the functionality of the Acrobat product suite. The SDK is available at:

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

1 Introduction

1.1 pdfmark Operator

The **pdfmark** operator is used in PostScript code to represent PDF features

It takes the following as its arguments:

- A mark object (that is, a **[** character)
- A variable number of arguments. Arguments are often in the form of *key–value pairs*, although they may take other forms as well.
- A name object.

It does not return any values. Hence, the general syntax of the **pdfmark** operator is:

```
[
...arguments...
/KIND pdfmark
```

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

Each instance of the **pdfmark** operator in a PostScript language file is referred to as a *marker*.

NOTE: Many, but not all, of the keys that appear in **pdfmark**s correspond to PDF dictionary keys. Some keys may be abbreviated when used in PDF files. For example, **Subtype** is filtered to **S**, **Dest** is filtered to **D**, and **File** is filtered to **F**. See the *PDF Reference (Version 1.3, 2nd Edition)* for full details about the PDF keys.

1.2 Compatibility With PostScript Devices

If you plan to use PostScript files containing **pdfmark** operators for purposes other than distilling to PDF, you need to ensure that other PostScript devices, such as printers, that do not implement the **pdfmark** operator, can use your files.

To accomplish this, you can place the following PostScript language code in the prolog of the PostScript file. If the PostScript interpreter processing the file does not implement the **pdfmark** operator, this code in effect makes each marker a no-op.

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

See Section 6.1 for an example.

Special care is also needed when using markers that encapsulate graphics. See Section 4.6.1 for information, as well as the example in Section 6.12.2.

1.3 Page Numbering

All pages in a PDF document are numbered sequentially; the first page in a document is page 1. When using the **pdfmark** operator, all page numbers must be specified using this sequence number, not the page number as it appears on the printed page.

1.4 Private Data

Some kinds of 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**.

2 Basic pdfmark Types

This chapter describes the basic 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. See the latest version of the *PDF Reference (Version 1.3, 2nd Edition)* for a description of the PDF file format.

The following **pdfmark** types are described in this chapter:

- Annotations (ANN)
- Bookmarks (OUT)
- Articles (ARTICLE)
- Page Cropping (PAGES, PAGE)
- Info Dictionary (DOCINFO)
- Document Open Options (DOCVIEW)
- Page Label and Plate Color (PAGELABEL)
- Marked Content (MP, DP, BMC, BDC, EMC)

Other **pdfmark** types are defined in the other chapters of this document.

2.1 Annotations (ANN)

PDF supports several types of annotations. The properties of each annotation is specified in an *annotation dictionary* containing various key–value pairs. Section 7.4 of the *PDF Reference (Version 1.3, 2nd Edition)* describes all the types of annotations, and their required and optional dictionary entries.

The **pdfmark** operator, in conjunction with the name **ANN**, can be used to specify an annotation in a PostScript file. The general syntax is:

```
[/Rect [xll yll xur yur]
/Subtype name
…Optional key-value pairs…
/ANN pdfmar
```

Table 2.1 describes the two required keys for annotations.

TABLE 2.1 *Required annotation keys*

Key	Type	Semantics
Rect	array	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 or link button.
Subtype	name	The annotation' s PDF subtype. If omitted, the value defaults to **Text**, indicating a note annotation. See Table 2.2 for the possible subtypes that can be used.

As of PDF 1.3, the following annotation types are supported:

TABLE 2.2 *PDF annotation types*

Value of Subtype key	Description
Text	Text annotation (note)
Link	Link annotation
FreeText	Free text annotation
Line	Line annotation
Square	Square annotation
Circle	Circle annotation
Highlight	Highlight annotation
Underline	Underline annotation
StrikeOut	Strikeout annotation

TABLE 2.2 PDF annotation types

Value of Subtype key	Description
Stamp	Rubber stamp annotation
Ink	Ink annotation
Popup	Pop-up annotation
FileAttachment	File attachment annotation
Sound	Sound annotation
Movie	Movie annotation
Widget	Widget annotation
TrapNet	Trap network annotation

Each type has its own set of key-value pairs that may be specified, as described in the *PDF Reference.* Future versions of PDF may introduce new types.

In addition to these types, *custom annotations* are supported, allowing the creation of arbitrary annotation types. Custom annotations may contain, in addition to the **Rect** and **Subtype** keys, arbitrary key–value pairs

See Section 6.3.7 for an example of a custom annotation.

Table 2.3 lists optional keys that are common to all annotations. Specific annotation types have additional keys that they use. See section 7.4 of the *PDF Reference (Version 1.3, 2nd Edition)* for complete information.

TABLE 2.3 Optional annotation keys

Key	Type	Semantics
SrcPg	integer	The sequence number of the page on which the annotation appears. (The first page in a document is always page 1.) If this key is present, the 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 annotation is to appear.
Color (PDF key = **C**)	array	A color value used for the background of the annotation's icon when closed; the title bar of the annotation's pop-up window; and the border of a link annotation. The value is an array containing three numbers (red, green, and blue), each of which must be between 0 and 1, inclusive, specifying a color in the DeviceRGB color space. (See Section 4.5.3 in the *PDF Reference (Version 1.3, 2nd Edition)* for a description of this color space.) If omitted, a default color is used.

TABLE 2.3 *Optional annotation keys*

Key	Type	Semantics
Title (PDF key = **T**)	string	The text label to be displayed in the title bar of the annotation's pop-up window when open and active. The encoding and character set used is either PDFDocEncoding (as described in Appendix D in the *PDF Reference (Version 1.3, 2nd Edition)* or Unicode. If Unicode, the string must begin with <FEFF>. For example, the string "ABC" is represented as (ABC) in PDFDocEncoding and <FEFF004100420043> in Unicode. **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 (PDF key = **M**)	string	The date and time the note was last modified. It should be of the form: (*D:YYYYMMDDHHmmSSOHH'mm'*) "*D:*" is an optional but strongly recommended 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.
Border	array	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].
F	integer	A set of flags specifying various characteristics. See Section 7.4.2 of the *PDF Reference.*

TABLE 2.3 Optional annotation keys

Key	Type	Semantics
AP	dictionary	An *appearance dictionary* specifying how the annotation is presented visually. See Section 7.4.4 of the *PDF Reference* for details.
AS	name	The annotation's *appearance state*. See Section 7.4.4 of the *PDF Reference* for details.
Action (PDF key = **A**)	name or dictionary	An action to be performed when the annotation is activated. See Section 3.1, "Actions," for details. NOTE: For links, this key is not permitted if the **Dest** key is present.

