Adobe Acrobat 7.0.5

Adobe PDF

pdfmark Reference Manual

October 2, 2005

Contents

Chapter 5   Examples 59

Preface

The Acrobat® Distiller® application is a PostScript-language interpreter that converts PostScript® language files into Portable Document Format (PDF) files. Because PDF has features (such as annotations, bookmarks, articles, and forms) that are not expressable using the standard PostScript operators, some PostScript language extension is necessary to describe features that are present in PDF, but not in standard PostScript.

To satisfy this need, Acrobat Distiller provides the operator **pdfmark** that is not found in standard PostScript interpreters.

The use of **pdfmark** allows an independent software vendor (ISV) already using the PostScript language to express, in PostScript syntax, idioms that are converted by Acrobat Distiller to PDF 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 for those who use, are about to use, or wish to understand **pdfmark** constructs in PostScript code intended for conversion to PDF by Acrobat Distiller. Those using **pdfmark** typically do so in one of the following ways:

● by manual creation or modification of PostScript code;

● by filtering or post-processing existing PostScript code; or,

● by an application that directly generates **pdfmark** constructs as part of its PostScript code generation.

Contents

Chapter 5, "Examples," gives several examples of using **pdfmark**.

Appendix A, "pdfmark for JDF," describes the use of **pdfmark** with JDF files.

Other Useful Documentation

Making full use of **pdfmark** requires knowledge of both PostScript and PDF. The following documents are available from both bookstores and online at http://partners.adobe.com/asn:

- *PostScript Language Reference*, 3rd edition, ISBN 0201379228

- *PDF Reference*, fifth edition, version 1.6, ISBN 0321 30474 8

In addition, a software development kit (the Acrobat SDK) consisting of documentation, headers, and sample code that enables software developers to access the functionality of the Acrobat product suite can be found at http://partners.adobe.com/asn.

1 pdfmark Syntax and Use

This chapter describes the **pdfmark** operator, its syntax, and its use by Acrobat Distiller and other PostScript interpreters. It also describes how built-in and user-defined PDF objects are referred to and defined.

What is the pdfmark Operator?

The **pdfmark** operator is not a standard PostScript operator, but is provided as a PostScript-language extension used by Acrobat Distiller to describe features that are present in PDF, but not in standard PostScript. The **pdfmark** operator has been available beginning with Acrobat Distiller 3.0, and, as an extensible operator, has evolved with each release of the PDF specification. This document describes **pdfmark** as it applies to PDF version 1.6 and corresponds to the Acrobat 7.0 suite of products.

NOTE: While the **pdfmark** operator provides for greater extensibility, it is not intended to define every feature that is present in PDF but not in standard PostScript.

Syntax of the pdfmark Operator

The **pdfmark** operator requires the following syntax:

> `[ `any_1` ... `any_n` feature pdfmark`

that:

- begins with a mark object (either `mark` or `[`);

- is followed by zero or more PostScript objects called the *arguments* of the **pdfmark** operator; and,

- concludes with a name object that indicates the particular *feature* that the **pdfmark** operator is to apply.

NOTE: Any instance of the **pdfmark** operator, the mark, its arguments, and the feature name in a PostScript program is referred to as a *pdfmark* in this document.

NOTE: Frequently, the arguments for a given feature are sequences of key-value pairs. Many of the pdfmark keys correspond directly to PDF dictionary keys. However, some keys may be new, entirely different, or abbreviated forms of keys as found in PDF dictionaries. For example, the PDF **Subtype** key may become the pdfmark key **S**, **Dest** may become **D**, and **File** may become **F**, and so forth. See the *PDF Reference* for more information on PDF keys.

The **pdfmark** operator does not change the operand nor the dictionary stacks, but may alter the execution, graphics state, or clipping stacks, depending on the particular pdfmark feature.

Using pdfmark with Standard PostScript Interpreters

The pdfmark operator is an extension of the PostScript language that is implemented in Acrobat Distiller, but is not available in many other PostScript products. Therefore, if a PostScript program containing pdfmark constructs is to used by these other products, it is necessary to ensure that these other products take some reasonable response when encountering the **pdfmark** operator.

One reasonable response is ignore pdfmark constructs. This can be accomplished by defining a **pdfmark** procedure that discards the pdfmark code for interpreters in which the **pdfmark** operator does not exist. One possible way to do this is to place the following code in the prolog of a PostScript program:

```
%%BeginProlog
/pdfmark where                          % Is pdfmark already available?
   { pop }                              % Yes: do nothing (use that definition)
   {                                    % No: define pdfmark as follows:
   /globaldict where                    % globaldict is preferred because
       { pop globaldict }               % globaldict is always visible; else,
       { userdict }                     % use userdict otherwise.
   ifelse
   /pdfmark /cleartomark load put
   }                                    % Define pdfmark to remove all objects
ifelse                                  % up to and including the mark object.
%%EndProlog
```

This code example works on PostScript Level 1 and above interpreters. In the examples that follow, PostScript Level 2 or higher is assumed to simplify the presentation of the examples.

Most pdfmark features are *atomic*. That is, the pdfmark construct stands alone and, if removed, does not affect surrounding PostScript. A few pdfmark features, on the other hand, are *modal*. A modal feature is one that, once completed, leaves the interpreter in a different state. Most modal features are paired: one feature shifts to a new state and a corresponding feature shifts back to the previous state. For example, consider:

[*any$_1$* ... *any$_n$* /BeginFeature pdfmark
additional PostScript code
[*any$_1$* ... *any$_m$* /EndFeature pdfmark

If it is desired to make the *additional PostScript code* conditional on the availability of the **pdfmark** operator, then the above definition of **pdfmark** needs to be improved.

```
%%BeginProlog
/pdfmark where
   { pop globaldict /?pdfmark /exec load put }   % pdfmark is built-in: exec code.
   {
   globaldict
      begin
      /?pdfmark /pop load def                     % pdfmark is absent: ignore code.
      /pdfmark /cleartomark load def
      end
   }
ifelse
%%EndProlog
```

With this the handling of modal code can be performed as:

```
[ any₁ ... anyₙ /BeginFeature pdfmark
{ additional PostScript code } ?pdfmark
[ any₁ ... anyₘ /EndFeature pdfmark
```

While the above solution is sufficient in most circumstances, it might be desirable to define a **pdfmark** procedure to handle individual features. The following example demonstrates a simple framework for handling individual pdfmark features:

```
%%BeginProlog
currentglobal currentpacking      % Because the pdfmark definition below uses
true setglobal true setpacking    % composite objects, we need to make sure the
                                  % procedure is defined in global VM mode.
/pdfmark where
   { pop globaldict /?pdfmark /exec load put}
   {
   globaldict
      begin
      /?pdfmark /pop load def
      /pdfmark
         {
            { counttomark pop }   % Check to see that a mark is on the stack.
         stopped
            { /pdfmark errordict /unmatchedmark get exec stop }
         if                       % Raise an error if no mark is found.
         dup type /nametype ne    % The topmost argument must be the feature.
            { /pdfmark errordict /typecheck get exec stop }
         if                       % The feature must be a name object.
            {
            dup /FEATURE₁ eq
               { (Interpreting FEATURE 1\n) print cleartomark exit }
            if                    % Replace the above code with actual code
            dup /FEATURE₂ eq
               { (Interpreting FEATURE 2\n) print cleartomark exit }
            if                    % Replace the above code with actual code
            (Feature not supported: ) print == cleartomark exit
                                  % Replace the above code with actual code
            }
         loop
         } bind def
      end
   }
   ifelse
   setpacking setglobal           % Restore to original modes.
   %%EndProlog
```

In the above code, the name objects FEATURE$_n$ would be replaced with actual pdfmark feature names and the code that follows the `dup /FEATURE`$_n$ `eq` would be replaced with code that consumes all of the arguments and the mark object.

In the examples that follow in this document, the `?pdfmark` definition is assumed to be as shown above. To work correctly with non-Distiller PostScript interpreters, any production implementation of these or additional definitions must take into account factors such as PostScript level, VM allocation modes, packing modes, and others.

Syntax for Private Keys

Some features can accept arbitrary key–value pairs, providing a way of placing *private* data into PDF files. All keys must be name objects. Unless otherwise stated, values must 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 Adobe's prefix **ADBE**.

Named Objects

This section describes how built-in and user-defined PDF objects are referred to and defined.

Built-In Named Objects

A PDF file contains built-in objects such as the Catalog and Page dictionaries. To refer to one of these dictionaries in a pdfmark construct, a syntax called a *named object*:

> {*objname*}

is used where objname is one of:

- **Catalog** — the PDF file's Catalog dictionary
- **DocInfo** — the PDF file's Info dictionary
- **Page***N* — 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

NOTE: The *objname* used here is not a standard PostScript name object; that is, it does not start with a slash "/" but instead is surrounded with braces "{ }"; it otherwise follows the syntax of PostScript name objects. The *objname* serves as a reference name to identify particular PDF objects and has no relationship to any identifier created in the resultant PDF file.

User-Defined Named Objects

In addition to built-in named objects, user-defined named objects may be created. The syntax to specify a user-defined named object is:

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

The name `/_objdef` indicates that a named object is to be defined and is followed by the `{objname}`. The object type, *objtype,* specifies the PDF type of named object that is to be created and must be one of the following name objects:

- **/array** — to create an array;
- **/dict** — to create a dictionary; or,
- **/stream** — to create a stream.

