Portable Document Format Publishing with GNU Troff

Keith Marshall
<keith.d.marshall@ntlworld.com>

A GNU MANUAL

Table of Contents

1. Introduction

It might appear that it is a fairly simple matter to produce documents in Adobe® "Portable Document Format", commonly known as PDF, using GNU Troff (`groff`) as the document formatter. Indeed, `groff`'s default output format is the native Adobe® PostScript® format, which PDF producers such as Adobe® Acrobat® Distiller,® or GhostScript, expect as their input format. Thus, the PDF production process would seem to entail simply formatting the document source with `groff`, to produce a PostScript® version of the document, which can subsequently be processed by Acrobat® Distiller® or GhostScript, to generate the final PDF document.

For many PDF production requirements, the production cycle described above may be sufficient. However, this is a limited PDF production method, in which the resultant PDF document represents no more than an on screen image of the printed form of the document, if `groff`'s PostScript® output were printed directly.

The Portable Document Format provides a number of features, which significantly enhance the experience of reading a document on screen, but which are of little or no value to a document which is merely printed. It *is* possible to exploit these PDF features, which are described in the Adobe® "pdfmark Reference Manual", with some refinement of the simple PDF production method, provided appropriate "feature implementing" instructions can be embedded into `groff`'s PostScript® rendering of the document. This, of course, implies that the original document source, which `groff` will process to generate the PostScript® description of the document, must include appropriate markup to exploit the desired PDF features. It is this preparation of the `groff` document source to exploit a number of these features, which provides the principal focus of this document.

The markup techniques to be described have been utilized in the production of the PDF version of this document itself. This has been formatted using `groff`'s `ms` macro package; thus, usage examples may be found in the document source file, `../../pdfmark.ms`, to which comments have been added, to help identify appropriate markup examples for implementing PDF features, such as:–

- Selecting a default document view, which defines how the document will appear when opened in the reader application; for example, when this document is opened in Acrobat® Reader, it should display the top of the cover sheet, in the document view pane, while a document outline should appear to the left, in the "Bookmarks" pane.
- Adding document identification "meta-data", which can be accessed, in Acrobat® Reader, by inspecting the "File/Document Properties/Summary".
- Creating a document outline, which will be displayed in the "Bookmarks" pane of Acrobat® Reader, such that readers may quickly navigate to any section of the document, simply by clicking on the associated heading in the outline view.
- Embedding active links in the body of the document, such that readers may quickly navigate to related material at another location within the same document, or in another PDF document, or even to a related Internet resource, specified by its URI.
- Adding annotations, in the form of "sticky notes", at strategic points within the PDF document.

All of the techniques described have been tested on *both* GNU/Linux, and on Microsoft® Windows™2000 operating platforms, using `groff 1.19.1`,[1] in association with `AFPL GhostScript 8.14`.[2] Other tools employed, which should be readily available on *any* Unix™ or GNU/Linux system, are `sed`, `awk` and `make`, together with an appropriate text editor, for creating and marking up the `groff` input files. These additional utilities are not provided, as standard, on the Microsoft® Windows™ platform, but several third party implementations are available. Some worth considering include the MKS® Toolkit,[3] Cygwin,[4] or MSYS.[5] This list is by no means exhaustive, and should in no way be construed as an endorsement of any of these packages, nor to imply that other similar packages, which may be available, are in any way inferior to them.

---

1. Later versions should, and some earlier versions may, be equally suitable. See http://www.gnu.org/software/groff for information and availability of the latest version.
2. Again, other versions may be suitable. See http://ghostscript.com for information and availability.
3. A commercial offering; see http://mkssoftware.com/products/tk/default.asp for information.
4. A *free* but comprehensive POSIX emulation environment and Unix™ toolkit for 32-bit Microsoft® Windows™ platforms; see http://cygwin.com for information and download.
5. Another free, but minimal suite of common Unix™ tools for 32-bit Microsoft® Windows™, available for download from https://mingw.osdn.io; it *does* include those tools listed above, and is the package which was actually used when performing the Windows™2000 platform tests referred to in the text.

2. Exploiting PDF Document Features

To establish a consistent framework for adding PDF features, a `groff` macro package, named `pdfmark.tmac`, has been provided. Thus, to incorporate PDF features in a document, the appropriate macro calls, as described below, may be placed in the `groff` document source, which should then be processed with a `groff` command of the form[6]

```
groff [-Tps|-Tpdf] [-m name] -m pdfmark [-options ...] file ...
```

It may be noted that the `pdfmark` macros have no dependencies on, and no known conflicts with, any other `groff` macro package; thus, users are free to use any other macro package, of their choice, to format their documents, while also using the `pdfmark` macros to add PDF features.

2.1. The `pdfmark` Operator

All PDF features are implemented by embedding instances of the **`pdfmark`** operator, as described in the Adobe® "pdfmark Reference Manual", into `groff`'s PostScript® output stream. To facilitate the use of this operator, the `pdfmark` macro package defines the primitive `pdfmark` macro; it simply emits its argument list, as arguments to a `pdfmark` operator, in the PostScript® output stream.

To illustrate the use of the `pdfmark` macro, the following is a much simplified example of how a bookmark may be added to a PDF document outline

```
.pdfmark \
  /Count 2 \
  /Title (An Example of a Bookmark with Two Children) \
  /View  [/FitH \n[PDFPAGE.Y]] \
  /OUT
```

In general, users should rarely need to use the `pdfmark` macro directly. In particular, the above example is too simple for general use; it *will* create a bookmark, but it does *not* address the issues of setting the proper value for the `/Count` key, nor of computing the `PDFPAGE.Y` value used in the `/View` key. The `pdfmark` macro package includes a more robust mechanism for creating bookmarks, (see section 2.4, "Creating a Document Outline"), which addresses these issues automatically. Nevertheless, the `pdfmark` macro may be useful to users wishing to implement more advanced PDF features, than those currently supported directly by the `pdfmark` macro package.

2.2. Selecting an Initial Document View

By default, when a PDF document is opened, the first page will be displayed, at the default magnification set for the reader, and outline and thumbnail views will be hidden. When using a PDF reader, such as Acrobat® Reader, which supports the `/DOCVIEW` class of the `pdfmark` operator, these default initial view settings may be overridden, using the `pdfview` macro. For example

```
.pdfview /PageMode /UseOutlines
```

will cause Acrobat® Reader to open the document outline view, to the left of the normal page view, while

```
.pdfview /PageMode /UseThumbs
```

will open the thumbnail view instead.

Note that the two `/PageMode` examples, above, are mutually exclusive — it is not possible to have *both* outline and thumbnail views open simultaneously. However, it *is* permitted to add `/Page` and `/View` keys, to force the document to open at a page other than the first, or to change the magnification at which the document is initially displayed; see the "pdfmark Reference Manual" for more information.

It should be noted that the view controlling meta-data, defined by the `pdfview` macro, is not written immediately to the PostScript® output stream, but is stored in an internal meta-data "cache", (simply implemented as a `groff` diversion). This "cached" meta-data must be written out later, by invoking the `pdfsync` macro, (see section 2.7, "Synchronizing Output and `pdfmark` Contexts").

---

6. Note that, if any `-T`*`dev`* option is specified, it should be either `-Tps`, or `-Tpdf`; any other explicit choice is unlikely to be compatible with `-m pdfmark`, and will have an unpredictable (possibly erroneous) effect on the output. If no `-T`*`dev`* option is specified, (in which case `-Tps` is implicitly assumed), or if `-Tps` is explicitly specified, then the output will be produced in PostScript® format, and will require conversion to PDF, (e.g. by using GhostScript tools); explicit specification of `-Tpdf` will result in direct output in PDF format, thus obviating the need for conversion.

2.3. Adding Document Identification Meta-Data

In addition to the /DOCVIEW class of meta-data described above, (see section 2.2, "Selecting an Initial Document View"), we may also wish to include document identification meta-data, which belongs to the PDF /DOCINFO class.

To do this, we use the pdfinfo macro. As an example of how it is used, the identification meta-data attached to this document was specified using a macro sequence similar to:–

```
.pdfinfo /Title     PDF Document Publishing with GNU Troff
.pdfinfo /Author    Keith Marshall
.pdfinfo /Subject   How to Exploit PDF Features with GNU Troff
.pdfinfo /Keywords  groff troff PDF pdfmark
```

Notice that the pdfinfo macro is repeated, once for each /DOCINFO record to be placed in the document. In each case, the first argument is the name of the applicable /DOCINFO key, which *must* be named with an initial solidus character; all additional arguments are collected together, to define the value to be associated with the specified key.