Section 2.1.1, "Notes," and Section 2.1.2, "Links," describe the syntax for two of the original and most commonly used annotation types in more detail.

2.1.1 Notes

Notes are known as *text annotations* in PDF. The syntax for creating a note is:

```
[/Contents string
/Rect [xll yll xur yur]
/SrcPg pagenum
/Open boolean
/Color array
/Title string
/ModDate datestring
/Name name
/Subtype /Text
/ANN pdfmark
```

In addition to the keys described in Table 2.1 and Table 2.3, the keys specific to text annotations are listed in Table 2.4. In addition to these keys, notes may also specify arbitrary key–value pairs.

TABLE 2.4 Keys specific to Text annotations

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 D in the *PDF Reference (Version 1.3, 2nd Edition)* or Unicode. If Unicode, the string must begin with <FEFF>.

TABLE 2.4 Keys specific to Text annotations

Key	Type	Semantics
Open	boolean	(*Optional*) If *true*, the note is open (that is, the text is visible). If *false* (the default if omitted), the note is closed (that is, displayed as an icon).
Name	name	*(Optional)* The name of an icon to be used in displaying the note. The values are: **Note** (default), **Comment**, **Help**, **Insert**, **Key**, **NewParagraph**, **Paragraph**.

See Section 6.2 for examples of notes.

2.1.2 Links

A link annnotation represents either a hypertext link to a destination in the document, or an action to be performed.

The usual syntax for creating a link is:

```
[/Rect [xll yll xur yur]
/Border [bx by c [d]]
/SrcPg pagenum
/Color array
/Subtype /Link
…Action-or-destination-specifying key–value pairs…
/ANN pdfmark
```

In addition to the keys described in Table 2.1 and Table 2.3, a link may also contain keys specifying destinations or actions, described in Chapter 3, “Specifying Actions and Destinations.”

See Section 6.3 for examples of links.

2.1.3 Other Annotations

Table 2.2 lists the other types of annotations that are available. Section 6.3.8 shows an example of a movie annotation, for instance.

One useful type of annotation is the *widget annotation*. Widgets are used by PDF *interactive forms* to represent the appearance of fields and to manage user interactions. See Section 7.6 of the *PDF Reference (Version 1.3, 2nd Edition)* for detailed information on using interactive forms.

See Section 6.13.3 for examples of using widget annotations to create interactive forms.

2.2 Bookmarks (OUT)

Bookmarks are known as *outline items* in PDF. They are specified by using the **pdfmark** operator in conjunction with the name **OUT**.

The syntax for a bookmark marker is:

```
[/Title string
/Count int
/Color array
/F integer
…Action-specifying key-value pairs…
/OUT pdfmark
```

TABLE 2.5 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 D in the *PDF Reference (Version 1.3, 2nd Edition)* 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, omitted otherwise*) This key's absolute value is the number of bookmarks immediately subordinate—that is, excluding subordinates of subordinates. If the value is positive, the bookmark is open, revealing its subordinates; if negative, the bookmark is closed, hiding its subordinates. NOTE: This differs from the PDF **Count** key, which represents the total number of open descendants at all lower levels of the outline hierarchy.
Color	array	*(Optional, effective beginning with Acrobat 5.0)* The bookmark's color. The value is an array containing three numbers (red, green, and blue), each of which must be between 0 and 1, inclusive, specifying a color in the DeviceRGB color space. (See Section 4.5.3 in the *PDF Reference (Version 1.3, 2nd Edition)* for a description of this color space.)

TABLE 2.5 Bookmark Attributes

Key	Type	Semantics
F	integer	*(Optional, effective beginning with Acrobat 5.0)* The style of the bookmark. Four styles are implemented: ● 0: Plain (the default) ● 1: Italic ● 2: Bold ● 3: Bold and Italic

In addition to the keys listed in Table 2.5, a bookmark must contain key–value pairs that specify an action. See Chapter 3, "Specifying Actions and Destinations," for more information.

Bookmark markers may begin anywhere in the PostScript language file. However, they must appear in sequential order.

See Section 6.4 for examples of bookmark markers.

2.3 Articles (ARTICLE)

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 [xll yll xur yur]
/Page pagenum
/ARTICLE pdfmark
```

TABLE 2.6 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 D in the *PDF Reference (Version 1.3, 2nd Edition)* 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.

TABLE 2.6 Article Bead 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 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 2.6 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.

See Section 6.5.3 for examples of articles.

2.4 Page Cropping (PAGES, PAGE)

Page cropping is used to specify the dimensions of a page or pages in a PDF file that will be displayed or printed (without altering the actual data in the file). Cropping is specified by using the **pdfmark** operator in conjunction with the names **PAGES** (for the entire document) or **PAGE** (for an individual page).

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

```
[/CropBox [xll yll xur yur]
/PAGES pdfmark
```

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

```
[/CropBox [xll yll xur yur]
/PAGE pdfmark
```

The **CropBox** key is an array representing 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 .04 × .04 inch (3 × 3 units) and the maximum allowed page size is 200 × 200 inches (14,400 × 14,400 units) in the default user space coordinate system.

The **PAGES pdfmark** 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 **PAGE pdfmark** 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, it affects only the first page of a document if it is placed before any marks are made on the first page.

See Section 6.6 for examples of cropping.

2.5 Info Dictionary (DOCINFO)

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
```

All the allowable keys are strings, and they are all optional. In addition to the keys listed in Table 2.7, arbitrary keys (which must also take string values) can be specified.

TABLE 2.7 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.

TABLE 2.7 Info Dictionary Attributes

Key	Type	Semantics
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.

See Section 6.7 for examples.

2.6 Document Open Options (DOCVIEW)

A PDF file can specify the following things regarding what happens when it is opened:

- The way the document is displayed. The options are: the document only, the document plus thumbnail images, the document plus bookmarks, or just the document in full screen mode.
- A location other than the first page that is to be displayed.
- An optional action that occurs.