NOTE: The feature `/OBJ` is used only to declare a particular objname and its associated type. Other pdfmark features are required to associate this objname with actual content and to have some existing PDF object refer to it.

Here is an example in which the named object "galaxy" is declared to be a dictionary type:

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

A few pdfmark features allow for the definition of a named object as part of the argument list. In these cases, the modified syntax is:

```
[ /_objdef {objname} any₁ ... anyₙ feature pdfmark
```

In this case, the objname is not only created, but also refers to the PDF object created as a result of the **pdfmark** feature. The **type** entry is not used because the *feature* implicitly provides this information. The following features support this syntax:

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

Named objects created in any of the above ways can be used in the definition of other named objects. That is, an {objname} can be used as an argument in a pdfmark construct as the value of a key–value pair or as an element in an array. In these cases, Distiller places an indirect reference to the object that {objname} is associated with in the PDF file.

NOTE: A **pdfmark** construct can make an object reference {*objname*} before defining the object {*objname*}. That is, the {*objname*} can be in the argument list of a **pdfmark** construct before it is defined. If {*objname*} is never defined, it is left as an unresolved reference in the cross-reference table. Hence, any consumer of such a PDF file must be able to handle unresolved references.

Namespaces

When using named objects in PostScript programs, it is possible that the same name might be used more than once. To help avoid conflicts in name object definitions, Acrobat Distiller provides a means for specifying the scope in which named objects have well-defined meaning.

In addition to the standard 5 PostScript stacks (see "PostScript Language Reference," third edition, Section 3.4, on page 45), Acrobat Distiller maintains a stack of *namespaces*. The namespace stack is similar to the PostScript dictionary stack, except that only the top-most namespace name objects are visible. The namespace stack is also similar to the graphics state stack, except that no `currentgstate` analog is provided. A namespace contains:

- Names for user-defined named objects (see "User-Defined Named Objects" on page 14)
- Names for stored implicit parent stacks (see "StStore" on page 57)
- Names for images (see "Naming Images (NI)" on page 32)

The appropriate use of namespaces can help ensure that PostScript file construction containing code from various sources that use pdfmark constructs does not have named-object conflicts. A common example is the handling of Encapsulated PostScript files (see "EPS Considerations" on page 57).

NOTE: The built-in named objects are managed separately from the namespace stack and are always visible.

The following pdfmark features are available for manipulating namespaces:

1. **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
   ```

2. **NamespacePop** pops the topmost namespace from the stack. Once a namespace has been popped, it can not be accessed again. The next lower namespace on the stack becomes the current namespace.

   The syntax for popping a namespace is:

   ```
   [ /NamespacePop pdfmark
   ```

   A warning will be issued by Acrobat Distiller if **NamespacePop** is encountered when the namespace stack is empty.

   ```
   %%[ Warning: /NamespacePop pdfmark ignored: No matching NamespacePush ]%%
   ```

NOTE: There are no pdfmark features to save or restore namespaces.

Adding Content to Named Objects

Once a named object has been declared, content can be added to the PDF object that it refers to. There are several pdfmark features to accomplish this for each of the types of named objects.

Arrays

There are several methods for adding content to array named objects. The most basic of these is the **PUT** feature, taking the syntax:

```
[ {arrayname} index value /PUT pdfmark
```

The **PUT** feature 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 created as *NULL* objects. For example:

```
[ /_objdef {MoonInfo} /type /array /OBJ pdfmark
[ {MoonInfo} 0 (Earth to Moon) /PUT pdfmark
[ {MoonInfo} 1 238855 /PUT pdfmark
[ {MoonInfo} 2 /miles /PUT pdfmark
```

The above code creates an array object and populates it with objects of various types. Note, at this point, the named object cannot be reached because there are no entries in the PDF file's cross-reference table or file trailer that lead to it.

Adding array objects as above can become tedious. When adding objects to contiguous array index positions, the pdfmark feature **PUTINTERVAL** can simplify this task. The syntax for this feature is:

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

The operation of this feature is the same as in PostScript: $value_1$ is placed in $arrayname_{index}$, $value_2$ is placed in $arrayname_{index+1}$, and so forth. The array is resized if necessary to hold the objects added. The previous example can be simplified to:

```
[ /_objdef {MoonInfo} /type /array /OBJ pdfmark
[ {MoonInfo} 0 [(Earth to Moon) 238855 /miles] /PUTINTERVAL pdfmark
```

One additional convenience for adding objects to array is available: the **APPEND** feature. This feature adds one additional entry immediately after the end of the array. Its syntax is:

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

Dictionaries

The **PUT** feature can also be used to add dictionary content. The named object can be either a built-in name, such as `{Catalog}` or `{Page37}`, or a user-defined object name.

For dictionary named objects, the syntax of the **PUT** feature is:

```
[ {dictname} <<key₁ value₁ ... keyₙ valueₙ >> /PUT pdfmark
```

For dictionary named objects, **PUT** adds the key–value pairs provided as arguments. Continuing the previous example:

```
[ {Catalog} << /TheMoon {MoonInfo} >> /PUT pdfmark
```

This adds a key–value pair to the PDF Catalog dictionary. The inserted key is `/TheMoon` and the value is an indirect object. To illustrate this, the resultant PDF file might have the following content:

```
trailer
<< … /Root 9 0 R … >>
…
9 0 obj << … /Type /Catalog … /TheMoon 3 0 R … >>
endobj
3 0 obj [(Earth to Moon)238855/miles]
endobj
```

Notice that the named object `MoonInfo` does not appear in the resultant PDF file, but the object it referred to, `3 0 obj` in this case, does.

Streams

For stream named objects, the syntax can take several forms:

```
[ {streamname} string /PUT pdfmark
[ {streamname} file /PUT pdfmark
[ {streamname} <<key₁ value₁ ... keyₙ valueₙ >> /PUT pdfmark
```

A stream object consists of a sequence of bytes, its character data, and an associated dictionary. When the stream named object is created, the character data is empty. The source of stream data can come from an explicit string or can be read from a PostScript file object (a file or filter), in which case reading proceeds until the end of file is reached.

In addition to the character data, a stream has an associated PDF dictionary. Some dictionary entries such as **Length** are created automatically. Key–value pairs that do not conflict with the keys common to PDF stream dictionaries can be added to this dictionary. The resultant PDF object associated with the stream named object is always compressed using a lossless method that can be specified in Acrobat Distiller's Adobe PDF Settings dialog.

The **CLOSE** feature closes a stream object created by **pdfmark** and has the syntax:

```
[ {streamname} /CLOSE pdfmark
```

The named stream object is closed and written to the PDF file. The `{streamname}` 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.

For example:

```
[ /_objdef {MoonNotes} /type /stream /OBJ pdfmark
[ {MoonNotes} (Hipparchus around 129 BC calculated the distance to the Moon.\n)
  /PUT pdfmark
[ {MoonNotes} (The Moon was first touched by Armstrong on July 20, 1969.\n)
  /PUT pdfmark
[ {MoonNotes} << /Author (Steve Amerige) /Company (Adobe) >> /PUT pdfmark
[ {Catalog} << /MoonNotes {MoonNotes} >> /PUT pdfmark
[ {MoonNotes} /CLOSE pdfmark
```

2 Basic pdfmark Features

This chapter describes the basic **pdfmark** features. In general, the key–value pairs used as arguments for **pdfmark** follow closely the key–value pairs that appear in the PDF file. See the latest version of the *PDF Reference* for a description of the PDF file format.

The following **pdfmark** features 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)
- Naming Graphics (BP, EP, SP)
- Naming Images (NI)
- Transparency (SetTransparency)
- Add Metadata to the Catalog (Metadata)
- Embedded File Content (EMBED)
- Building an Output Intents Array

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

Annotations (ANN)

PDF supports several types of annotations. The properties of each annotation are specified in an *annotation dictionary* containing various key–value pairs. Section 8.4 of the *PDF Reference* describes all the types of annotations, and their required and optional dictionary entries.

The **pdfmark** operator using the feature name **ANN** is 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 pdfmark
```

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
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, annotations with unrecognized **Subtype** values, called *custom annotations*, are supported. Custom annotations may contain, in addition to the **Rect** and **Subtype** keys, arbitrary key–value pairs.

EXAMPLE 2.1 **Custom Annotation**

```
[/Rect [ 400 435 500 535 ]
/Subtype /ADBETest_DummyType
/ADBETest_F8Array [ 0 1 1 2 3 5 8 13 ]
/ANN pdfmark
```

When viewed with Acrobat Viewer, this annotation appears with an unknown annotation icon.

Table 2.3 lists optional keys that are common to all annotations. Specific annotation types have additional keys that they use. See section 8.4 of the *PDF Reference (Version 1.4, 3rd 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 pdfmark may be placed anywhere in the PostScript language file. If omitted, the pdfmark 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* for a description of this color space.) If omitted, a default color is used.
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*) 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.

TABLE 2.3   *Optional annotation keys*

Key	Type	Semantics
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 - (minus) 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 8.4.2 of the *PDF Reference*.
AP	dictionary	An *appearance dictionary* specifying how the annotation is presented visually. See Section 8.4.4 of the *PDF Reference* for details.
AS	name	The annotation's *appearance state*. See Section 8.4.4 of the *PDF Reference* for details.
Action (PDF key = **A**)	name or dictionary	An action to be performed when the annotation is activated. See "Actions" on page 41 for details. **NOTE:** For links, this key is not permitted if the **Dest** key is present.