As is the case with the pdfview macro, (see section 2.2, "Selecting an Initial Document View"), the /DOCINFO records specified with the pdfinfo macro are not immediately written to the PostScript® output stream; they are stored in the same meta-data cache as /DOCVIEW specifications, until this cache is explicitly flushed, by invoking the pdfsync macro, (see section 2.7, "Synchronizing Output and pdfmark Contexts").

2.4. Creating a Document Outline

A PDF document outline comprises a table of references, to "bookmarked" locations within the document. When the document is viewed in an "outline aware" PDF document reader, such as Adobe® Acrobat® Reader, this table of "bookmarks" may be displayed in a document outline pane, or "Bookmarks" pane, to the left of the main document view. Individual references in the outline view may then be selected, by clicking with the mouse, to jump directly to the associated marked location in the document view.

The document outline may be considered as a collection of "hypertext" references to "bookmarked" locations within the document. The pdfmark macro package provides a single generalized macro, pdfhref, for creating and linking to "hypertext" reference marks. This macro will be described more comprehensively in a later section, (see section 2.5, "Adding Reference Marks and Links"); the description here is restricted to its use for defining document outline entries.

2.4.1. A Basic Document Outline

In its most basic form, the document outline comprises a structured list of headings, each associated with a marked location, or "bookmark", in the document text, and a specification for how that marked location should be displayed, when this bookmark is selected.

To create a PDF bookmark, the pdfhref macro is used, at the point in the document where the bookmark is to be placed, in the form

.pdfhref O *<level> descriptive text ...*

in which the reference class "**O**" stipulates that this is an outline reference.

Alternatively, for those users who may prefer to think of a document outline simply as a collection of bookmarks, the pdfbookmark macro is also provided — indeed, pdfhref invokes it, when processing the "**O**" reference class operator. It may be invoked directly, in the form

.pdfbookmark *<level> descriptive text ...*

Irrespective of which of the above macro forms is employed, the *<level>* argument is required. It is a numeric argument, defining the nesting level of the "bookmark" in the outline hierarchy, with one being the topmost level. Its function may be considered analogous to the *heading level* of the document's section headings, for example, as specified with the NH macro, if using the ms macros to format the document.

All further arguments, following the *<level>* argument, are collected together, to specify the heading text which will appear in the document's outline view. Thus, the outline entry for this section of this document, which has a level three heading, might be specified as

```
.pdfhref O 3 2.4.1. A Basic Document Outline
```

or, in the alternative form using the pdfbookmark macro, as

```
.pdfbookmark 3 2.4.1. A Basic Document Outline
```

2.4.2. Hierarchical Structure in a Document Outline

When a document outline is created, using the `pdfhref` macro as described in section 2.4.1, and any entry is added at a nesting level greater than one, then a hierarchical structure is automatically defined for the outline. However, as was noted in the simplified example in section 2.1, the data required by the `pdfmark` operator to create the outline entry may not be fully defined, when the outline reference is defined in the `groff` document source. Specifically, when the outline entry is created, its `/Count` key must be assigned a value equal to the number of its subordinate entries, at the next inner level of the outline hierarchy; typically however, these subordinate entries will be defined *later* in the document source, and the appropriate `/Count` value will be unknown, when defining the parent entry.

To resolve this paradox, the `pdfhref` macro creates the outline entry in two distinct phases — a destination marker is placed in the PostScript® output stream immediately, when the outline reference is defined, but the actual outline entry is stored in an internal "outline cache", until its subordinate hierarchy has been fully defined; it can then be inserted in the output stream, with its `/Count` value correctly assigned. Effectively, to ensure integrity of the document outline structure, this means that each top level outline entry, and *all* of its subordinates, are retained in the cache, until the *next* top level entry is defined.

One potential problem, which arises from the use of the "outline cache", is that, at the end of any document formatting run, the last top level outline entry, and any subordinates defined after it, will remain in the cache, and will *not* be automatically written to the output stream. To avoid this problem, the user should follow the guidelines given in section 2.7, to synchronize the output state with the cache state, (see section 2.7, "Synchronizing Output and `pdfmark` Contexts"), at the end of the `groff` formatting run.

2.4.3. Associating a Document View with an Outline Reference

Each "bookmark" entry, in a PDF document outline, is associated with a specific document view. When the reader selects any outline entry, the document view changes to display the document context associated with that entry.

The document view specification, to be associated with any document outline entry, is established at the time when the outline entry is created. However, rather than requiring that each individual use of the `pdhref` macro, to create an outline entry, should include its own view specification, the actual specification assigned to each entry is derived from a generalized specification defined in the string `PDFBOOKMARK.VIEW`, together with the setting of the numeric register `PDFHREF.VIEW.LEADING`, which determine the effective view specification as follows:–

`PDFBOOKMARK.VIEW`
: Establishes the magnification at which the document will be viewed, at the location of the "bookmark"; by default, it is defined by

```
.ds PDFBOOKMARK.VIEW /FitH \\n[PDFPAGE.Y] u
```

: which displays the associated document view, with the "bookmark" location positioned at the top of the display window, and with the magnification set to fit the page width to the width of the window.

`PDFHREF.VIEW.LEADING`
: Specifies additional spacing, to be placed between the top of the display window and the actual location of the "bookmark" on the displayed page view. By default, it is set as

```
.nr PDFHREF.VIEW.LEADING 5.0p
```

: Note that `PDFHREF.VIEW.LEADING` does not represent true "leading", in the typographical sense, since any preceding text, set in the specified display space, will be visible at the top of the document viewing window, when the reference is selected.

: Also note that the specification of `PDFHREF.VIEW.LEADING` is shared by *all* reference views defined by the `pdfhref` macro; whereas `PDFBOOKMARK.VIEW` is applied exclusively to outline references, there is no independent `PDFBOOKMARK.VIEW.LEADING` specification.

If desired, the view specification may be changed, by redefining the string `PDFBOOKMARK.VIEW`, and possibly also the numeric register `PDFHREF.VIEW.LEADING`. Any alternative definition for `PDFBOOKMARK.VIEW` *must* be specified in terms of valid view specification parameters, as described in the Adobe® "pdfmark Reference Manual".

Note the use of the register `PDFPAGE.Y`, in the default definition of `PDFBOOKMARK.VIEW` above. This register is computed by `pdfhref`, when creating an outline entry; it specifies the vertical position of the "bookmark", in basic `groff` units, relative to the *bottom* edge of the document page on which it is defined, and is followed, in the `PDFBOOKMARK.VIEW` definition, by the `grops` "`u`" operator, to convert it to PostScript® units on output. It may be used in any redefined specification for `PDFBOOKMARK.VIEW`, (or in the analogous definition of `PDFHREF.VIEW`,

described in section 2.5.2.2, "Associating a Document View with a Reference Mark"), but *not* in any other context, since its value is undefined outside the scope of the `pdfhref` macro.

Since `PDFPAGE.Y` is computed relative to the *bottom* of the PDF output page, it is important to ensure that the page length specified to `troff` correctly matches the size of the logical PDF page. This is most effectively ensured, by providing *identical* page size specifications to `groff`, `grops` and to the PostScript® to PDF converter employed, and avoiding any page length changes within the document source.

Also note that `PDFPAGE.Y` is the only automatically computed "bookmark" location parameter; if the user redefines `PDFBOOKMARK.VIEW`, and the modified view specification requires any other positional parameters, then the user *must* ensure that these are computed *before* invoking the `pdfhref` macro.

2.4.4. Folding the Outline to Conceal Less Significant Headings

When a document incorporates many subheadings, at deeply nested levels, it may be desirable to "fold" the outline such that only the major heading levels are initially visible, yet making the inferior subheadings accessible, by allowing the reader to expand the view of any heading branch on demand.

The `pdfmark` macros support this capability, through the setting of the `PDFOUTLINE.FOLDLEVEL` register. This register should be set to the number of heading levels which it is desired to show in expanded form, in the *initial* document outline display; all subheadings at deeper levels will still be added to the outline, but will not become visible until the outline branch containing them is expanded. For example, the setting used in this document:

```
.\" Initialize the outline view to show only three heading levels,
.\" with additional subordinate level headings folded.
.\"
.nr PDFOUTLINE.FOLDLEVEL 3
```

results in only the first three levels of headings being displayed in the document outline, *until* the reader chooses to expand the view, and so reveal the lower level headings in any outline branch.

The initial default setting of `PDFOUTLINE.FOLDLEVEL`, if the document author does not choose to change it, is 10,000. This is orders of magnitude greater than the maximum heading level which is likely to be used in any document; thus the default behaviour will be to show document outlines fully expanded, to display all headings defined, at all levels within each document.