The above information is contained in key–value pairs in the document's Catalog dictionary. It can be set 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
```

The **PageMode** key specifies how the document is to be displayed when opened. It can take the following values:

- *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.

If **PageMode** is not specified, the value defaults to *UseNone*.

The **DOCVIEW pdfmark** can also specify a destination (a page to which the document should be opened) or an action, by using additional key–value pairs. See Chapter 3, “Specifying Actions and Destinations,” for details about the key–value pairs that can be used.

DOCVIEW pdfmarks may occur anywhere in the PostScript language file.

See Section 6.8 for an example.

2.7 Page Label and Plate Color (PAGELABEL)

The **PAGELABEL pdfmark** allows specification of the *page label* for a given page. Page labels can strings like “iv” or “3-24”, and do not necessarily correspond to the actual page numbers, which run consecutively. See Section 7.3.2 of the *PDF Reference* for details.

Its syntax is:

```
[/Label string
/PlateColor string
/PAGELABEL pdfmark
```

Both the **Label** and **PlateColor** keys are optional. **Label** takes a string representing the page label for the page on which the **pdfmark** appears.

PlateColor takes an optional string representing a device colorant. It is used in high-end printing situations where the pages are pre-separated prior to generating PDF. This means that there are multiple page objects in the PDF file (each representing a different colorant) corresponding to a single physical page.The color for each separation must be specified in a *separation dictionary*; see Section 8.6.2 of the *PDF Reference* for details.

Consecutive pages that specify **PlateColor**, with the same value for **Label**, are placed in the same separation group. The last instance of a **Label** or **PlateColor** on a page overrides any earlier settings of the same key on the same page.

Section 6.9 gives examples of the use of this marker.

2.8 Marked Content (MP, DP, BMC, BDC, EMC)

PDF 1.2 introduced *marked content operators*, which identify (mark) a portion of a PDF document as elements that can be processed by an application or plug-in.

Several **pdfmark** names can be used to specify marked content:

- MP and DP designate a single marked-content point in the document's content stream.
- BMC, BDC, and EMC bracket a marked-content sequence of objects in the content stream. Note that these are complete graphics objects, not just a sequence of bytes.

NOTE: Marked content can also be used in conjunction with PDF's logical structure facilities. See Chapter 5, "Logical Structure," for information about **pdfmark**s that implement logical structure.

2.8.1 Marked-Content Points

MP creates a marked-content point in the PDF file. **DP** creates a marked-content point, with an associated property list. Their syntax is:

```
[tag
/MP pdfmark

[tag
property-list
/DP pdfmark
```

tag is an optional name object indicating the role or significance of the point.*property-list* is a dictionary containing key-value pairs that are meaningful to the program creating the marked content.

2.8.2 Marked-content sequences

BMC and BDC begin a marked-content sequence, and EMC ends a sequence. Their syntax is:

```
[tag
/BMC pdfmark

[tag
property-list
/BDC pdfmark

[/EMC pdfmark
```

tag is an optional name for the sequence. *property-list* is a dictionary containing key-value pairs that are meaningful to the program creating the marked content.

2.9 Compatibility Notes

The following pdfmark names are compatible with older versions of Distiller, but are not recommended to be used with the current version.

2.9.1 Old-style Links (LNK)

Prior to version 2.1 of Distiller, links could only be specified by using the **LNK pdfmark**. This form is still supported for backward compatibility, but should no longer be used. Instead, use **ANN**, with **Link** as the value of the **Subtype** key. The syntax is:

```
[/Rect array
/Page integer
/View array
/LNK pdfmark
```

2.9.2 Pass-through PostScript Commands (PS)

NOTE: As of Acrobat version 5.0, this feature is no longer supported in any PDF files of version 1.4 and later.

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 2.8 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.

3 Specifying Actions and Destinations

When a user opens a file, clicks on a link, or clicks on a bookmark, there are several types of information that need to be specified in order to indicate what should happen. Different **pdfmark** types require one or more of the following:

- *Actions* specify what type of action should be taken. They are indicated by the **Action** key in a **pdfmark**. See Section 3.1, "Actions."
- *Destinations* specify a particular location in a file, and a zoom factor. See Section 3.2, "Destinations."

  View destinations require a **Page** key and a **View** key. Typically they are used along with an **Action** key; if there is no **Action** key, the action is the equivalent of **GoTo**, meaning to jump to the destination in the current file.

  Alternatively, *named destinations* can be used, specified by the **Dest** key. They specify a destination in the same file or another file, by name.
- *File specifiers* are indicate the target of an action when it is not the current file. See Table 3.2, "File specifier keys."

3.1 Actions

PDF defines several types of actions that can be specified for bookmarks and annotations. The types defined as of PDF 1.3 are:

TABLE 3.1 Action types

Action type	Description
GoTo	Go to a destination in the current document
GoToR	Go to a destination in another document
Launch	Launch an application, usually to open a file
Thread	Begin reading an article thread
URI	Resolve a uniform resource identifier
Sound	Play a sound
Movie	Play a movie
Hide	Set an annotation's Hidden flag
Named	Execute an action predefined by the viewer application

TABLE 3.1 Action types

Action type	Description
SubmitForm	Send data to a URL
ResetForm	Set fields to their default values
ImportData	Import field values from a files
JavaScript	Execute a JavaScript script.

When using **pdfmark**, the type of action for the annotation or bookmark is specified by the **Action** key. It takes one of the following values:

- A predefined name corresponding to one of the first four items in Table 3.1: **GoTo**, **GoToR**, **Launch**, or **Article** (which corresponds to the **Thread** type in PDF).
- A dictionary specifying one of the other types, or a custom action. This dictionary must contain the key–value pairs that are to be placed into the *action dictionary* in the PDF file. See Section 7.5 in the *PDF Reference (Version 1.3, 2nd Edition)* for a detailed description of all the actions and their dictionaries. The syntax for this type of **Action** key is:

  ```
  /Action << / Subtype actiontype
  ...other action dictionary key-value pairs... >>
  ```

  Section 6.3.5 shows a note marker containing a **URI** action.

If the **Action** key is not present, the action is assumed to be the equivalent of **GoTo**; that is, jumping to a location in the current document. Actions other than **GoTo** may require a file-specifier key to specify an external document (see Table 3.2, “File specifier keys”).

3.1.1 GoTo Actions

GoTo actions jump to a specified page and zoom factor within the current document. They require the **Dest** key, or both the **Page** and **View** keys. See Section 3.2, “Destinations,” for more information on these keys.

3.1.2 GotoR Actions

GoToR actions specify a location in another PDF file. They require the **Dest** key, or both the **Page** and **View** keys, plus one or more file-specifier keys (see Table 3.2).

See Section 6.4 for an example of a **GoToR** action.

The following table specifies keys that can be used with the **GoToR**, **Launch**, and **Article** actions to specify the target file.

TABLE 3.2 File specifier keys

Key	Type	Semantics
File	string	(*Required*) The device-independent pathname of the PDF file.
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.
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.
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.
URI	string	(*Optional*) The uniform resource identifier (URI) of a file on the internet. It can be an HTML file as well as a PDF file. 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#name*. The Acrobat viewer displays the part of the PDF file specified by the named destination. NOTE: This key is used with the **Launch** action. URIs can also be specified with an action dictionary where the value of the **Subtype** key is **/URI** (see Sections 6.3.5 and for examples.)
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.

Section 3.10 of the *PDF Reference (Version 1.3, 2nd Edition)* provides more information about the above specifiers.

3.1.3 Launch Actions

Launch actions launch an arbitrary application or document, specified by the **File** key. If an application is specified, some platforms allow passing options or filenames to the application that is launched. See Section 6.3.4 for an example of a launch action.

See Table 3.2 for the file specifier keys that can be used by Launch actions. In addition, the following optional keys can be used:

TABLE 3.3 Optional keys for Launch actions

Key	Type	Semantics
Dir	string	(*Optional*) The default directory of a Windows application.
Op	string	(*Optional*) The operation to perform; used only under Windows. The string must be open (the default) or print. If **WinFile** specifies an application, not a document, this key is ignored and the application is launched.
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. If the **WinFile** key specifies an application, **Params** must not be present.

NOTE: Acrobat 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*.

3.1.4 Article Actions

Article actions set the Acrobat viewer to article-reading mode, at the beginning of a specified article in the current document or another PDF document.

They require the **Dest** key, which takes one of the following values:

- An integer that specifies the article's index in the document (the first article in a document has an index of 0)
- A string that matches the article's Title.

In addition, article actions require one or more file-specifier keys if the article is in a different PDF file (see Table 3.2).

See Section 6.5.1 for an example of an article action.

3.2 Destinations

There are two ways of specifying a location within a document that is the target of an action:

- *View destinations* explicitly specify a page, a location on the page, and a fit type. See Section 3.2.1
- *Named destinations* specify the target as a name which has been defined. See Section 3.2.2.

3.2.1 View Destinations

View destinations require the following two keys:

TABLE 3.4 Keys for view destinations

Key	Type	Semantics
Page	integer or name	The destination page. An integer value represents the sequence number of the page within the PDF file. NOTE: The first page in a file is page 1, not page 0. The name objects **Next** and **Prev** are valid destination page values for links and articles. If the destination of a link is on 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.
View	array	Specifies a link or bookmark's destination on a page, and its fit type. The first array entry is one of the fit type names shown in Table 3.5. The remaining entries, if any, specify the location as either a rectangle, a point, or an *x*– or *y*–coordinate, depending on the fit type.

All distances and coordinates specified in Table 3.5 are in default user space.

TABLE 3.5 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 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 from the page origin to the top of the window.

TABLE 3.5 Fit type names and parameters*(Continued)*

Name	Parameters and semantics
FitR	*x1 y1 x2 y2* Fit the rectangle specified by the parameters to the window.
FitV	*left* Fit the height of the page to the window. *left* specifies the distance in 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 from the page origin to the left edge of the window.
XYZ	*left top zoom* *left* and *top* specify the distance 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.

Sections 6.3, 6.4, 6.8 and 6.10 show examples of destinations.

3.2.2 Defining Named Destinations

Locations in PDF files can be specified by name instead of by page number and view. These names can then be used as destinations of bookmarks or links. Using named destinations is particularly advantageous for cross-document links, because if the document containing a link’s destination is revised, the link will still work, regardless of whether its location in the file has changed.

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 3.6 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 Section 3.2 for information on specifying a view destination.

In addition to the keys listed in Table 3.6 named destinations may also specify arbitrary key–value pairs.

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.

See Section 6.10 for examples of named destinations.

3.2.3 Referencing Named Destinations

Named destinations that have been defined with the **DEST pdfmark** can be used as the target of a bookmark or link, or by the optional open action in a document's Catalog dictionary. They are specified using the **Dest** key.

See Section 6.10 for examples of named destinations.

Note: When used with the *Article* action, **Dest** has a different syntax. See Section 3.1.4, "Article Actions."

4 Cos Objects

Cos objects are the building blocks of PDF files. Acrobat Distiller has the ability to define composite Cos objects (arrays, dictionaries, and streams) directly. Using **pdfmark**, a PostScript language program can create Cos objects, name them, and create references to them in other objects.

The **OBJ pdfmark** allows Cos objects to be created and named directly. Several of the other **pdfmark** operators discussed in this document also have the ability to create and name objects. Information can be put into these objects using the **PUT**, **PUTINTERVAL**, **APPEND**, or **CLOSE pdfmark**s.

A PDF file contains built-in objects (such as the Catalog and page dictionaries) which can also have information put into them by the same methods (see Section 4.2). This allows access to many features of PDF, even those not mentioned explicitly in this document. To do this effectively requires knowledge of the specific dictionary keys, as described in the *PDF Reference (Version 1.3, 2nd Edition)*.

See Section 6.11 for examples of how to create and reference Cos objects with **pdfmark**.

4.1 Naming Objects with _objdef

4.1.1 OBJ

The syntax for specifying a Cos object is:

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

The **_objdef** key is required. `{objname}` is a string representing the name given to the Cos object (the curly braces are required).

NOTE: The Cos object name used here is not a standard name object; that is, it does not start with a slash “/”. The name exists *only* during the distillation process and has no relationship to any identifier created in the output PDF file.

The **type** key is also required. Its value is a name object specifying the type of Cos object created, and it must be either **/array** to create an array, **/dict** to create a dictionary, and **/stream** to create a stream.

Cos objects created with an **OBJ pdfmark** can be used to define other Cos objects. An {*objname*} 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, Distiller places an indirect reference to that object in the PDF file. Sections 6.11.4 through 6.11.7 show examples of this.

Names defined by **_objdef** are in the namespace governed by the stack operators NamespacePush and NamespacePop, defined in Section 4.5, “Namespace Commands.”

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.

4.1.2 Other Commands

In addition to **OBJ,** other **pdfmark** commands can also use the **_objdef** {*objname*} key–value pair to create objects and give them names, as follows:

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

These commands are:

- **ANN** — annotation
- **BP** — encapsulated graphic
- **DEST** — named destination
- **NI** — encapsulated image
- **StPNE** — structure element

6.11.3 and subsequent sections show examples.

NOTE: All the Cos objects created with these **pdfmark** commands are dictionaries.

4.2 Implicitly Named Objects

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

- **{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

Using the **pdfmark**s described in the next section, it is possible to modify the information in these dictionaries. Sections 6.11.5 and 6.11.6 show examples.

4.3 Adding Information to Cos Objects

4.3.1 PUT

The **PUT pdfmark** allows information to be added to Cos objects. The syntax for **PUT** has several forms, depending on the object added to:

```
[{dictname} <<key1 value1...>> /PUT pdfmark
[{streamname} string /PUT pdfmark
[{streamname} file /PUT pdfmark
[{arrayname} index value /PUT pdfmark
```

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

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 Distiller).

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.

4.3.2 PUTINTERVAL

The **PUTINTERVAL pdfmark** adds multiple entries to an array object, starting at an index. Its syntax is:

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

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

4.3.3 APPEND

The **APPEND pdfmark** adds an entry at the end of an array object. Its syntax is:

```
[arrayname} value /APPEND pdfmark
```

4.4 Closing a Stream Object

The **CLOSE pdfmark** 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 Distiller completes writing a PDF file, any open streams are closed and written automatically.

4.5 Namespace Commands

Acrobat Distiller maintains a pushdown stack of namespaces. The stack starts with a default outermost namespace on top, which cannot be popped off the stack.

A namespace contains:

- Names for Cos objects, defined by the **_objdef** key.
- Names for stored implicit parent stacks, defined in Section 5.7.1.
- Names for images, defined in Section 5.2.

The commands **NamespacePush** and **NameSpacePop** can be used to push namespaces onto and pop them off the stack. At any point in processing a PostScript file, the only names visible are those in the namespace on top of the stack.

NOTE: The implicitly-named objects described in Section 4.2 are always visible. They are not subject to namespace pushing and popping.

The correct use of namespaces can ensure that embedded EPS files do not interfere with structure element processing of the containing file (see Section 5.8, "EPS Considerations.").

4.5.1 NamespacePush

NamespacePush causes a new, empty namespace to be pushed onto the namespace stack and causes all other namespaces to be hidden. The syntax for pushing a namespace is:

```
[ /NamespacePush pdfmark
```

4.5.2 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. It is an error if **NamespacePop** is encountered when the outermost namespace is the only occupant of the stack.

The syntax for popping a namespace is:

```
[ /NamespacePop pdfmark
```

There is no way to save and restore namespaces.

4.6 Naming Graphics and Images

4.6.1 Naming Graphics - BP, EP, SP

Acrobat Distiller allows a PostScript language program to specify that a given set of graphical operations should be encapsulated and treated as a single object. The **pdfmark**s **BP** (Begin Picture) and **EP** (End Picture) enclose a set of graphic operations. The **SP** (Show Picture) **pdfmark** indicates where to insert an object (which may be inserted in more than one place).

The syntax for the graphics encapsulation commands is:

```
[/BBox [xll yll xur yur] /_objdef {objname} /BP pdfmark
[/EP pdfmark
[{objname} /SP pdfmark
```

The **_objdef** {*objname*} key–value pair in the **BP pdfmark** names the picture *objname* (see Section 4.1.1). Any subsequent **pdfmark** can refer to this object.

NOTE: Graphics names are in the namespace governed by **NamespacePush** and **NamespacePop**, defined in Section 4.5.

The **BBox** key 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—in user space—of the rectangle defining the graphic's bounding box.

When Distiller sees a **BP pdfmark**, it forks the distillation from the current context and distills subsequent graphics into a PDF Form object. When it encounters an **EP pdfmark**, Distiller finishes the Form object, and distillation continues in the original context. **BP** and **EP pdfmark** operators can be nested.

The **SP pdfmark** tells Distiller to insert a use of a named 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 current transformation matrix (CTM) to position the graphic.

In addition to using **SP** to insert pictures, other **pdfmark**s that allow specifying named objects can add pictures built using **BP** and **EP** to a page.

See Section 6.12 for an example.

Even if you define the **pdfmark** operator so that a PostScript interpreter ignores any text between a mark and a **pdfmark**, any PostScript operators between the **BP** and **EP pdfmark**s will still be processed. To avoid printing anything between the **BP** and **EP pdfmark**s, use a conditional construct like the one shown in Section 6.12.2.

4.6.2 Naming Images - NI

The **NI pdfmark** gives a name to a PostScript image. Subsequently, the name can be used to refer to the image in the same way that a Cos object is referenced. For example, an image can be included in PDF logical structure via **StOBJ** (see

Section 5.5.4), so that it can be included later in element content. The example in Section 6.11.8 shows using **NI** with an alternate image.

The syntax for defining an image name is:

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

NI takes the standard **_objdef** key to name the image within Distiller; see Section 4.1.1. Image names are in the namespace governed by **NamespacePush** and **NamespacePop**, defined in Section 4.5.

The image named by an **NI** command is to be found subsequently in the PostScript source file, but it does not need to immediately follow the **NI**. An image is assigned the name given by the most recent **NI** not yet paired with an image.

Another way of understanding this: Distiller maintains a stack of names pushed by **NI** and popped by the occurrence of an image. If an image is encountered when this stack is empty, it is not an error: the image simply does not receive a name.

5 Logical Structure

PDF files (in Versions 1.3 and beyond) can contain *structure trees* giving a logical structure to the information in a document. The facilities for logical structure in PDF are described in Section 8.4.3 of the *PDF Reference (Version 1.3, 2nd Edition).*

There is a *structure suite* of names used in conjunction with the **pdfmark** operator that can be used to specify logical structure within PDF files.

Section 6.14 gives a variety of examples of using the structure suite.

5.1 Elements and Parents

A document's logical structure consists of a hierarchy of *structure elements.* Elements can contain contents and attributes. At the root of the hierarchy is a dictionary object called the Structure Tree Root.

When using the structure suite, the hierarchy is established by means of the *implicit parent stack* of elements. Elements can be pushed onto or popped off of this stack. When an element is created, its parent is the current top item on the stack. (If the 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.) When element content is created, its containing element is the current top item on the stack.

NOTE: Some operators that specify an element cannot accept the Structure Tree Root as the implicit argument; therefore 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. These cases are noted in the command descriptions.

5.2 Structure Operators

This section lists the **pdfmark** names that make up the structure suite. Most of these are directly related to PDF logical structure features, but some only manipulate the state of the PDF creation process, without corresponding to any particular output.

- Structure Tree Root
  - StRoleMap adds entries to the role map.
  - StClassMap adds entries to the class map.
- Elements
  - 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.
- Element Content
  - 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.
- Attributes
  - StAttr enables the attachment of attribute objects to elements.
- Saving and restoring the stack
  - StStore saves the current state of the implicit parent stack.
  - StRetrieve restores the implicit parent stack from a saved state.

Section 5.3 through Section 5.7 fill in the details of the structure suite.

5.3 Structure Tree Root

Acrobat Distiller automatically creates a new Structure Tree Root the first time it creates a new element with **StPNE** (see Section 5.4.1).

The Structure Tree Root contains a *role map* and a *class map* (see Section 8.4.3 of the *PDF Reference* for details). The following two **pdfmark**s can be used to add information to these maps.

5.3.1 StRoleMap

StRoleMap specifies key-value pairs to be added as dictionary entries to the Structure Tree Root's role map. If the Structure Tree Root doesn't already exist, it is created; if the Structure Tree Root doesn't have a role map dictionary, one is created. A given key–value pair always modifies the role map, even if the key is already in the dictionary.

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
```