"Notes" on page 23 and "Links" on page 24 describe the syntax for two of the original and most commonly used annotation types in more detail.

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* or Unicode. If Unicode, the string must begin with <FEFF>.
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**.

EXAMPLE 2.2 **Text Annotation**

```
[ /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
```

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."

EXAMPLE 2.3   ***Link Annotation***

```
[ /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
```

Other Annotations

Table 2.2 lists the other types of annotations that are available. For example, consider the following movie annotation.

EXAMPLE 2.4   ***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
```

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 8.6 of the *PDF Reference* for detailed information on using interactive forms.

See "Define the Widget annotations, which are also field dictionaries for this form" on page 73 for examples of using widget annotations to create interactive forms.

Bookmarks (OUT)

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

The syntax for a bookmark pdfmark 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)* 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* 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.

The bookmark **pdfmark**s may begin anywhere in the PostScript language file. However, they must appear in sequential order.

See "Bookmarks" on page 62 for examples of bookmark **pdfmark**s.

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 feature 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 pdfmark 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)* or Unicode. If Unicode, the string must begin with <FEFF>. For example, the Unicode string for (ABC) is <FEFF004100420043>. **Title** has a maximum length of 255 PDFDocEncoding characters or 126 Unicode values, although a practical limit of 32 characters is advised so that it can be read easily in the Acrobat viewer.
Rect	array	*(Required)* An array of four numbers [*xll, yll, xur, yur*] specifying the lower-left x, lower-left y, upper-right x, and upper-right y coordinates—in user space—of the rectangle defining the bead.

TABLE 2.6   ***Article Bead Attributes***

Key	Type	Semantics
Page	integer	(*Optional*) The sequence number of the page on which the bead is located. A bead pdfmark that contains the optional **Page** key may be placed anywhere in the PostScript language file. A bead pdfmark 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 "Articles" on page 63 for examples of articles.

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 x .04 inch (3 x 3 units) and the maximum allowed page size is 200 x 200 inches (14,400 x 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 "Page Cropping" on page 64 for examples of cropping.

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 - (minus) 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 **pdfmark**s may occur anywhere in the PostScript language file.

See "Info Dictionary" on page 64 for examples.

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 "File Open Action" on page 65 for an example.

Page Label and Plate Color (PAGELABEL)

The **PAGELABEL pdfmark** allows specification of the *page label* for a given page. Page labels can be strings like "iv" or "3-24", and do not necessarily correspond to the actual page numbers, which run consecutively. See Section 8.3.1 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 9.10.3 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.

"Page Label" on page 65 gives examples of the use of this pdfmark.

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 4, "Logical Structure," for information about pdfmark features that implement logical structure.

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.

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.

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** features **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:

```
[ /_objdef {objname} /BBox [xll yll xur yur] /BP pdfmark
... page marking instructions ...
```

```
[ /EP pdfmark
[ {objname} /SP pdfmark
```

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

NOTE: Graphics names are in the namespace governed by **NamespacePush** and **NamespacePop**, defined in "Namespaces" on page 15.

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 features that allow specifying named objects can add pictures built using **BP** and **EP** to a page.

See "Using the Graphics Encapsulation pdfmark Names (BP, EP, SP)" on page 68 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 "Using BP and EP pdfmarks to define button faces for forms" on page 70.

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 named object is referenced. For example, an image can be included in PDF logical structure via **StOBJ** (see "StOBJ" on page 55), so that it can be included later in element content. The example in "Using OBJ and PUT pdfmarks to create an alternate image" on page 67 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. Image names are in the namespace governed by **NamespacePush** and **NamespacePop**, defined in "Namespaces" on page 15.

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.

Transparency (SetTransparency)

PDF 1.4 extends the Adobe imaging model to include the notion of transparency. See Chapter 7 in the *PDF Reference* for complete information on transparency. To produce PDF files with transparency from PostScript files, use the **SetTransparency pdfmark** feature. This feature provides a mechanism for specifying the following graphics state parameters:

TABLE 2.8   ***Graphics State Parameters for Transparency***

Key	Value	Meaning
BM	name or array of names	Current blend mode. Default is **Normal**.
SMask	dictionary or **None**	Current soft mask, specifying the mask shape or mask opacity values. Default is **None**.
CA	number	Current stroking alpha constant, specifying the constant shape or constant opacity value to be used for stroking operations. Default is **1.0**.
ca	number	Same as **CA**, but for nonstroking operation. Default is **1.0**.
AIS	boolean	The alpha source flag ("alpha is shape"), specifying whether the current soft mask and alpha constant are to be interpreted as shape values (**true**) or opacity values (**false**). Default is **false**.
TK	boolean	The text knockout flag, which determines the behavior of overlapping glyphs within a text object. Default is **true**.

The syntax of the **SetTransparency** feature is:

```
[ key-value pairs /SetTransparency pdfmark
```

where recognized key-value pairs are found in Table 2.8.

NOTE: The keys used by this **pdfmark** feature are the same as are found in PDF documents.

The arguments to the SetTransparency feature are checked for correct types and values. Unrecognized keys are ignored and their values are not checked or written to the PDF document. If no recognized key-value pairs are presented, then this feature adds no transparency information to the PDF document.

The values set by this feature are subject to **gsave**/**grestore**. For example:

```
[ /ca .8 /SetTransparency pdfmark      % Nonstroking alpha is now .8
  gsave
   [ /ca .7 /SetTransparency pdfmark    % Nonstroking alpha is now .7
  grestore
                                        % Nonstroking alpha is now .8
```

The **initgraphics** operator resets all of the graphics state parameters for transparency to the defaults as shown in Table 2.8.

The following PostScript demonstrates a use of the **SetTransparency** feature using Normal blend mode with differing opacities:
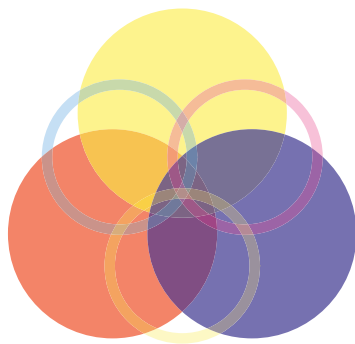
```
/DeviceCMYK setcolorspace 15 setlinewidth
[ /ca .6 /CA .3 /BM /Normal /SetTransparency pdfmark

0 1 1 0 setcolor 220 330 150 0 360 arc fill    % red
0 0 1 0 setcolor 320 503 150 0 360 arc fill    % yellow
1 1 0 0 setcolor 420 330 150 0 360 arc fill    % blue

1 0 0 0 setcolor 230 440 104 0 360 arc stroke % cyan
0 1 0 0 setcolor 410 440 104 0 360 arc stroke % magenta
0 0 1 0 setcolor 320 284 104 0 360 arc stroke % yellow
```

Compare this to the following in which the blend mode has been changed:
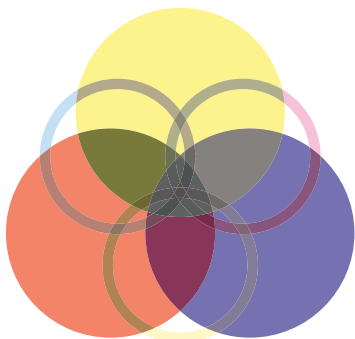
```
/DeviceCMYK setcolorspace 15 setlinewidth
[ /ca .6 /CA .3 /BM /Difference /SetTransparency pdfmark

0 1 1 0 setcolor 220 330 150 0 360 arc fill    % red
0 0 1 0 setcolor 320 503 150 0 360 arc fill    % yellow
1 1 0 0 setcolor 420 330 150 0 360 arc fill    % blue

1 0 0 0 setcolor 230 440 104 0 360 arc stroke % cyan
0 1 0 0 setcolor 410 440 104 0 360 arc stroke % magenta
0 0 1 0 setcolor 320 284 104 0 360 arc stroke % yellow
```

Note that filling and stroking the *same* path results in the use of the PDF **f** and **S** operators and not the **B** operator. This produces a "double border" effect and is not usually desirable.
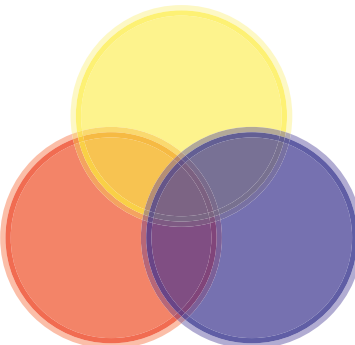
```
/DeviceCMYK setcolorspace 15 setlinewidth
[ /ca .6 /CA .3 /BM /Normal /SetTransparency pdfmark

0 1 1 0 setcolor 220 330 150 0 360 arc         % red path
                 gsave fill grestore stroke    % fill, then stroke
0 0 1 0 setcolor 320 503 150 0 360 arc         % yellow path
                 gsave fill grestore stroke    % fill, then stroke
1 1 0 0 setcolor 420 330 150 0 360 arc         % blue path
                 gsave fill grestore stroke    % fill, then stroke
```

Transparency Group XObject and Soft Mask

To specify a soft mask dictionary in a graphics state, it is necessary to *define* and *access* a transparency group XObject—a form XObject with a **Group** entry. See the *PDF Reference*, Section 7.5.5, for complete information.

Transparency Group XObject

There are two PostScript idioms that create a Form XObject with Distiller: the **execform** operator and the **BP pdfmark** feature. In Distiller 6.0, each of these recognize the **Group** key that is used to indicate a transparency group. Two forms with differing **Group** content are considered to be different. The syntax for these two idioms are:

```
<< /FormType 1
   /BBox [xll yll xur yur]
   /Group group-dictionary
   ...
>>


[ /_objdef {myForm}
   /BBox [xll yll xur yur]
   /Group group-dictionary
   ...
   /BP pdfmark
```

Soft Mask Dictionaries

Because Distiller is configured to use **execform** (not **/Form defineresource**), there is no direct way for Distiller to access a PostScript form dictionary if it is not used by **execform**. But a form used by **execform** will always leave marks on the page. So the way to create a soft mask dictionary is to create a transparency group form XObject using the **BP pdfmark** feature, then refer to this form in the soft mask dictionary in the Graphics state. For example:

```
[ /_objdef {myForm}                % Name to be used by G in Soft Mask below
   /BBox [xll yll xur yur]
   /Group dict
   /BP pdfmark
   ... define the shapes here
   /EP pdfmark

% Set the soft mask in Graphics state
[ /SMask << /S /Alpha /G {myForm} >> /SetTransparency pdfmark
```

Here is another example:

```
280 0 translate
/DeviceCMYK setcolorspace
% Draw the background...
0 0 0 0.2 setcolor 10 540 100 200 rectfill
1 1 1 0 setcolor 10 540 200 200 rectstroke
% Define the form...
[ /_objdef {aForm} /BBox [ 10 540 210 740]
   /Group << /S /Transparency /K true>> /BP pdfmark
/DeviceCMYK setcolorspace
```

```
0.14 0.85 0.77 0.03 setcolor 72 600 50 0 360 arc fill
0.12 0.02 0.78 0 setcolor 110 650 50 0 360 arc fill
0.93 0.69 0.07 0.01 setcolor 147 600 50 0 360 arc fill
[ /EP pdfmark
% Draw the form...
gsave
[ /ca 0.5 /BM /Normal /SetTransparency pdfmark
[ {aForm} /SP pdfmark
grestore
% Use the Form as Soft Mask...
[ /SMask << /S /Alpha /G {aForm} >> /SetTransparency pdfmark
...
```

Soft Mask Images

There are two ways to specify a soft mask in PDF: a soft-mask dictionary in the Graphics state as described above, and a soft-mask image associated with another image XObject (as an **SMask** entry).

A soft-mask image XObject has the same entries as an ordinary image XObject, with some restrictions:

1. **ColorSpace** must be **DeviceGray**.

2. **Matte** is a array of component values in the color space of the parent image.

3. **Width** and **Height** must be the same as in the parent image if **Matte** is present.

4. **ImageMask** must be `false` or absent.

5. **Mask** and **SMask** must be absent.

6. **BitsPerComponent** is required.

Distiller already has a mechanism for naming and identifying image objects using the **NI pdfmark** feature. To support softmasks, **NI** now recognizes three additional entries: **IsSoftMask**, **Matte**, and **SMask**.

TABLE 2.9   *NI pdfmark*

Key	Value	Comments
/_objdef	{nameobject}	A name object assigned to the next image.
IsSoftMask	boolean	Default is **false**.
Matte	array	Array of component values specifying matte color with which the parent image data has been pre-blended.
SMask	{SoftMaskImageName}	{SoftMaskImageName} must be defined already by another **NI pdfmark**. If **SMask** is present, **IsSoftMask** must be **false**.

Using NI pdfmark, one must define the soft-mask image first and then use it as the SMask entry for the parent image. For example:

```
[ /_objdef {mySoftMask}           % Name assigned to the next image.
  /SoftMask true                  % Next image {mySoftMask123} is a soft mask.
  /Matte [.1 .2 .3]
  /NI pdfmark
... define the soft mask image (ColorSpace must be /DeviceGray)

[ /_objdef {myImage}              % Name assigned to next image.
  /SMask {mySoftMask}             % Associate soft mask {mySoftMask123}
  /NI pdfmark
... define the image here
```

In this example, the image's **ColorSpace** must have three components and the image data must be pre-blended with **[.1 .2 .3]**.

Add Metadata to the Catalog (Metadata)

The ability to add metadata to the **Catalog** has been added for Distiller 6.0. The syntax for the **Metadata** feature is:

```
[ {Catalog} {XMPStreamName} /Metadata pdfmark
```

In future releases of Distiller, metadata may be attached to objects other than the **Catalog** object.

If the target is not the **Catalog** object or if DSC processing is enabled and this feature is located within Encapsulated PostScript (EPS), then this feature is ignored. Otherwise, the metadata associated with the stream **XMPStreamName** is added to the **Catalog** object with the key **Metadata**. See the *PDF Reference*, Section 9.2.2, for more information.

For example:

```
[ /_objdef {myMetadata} /type stream /OBJ pdfmark
[ {myMetadata} currentfile 0 (% -- end --) /SubFileDecode filter /PUT pdfmark
<?xpacket begin='' id='W5M0MpCehiHzreSzNTczkc9d'?>
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
...
% -- end --
[ {myMetadata} << /Type /Metadata /Subtype /XML>> /PUT pdfmark
[ {Catalog} {myMetadata} /Metadata pdfmark
```

Embedded File Content (EMBED)

With Distiller 6.0, the **pdfmark** feature **EMBED** enables the embedding of file content into a PDF document. The syntax for this feature is:

```
[ /Name (Unicode Name)
  /FS << /Type /Filespec /F (name) /EF << /F {streamName} >> >>
  EMBED pdfmark
```

This use of this feature embeds file content into a name tree within the **EmbeddedFiles** dictionary of the name dictionary (**Names**, a collection of name trees). This corresponds to the following structure as found in PDF documents:

```
<< /Type /Catalog             % The catalog dictionary
   /Names                     % The name dictionary; see PDF 1.4, p. 92
   <<  /EmbeddedFiles         % One particular name tree; see p. 93, 101-102
       <<  /Names             % The name tree node; p. 102
           [
           (Unicode Name)     % Unique Unicode string used for Acrobat access
               <<             % The file specification dictionary; p. 122
               /F (name)      % The file name for export
               /EF << ... >>  % Embedded file stream dictionary; p. 124
               >>
           ]
       >>
   >>
>>
```

For example:

```
[ /NamespacePush pdfmark
[ /_objdef {fstream} /type /stream /OBJ pdfmark
[ {fstream} << /Type /EmbeddedFile >> /PUT pdfmark
[ {fstream} (Simulating file content here) /PUT pdfmark
[ /Name (Unicode Unique Name)       % e.g., <feff 0041 0073> is Unicode for "As"
  /FS <<
      /Type /Filespec
      /F (myfile.txt)
      /EF << /F {fstream} >>
      >>
  /EMBED pdfmark
[ {fstream} /CLOSE pdfmark
[ /NamespacePop pdfmark
```

Building an Output Intents Array

The following Windows and Macintosh examples demonstrate how to build an Output Intents array. The hard-coded file and directory path should be applicable to most users.

Windows Example

```
% Define the profile object. The file is set up using a Windows path.
% You can also use a Macintosh or Unix path, or embed the profile data
% in the PostScript.

[ /NamespacePush pdfmark
[ /_objdef {Profile} /type /stream /OBJ pdfmark
[ {Profile} <</N 4>> /PUT pdfmark
[ {Profile}
(c:/Program Files/Common
```

```
Files/Adobe/Color/Profiles/Recommended/EuroscaleCoated.icc)
(r) file /PUT pdfmark

% Build the OutputIntent objects
[ /_objdef {OIDict} /type /dict /OBJ pdfmark
[ /_objdef {OIArray} /type /array /OBJ pdfmark
[ {OIDict} << /Type /OutputIntent /OutputCondition (Test) /S
/GTS_PDFX /OutputConditionIdentifier (Custom) /DestOutputProfile
{Profile} >> /PUT pdfmark
[ {OIArray} 0 {OIDict} /PUT pdfmark

% Store the OutputIntents array in the catalog.
[ {Catalog}<< /OutputIntents {OIArray} >> /PUT pdfmark
[ /NamespacePop pdfmark
```

Macintosh Example

```
% Define the profile object. The file is set up using a Windows path.
% You can also use a Macintosh or Unix path, or embed the profile data
% in the PostScript.

[ /NamespacePush pdfmark
[ /_objdef {Profile} /type /stream /OBJ pdfmark
[ {Profile} <</N 4>> /PUT pdfmark
[ {Profile}
(/Library/Application Support/Adobe/Color/Profiles/JapanStandard.icc)
(r) file /PUT pdfmark

% Build the OutputIntent objects
[ /_objdef {OIDict} /type /dict /OBJ pdfmark
[ /_objdef {OIArray} /type /array /OBJ pdfmark
[ {OIDict} << /Type /OutputIntent /OutputCondition (Test) /S
/GTS_PDFX /OutputConditionIdentifier (Custom) /DestOutputProfile
{Profile} >> /PUT pdfmark
[ {OIArray} 0 {OIDict} /PUT pdfmark

% Store the OutputIntents array in the catalog.
[ {Catalog}<< /OutputIntents {OIArray} >> /PUT pdfmark
[ /NamespacePop pdfmark
```

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 "Actions" on page 41."

- *Destinations* specify a particular location in a file, and a zoom factor. See "Destinations" on page 44."

  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."

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

TABLE 3.1 *Action types*

Action type	Description
Named	Execute an action predefined by the viewer application
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 8.5 in the *PDF Reference* 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... >>
```

"Custom link action (URI link for the Acrobat WebLink plug-in)" on page 61 shows a note pdfmark 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").