The setting of `PDFOUTLINE.FOLDLEVEL` may be changed at any time; however, the effect of each such change may be difficult to predict, since it is applied not only to outline entries which are defined *after* the setting is changed, but also to any entries which remain in the outline cache, *at* this time. Therefore, it is recommended that `PDFOUTLINE.FOLDLEVEL` should be set *once*, at the start of each document; if it *is* deemed necessary to change it at any other time, the outline cache should be flushed, (see section 2.7, "Synchronizing Output and `pdfmark` Contexts"), *immediately* before the change, which should immediately precede a level one heading.

2.4.5. Outlines for Multipart Documents

When a document outline is created, using the `pdfhref` macro, each reference mark is automatically assigned a name, composed of a fixed stem followed by a serially generated numeric qualifier. This ensures that, for each single part document, every outline reference has a uniquely named destination.

As the overall size of the PDF document increases, it may become convenient to divide it into smaller, individually formatted PostScript® components, which are then assembled, in the appropriate order, to create a composite PDF document. While this strategy may simplify the overall process of creating and editing larger documents, it does introduce a problem in creating an overall document outline, since each individual PostScript® component will be assigned duplicated sequences of "bookmark" names, with each name ultimately referring to multiple locations in the composite document. To avoid such reference naming conflicts, the `pdfhref` macro allows the user to specify a "tag", which is appended to the automatically generated "bookmark" name; this may be used as a discriminating mark, to distinguish otherwise similarly named destinations, in different sections of the composite document.

To create a "tagged" document outline, the syntax for invocation of the `pdfhref` macro is modified, by the inclusion of an optional "tag" specification, *before* the nesting level argument, i.e.

`.pdfhref O` `[`**`-T`** *`<tag>`*`] <level> descriptive text ...`

The optional *`<tag>`* argument may be composed of any characters of the user's choice; however, its initial character *must not* be any decimal digit, and ideally it should be kept short — one or two characters at most.

By employing a different tag in each section, the user can ensure that "bookmark" names remain unique, throughout all the sections of a composite document. For example, when using the `spdf.tmac` macro package, which adds

pdfmark capabilities to the standard ms package, (see section 3.1, "Using pdfmark Macros with the ms Macro Package"), the table of contents is collected into a separate PostScript® section from the main body of the document. In the "body" section, the document outline is "untagged", but in the "Table of Contents" section, a modified version of the TC macro adds an outline entry for the start of the "Table of Contents", invoking the pdfhref macro as

```
.pdfhref O -T T 1 \\*[TOC]
```

to tag the associated outline destination name with the single character suffix, "T". Alternatively, as in the case of the basic outline, (see section 2.4.1, "A Basic Document Outline"), this may equally well be specified as

```
.pdfbookmark -T T 1 \\*[TOC]
```

2.4.6. Delegation of the Outline Definition

Since the most common use of a document outline is to provide a quick method of navigating through a document, using active "hypertext" links to chapter and section headings, it may be convenient to delegate the responsibility of creating the outline to a higher level macro, which is itself used to define and format the section headings. This approach has been adopted in the spdf.tmac package, to be described later, (see section 3.1, "Using pdfmark Macros with the ms Macro Package").

When such an approach is adopted, the user will rarely, if ever, invoke the pdfhref macro directly, to create a document outline. For example, the structure and content of the outline for this document has been exclusively defined, using a combination of the NH macro, from the ms package, to establish the structure, and the XN macro from spdf.tmac, to define the content. In this case, the responsibility for invoking the pdfhref macro, to create the document outline, is delegated to the XN macro.

2.5. Adding Reference Marks and Links

Section 2.4 has shown how the pdfhref macro may be used to create a PDF document outline. While this is undoubtedly a powerful capability, it is by no means the only trick in the repertoire of this versatile macro.

The macro name, pdfhref, which is a contraction of "PDF HyperText Reference", indicates that the general purpose of this macro is to define *any* type of dynamic reference mark, within a PDF document. Its generalized usage syntax takes the form

.pdfhref <***class***> [*-options ...*] [*--*] [*descriptive text ...*]

where <*class*> represents a required single character argument, which defines the specific reference operation to be performed, and may be selected from:–

- **O** Add an entry to the document outline. This operation has been described earlier, (see section 2.4, "Creating a Document Outline").
- **M** Place a "named destination" reference mark at the current output position, in the current PDF document, (see section 2.5.2, "Marking a Reference Destination").
- **D** Specify the content of a PDF document reference dictionary entry; typically, such entries are generated automatically, by transformation of the intermediate output resulting from the use of pdfhref "**M**", with the "**-X**" modifier, (see section 4.1.1, "Creating a Document Reference Map"); however, it is also possible to specify such entries manually, (see section 2.5.5.2, "Specifying Reference Text Explicitly").
- **L** Insert an active link to a named destination, (see section 2.5.3, "Linking to a Marked Reference Destination"), at the current output position in the current PDF document, such that when the reader clicks on the link text, the document view changes to show the location of the named destination.
- **W** Insert an active link to a "web" resource, (see section 2.5.4, "Linking to Internet Resources"), at the current output position in the current PDF document. This is effectively the same as using the "**L**" operator to establish a link to a named destination in another PDF document, (see section 2.5.3.2, "References to Destinations in Other PDF Documents"), except that in this case, the destination is specified by a "uniform resource identifier", or URI; this may represent any Internet or local resource which can be specified in this manner.
- **F** Specify a user defined macro, to be called by pdfhref, when formatting the text in the active region of a link, (see section 2.5.5, "Establishing a Format for References").
- **K** Define one or more location keywords, and associated format-string names, which should be interpreted by the pdfhref reference text formatting routine, (see section 2.5.5.4, "Customizing Automatically Formatted Reference Text").

Z Define the absolute position on the physical PDF output page, where the "hot-spot" associated with an active link is to be placed. Invoked in pairs, marking the starting and ending PDF page co-ordinates for each link "hot-spot", this operator is rarely, if ever, specified directly by the user; rather, appropriate `pdfhref` "**Z**" specifications are inserted automatically into the document reference map during the PDF document formatting process, (see section 4.1.1, "Creating a Document Reference Map").

I Initialize support for `pdfhref` features. The current `pdfhref` implementation provides only one such feature which requires initialization — a helper macro which must be attached to a user supplied page trap handler, in order to support mapping of reference "hot-spots" which extend through a page transition; (see section 2.5.6.1, "Links with a Page Transition in the Active Region").

2.5.1. Optional Features of the `pdfhref` Macro

The behaviour of a number of the `pdfhref` macro operations can be modified, by including "*option specifiers*" after the operation specifying argument, but *before* any other arguments normally associated with the operation. In *all* cases, an option is specified by an "*option flag*", comprising an initial hyphen, followed by one or two option identifying characters. Additionally, *some* options require *exactly one* option argument; for these options, the argument *must* be specified, and it *must* be separated from the preceding option flag by one or more *spaces*, (tabs *must not* be used). It may be noted that this paradigm for specifying options is reminiscent of most Unix™ shells; however, in the case of the `pdfhref` macro, omission of the space separating an option flag from its argument is *never* permitted.

A list of *all* general purpose options supported by the `pdfhref` macro is given below. Note that not all options are supported for all `pdfhref` operations; the operations affected by each option are noted in the list. For *most* operations, if an unsupported option is specified, it will be silently ignored; however, this behaviour should not be relied upon.

The general purpose options, supported by the `pdfhref` macro, are:–

-N <*name*>
Allows the <*name*> associated with a PDF reference destination to be defined independently from the following text, which describes the reference. This option affects only the "**M**" operation of the `pdfhref` macro, (see section 2.5.2, "Marking a Reference Destination").

-E Also used exclusively with the "**M**" operator, the **-E** option causes any specified *descriptive text* arguments, (see section 2.5.2, "Marking a Reference Destination"), to be copied, or *echoed*, in the body text of the document, at the point where the reference mark is defined; (without the **-E** option, such *descriptive text* will appear *only* at points where links to the reference mark are placed, and where the standard reference display format, (see section 2.5.5, "Establishing a Format for References"), is used).

-D <*dest*>
Specifies the `URI`, or the destination name associated with a PDF active link, independently of the following text, which describes the link and demarcates the link "hot-spot". This option affects the behaviour of the `pdfhref` macro's "**L**" and "**W**" operations.

When used with the "**L**" operator, the <*dest*> argument must specify a PDF "named destination", as defined using `pdfhref` with the "**M**" operator.