5.3.2 StClassMap

StClassMap behaves like **StRoleMap**, except that it adds entries to the Structure Tree Root’s class map, 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
```

5.4 Elements

The structure suite provides several commands to create elements and link them into structure trees.

5.4.1 StPNE

StPNE (“Push New Element”) creates a new element whose parent is the element on the top of the implicit parent stack. Its syntax is:

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

These keys are described in Table 5.1.

TABLE 5.1 Common element keys

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.

TABLE 5.1 Common element keys

Key	Type	Semantics
Class	name	(*Optional*) The class name to be associated with the element.
At	integer	(*Optional*) Index at which to insert this item within its parent. If omitted, or greater than or equal to the parent's current number of children, the 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 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.
Bookmark	dictionary	(*Optional*) Specifies a bookmark that is generated for this structural element. Table 5.2 describes this dictionary.

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

NOTE: If the implicit parent stack is empty, the Structure Tree Root is pushed onto the stack and used as the new element's parent. If there is no Structure Tree Root, one is created, pushed onto the stack, and used as the new element's parent.

StPNE may also take the key **_objdef** to specify an object name for the element (see Section 4.1). Once an element is named, it can be referenced with the **E** key of the **StPush pdfmark** (see Section 5.4.3).

The **Bookmark** key allows a bookmark to be automatically generated for an element and added to the Structured Bookmark subtree. Its value is a bookmark dictionary, which may contain the **Title** and **Open** keys described in Table 5.2.

TABLE 5.2 Bookmark dictionary / bookmark tree root

Key	Type	Semantics
Title	string	(*Optional*) Bookmark title. The encoding and character set used is either PDFDocEncoding (as described in Appendix D in the *PDF Reference (Version 1.3, 2nd Edition)* 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.

TABLE 5.2 Bookmark dictionary / bookmark tree root

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.

If the **Title** key is absent, the title is the title of the element or its subtype.

The bookmark dictionary may also contain key-value pairs that specify an action to be taken when the bookmark is activated (see Chapter 3, "Specifying Actions and Destinations). If none of the action keys are present, the bookmark's action is to go to either the first page where a marked content is a child of this element or a child in one of its descendant elements.

Section 6.14.3 defines a bookmark for an element.

5.4.2 StBookmarkRoot

StBookmarkRoot creates the root bookmark for structure bookmarks added by a **StPNE** with a **Bookmark** key. Its syntax is:

```
[ /Title string
/Open boolean
action-specifying-keys...
/StBookmarkRoot pdfmark
```

It contains the **Title** and **Open** keys shown in Table 5.2. If the **Title** key is absent, the title is "Untitled".

It may also contain the action keys in Chapter 3, "Specifying Actions and Destinations." 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.

5.4.3 StPush

StPush pushes an existing element onto the implicit parent stack. The syntax for pushing an element is:

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

The **E** key specifies an existing element, given as an object name of the special form {*objname*} used to refer to Cos objects. It must be a name that was created by a previous **StPNE** using the **_objdef** key (see Section 5.4.1).

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.

5.4.4 StPop

StPop removes the element at the top of the implicit parent stack. It is an error for **StPop** to be encountered when the implicit parent stack is empty.

The syntax for popping an element is:

```
[/StPop pdfmark
```

5.4.5 StPopAll

StPopAll completely empties the implicit parent stack. The syntax for emptying the stack is:

```
[/StPopAll pdfmark
```

5.5 Specifying Element Content

Elements may have two kinds of document content: marked content and references to PDF objects.

Use **StBDC** and **StBMC** to indicate the beginning of marked content and **EMC** to delimit 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 structure hierarchy.

NOTE: Marked content can be specified independently of the structure suite, using the operators described in Section 2.8, "Marked Content (MP, DP, BMC, BDC, EMC)."

It is possible to nest marked content by nesting the **StBMC**/**BDC** and **EMC** operators. This is different than the nesting maintained by the tree structure of elements, which is implemented using **StPNE** and **StPop**. Note that nested marked content may belong to elements in different branches of a Structure Tree.

To specify references to PDF objects, use the **StOBJ** operator.

5.5.1 StBMC

StBMC marks the beginning of a sequence of marked content objects. Its syntax is:

```
[/T tag
/At integer
/StBMC pdfmark
```

The marked content is added to its containing element (the top element of the implicit parent stack) at the position optionally specified by the **At** key (see Table 5.1). The **T** key is described in Table 5.3. It is an error if the implicit parent stack is empty when **StBMC** is encountered.

TABLE 5.3 Specifying tags and property list entries for Marked Content

Key	Type	Semantics
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.
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.

5.5.2 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. Its syntax is:

```
[/T tag
/P properties-dictionary
/At integer
/StBDC pdfmark
```

The marked content is added to its containing element (the element on top of the implicit parent stack) at the position optionally specified by the **At** key (see Table 5.1). The **P** (**Properties**) and **T** (**Tag**) keys are described in Table 5.3. It is an error if the implicit parent stack is empty when **StBDC** is encountered.

5.5.3 EMC

EMC signals the end of a marked sequence of page content operators. Its syntax is:

```
[ /EMC pdfmark
```

5.5.4 StOBJ

StOBJ adds an existing PDF object to the content of the top element of the implicit parent stack, using the Cos object reference mechanism. Its syntax is:

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

The **Obj** key specifies the object to be added as data to the specified element, given as an object name of the special form {*objname*} used to refer to Cos objects. This object must have been created previously (see Section 4.1, "Naming Objects with _objdef.") and must be a dictionary or stream.

The **At** key (see Table 5.1) specifies the position of the new content within the containing element.

It is an error if the implicit parent stack is empty when **StOBJ** is encountered.

5.6 Attribute Objects

Elements can have additional information, or attributes, associated with them. Attributes are held in *attribute objects*, which can be associated with either a single element by using **StAttr** (see Section 5.6.1), or with a group of objects by storing it in the **ClassMap** of the Structure Tree Root, using **StClassMap** (see Section 5.3.2).

5.6.1 StAttr

StAttr creates a new attribute object and adds it to the element on top of the implicit parent stack.

The syntax to create a new attribute object is:

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

The **Obj** key specifies the object to be added as an attribute object to the specified element, given as an object name of the special form {*objname*} used to refer to Cos objects. This object must have been created previously (see Section 4.1, "Naming Objects with _objdef.") and must be a dictionary or stream.

NOTE: In the PDF file, the attribute object is stored in the **A** key in the element's dictionary.

It is an error if the implicit parent stack is empty when **StAttr** is encountered.

5.7 Storing and Retrieving the Implicit Parent Stack

Structure suite operators specify parents implicitly by means of the stack. However, it is not always possible to mimic a tree's structure by nesting the structure within the document. 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 applications flexibility in their page output while allowing them the convenience of specifying tree structure, the structure suite provides a way of storing and later retrieving the tree's context.

See Section 6.14.4 for an illustration of storing and retrieving the implicit parent stack.

NOTE: The names under which implicit parent stacks are stored and retrieved are in the current namespace governed by the stack operators **NamespacePush** and **NamespacePop**, defined in Section 4.5, “Namespace Commands.”

5.7.1 StStore

StStore saves the current state of the implicit parent stack (without changing it). Its syntax is:

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

The **StoreName** key specifies a name object to be 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.

5.7.2 StRetrieve

StRetrieve restores the implicit parent stack from a saved state, whose name is specified by the **StoreName** key (as described in Section 5.7.1). The syntax for a restoring the current state is:

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

The previous state of the implicit parent stack is overwritten by the restored state. 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.**

5.8 EPS Considerations

Encapsulated PostScript (EPS) is a special form of PostScript used to embed 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.

6 Examples

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

6.1 Ignore pdfmark if not defined in the PostScript interpreter

```
%!PS-Adobe-3.0
%%BeginProlog
/pdfmark where
{pop} {userdict /pdfmark /cleartomark load put} ifelse
%%EndProlog
```

6.2 Notes

6.2.1 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
```

6.2.2 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
```

6.2.3 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
```

6.3 Links

6.3.1 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
```

6.3.2 Link

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

6.3.3 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
```

6.3.4 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
```

6.3.5 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

%Equivalent link using Launch action
[/Rect [ 50 425 295 445 ]
/Action /Launch
/Border [ 0 0 2 ]
/Color [ .7 0 0 ]
/URI (http://www.adobe.com)
/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
```

6.3.6 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
```

6.3.7 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
```

6.3.8 Movie annotation

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

6.4 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 color and style (new in Acrobat 5.0)
[/Count 0
/Title (Adobe’s Home Page)
/Action /Launch
/URI (http://www.adobe.com)
/C [1 0 0]
/F 3
/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
```

6.5 Articles

6.5.1 Article action

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

6.5.2 Create text for the article “Now is the Time”

```
/Helvetica 12 selectfont
(Now is the Time \(Article\)) 230 690  moveto show
(Now is the time for all good men to come to the aid of their
country.) 230 670  moveto show
(Now is the time for all good men to come to the aid of their
country.) 230 655 moveto show
%... additional text
(Click here to go to Adobe's Home Page on the Web) 55 430 moveto show
```

6.5.3 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
```

6.6 Page Cropping

6.6.1 Crop this page

```
[/CropBox [0 0 288 288] /PAGE pdfmark
/Helvetica findfont 12 scalefont setfont
/DrawBorder
{10 278 moveto 278 278 lineto 278 10 lineto
10 10 lineto closepath stroke
} bind def
%%EndSetup
%%Page: 1 1
DrawBorder
75 250 moveto
(This is Page 3) show
75 230 moveto
(Click here to go to page 1.) show
75 200 moveto
(Click here to open test.doc.) show
```

6.6.2 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
} bind def
/Helvetica findfont 10 scalefont setfont
%%EndSetup
%%Page: 1 1
DrawBorder
75 690 moveto
(This is Page 1) show
75 670 moveto
(Below is a closed, default note created using pdfmark:) show
75 570 moveto
(Below is an open note with a custom color and label:) show
400 670 moveto
(Below is a closed note) show
```