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 "Destinations" on page 44 for more information on these keys.

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 "Bookmarks" on page 62 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. N O T E : 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 examples on page 61.)
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* provides more information about the above specifiers.

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 "Link that launches another file" on page 61 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*.

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 "Article action" on page 63 for an example of an article action.

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 "View Destinations" on page 45.

- *Named destinations* specify the target as a name which has been defined. See "Defining Named Destinations" on page 46.

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.
FitR	*x1 y1 x2 y2* Fit the rectangle specified by the parameters to the window.

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

Name	Parameters and semantics
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.

The following examples illustrate destinations: , , and .

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 pdfmark 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 pdfmark may be placed anywhere in the PostScript language file. If omitted, the pdfmark 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 "Destinations" on page 44 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 "Named Destinations" on page 65 for examples of named destinations.

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 "Named Destinations" on page 65 for examples of named destinations.

Note: When used with the *Article* action, **Dest** has a different syntax. See "Article Actions" on page 44.

4 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 9.6 of the *PDF Reference (Version 1.4, 3rd 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.

"Structure Examples" on page 76 gives a variety of examples of using the structure suite.

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.

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.

The following sections fill in the details of the structure suite.

Structure Tree Root

Acrobat Distiller automatically creates a new Structure Tree Root the first time it creates a new element with **StPNE** (see "StPNE" on page 51).

The Structure Tree Root contains a *role map* and a *class map* (see Section 9.6 of the *PDF Reference* for details). The following two pdfmark features can be used to add information to these maps.

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
```

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
```

Elements

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

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 4.1.

TABLE 4.1   *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.
Class	name	(*Optional*) The class name to be associated with the element.

TABLE 4.1   **Common element keys**

Key	Type	Semantics
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 4.2 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. Once an element is named, it can be referenced with the **E** key of the **StPush pdfmark** (see "StPush" on page 53).

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 4.2.

TABLE 4.2   **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*) 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.
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.

"A bookmark for a structural element" on page 78 defines a bookmark for an element.

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 4.2. 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.

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 "StPNE" on page 51).

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.

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

StPopAll

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

    [/StPopAll pdfmark

StUpdate

StUpdate updates the entries of the current structure element. The syntax is:

    [<< /S /Span... >> /StUpdate pdfmark

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 "Marked Content (MP, DP, BMC, BDC, EMC)" on page 30."

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.

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 4.1). The **T** key is

described in Table 4.3. It is an error if the implicit parent stack is empty when **StBMC** is encountered.

TABLE 4.3   ***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.

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 4.1). The **P** (**Properties**) and **T** (**Tag**) keys are described in Table 4.3. It is an error if the implicit parent stack is empty when **StBDC** is encountered.

EMC

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

```
[ /EMC pdfmark
```

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 and must be a dictionary or stream.

The **At** key (see Table 4.1) 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.

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 "StAttr" on page 56), or with a group of objects by storing it in the **ClassMap** of the Structure Tree Root, using **StClassMap** (see "StClassMap" on page 51).

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 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.

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 "Interrupted structure" on page 78 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 "Namespaces" on page 15."

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.

StRetrieve

StRetrieve restores the implicit parent stack from a saved state, whose name is specified by the **StoreName** key (as described in "StStore" on page 57). 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.**

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.

Tagged PDF

PDF 1.4 introduced the concept of *tagged PDF*. Tagged PDF is a type of structured PDF that allows page content to be extracted and reused for various purposes, such as reflow of text and graphics, conversion to various file formats such as HTML and XML, and accessibility to the visually impaired.

For detailed information on tagged PDF, see section 9.7 of the *PDF Reference*.

In PDF 1.4, the Catalog dictionary contains a **MarkInfo** entry whose value is a dictionary. That dictionary has a single key called **Marked** whose value is a boolean; a value of **true** indicates that the document is a tagged PDF.

The syntax for indicating tagged PDF using **pdfmark** is:

```
[ {Catalog} <</MarkInfo <</Marked true >> >> /PUT pdfmark
```

See for an example of using tagged PDF.

5 Examples

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

Ignore pdfmark if not defined in the PostScript interpreter

```
%%BeginProlog
/pdfmark where
   { pop globaldict /?pdfmark /exec load put } % pdfmark is built-in: exec code.
   {
   globaldict
      begin
      /?pdfmark /pop load def                 % pdfmark is absent: ignore code.
      /pdfmark /cleartomark load def
      end
   }
ifelse
%%EndProlog
```

Notes

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
```

Fancy note

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

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

Private Data in Note

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

Links

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

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

Link

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

Fancy link

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

Link that launches another file

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

Custom link action (URI link for the Acrobat WebLink plug-in)

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

% 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
```

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
```

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
```

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
```

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
```

Articles

Article action

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

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 people 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
```

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
```

Page Cropping

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
```

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
```

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
```

File Open Action

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

Page Label

```
%%Page: Sec1:2 1
%%PlateColor: Cyan
[ /Label (Sec1:1) /PlateColor (Cyan) /PAGELABEL pdfmark

%%Page: iii 3
[ /Label (iii) /PAGELABEL pdfmark
```

Named Destinations

Definition of named destination

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

Link to a named destination

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

Named Objects

Creating user-defined named objects

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

Adding values to named 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 37's dictionary
[ {Page37} << /SpecialKey (special string) >> /PUT pdfmark
% insert key-value pair into the current page's dictionary
[ {ThisPage} << /NewKey (new string) >> /PUT pdfmark
```

Creating an annotation as a named object and adding content 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
```

Using a named 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
```

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
```

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
```

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
```

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).

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

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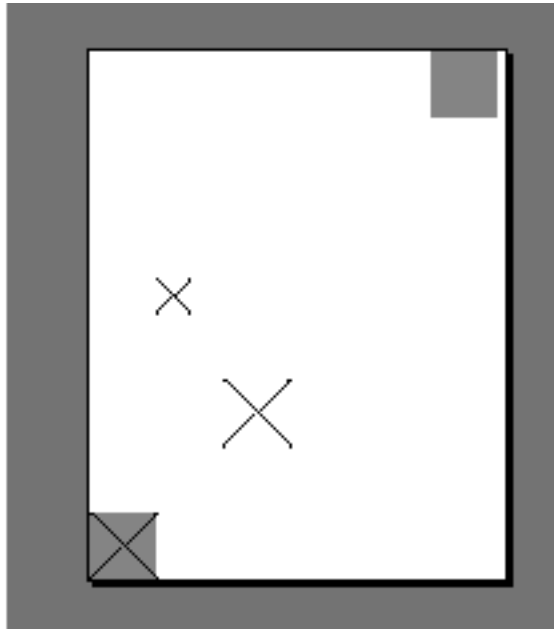
Using BP and EP pdfmarks to define button faces for forms

These examples illustrate how to use encapsulated graphics in forms. (See "Structure Examples" on page 76 for more information on forms). Note that the definition of **?pdfmark** is that as defined in Chapter 1.

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
   {0 0 1 setrgbcolor /ZapfDingbats 119 selectfont 0 7 moveto (4) show}
?pdfmark
[ /EP pdfmark

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

% Up/Down button appearances
[ /BBox [0 0 200 100] /_objdef {Up} /BP 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
  {
  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
  }
?pdfmark
[ /EP pdfmark
% Submit button appearances
[ /BBox [0 0 250 100] /_objdef {Submit} /BP 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
  }
?pdfmark
[ /EP pdfmark

[ /BBox [0 0 250 100] /_objdef {SubmitP} /BP 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
  }
?pdfmark
[ /EP pdfmark
```

Forms Examples

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

Define the AcroForm dictionary at the document Catalog

The AcroForm dictionary includes these required entries (see Section 8.6 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.

```
[ /_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 is 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
```

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.

```
[ /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
```

Structure Examples

This section gives examples illustrating various uses of the structure **pdfmark** suite. Example shows an entire structure tree, consisting of one section containing two paragraphs. It illustrates both how to create the tree structure and how the structure is related to the page content of the PDF file. Example shows the parts of the output PDF file that result from the PostScript language code.

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

[ /StBMC 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 instead.

[ /StPop pdfmark
[ /StPop pdfmark
```

PDF output resulting from code in previous section

This example is for illustration purposes only. The PDF code actually produced by Adobe Acrobat Distiller would not include comments and would differ in other ways.

```
% 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
```

A bookmark for a structural element

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

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.
[ /_objdef {LOT} /Subtype /ListOfTables /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
[ /StBMC 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

[ /StBMC 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
```

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.
[ /_objdef {FWL}/Subtype /List /Title (Funny Words) /StPNE pdfmark

[ /StPop pdfmark

[/Subtype /Section /StPNE pdfmark

[ /Subtype /P /StPNE pdfmark

% Begin PostScript code for the paragraph
[ /StBMC 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.
[ /StBMC 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
[ /StBMC 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
```

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

[ /StBMC 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.
[ /StBMC pdfmark
% ... write the portion of the paragraph that's on Page 2 ...
[ /EMC pdfmark
```

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 /StPNE pdfmark
                % No /At key, so defaults to being inserted
                % as last child of its parent.