When used with the "**W**" operator, <*dest*> must specify a link destination in the form of a "uniform resource identifier", or `URI`, (see section 2.5.4, "Linking to Internet Resources").

-F <*file*>
When used with the "**L**" `pdfhref` operator, <*file*> specifies an external PDF file in which the named destination for the link reference is defined. This option *must* be specified with the "**L**" operator, to create a link to a destination in a different PDF document; when the "**L**" operator is used *without* this option, the link destination is assumed to be defined within the same document.

-P <*"prefix-text"*>
Specifies <*"prefix-text"*> to be attached to the *start* of the text describing an active PDF document link, with no intervening space, but without itself being included in the active area of the link "hot-spot"; it is effective with the "**L**" and "**W**" `pdfhref` operators.

Typically, this option would be used to insert punctuation before the link "hot-spot". Thus, there is little reason for the inclusion of spaces in <*"prefix-text"*>; however, if such space is required, then the enclosing double quotes *must* be specified, as indicated.

-A <*"affixed-text"*>

Specifies <*"affixed-text"*> to be attached to the *end* of the text describing an active PDF document link, with no intervening space, but without itself being included in the active area of the link "hot-spot"; it is effective with the "**L**" and "**W**" pdfhref operators.

Typically, this option would be used to insert punctuation after the link "hot-spot". Thus, there is little reason for the inclusion of spaces in <*"affixed-text"*>; however, if such space is required, then the enclosing double quotes *must* be specified, as indicated.

-T <***tag***>
When specified with the "**O**" operator, <*tag*> is appended to the "bookmark" name assigned to the generated outline entry. This option is *required*, to distinguish between the series of "bookmark" names generated in individual passes of the groff formatter, when the final PDF document is to be assembled from a number of separately formatted components; (see section 2.4.5, "Outlines for Multipart Documents").

-X This pdfhref option is used with either the "**M**" operator, or with the "**L**" operator.

When used with the "**M**" operator, (see section 2.5.2, "Marking a Reference Destination"), it ensures that a cross reference record for the marked destination will be included in the document reference map, (see section 2.5.2.1, "Mapping a Destination for Cross Referencing").

When used with the "**L**" operator, (see section 2.5.3, "Linking to a Marked Reference Destination"), it causes the reference to be displayed in the standard cross reference format, (see section 2.5.5, "Establishing a Format for References"), but substituting the *descriptive text* specified in the "pdfhref L" argument list, for the description specified in the document reference map.

-- Marks the end of the option specifiers. This may be used with all pdfhref operations which accept options, to prevent pdfhref from interpreting any following arguments as option specifiers, even if they would otherwise be interpreted as such. It is also useful when the argument list to pdfhref contains special characters — any special character, which is not valid in a groff macro name, will cause a parsing error, if pdfhref attempts to match it as a possible option flag; using the "--" flag prevents this, so suppressing the groff warning message, which would otherwise ensue.

Using this flag after *all* sequences of macro options is recommended, even when it is not strictly necessary, if only for the entirely cosmetic benefit of visually separating the main argument list from the sequence of preceding options.

In addition to the pdfhref options listed above, a supplementary set of two character options are defined. These supplementary options, listed below, are intended for use with the "**L**" operator, in conjunction with the **-F** <***file***> option, to specify alternate file names, in formats compatible with the file naming conventions of alternate operating systems; they will be silently ignored, if used in any other context.

The supported alternate file name options, which are ignored if the **-F** <***file***> option is not specified, are:–

-DF <***dos-file***>
Specifies the name of the file in which a link destination is defined, using the file naming semantics of the MS-DOS® operating system. When the PDF document is read on a machine where the operating system uses the MS-DOS® file system, then <*dos-file*> is used as the name of the file containing the reference destination, overriding the <*file*> argument specified with the **-F** option.

-MF <***mac-file***>
Specifies the name of the file in which a link destination is defined, using the file naming semantics of the Apple® Macintosh® operating system. When the PDF document is read on a machine where the operating system uses the Macintosh® file system, then <*mac-file*> is used as the name of the file containing the reference destination, overriding the <*file*> argument specified with the **-F** option.

-UF <***unix-file***>
Specifies the name of the file in which a link destination is defined, using the file naming semantics of the Unix™ operating system. When the PDF document is read on a machine where the operating system uses POSIX file naming semantics, then <*unix-file*> is used as the name of the file containing the reference destination, overriding the <*file*> argument specified with the **-F** option.

-WF <***win-file***>
Specifies the name of the file in which a link destination is defined, using the file naming semantics of the MS-Windows® 32-bit operating system. When the PDF document is read on a machine where the operating system uses any of the MS-Windows® file systems, with long file name support, then <*win-file*> is used as the name of the file containing the reference destination, overriding the <*file*> argument specified with the **-F** option.

2.5.2. Marking a Reference Destination

The pdfhref macro may be used to create active links to any Internet resource, specified by its URI, or to any "named destination", either within the same document, or in another PDF document. Although the PDF specification allows link destinations to be defined in terms of a page number, and an associated view specification, this style of reference is not currently supported by the pdfhref macro, because it is not possible to adequately bind the specification for the destination with the intended reference context.

References to Internet resources are interpreted in accordance with the W3C standard for defining a URI; hence the only prerequisite, for creating a link to any Internet resource, is that the URI be properly specified, when declaring the reference; (see section 2.5.4, "Linking to Internet Resources"). In the case of references to "named destinations" in PDF documents, however, it is necessary to provide a mechanism for creating such "named destinations". This may be accomplished, by invoking the pdfhref macro in the form

.pdfhref M [**-N** *<name>*] [**-X**] [**-E**] [*descriptive text ...*]

This creates a "named destination" reference mark, with its name specified by *<name>*, or, if the **-N** option is not specified, by the first word of *descriptive text*; (note that this imposes the restriction that, if the **-N** option is omitted, then *at least* one word of *descriptive text must* be specified). Additionally, a reference view will be automatically defined, and associated with the reference mark, (see section 2.5.2.2, "Associating a Document View with a Reference Mark"), and, if the **-X** option is specified, and no document cross reference map has been imported, (see section 4.1.2, "Deploying a Document Reference Map"), then a cross reference mapping record, (see section 2.5.2.1, "Mapping a Destination for Cross Referencing"), will be written to the stdout stream; this may be captured, and subsequently used to generate a cross reference map for the document, (see section 4.1.1, "Creating a Document Reference Map").

When a "named destination" reference mark is created, using the pdfhref macro's "**M**" operator, there is normally no visible effect in the formatted document; any *descriptive text* which is specified will simply be stored in the cross reference map, for use when a link to the reference mark is created. This default behaviour may be changed, by specifying the **-E** option, which causes any specified *descriptive text* to be "echoed" in the document text, at the point where the reference mark is placed, in addition to its inclusion in the cross reference map.

2.5.2.1. Mapping a Destination for Cross Referencing

Effective cross referencing of *any* document formatted by groff requires multiple pass formatting. Details of how this multiple pass formatting may be accomplished, when working with the pdfmark macros, will be discussed later, (see section 4.1, "Resolving Cross References"); at this stage, the discussion will be restricted to the initial preparation, which is required at the time when the cross reference destinations are defined.

The first stage, in the process of cross referencing a document, is the generation of a cross reference map. Again, the details of *how* the cross reference map is generated will be discussed in section 4.1; however, it is important to recognize that *what* content is included in the cross reference map is established when the reference destination is defined — it is derived from the reference data exported on the stderr stream by the pdfhref macro, when it is invoked with the "**M**" operator, and is controlled by whatever definition of the string PDFHREF.INFO is in effect, when the pdfhref macro is invoked.

The initial default setting of PDFHREF.INFO is

```
.ds PDFHREF.INFO page \\n% \\$*
```

which ensures that the cross reference map will contain at least a page number reference, supplemented by any *descriptive text* which is specified for the reference mark, as defined by the pdfhref macro, with its "**M**" operator; this may be redefined by the user, to export additional cross reference information, or to modify the default format for cross reference links, (see section 2.5.5, "Establishing a Format for References").

2.5.2.2. Associating a Document View with a Reference Mark

In the same manner as each document outline reference, defined by the pdfhref macro with the "**O**" operator, (see section 2.4, "Creating a Document Outline"), has a specific document view associated with it, each reference destination marked by pdfhref with the "**M**" operator, requires an associated document view specification.