```
400 655 moveto
(containing private data:) show
400 570 moveto
(Below is a custom annotation.) show
400 555 moveto
(It should appear as an unknown) show
400 540 moveto
(annotation icon:) show
```

6.7 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
```

6.8 File Open Action

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

6.9 Page Label

```
%%Page: Sec1:2 1
%%PlateColor: Cyan
/pdfmark where {pop [ /Label (Sec1:1) /PlateColor (Cyan) /PAGELABEL
pdfmark } if

%%Page: iii 3
/pdfmark where {pop [ /Label (iii) /PAGELABEL pdfmark } if
```

6.10 Named Destinations

6.10.1 Definition of named destination

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

6.10.2 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
```

6.11 Cos Objects

6.11.1 Creating composite objects

```
[/_objdef {myarrayname} /type/ array /OBJ pdfmark
[/_objdef {mydictname} /type /dict /OBJ pdfmark
[/_objdef {mystreamname} /type /stream /OBJ pdfmark
```

6.11.2 Adding values to objects

```
% insert 132 at location 0
[{myarrayname} 0 132 /PUT pdfmark
[{myarrayname} 100 /APPEND pdfmark
[{myarrayname} /name2 /APPEND pdfmark
[{myarrayname} 2 [200 300] /PUTINTERVAL pdfmark
% At the end of the above examples, the array {myarrayname}
% has the value [132 100 200 300 /name2]
% 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
```

6.11.3 Creating an annotation with a name and adding 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
```

6.11.4 Using an object as a value

This example creates a text annotation on the current page with extra keys in the annotation dictionary. These keys, **MyPrivateAnnotArrayData** and **MyPrivateAnnotDictData**, have values that are indirect references to the array and dictionary objects created by the previous **pdfmark** entries.

```
[/_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
```

6.11.5 Putting 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
```

6.11.6 Using OBJ 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
```

6.11.7 Using OBJ 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
```

6.11.8 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 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).

6.12 Using the Graphics Encapsulation pdfmark Names (BP, EP, SP)

6.12.1 Creating a picture

This PostScript language sample draws a gray rectangle, then builds a picture enclosed by the **BP** and **EP pdfmark**s. (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.

```
% 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
```

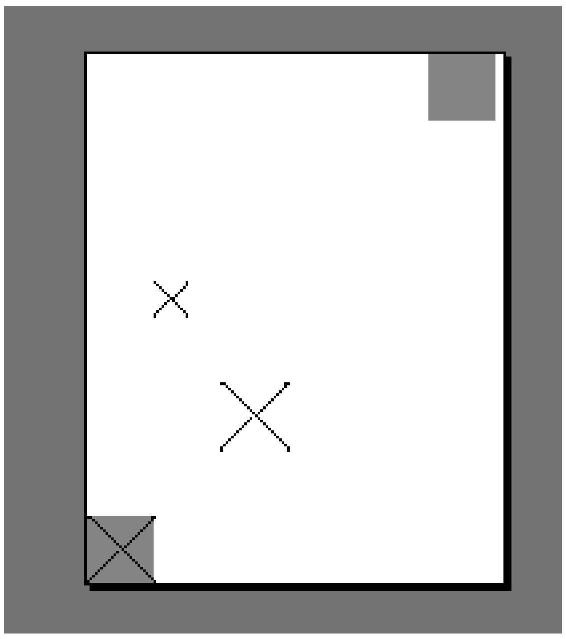
The resulting page stream in the PDF file contains the following:

```
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:



6.12.2 Using BP and EP pdfmarks to define button faces for forms

These examples illustrate how to use encapsulated graphics in forms. (See Section 6.14 for more information on forms).

This code defines the variable **__pdfMark__** as false if **pdfmark** is not defined. Subsequent code uses the variable to disallow printing PostScript code between the BP and EP **pdfmark**s..

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

This code defines common objects that can be used used by widgets for forms.

```
%%<<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
```

6.13 Forms Examples

This examples in this section show how to use the Forms **pdfmark** suite.

6.13.1 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
```

6.13.2 Define the AcroForm dictionary at the document Catalog

The AcroForm dictionary includes these required entries (see Section 7.6 of the *PDF Reference* for more information):

- **Fields** (the array from where all widgets in the form can be found)
- **DA** (Default Appearance)

- **DR** (Default Resources)
- **NeedAppearances** 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

%Define Fields array of Acroform dictionary. It will
%contain entries for each of the widgets defined below.
%NOTE: it's not necessary to explicitly assign the widget
%annotations to the Fields array; Acrobat does it automatically
%when the file is opened.

[ /_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

%Put Acroform entry in catalog dictionary
[ {Catalog}
  <<
    /AcroForm {aform}
  >>
/PUT pdfmark
%%>> %End AcroForm
```

6.13.3 Define the Widget annotations, which are also field dictionaries for this form

This is 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
```

6.14 Structure Examples

This section gives examples illustrating various uses of the structure **pdfmark** suite. Example 6.14.1 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 6.14.2 shows the parts of the output PDF file that result from the PostScript language code.

6.14.1 A simple structure

This example has 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 impicit 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
```

6.14.2 PDF output resulting from code in Section 6.14.1

This example is for illustration only. The PDF code actually produced by Adobe Acrobat Distiller 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 child.
/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 represents 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 children.
/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
```

6.14.3 A bookmark for a structural element

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

6.14.4 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
```

6.14.5 Independence of logical and physical structure

This example 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
```

6.14.6 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
```

6.14.7 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
```

7 Changes Since Last Version

This document has been reorganized and streamlined since its previous version, dated November 1999. Several errors in the content and examples have been fixed. In addition, the following additions and changes have been made:

- All examples are now in Chapter 6, with links to them from the relevant sections of the document. Many of the examples were previously scattered throughout the document.
- The following sections were added on previously undocumented material: Section 2.7, “Page Label and Plate Color (PAGELABEL),” Section 2.8, “Marked Content (MP, DP, BMC, BDC, EMC),” and Section 4.3.3, “APPEND.”
- Table 2.2, “PDF annotation types,” and Table 3.1, “Action types,” were added.
- The new color color and style attributes of bookmarks were added to Table 2.5, “Bookmark Attributes.”