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

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

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

Tagged PDF

This is a sample PostScript file that illustrates the use of tagged PDF.

```
% There are three things that should be added to this example:
% 1. A small table (just 2 row, three column)
% 2. A figure (either standalone, or actually embedded in the text)
% 3. If possible, the encoding of a font so that the soft hyphen really works
% without the "actual text"

[ /Creator (Hand Created)
  /CreationDate (D:20010508130548)
  /ModDate (D:20010508145339)
  /Author (Adobe Developer)
  /Title (Sample Document 1 for tagged PDF creation)
  /Subject (A base document for the creation of some simple PostScript and
PDFMarks to show tagged PDF)
  /Session (Tagged PDF Dev Tech Seminar)
  /Purpose (Demonstration)
  /DOCINFO pdfmark

[ {Catalog} <</MarkInfo <</Marked true>>>> /PUT pdfmark

% Layout class for documenttitle below
[ /_objdef {C1} /type /dict /OBJ pdfmark
[ {C1} <</O /Layout /SpaceAfter 10 /SpaceBefore 10 /TextAlign /Center>>
/  PUT pdfmark
[ /CM1 {C1} /StClassMap pdfmark

% Layout class for topichead
[ /_objdef {C2} /type /dict /OBJ pdfmark
[ {C2} <</O /Layout /SpaceAfter 5 /SpaceBefore 5 /TextAlign /Left>>
  /PUT pdfmark
[ /CM2 {C2} /StClassMap pdfmark

% Layout class for topichead2
[ /_objdef {C3} /type /dict /OBJ pdfmark
[ {C3} <</O /Layout /SpaceAfter 3 /SpaceBefore 3 /TextAlign /Left>>
  /PUT pdfmark
[ /CM3 {C3} /StClassMap pdfmark

% Layout class for p
[ /_objdef {C4} /type /dict /OBJ pdfmark
[ {C4} <</O /Layout /SpaceAfter 1 /SpaceBefore 3 /TextAlign /Left>>
  /PUT pdfmark
[ /CM4 {C4} /StClassMap pdfmark

[ /Subtype /document /Lang (en-US) /StPNE pdfmark

[ /_objdef {dta1} /type /dict /OBJ pdfmark

[ {dta1} <</O /XML-1.00 /Author (Joe)>> /PUT pdfmark
[ /Subtype /documenttitle /Class /CM1 /StPNE pdfmark
[ /Obj {dta1} /StAttr pdfmark
```

```
[ /StBMC pdfmark

/Helvetica-Bold findfont 24 scalefont setfont
216 720 moveto
(Title of Document) show

[ /EMC pdfmark
[ /StPop pdfmark

[ /Subtype /topic /StPNE pdfmark
[ /Subtype /topichead /Class /CM2 /StPNE pdfmark
[ /StBMC pdfmark

/Helvetica-Bold findfont 18 scalefont setfont
72 690 moveto
(First Topic) show

[ /EMC pdfmark
[ /StPop pdfmark

[ /Subtype /p /Class /CM4 /StPNE pdfmark
[ /StBMC pdfmark

/Helvetica findfont 12 scalefont setfont
72 674 moveto
(Some text in a paragraph in the first topic. These lines may not be justified,
but are illustrative.) show

[ /EMC pdfmark
[ /StPop pdfmark
[ /StPop pdfmark

[ /Subtype /topic /StPNE pdfmark
[ /Subtype /topichead /Class /CM2 /StPNE pdfmark
[ /StBMC pdfmark

/Helvetica-Bold findfont 18 scalefont setfont
72 648 moveto
(Second Topic) show

[ /EMC pdfmark
[ /StPop pdfmark
[ /Subtype /p /Class /CM4 /StPNE pdfmark
[ /StBMC pdfmark

/Helvetica findfont 12 scalefont setfont
72 632 moveto
(This is a paragraph of text in the second topic. ) show

[ /EMC pdfmark
[ /Subtype /emph /StPNE pdfmark
[ /StBMC pdfmark

/Helvetica-Oblique findfont 12 scalefont setfont
```

```
(Emphasized ) show

[  /EMC pdfmark
[  /StPop pdfmark
[  /StBMC pdfmark

/Helvetica findfont 12 scalefont setfont
(words ) show

72 618 moveto
(here.) show

[  /EMC pdfmark
[  /StPop pdfmark

[  /Subtype /topic /StPNE pdfmark
[  /Subtype /topichead2 /Class /CM3 /StPNE pdfmark

[  /StBMC pdfmark

/Helvetica-Bold findfont 14 scalefont setfont
72 596 moveto
(Subtopic of second topic) show

[  /EMC pdfmark
[  /StPop pdfmark

[  /Subtype /p /Class /CM4 /StPNE pdfmark
[  /StBMC pdfmark

/Helvetica findfont 12 scalefont setfont
72 580 moveto
(This paragraph of text is the second topic, first subtopic. ) show
72 566 moveto
(Hyphenated words make up this para) show

[  /EMC pdfmark
[  /Subtype /Span /ActualText <FEFF00AD> /StPNE pdfmark
[  /StBMC pdfmark

(-) show
[  /EMC pdfmark
[  /StPop pdfmark
[  /StBMC pdfmark

72 552 moveto
(graph also.) show

[  /EMC pdfmark
[  /StPop pdfmark
[  /StPop pdfmark
[  /StPop pdfmark

% And now we put in another topic with line numbers
```

```
[ /Subtype /topic /StPNE pdfmark
[ /Subtype /topichead /Class /CM2 /StPNE pdfmark
[ /StBMC pdfmark

/Helvetica-Bold findfont 18 scalefont setfont
72 510 moveto
(Line Numbered Topic) show

[ /EMC pdfmark
[ /StPop pdfmark

[ /Subtype /p /Class /CM4 /StPNE pdfmark

/Helvetica findfont 12 scalefont setfont
[ /Artifact <</Type /Layout>> /BDC pdfmark
48 494 moveto (1) show

[ /EMC pdfmark
[ /StBMC pdfmark

72 494 moveto
(This is some text such as would appear in a legal bill. ) show

[ /EMC pdfmark
[ /Artifact <</Type /Layout>> /BDC pdfmark

48 478 moveto (2) show

[ /EMC pdfmark
[ /StBMC pdfmark

72 478 moveto
(Note that this text has line numbers, but that ) show

[ /EMC pdfmark
[ /Artifact <</Type /Layout>> /BDC pdfmark

48 464 moveto (3) show

[ /EMC pdfmark
[ /StBMC pdfmark

72 464 moveto
(the numbers disappear when you reflow ) show

[ /EMC pdfmark
[ /Artifact <</Type /Layout>> /BDC pdfmark

48 450 moveto (4) show

[ /EMC pdfmark

[ /StBMC pdfmark

72 450 moveto
```

```
(the text our save the text as XML.) show

[ /EMC pdfmark
[ /StPop pdfmark
[ /StPop pdfmark


% =================================================================
% Create a simple link example
% =================================================================

[ /Subtype /P /StPNE pdfmark
[ /Subtype /Link /StPNE pdfmark

[ /_objdef {annotObj} /Rect [70 398 202 412]
  /Action << /Subtype /URI /URI (http://www.adobe.com) >>
  /Border [0 0 0]
  /Subtype /Link
  /ANN pdfmark

[ /Obj {annotObj} /StOBJ pdfmark

[ /StBMC pdfmark
  0 0 1 setrgbcolor
  72 400 moveto

(http://www.adobe.com.) show

[ /EMC pdfmark

[ /StPop pdfmark
[ /StPop pdfmark

% Set the tab order for the page to structure order.
[ {ThisPage} << /Tabs /S >> /PUT pdfmark


% =================================================================
% Create figure with a bounding box
% =================================================================

[ /Subtype /Figure /Alt (Logo.) /Title (Company Logo) /StPNE pdfmark

% Generate attribute dictionary for figure
[ /_objdef {layoutObj} /type /dict /OBJ pdfmark
[ {layoutObj} <</O /Layout /Height 70 /Width 140 /BBox [90 290 250 360]
  /Placement /Block>> /PUT pdfmark

% Attach attributes to figure
[ /Obj {layoutObj} /StAttr pdfmark

[ /StBMC pdfmark

/Helvetica findfont 48 scalefont setfont
0 0 0 setrgbcolor
90 290 moveto
```

```
90 360 lineto
250 360 lineto
250 290 lineto
closepath
stroke
100 300 moveto
1 0 0 setrgbcolor
(LOGO) false charpath
2 setlinewidth stroke

[  /EMC pdfmark

[  /StPop pdfmark


% ================================================================
% Simple List Example
% ================================================================

/Helvetica-Bold findfont 18 scalefont setfont
0 0 0 setrgbcolor

[  /Subtype /L /Lang (en-US) /Title (Some salutations) /StPNE pdfmark

% Create a list attribute which specifies the type of label to use
[  /_objdef {firstAttrObj} /type /dict /OBJ pdfmark
[  {firstAttrObj} <</O /List /ListNumbering /LowerRoman>> /PUT pdfmark

% Create an attribute specifing the writing direction
[  /_objdef {secondAttrObj} /type /dict /OBJ pdfmark
[  {secondAttrObj} <</O /Layout /WritingMode /LrTb>> /PUT pdfmark

% Set attribute dict on list
[  /Obj {firstAttrObj} /StAttr pdfmark
[  /Obj {secondAttrObj} /StAttr pdfmark

/Helvetica-Oblique findfont 12 scalefont setfont

[  /Subtype /LI /StPNE pdfmark
   [   /Subtype /Lbl /StPNE pdfmark
      [   /StBMC pdfmark
          48 238 moveto
          (i ) show
      [   /EMC pdfmark
   [   /StPop pdfmark
   [   /Subtype /LBody /Lang (en-cockney) /StPNE pdfmark
      [   /StBMC pdfmark
          72 238 moveto
          (whatcha) show
      [   /EMC pdfmark
   [   /StPop pdfmark
[  /StPop pdfmark

[  /Subtype /LI /StPNE pdfmark
   [   /Subtype /Lbl /StPNE pdfmark
```

```
        [   /StBMC pdfmark
            48 226 moveto
            (ii ) show
        [   /EMC pdfmark
    [   /StPop pdfmark
    [   /Subtype /LBody /Lang (fr) /StPNE pdfmark
        [   /StBMC pdfmark
            72 226 moveto
            (bon jour) show
        [   /EMC pdfmark
    [   /StPop pdfmark
[   /StPop pdfmark

[   /StPop pdfmark


% ==============================================================
% Simple Table Example
% ==============================================================

% Create a table element
[ /Subtype /Table /Lang (en-US) /StPNE pdfmark

    % Stick the frame of the table in an artifact
    [   /Artifact <</Type /Layout /BBox [40 175 340 220] >> /BDC pdfmark
        40 220 moveto 340 220 lineto 340 175 lineto 40 175 lineto closepath
        40 196 moveto 340 196 lineto
        190 220 moveto 190 175 lineto
        stroke
    [   /EMC pdfmark

    % Create a table attribute which specifies the type of label to use
    [   /_objdef {tableattrObj} /type /dict /OBJ pdfmark
    [   {tableattrObj} <</O /Layout /Placement /Block /SpaceAfter 10 /BorderColor
[0 0 0]>> /PUT pdfmark

    % Attach attribute to table
    [   /Obj {tableattrObj} /StAttr pdfmark

% Create an attribute object with the common settings for each table data cell
    [   /_objdef {tableCellsObj} /type /dict /OBJ pdfmark
    [   {tableCellsObj} <</O /Layout /Width 150 /BorderStyle /Solid
/BorderThickness 2 /BorderColor [0 0 0]>> /PUT pdfmark

    % Add it to the classmap
    [   /CommonTableInfo {widthObj} /StClassMap pdfmark

    [   /Subtype /THead /StPNE pdfmark
    [   /Subtype /TR /StPNE pdfmark
        [   /Subtype /TH /Class /CommonTableInfo /StPNE pdfmark
            [   /StBMC pdfmark
                48 200 moveto
                (Item) show
            [   /EMC pdfmark
        [   /StPop pdfmark
        [   /Subtype /TH /Class /CommonTableInfo /StPNE pdfmark
```

```
            [    /StBMC pdfmark
                200 200 moveto
                (Description) show
            [    /EMC pdfmark
        [    /StPop pdfmark
      [  /StPop pdfmark
      [  /StPop pdfmark

      [  /Subtype /TBody /StPNE pdfmark
      [  /Subtype /TR /StPNE pdfmark
        [    /Subtype /TD /Class /CommonTableInfo /StPNE pdfmark
            [    /StBMC pdfmark
                48 180 moveto
                (Thing) show
            [    /EMC pdfmark
        [    /StPop pdfmark

        [    /Subtype /TD /Class /CommonTableInfo /StPNE pdfmark
            [    /StBMC pdfmark
                200 180 moveto
                (Things) show
            [    /EMC pdfmark
        [    /StPop pdfmark
      [  /StPop pdfmark
      [  /StPop pdfmark

[  /StPop pdfmark


[  /StPop pdfmark

% Now that the text is done, let's make the outlines
% The first bookmark magnifies 400 percent, while the others go to their
% line in the text
[ /Count 4 /Page 1 /View [/XYZ 216 744 4.0] /Title (Title of Document)
/OUT pdfmark
[ /Page 1 /View [/XYZ 0 704 1.0] /Title (First Topic) /OUT pdfmark
[ /Count -1 /Page 1 /View [/XYZ 0 662 1.0] /Title (Second Topic) /OUT pdfmark
[ /Page 1 /View [/XYZ 0 610 1.0] /Title (Subtopic of second Topic) /OUT pdfmark
[ /Page 1 /View [/XYZ 0 530 1.0] /Title (Line Numbered Topic) /OUT pdfmark
[ /PageMode /UseOutlines /Page 1 /View [/XYZ null null null] /DOCVIEW pdfmark

% And finally the rolemap, with every tag that I have used defined.
[ /document /Document
  /documenttitle /H
  /p /P
  /emph /Span
  /topic /Div
  /topic2 /Div
  /topichead /H1
  /topichead2 /H2
  /StRoleMap pdfmark

showpage
(%%[Page: 1]%%) =
```