The mechanism whereby a document view is associated with a reference mark is entirely analogous to that employed for outline references, (see section 2.4.3, "Associating a Document View with an Outline Reference"), except that the PDFHREF.VIEW string specification is used, in place of the PDFBOOKMARK.VIEW specification. Thus, the reference view is defined in terms of:–

PDFHREF.VIEW
A string, establishing the position of the reference mark within the viewing window, and the magnification at

which the document will be viewed, at the location of the marked reference destination; by default, it is defined by

```
.ds PDFHREF.VIEW /FitH \\n[PDFPAGE.Y] u
```

which displays the reference destination at the top of the viewing window, with the magnification set to fit the page width to the width of the window.

PDFHREF.VIEW.LEADING
A numeric register, specifying additional spacing, to be placed between the top of the display window and the actual position at which the location of the reference destination appears within the window. This register is shared with the view specification for outline references, and thus has the same default initial setting,

```
.nr PDFHREF.VIEW.LEADING 5.0p
```

as in the case of outline reference views.

Again, notice that PDFHREF.VIEW.LEADING does not represent true typographic "leading", since any preceding text, set in the specified display space, will be visible at the top of the viewing window, when the reference is selected.

Just as the view associated with outline references may be changed, by redefining PDFBOOKMARK.VIEW, so the view associated with marked reference destinations may be changed, by redefining PDFHREF.VIEW, and, if desired, PDFHREF.VIEW.LEADING; such changes will become effective for all reference destinations marked *after* these definitions are changed. (Notice that, since the specification of PDFHREF.VIEW.LEADING is shared by both outline reference views and marked reference views, if it is changed, then the views for *both* reference types are changed accordingly).

It may again be noted, that the PDFPAGE.Y register is used in the definition of PDFHREF.VIEW, just as it is in the definition of PDFBOOKMARK.VIEW; all comments in section 2.4.3 relating to its use, and indeed to page position computations in general, apply equally to marked reference views and to outline reference views.

2.5.3. Linking to a Marked Reference Destination

Any named destination, such as those marked by the pdfhref macro, using it's "**M**" operator, may be referred to from any point in *any* PDF document, using an *active link*; such active links are created by again using the pdfhref macro, but in this case, with the "**L**" operator. This operator provides support for two distinct cases, depending on whether the reference destination is defined in the same document as the link, (see section 2.5.3.1, "References within a Single PDF Document"), or is defined as a named destination in a different PDF document, (see section 2.5.3.2, "References to Destinations in Other PDF Documents").

2.5.3.1. References within a Single PDF Document

The general syntactic form for invoking the pdfhref macro, when creating a link to a named destination within the same PDF document is

```
.pdfhref L [-D <dest-name>] [-P <prefix-text>] [-A <affixed-text>] \
    [-X] [--] [descriptive text ...]
```

where <*dest-name*> specifies the name of the link destination, as specified using the pdfhref "**M**" operation; (it may be defined either earlier in the document, to create a backward reference, or later, to create a forward reference).

If any *descriptive text* arguments are specified, then they will be inserted into the groff output stream, to define the text appearing in the "hot-spot" region of the link; this will be printed in the link colour specified by the string, PDFHREF.TEXT.COLOUR, which is described in section 2.5.5.1, "Using Colour to Demarcate Link Regions". If the **-X** option is also specified, then the *descriptive text* will be augmented, by prefacing it with page and section number indicators, in accordance with the reference formatting rules which are in effect, (see section 2.5.5, "Establishing a Format for References"); such indicators will be included within the active link region, and will also be printed in the link colour.

Note that *either* the **-D** <***dest-name***> option, *or* the *descriptive text* arguments, *but not both*, may be omitted. If the **-D** <***dest-name***> option is omitted, then the first word of *descriptive text*, i.e. all text up to but not including the first space, will be interpreted as the <***dest-name***> for the link; this text will also appear in the running text of the document, within the active region of the link. Alternatively, if the **-D** <***dest-name***> option *is* specified, and *descriptive text* is not, then the running text which defines the reference, and its active region, will be derived from the reference description which is specified when the named destination is marked, (see section 2.5.2, "Marking a Reference Destination"), and will be formatted according to the reference formatting rules which are in

effect, when the reference is placed, (see section 2.5.5, "Establishing a Format for References"); in this case, it is not necessary to specify the **-X** option to activate automatic formatting of the reference — it is implied, by the omission of all *descriptive text* arguments.

The **-P** <***prefix-text***> and **-A** <***affixed-text***> options may be used to specify additional text which will be placed before and after the linked text respectively, with no intervening space. Such prefixed and affixed text will be printed in the normal text colour, and will not be included within the active region of the link. This feature is mostly useful for creating parenthetical references, or for placing punctuation adjacent to, but not included within, the text which defines the active region of the link.

The operation of the pdfhref macro, when used with its "**L**" operator to place a link to a named PDF destination, may best be illustrated by an example. However, since the appearance of the link will be influenced by factors established when the named destination is marked, (see section 2.5.2, "Marking a Reference Destination"), and also by the formatting rules in effect when the link is placed, the presentation of a suitable example will be deferred, until the formatting mechanism has been explained, (see section 2.5.5, "Establishing a Format for References").

2.5.3.2. References to Destinations in Other PDF Documents

The pdfhref macro's "**L**" operator is not restricted to creating reference links within a single PDF document. When the link destination is defined in a different document, then the syntactic form for invoking pdfhref is modified, by the addition of options to specify the name and location of the PDF file in which the destination is defined. Thus, the extended pdfhref syntactic form becomes

```
.pdfhref L -F <file> [-D <dest-name>] \
    [-DF <dos-file>] [-MF <mac-file>] [-UF <unix-file>] \
    [-WF <win-file>] [-P <prefix-text>] [-A <affixed-text>] \
    [-X] [--] [descriptive text ...]
```

where the **-F** <***file***> option serves *two* purposes: it both indicates to the pdfhref macro that the specified reference destination is defined in an external PDF file, and it also specifies the normal path name, which is to be used to locate this file, when a user selects the reference.

In addition to the **-F** <***file***> option, which *must* be specified when referring to a destination in an external PDF file, the **-DF** <***dos-file***>, **-MF** <***mac-file***>, **-UF** <***unix-file***> and **-WF** <***win-file***> options may be used to specify the location of the file containing the reference destination, in a variety of operating system dependent formats. These options assign their arguments to the /DosFile, /MacFile, /UnixFile and /WinFile keys of the generated pdfmark respectively; thus when any of these options are specified, *in addition to* the **-F** <***file***> option, and the document is read on the appropriate operating systems, then the path names specified by <***dos-file***>, <***mac-file***>, <***unix-file***> and <***win-file***> will be searched, *instead* of the path name specified by <***file***>, for each of the MS-DOS®, Apple® Macintosh®, Unix™ and MS-Windows® operating systems, respectively; see the "pdfmark Reference Manual", for further details.

Other than the use of these additional options, which specify that the reference destination is in an external PDF file, the behaviour of the pdfhref "**L**" operator, with the **-F** <***file***> option, remains identical to its behaviour *without* this option, (see section 2.5.3.1, "References within a Single PDF Document"), with respect to the interpretation of other options, the handling of the *descriptive text* arguments, and the formatting of the displayed reference.

Once again, since the appearance of the reference is determined by factors specified in the document reference map, and also by the formatting rules in effect when the reference is placed, the presentation of an example of the placing of a reference to an external destination will be deferred, until the formatting mechanism has been explained, (see section 2.5.5, "Establishing a Format for References").

2.5.4. Linking to Internet Resources

In addition to supporting the creation of cross references to named destinations in PDF documents, the pdfhref macro also has the capability to create active links to Internet resources, or indeed to *any* resource which may be specified by a Uniform Resource Identifier, (which is usually abbreviated to the acronym "URI", and sometimes also referred to as a Uniform Resource Locator, or "URL").

Since the mechanism for creating a link to a URI differs somewhat from that for creating PDF references, the `pdfhref` macro is invoked with the "**W**" (for "web-link") operator, rather than the "**L**" operator; nevertheless, the invocation syntax is similar, having the form

.pdfhref W [**-D** <***URI***>] [**-P** <***prefix-text***>] [**-A** <***affixed-text***>] \
    [**--**] *descriptive text ...*

where the optional **-D** <***URI***> modifier specifies the address for the target Internet resource, in any appropriate *Uniform Resource Identifier* format, while the *descriptive text* argument specifies the text which is to appear in the "hot-spot" region, and the **-P** <***prefix-text***> and **-A** <***affixed-text***> options have the same effect as in the case of local document links, (see section 2.5.3.1, "References within a Single PDF Document").

Notice that it is not mandatory to include the **-D** <***URI***> in the link specification; if it *is* specified, then it is not necessary for the URI to appear, in the running text of the document — the *descriptive text* argument exactly defines the text which will appear within the "hot-spot" region, and this need not include the URI. However, if the **-D** <***URI***> specification is omitted, then the *descriptive text* argument *must* be an *exact* representation of the URI, which *will*, therefore, appear as the entire content of the "hot-spot". For example, we could introduce a reference to the groff web site, in which the actual URI is concealed, by using mark up such as:–

```
For example, we could introduce a reference to
.pdfhref W -D http://www.gnu.org/software/groff -A , the groff web site
in which the actual URI is concealed,
```

Alternatively, to refer the reader to the groff web site, making it obvious that the appropriate URI is http://www.gnu.org/software/groff, the requisite mark up might be:–

```
to refer the reader to the groff web site,
making it obvious that the appropriate URI is
.pdfhref W -A , http://www.gnu.org/software/groff
the requisite mark up might be:\(en
```

2.5.5. Establishing a Format for References

There are two principal aspects to be addressed, when defining the format to be used when displaying references. Firstly, it is desirable to provide a visual cue, to indicate that the text describing the reference is imbued with special properties — it is dynamically linked to the reference destination — and secondly, the textual content should describe where the link leads, and ideally, it should also describe the content of the reference destination.

The visual cue, that a text region defines a dynamically linked reference, is most commonly provided by printing the text within the active region in a distinctive colour. This technique will be employed automatically by the `pdfhref` macro — see section 2.5.5.1, "Using Colour to Demarcate Link Regions" — unless the user specifically chooses to adopt, and implement, some alternative strategy.

2.5.5.1. Using Colour to Demarcate Link Regions

Typically, when a PDF document contains *active* references to other locations, either within the same document, or even in other documents, or on the World Wide Web, it is usually desirable to make the regions where these active links are placed stand out from the surrounding text.

The mechanism, which is apparently advocated by Adobe®, as the default for indicating any active link region, is to draw a coloured border around the region. This is a most unfortunate default choice: not only does it look hideously ugly, but it also seems very distracting to the reader! Consequently, while it does support this mechanism for link visualization, `groff`'s `pdfmark` macros disable it, by default; it is controlled by a pair of strings:–

PDFHREF.BORDER
: This string comprises a space-separated triplet of numeric values, optionally followed by a further space-separated `pdfmark array`, (see the Adobe® "pdfmark Reference Manual" for details), which together specify the link border style, in terms of its elliptical corner horizontal radius, vertical radius, line thickness, and line style mark-to-space ratio array; by default, it is defined as

  ```
  .ds PDFHREF.BORDER 0 0 0
  ```

  which has the effect of specifying an invisible link border, (a solid zero-width line, with rectanuglar corners),

thus appearing to disable the use of borders for link visualization. This differs from the Adobe® default, which represents a solid (visible) line, one pixel in width, and with rectangular corners; this Adobe® default may be reinstated, by explicitly defining

```
.ds PDFHREF.BORDER 0 0 1
```

before specifying any link references, which it is desired to have rendered in the Adobe® style.

PDFHREF.COLOUR

This string[7] comprises a triplet of space-separated decimal numeric values, each in the range 0.0..1.0; together, they represent, in RGB colour space, the colour in which link borders should be rendered, in the event that the PDFHREF.BORDER property is specified to make them visible; by default, it is defined as

```
.ds PDFHREF.COLOUR 0.35 0.00 0.60
```

which represents a deep lilac colour.

While the foregoing discussion of PDFHREF.BORDER, and PDFHREF.COLOUR, may seem sufficient for those users who are willing to adopt the Adobe® convention of drawing a border to offset links from the surrounding text, it is *not* the preferred way of doing so, in groff's pdfmark implementation. Given the perceived ugliness of the Adobe® convention, the preferred technique for visualizing links is to disable the rendition of the link border, (by making it invisible, as groff's pdfmark implementation does by default), and to simply print the text, within the link "hot-spot" region, in a colour which contrasts with that of the surrounding text.

It may noted that, whereas the preceding PDFHREF.BORDER, and PDFHREF.COLOUR properties exert their influence within the Adobe® pdfmark infrastructure, that infrastructure provides no mechanism for control of the text colour within a link "hot-spot" region; however, the desired effect may be readily achieved, simply by assignment of groff colour properties. In groff's pdfmark implementation, the text colour, for use within link "hot-spot" regions, is established by a further string assignment, viz.:–

PDFHREF.TEXT.COLOUR

Specifies the text colour, for rendition of PDF reference links.

Unlike PDFHREF.COLOUR, this string[8] must be assigned a value which represents a groff colour name, rather than an RGB colour-space triplet; by default, it is assigned the name of a custom colour, which is internally derived from, and is thus chromatically identical to the deep lilac colour,[9] as represented by the default RGB colour-space triplet which is specified as the default value of PDFHREF.COLOUR.

2.5.5.2. Specifying Reference Text Explicitly

Although the use of colour within, and/or borders around, pdfhref link "hot-spot" regions may be considered to be a necessary visual indication of the location of such "hot-spots", for users of on-screen" PDF readers, such visual indicators alone are insufficient to convey any necessary information regarding the context to which the link refers; neither do they offer any particular benefit to readers of documents in printed hard-copy formats. To address these limitations, it is necessary to specify appropriate text within each "hot-spot" region, to identify the link context.

Depending on the type of contextual information, which it is desired to include within any link "hot-spot" region, groff's pdfmark macro suite provides a variety of mechanisms to specify it; the simplest of these is to simply specify the desired text *explicitly*, at the point of insertion of the reference. For example, given that the "use of colour" reference, in the initial paragraph of this section, points to a destination named by mark up similar to:–

```
.pfdhref M -X -N set-colour -- ...
```

the reference text was specified explicitly, (ignoring recorded location information), using the mark up:–

```
.pdfhref L -D set-colour -- use of colour
```

---

7. For authors who may prefer American English spelling, PDFHREF.COLOR will be recognized as an alias for PDFHREF.COLOUR. However, should the alias be broken, (by deletion of either of the alternative names, prior to redefining it), it is the World English spelling, PDFHREF.COLOUR, which will be honoured when rendering links.

8. Just as PDFHREF.COLOR is defined as an alias for PDFHREF.COLOUR, the alias PDFHREF.TEXT.COLOR may be used as an American English spelling alternative to World English PDFHREF.TEXT.COLOUR; once again, should the alias be broken, the World English spelling will prevail.

9. This deep lilac colour has been chosen on the basis that it will provide sufficient contrast, when the PDF document is viewed on a colour display screen, to be discernable by readers with normal colour perception, but not so much contrast as to be distracting; conversely, if the document is printed on a monochrome hard-copy device, since links cannot then be clicked, it is anticipated that the contrast will be barely discernable, if at all.

2.5.5.3. Using Automatically Formatted Reference Text

When the text within a link "hot-spot" is specified explicitly, using a `pdfhref` macro call of the form

```
.pdfhref L -D <dest-name> -- <explicit-text>
```

as described in the preceding section, then `<explicit-text>` will appear in the formatted document, *exactly* as specified. This may be the author's intent, but it does suffer from the disadvantage that, in spite of location information having been recorded when `<dest-name>` was marked, none is included in `<explicit-text>`, *unless* the author *explicitly* includes it; this places the onus on the author, if inclusion of such location information is desired, to track it *manually*, and to specify it within `<explicit-text>`, in the desired format.

To mitigate this limitation, of *explicitly* specified reference text, `groff`'s `pdfmark` macro suite provides a capability for *automatic* formatting of reference text, based on the content of a "`PDFHREF.INFO` record",[10] which is generated by, and is specific to any named link-destination marked by a `pdfhref` macro request of the form

```
.pdfhref M -X -N <dest-name> [[--] <default-text>]
```

When a `pdfhref` link destination has been marked by a macro request of this form,[11] a subsequent request of the simplified form

```
.pdfhref L -D <dest-name>
```

(excluding *any* `<explicit-text>` specification), then the reference text will be generated *automatically*, by passing the content of the "`PDFHREF.INFO` record" associated with `<dest-name>`, as arguments to a designated[12] `pdfhref` reference text formatting macro.

To illustrate this capability, if we revisit the example offered in section 2.5.5.2, "Specifying Reference Text Explicitly", but instead of specifying the reference as

```
.pdfhref L -D set-colour -- use of colour
```

we simply specify it, *without* the explict reference text arguments, as

```
.pdfhref L -D set-colour
```

and if neither `PDFHREF.INFO`, nor the designated `pdfhref` reference text formatting macro, have been changed from their original default settings, then we should see a reference formatted as

> see page 12, "Using Colour to Demarcate Link Regions"

Alternatively, location data from the "`PDFHREF.INFO` record" may be combined with explicitly specified text, by adding the **`-X`** option to the explicit form of the `pdfhref` macro request, e.g.

```
.pdfhref L -X -D set-colour -- Using Colour ...
```

will cause the reference to be displayed as

> see page 12, Using Colour ...