A

pdfmark for JDF

The use of **pdfmark** in PostScript has been expanded to include representations of JDF features. JDF is an extensible XML-based job ticketing format designed for use bythe printing industry. Information about JDF can be obtained from http://www.cip4.org.

In particular, pdfmark for JDF (henceforth called JDF pdfmark) allows the PostScript file/stream to specify elements and attributes to be added to a JDF document being used for a job. Applications that support JDF pdfmark include Acrobat Distiller 6.0 and later. See the document *Acrobat Distiller Parameters* (technical note 5151) for further information about Distiller.

Syntax

The syntax for the JDF pdfmark operator is as follows:

```
[   /Attribute string
    /Value string
    /Subtype /CreateAttribute
    /JDF pdfmark
```

Where the **Attribute** and **Value** keys are as described in Table A.1. Also, Table A.2 presents examples of XPath expressions.

TABLE A.1   *Keys supported by JDF pdfmark*

Key	Type	Semantics
Attribute	string	An XPath expression that identifies the location of the attribute absolutely from the root of the JDF. If any portion of the hierarchy of elements containing the attribute is not present in the JDF, they are created. *XPath* is a language for addressing parts of an XML document, as defined in *XML Path Language (XPath) Version1.0*, which is available from http://www.w3.org/TR/xpath. JDF pdfmark supports the following subset of XPath expressions: Expression ::= JDFRoot'/'Attribute | JDFRoot'/'Children'/'Attribute JDFRoot ::='//JDF' Children ::= Element | Element'/'Children Element ::= element | element'['FilterExpression']' FilterExpression ::= Filter | Filter'and'FilterExpression | Filter'or'FilterExpression Filter ::= Attribute'='Value Attribute ::='@'attribute
Value	string	The value to be assigned to the attribute, using the XPath expression: Value ::='"' 'value'"'

TABLE A.2 **Examples of XPath expressions**

Expression	Interpretation
//JDF/@JobID	Selects the **JobID** attribute in root JDF node
//JDF/JDFResourceLinkPool /ComponentLink/rRef	Selects the **rRef** attribute of the **ComponentLink** found in the **ResourcePool** in the root JDF node
//JDF/JDF[@Type="Trapping"]/@Status	Selects the **Status** attribute of the **Trapping** node that is a child of the root JDF node
//JDF/JDFResourceLinkPool /ComponentLink[@Usage="Output" and @ProcessUsage="Good"]/rRef	First identifies the **ResourceLinkPool** of the root JDF node. It then selects the **rRef** attribute of the **ComponentLink** with both a **Usage** attribute value "Output" and a **ProcessUsage** attribute with value "Good"

NOTE: In actual use, all XPath expressions should end with **@attribute** because they must define the location of an attribute.

The JDF pdfmark commands shown in Example A.1 cause supporting applications to modify the current JDF document as illustrated in Example A.2.

EXAMPLE A.1 **Using JDF pdfmark to set Trapping element and subelement attributes**

```
[ /Attribute (//JDF/JDF[@Type="Trapping"]/@Type)
  /Value (Trapping)
  /Subtype /CreateAttribute /JDF pdfmark

[ /Attribute
  (//JDF/JDF[@Type="Trapping"]/ResourceLinkPool/TrappingDetailsLink/@rRef)
  /Value (TD1)
  /Subtype /CreateAttribute /JDF pdfmark

[ /Attribute
  (//JDF/JDF[@Type="Trapping"]/ResourceLinkPool/TrappingDetailsLink[@rRef="TD1"]/@Usage)
  /Value (Input)
  /Subtype /CreateAttribute /JDF pdfmark
```

EXAMPLE A.2 ***JDF structure created through JDF pdfmark in*** ***Example A.1***

```
┌─────────────────┐
│ JDFRoot         │
│                 │
│                 │
└─────────────────┘
    │
    │  ┌─────────────────────┐
    │  │ JDF                 │
    └──│ Type = "Trapping"   │
       │                     │
       └─────────────────────┘
          │
          │  ┌─────────────────────┐
          │  │ ResourceLinkPool    │
          └──│                     │
             │                     │
             └─────────────────────┘
                │
                │  ┌──────────────────────┐
                │  │ TrappingDetailsLink  │
                └──│ rRef = "TD1"         │
                   │ Usage="Input"        │
                   └──────────────────────┘
```
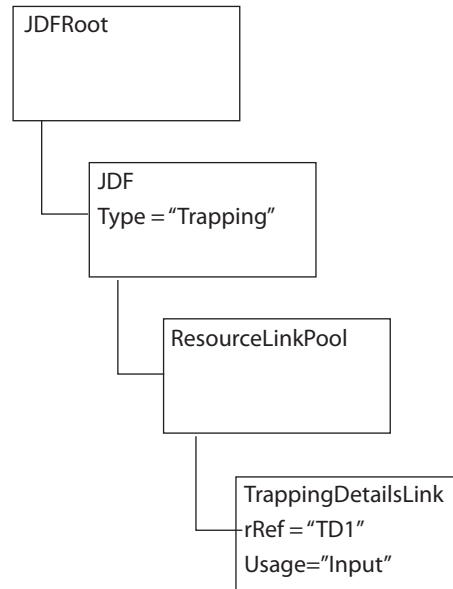
B Distilling Optional Content

Introduction

This appendix describes the ProcSet used to build Optional Content (OC) into PDF through Distiller. Optional content refers to content in a PDF document that can be selectively viewed or hidden. Optional content is a new feature that became available with Acrobat 6. See Section 4.10 of the *PDF Reference*, fifth edition, PDF 1.6, for more information on OC.