Notice that, when the displayed form of the reference incorporates the assigned `<default-text>`, as derived from the "`PDFHREF.INFO` record", this is enclosed in double quotation marks, but explicitly specified text is not; if quotation of explicitly specified text is desired, then appropriate quotation marks should simply be included within the `<explicit-text>` arguments specification.

Finally, to conclude this introduction to the automatic reference text formatting capabilities of `groff`'s `pdfmark` macro suite, it may be noted that, while the default provisions may be adequate, in many cases, this will not always be so. In the event of these default provisions being inadequate, customization is readily supported, as explained in the following section.

---

10. The conceptual nomenclature "`PDFHREF.INFO` record" has been adopted here, since the content of the record is dictated by the definition of the `PDFHREF.INFO` string, as described in section 2.5.2.1, "Mapping a Destination for Cross Referencing".

11. The specification of the **`-X`** option is imperative, within this `pdfhref` macro request; without it, no "`PDFHREF.INFO` record" will be generated.

12. A suitable `pdfhref` reference text formatting macro is provided, within `groff`'s `pdfmark` macro suite; it will be used by default, unless the author has designated an alternative, as described in section 2.5.5.4, "Customizing Automatically Formatted Reference Text".

2.5.5.4. Customizing Automatically Formatted Reference Text

"Automatically formatted" reference text is interpolated, within the running text of a published PDF document, when the `pdfhref` macro is invoked with its "**`L`**" operator, to refer to a destination named by the "**`-D`**" option, *and either*:–

- no explicit reference text is specified, (in which case, specification of the "**`-X`**" option is implied), *or*,
- the "**`-X`**" option is specified, in conjunction with explicit reference text.

In each of these cases, the reference text is derived, (in its entirety, in the first case, or partially, in the second), from a reference dictionary record for the named destination.

The reference dictionary, itself, comprises a collection of `PDFHREF.INFO` records, one per destination, indexed by destination name. While it is possible to create a reference dictionary record manually, using a macro call of the form:–

`.pdfhref D -N <dest-name> [<keyword> <value>] ... <text> ...`

(in which the in-order aggregate of all specified `<keyword> <value>` pairs, and all following `<text>` arguments, comprises the `PDFHREF.INFO` record, and `<dest-name>` represents the destination name on which it is indexed), it is generally more convenient to have the dictionary compiled automatically, by specifying the "**`-X`**" option, when using a macro call of the form:–

`.pdfhref M -X -N <dest-name> -- <text> ...`

to mark the location of each named destination; the procedure is described, in detail, in section 4.1.1, "Creating a Document Reference Map".

Interpolation of automatically formatted reference text is delegated to a specialized formatting macro, which assumes responsibility for storing the formatted representation of the reference text, into the `PDFHREF.TEXT` string; this formatting macro may be defined by the user, (if specialized formatting is required), or, in most cases, a standardized default macro, provided by `pdfmark.tmac`, may be used. In either case, the content of the `PDFHREF.INFO` record, which is associated with the named reference destination, will be passed to the formatting macro as a sequence of macro arguments, while any explicit reference text, which has been specified in the initiating "`.pdfhref L ...`" call, will be passed in the `PDFHREF.DESC` string; (if no reference text is explicitly specified, then any pre-existing definition of `PDFHREF.DESC` is explicitly deleted, prior to calling the formatting macro).

When the `pdfmark.tmac` default formatting macro is used, formatting progresses as follows:–

1. `PDFHREF.TEXT` is initialized to the content of the `PDFHREF.PREFIX` string; this has a default value of "see", but may be redefined by the user, to any suitable alternative content, (including the empty string, if desired).
2. The `PDFHREF.INFO` record, as passed in the argument list, is inspected to determine whether the first argument (`\$1`) matches any of the known formatting keywords, or otherwise it, and any additional arguments which follow it, will be interpreted as representing the text of an implicit reference description.
3. If `\$1` *does* match any of the known formatting keywords, the argument which follows (`\$2`) is interpreted as the `<value>`, (which completes a `<keyword> <value>` pair); `\$2` is interpolated into the format string which is associated with keyword `\$1`, and the result is appended to `PDFHREF.TEXT`. The matched `\$1`, and accompanying `\$2`, are then shifted out of the argument list, promoting `\$3`, (if any further arguments are present), to the `\$1` position, and the keyword matching process is repeated, from step 2.
4. If, during *any* execution cycle of step 2, `\$1` is found *not* to match any known formatting keyword, *and* `PDFHREF.DESC` has *not* been assigned any explicit content, then any remaining arguments are assigned to `PDFHREF.DESC`; thus, `PDFHREF.DESC` becomes the descriptive component of the reference text, either as explicitly specified by the originating "`.pdfhref L ...`" call, or implicitly deduced from the reference dictionary `PDFHREF.INFO` record for the named link destination.
5. Finally, an interpolating reference to `PDFHREF.DESC` is appended to `PDFHREF.TEXT`, from which any accumulated initial spaces are then removed, and the resultant `PDFHREF.TEXT` string is handed back to the originating "`.pdfhref L ...`" call, for interpolation as the formatted content within, and which defines the extent of, the link "hot-spot" region.

From the foregoing, it may be deduced that reference text, formatted by the default formatting macro, will commence with the content of the `PDFHREF.PREFIX` string, followed by the result of interpolation of any `keyword/value` pairs found in the `PDFHREF.INFO` record component of the relevant reference dictionary entry, and ends with a `PDFHREF.DESC` component, either as implicitly defined within that same `PDFHREF.INFO` record, or as explicitly specified as final arguments to "`.pdfhref L ...`". Within the formatted text, the keyword/value pair interpolations

appear in the order in which "known" keywords are found, while parsing the PDFHREF.INFO record; the default set of known formatting keywords comprises:–

Keyword	Format Name	Default Format
page	PDFHREF.PAGEREF	page \\$1,
section	PDFHREF.SECTREF	section \\$1,
file	PDFHREF.FILEREF	\\$1

The format of any of these known keyword interpolations may be customized, by redefinition of their corresponding "Format Name" strings; each may incorporate any text of the user's choice — inclusion of the keyword itself is *not* necessary, however, inclusion of the \\$1 placeholder, while not mandatory, *is* a necessary prerequisite for interpolation of the value component of the keyword/value pair, from the PDFHREF.INFO record.

In the case of any PDFHREF.INFO record which originates from a ".pdfhref M -X ..." call, the precise gamut of keyword interpolations which do, and the order in which they will, appear within automatically formatted reference text, may be manipulated by redefinition of the PDFHREF.INFO string itself. For example, with the default PDFHREF.PREFIX, PDFHREF.PAGEREF, PDFHREF.SECTREF, and PDFHREF.INFO definitions:–

```
.ds PDFHREF.PREFIX see
.ds PDFHREF.PAGEREF page \\$1,
.ds PDFHREF.SECTREF section \\$1,
.ds PDFHREF.INFO page \\n% \\$*
```

a request such as

```
.pdfhref L -D set-colour
```

may, (as indicated in section 2.5.5.3, "Using Automatically Formatted Reference Text"), result in formatted reference text similar to:–

see page 12, "Using Colour to Demarcate Link Regions"

whereas, a simple redefinition of PDFHREF.INFO, *before* the "set-colour" destination is marked:–

```
.ds PDFHREF.INFO section \\*[SN-NO-DOT] \\$*
```

may[13] result in alternative formatting similar to:–

see section 2.5.5.1, "Using Colour to Demarcate Link Regions"

(as is used in this document itself).

In addition to these default formatting capabilities, pdfmark.tmac also offers support[14] for interpolation of user-defined keyword/value pairs; these may be defined, using a macro call of the form:–

```
.pdfhref K <keyword> <format-name> [<keyword> <format-name>] ...
```

accompanied by string definitions, similar to that for PDFHREF.PAGEREF, for each format-name argument specified; keyword/value pairs, corresponding to such user-defined keywords, may be incorporated into the PDFHREF.INFO record definition, and they will be interpreted by the default formatting macro, in the same manner as the default set of keywords. For example, we might wish to add add a chapter reference capability. We *could* accomplish this by subverting the effect of one the default keywords;[15] however, as a (possibly undesirable) side effect of such a customization, we would lose the normal behaviour of the selected default keyword, while also introducing an element of obfuscation around the use of that keyword; we may prefer not to do this.