NOTE: While the OC ProcSet makes extensive use of pdfmark internally, clients of the OC ProcSet should not have to use pdfmark to add OC to a PDF file.

Initialization and Termination Code

To use the OC ProcSet, clients must insert the following PostScript into the document setup section of the PostScript job. This places definitions of the OC ProcSet procedures in **userdict** for easy access by the client.

```
{/OCProcSet /ProcSet findresource} stopped not
{/initialize get exec}
{
    /BeginOC /pop load def
    /EndOC {} def
    /SimpleOC /pop load def
    /SetOCGInitState {pop pop} bind def
    /OCEndPage {} def
    /SetOCGIntent {pop pop} bind def
    /SetOCGUsage {pop pop} bind def
    /AddASEvent {pop pop pop} bind def
    /GetOCGPdfmarkTag {{---invalidpdfmarkname---}} bind def
}
ifelse
```

Also, the following PostScript must be inserted into the trailer section of the PS file:

```
{/OCProcSet /ProcSet findresource /terminate get exec} stopped pop
```

When using the OC ProcSet, the Optional Content Group (OCG) is the primary data item. It is referred to by the OCG's name, which is a string object. (See the description of the entries in the OCG dictionary in the *PDF Reference*.) Clients of the ProcSet don't need to do anything to set up OCGs—they simply refer to them by their name strings, and the ProcSet takes care of creating them on-the-fly. Clients can set the initial state, intent, and usage info for OCGs, using SetOCGInitState, SetOCGIntent, and SetOCGUsage, respectively.

There are two techniques for using the ProcSet to make content optional: one for non-nested OC, and one for nested OC:

- Non-nested OC. The simplest technique is for non-nested OC, using the SimpleOC procedure. Simply pass in a string for the OCG name, and all marks up to the next SimpleOC belong to the OCG with that name. Passing in **null SimpleOC** makes subsequent content non-optional. At the end of the page, before the **showpage**, issue **null SimpleOC**.

- Nested OC. The second way is for documents that have nested visibility control. For these the ProcSet provides stack-style OC control. This is also the style of control used if you have content that requires an Optional Content Membership Dictionary (OCMD), because it belongs to more than one OCG and/or requires a visibility policy entry in the OCMD. For this sort of OC, use the BeginOC and EndOC procedures. With this style, you should call OCEndPage at the end of the page (before **showpage**). This ensures that the marked content is closed properly.

Procedure Definitions

AddASEvent

```
event_type event_categories ocgnames AddASEvent
```

Description

The **AddASEvent** procedure adds an auto state event to the PDF's default configuration. See the *PDF Reference* for a description of auto state control for OC.

Parameters

event_type	Must be a PostScript name. Either **/View**, **/Print**, or **/Export**.
event_categories	Must be a PostScript array of name objects (typically matching keys in OCG usage dictionaries). For a description of usage dictionaries, see Section 4.10.1 of the *PDF Reference*, fifth edition, PDF 1.6.
ocgnames	Array of valid PostScript string variables representing OCGs.

Returns

None.

Example

```
/View [/Zoom] [(30,000 Feet) (5,000 Feet) (100 Feet)] AddASEvent
```

The example declares that the three OCGs named **30,000 Feet**, **5,000 Feet**, and **100 Feet** are to be controlled for on-screen viewing, based on the current zoom level and the **/Zoom** information in each OCG's **Usage** dictionary.

BeginOC

ocgname BeginOC
ocgnames BeginOC
ocgname policy BeginOC
ocgnames policy BeginOC

Description

The **BeginOC** procedure is used to begin a span of content that belongs to the OCG(s) supplied. Used for nested visibility control. It is used when content can belong to more than one OCG. Both multiple membership (using the array of OCG names) and stack-based nesting are supported. The EndOC calls must come before the **showpage** call of any page. Every **BeginOC** call should have a matching **EndOC** call.

NOTE: It is illegal to mix **SimpleOC** and **BeginOC**/**EndOC** on the same page.

Parameters

ocgnames	String object identifying an OCG.
ocgname	Array of string objects identifying a set of OCGs.
policy	(Optional) One of the following names: **/AllOn**, **/AnyOn**, **/AllOff**, or **/AnyOff**, identifying the visibility policy to use. If no *policy* is specified, **/AnyOn** is used by default.

Returns

None.

See Also

EndOC

EndOC

EndOC

Description

The **EndOC** procedure is used to end a span of optional content. It is used to close a span of optional content started by BeginOC. Both multiple membership (using an array of OCG names) and stack-based nesting are supported. The **EndOC** calls must come before the **showpage** call of any page.

Every **BeginOC** call should have a matching **EndOC** call.

NOTE: It is illegal to mix **SimpleOC** and **BeginOC**/**EndOC** on the same page.

Returns

None.

See Also

BeginOC

GetOCGPdfmarkTag

ocgname `GetOCGPdfmarkTag` --> *procedure*

Description

The **GetOCGPdfmarkTag** returns the object that the ProcSet implementation uses to identify the OCG object for pdfmark. Using this object, the client can use the **/PUT** pdfmark command to add additional key/value pairs to the OCG dictionary.

Parameters

ocgname	String object identifying an OCG.

Returns

The OCG's **/OBJ** pdfmark tag.

See Also

pdfmark **/OBJ**, **/PUT** commands.

Example

```
[(MyLayer) GetOCGPdfmarkTag <</key1 (easy as) /key2 3.14159>> /PUT
pdfmark
```

This example adds the key/value pairs:

```
/key1 (easy as)
/key2 3.14159
```

to the dictionary for the OCG with the name **MyLayer**.

OCEndPage

`OCEndPage`

Description

The **OCEndPage** is called at the end of the page in a multi-page PostScript file to allow the ProcSet to close any open optional content on the current page. It can be used to close a call to either SimpleOC or BeginOC.

Returns

None.

SetOCGInitState

> *ocgname bool* SetOCGInitState

Description

The **SetOCGInitState** procedure sets the initial state of an OCG to be either ON (**true**) or OFF (**false**).

Parameters

ocgname	Valid PostScript string variable representing an OCG.
bool	**true** or **false**. Value of *ocgname*'s initial state. For a description of an OCG's state, see Section 4.10.1 of the *PDF Reference*, fifth edition, PDF 1.6.

Returns

None.

SetOCGIntent

> *ocgname intent* SetOCGIntent

Description

The **SetOCGIntent** procedure sets the **Intent** key in *ocgname* to *intent*.

Parameters

ocgname	Valid PostScript string variable representing an OCG.
intent	Value of *ocgname*'s **Intent** key, such as **/Design**, **/View**, **/All**, or **/None**, or an array of names, excluding **/All** and **/None**. For a description of an OCG's **Intent** key, see Section 4.10.1 of the *PDF Reference*, fifth edition, PDF 1.6. The default value is **/View**.

Returns

None.

SetOCGUsage

> *ocgname dict* `SetOCGUsage`

Description

The `SetOCGUsage` procedure sets the **Usage** key in *ocgname* to the *dict* supplied. This is the top level usage dictionary, not a usage category dictionary. Only one call per OCG is honored, so the client must collect all usage subdictionaries and issue a single call to set the **Usage** dictionary for the OCG.

Parameters

ocgname	Valid PostScript string variable representing an OCG.
dict	Value of *ocgname*'s **Usage** key, which is a dictionary. For a description of an OCG's **Usage** key, see Section 4.10.1 of the *PDF Reference*, fifth edition, PDF 1.6. By default, there is no **Usage** key in the OCG's dictionary. This procedure simply adds the key to *dict*.

Returns

None.

Example

```
(30,000 Feet) <</Zoom << /max 0.5 >> >> SetOCGUsage
(5,000 Feet) <</Zoom << /min 0.5 /max 4>> >> SetOCGUsage
(100 Feet) <</Zoom << /min 4 >> >> SetOCGUsage
```

This example specifies, in conjunction with the AddASEvent example, that, when viewing, the objects in the **30,000 Feet** OCG should be visible when the zoom level is less than 50%, the objects in the **5,000 Feet** OCG should be visible between 50% and 400%, and the objects in the **100 Feet** OCG should be visible when the zoom level is at least 400%.

SimpleOC

> *ocgname* `SimpleOC`

Description

The `SimpleOC` procedure ends any current optional content span, and begins a new one where the content belongs to *ocgname*.

To use the **SimpleOC** procedure, simply pass in a string for the OCG name, and all marks up to the next **SimpleOC** belong to the OCG with that name. Passing in **null SimpleOC** makes subsequent content non-optional. At the end of the page, before the **showpage**, issue **null SimpleOC**.

NOTE: It is illegal to mix **SimpleOC** and **BeginOC/EndOC** on the same page.

Parameters

ocgname	Valid PostScript string variable representing an OCG.

Returns

None.

Example

To show content on all layers (at all times):

```
null SimpleOC
```