If we wish to avoid subversion of any default keyword, with the attendant obfuscation of intent for the chosen keyword, (and we have a sufficiently recent version of pdfmark.tmac), then the preferred method for implementing a custom

13. Assuming that the SN-NO-DOT string represents the effective section number, at the point where the link destination is marked, as it does when formatting with the ms macros provided with groff-1.19.2 and later.

14. Available only in versions of pdfmark.tmac as distributed with groff from groff-1.23.0 onwards.

15. We might choose to implement the effect of a chapter keyword by subverting the default behaviour of, e.g. the section keyword; we could achieve this by redefining the associated PDFHREF.SECTREF format string, in conjunction with a modified PDFHREF.INFO template:–

```
.ds PDFHREF.SECTREF chapter \\$1,
.ds PDFHREF.INFO section \\n[chapter] page \\n% \\$*
```

and, (assuming for the purpose of this example, that the chapter number matches the current top-level section heading number),

```
.pdfhref L -D set-colour
```

would be expected to yield a reference similar to:–

see chapter 2, page 12, "Using Colour to Demarcate Link Regions"

keyword feature, such as automatic interpretation of a `chapter` reference keyword, would be to make use of the "`.pdfhref K ...`" capability; e.g.:–

```
.pdfhref K chapter PDFHREF.CHAPTER
.ds PDFHREF.CHAPTER chapter \\$1,
```

With these definitions in place, and assuming that, at the point where the named destination is marked, the effective chapter number is made available in the `\n[CH]` numeric register, and also that the effective `PDFHREF.INFO` definition has been preset to:–

```
.ds PDFHREF.INFO chapter \\n[CH] page \\n% section \\*[SN-NO-DOT] \\$*
```

(assuming that `SN-NO-DOT` represents a section number, as it does when the `-ms` macros are used for document formatting), the example from the preceding section, viz.:–

```
.pdfhref L -D set-colour
```

will now produce a reference similar to

see chapter 2, page 12, section 2.5.5.1, "Using Colour to Demarcate Link Regions"

Finally, in the event that the default reference text formatting macro, combined with any user-defined `PDFHREF.INFO` specification, user-defined keyword-specific format strings, and combination of default or user-defined keywords, is insufficient to achieve a required formatting effect, the "`.pdfhref F [<`*macro-name*`>]`" facility allows the user to define an alternative formatting macro, and substitute it in place of the default. For example, within this document itself, some internal references are displayed as a section number reference alone; such references are derived from the associated `PDFHREF.INFO` record, but are formatted by the document-local `SECREF` macro:–

```
.de SECREF
.   while \\n(.$ \{\
.      ie '\\$1'section' \{\
.            if !dSECREF.BEGIN .ds SECREF.BEGIN \\$1
.            ds PDFHREF.TEXT \\*[SECREF.BEGIN]\~\\$2
.            rm SECREF.BEGIN
.            shift \\n(.$
.         \}
.      el \{\
.            shift
.            if \\n(.$ shift
.         \}
.      \}
..
```

to filter all but the "section" reference out of the `PDFHREF.INFO` record, which is then displayed as the reference text; used thus:–

```
.pdfhref F SECREF
.pdfhref L -D <reference-name>
.pdfhref F
```

it will emit reference text similar to:–

section 2.5.5.4

while, when used with the additional qualifying definition of `SECREF.BEGIN`:–

```
.pdfhref F SECREF
.ds SECREF.BEGIN Section
.pdfhref L -D <reference-name>
.pdfhref F
```

it will capitalize the emitted reference text, such that it becomes suitable for use at the beginning of a sentence:–

Section 2.5.5.4

Notice that the preceding `SECREF` macro exhibits *identical* semantics to those of the default reference formatting macro, as described above, (and as *any* user-defined reference formatting macro *must*), insofar as it expects to be passed the content of a `PDFHREF.INFO` record as its arguments, and it returns the formatted reference text as the definition of

the PDFHREF.TEXT string; however, while the PDFHREF.DESC, PDFHREF.PREFIX, PDFHREF.PAGEREF, PDFHREF.SECTREF, and PDFHREF.FILEREF strings (and any other custom format strings which the user may have defined) remain available, the SECREF macro simply ignores them.

Further note that the effect of invoking "`.pdfhref F <macro-name>`" is persistent; if it is desired to revert to use of the default reference formatting macro, after temporary use of a user-defined alternative, this may be accomplished by invoking "`.pdfhref F`" *without* specifying any "`<macro-name>`" argument, as shown in each of the two preceding usage examples.

2.5.6. Problematic Links

Irrespective of whether a pdfhref reference is placed using the "**L**" operator, or the "**W**" operator, there may be occasions when the resulting link does not function as expected. A number of scenarios, which are known to be troublesome, are described below.

2.5.6.1. Links with a Page Transition in the Active Region

When a link is placed near the bottom of a page, it is possible that its active region, or "hot-spot", may extend on to the next page. In this situation, a page trap macro is required to intercept the page transition, and to restart the mapping of the "hot-spot" boundary on the new page.

The pdfmark macro package includes a suitable page trap macro, to satisfy this requirement. However, to avoid pre-empting any other requirement the user may have for a page transition trap, this is *not* installed as an active page trap, unless explicitly requested by the user.

To enable proper handling of page transitions, which occur within the active regions of reference links, the user should:–

1. Define a page transition macro, to provide whatever features may be required, when a page transition occurs — e.g. printing footnotes, adding page footers and headers, etc. This macro should end by setting the output position at the correct vertical page offset, where the printing of running text is to restart, following the page transition.
2. Plant a trap to invoke this macro, at the appropriate vertical position marking the end of normal running text on each page.
3. Initialize the pdfhref hook into this page transition trap, by invoking

   `.pdfhref I -PT <macro-name>`

   where **`<macro-name>`** is the name of the user supplied page trap macro, to ensure that pdfhref will correctly restart mapping of active link regions, at the start of each new page.

It may be observed that this initialization of the pdfhref page transition hook is, typically, required only once *before* document formatting begins. Users of document formatting macro packages may reasonably expect that this initialization should be performed by the macro package itself. Thus, writers of such macro packages which include pdfmark bindings, should provide appropriate initialization, so relieving the end user of this responsibility. The following example, abstracted from the sample ms binding package, spdf.tmac, illustrates how this may be accomplished:–

```
.\" groff "ms" provides the "pg@bottom" macro, which has already
.\" been installed as a page transition trap.  To ensure proper
.\" mapping of "pdfhref" links which overflow the bottom of any
.\" page, we need to install the "pdfhref" page transition hook,
.\" as an addendum to this macro.
.
.pdfhref I -PT pg@bottom
```

2.6. Annotating a PDF Document using Pop-Up Notes

The Adobe® PDF specification defines several types of annotation, which may be associated with a PDF document; of these defined annotation types, *two* are *explicitly* supported by groff's pdfmark macros. Of these, although it is not explicitly identified as such, in the preceding discussion, it is the "Link" annotation type which underpins the operation of the pdfhref macro, as it is extensively described in section 2.5, "Adding Reference Marks and Links".

In addition to supporting the "Link" annotation type, through the use of the pdfhref macro, (see section 2.5, "Adding Reference Marks and Links"), the pdfmark macros offer support for the "Text" annotation type; primarily

useful as a means of adding editorial comments, this creates an annotation similar to a “sticky note”, attached to the document page, and represented by an icon, at the attachment point, which, when clicked, opens the annotation itself, in a pop-up window.

It may be noted that some — but not all[16] — PDF viewer applications may provide support for adding, and editing “`Text`” annotations. While such support, within a viewer application, may be convenient for 3rd-party editorial annotation, it may not be the most convenient method for the document author, should he, or she, wish to insert such annotations at the point of document origin. Thus, the `pdfmark` macros provide the `pdfnote` macro, for direct insertion of “`Text`” annotations, such as this,  created as in the following example:[17]

```
Thus, the
.CW pdfmark
macros provide the
.CW pdfnote
macro, for direct insertion of
.CW Text \(rq \(lq
annotations,
such as this,
.pdfnote -T "An Example Text Annotation" -PD 1 \# continued/...
Please do not move, modify, or remove this note; doing \#
so may invalidate the example to which it refers.\#
\*[PDFNOTE.PILCROW]\#
This is an illustration of an editorial comment, \#
placed directly by the document author, \#
using the exact markup as specified in \#
the example of the usage of the pdfnote macro, \# .../continuation ends
which immediately follows the note's icon.\"          here.
\h'5n'created as in the following example:
```

In addition to illustrating the technique for spreading the `pdfnote` text content over several *input* lines, this example of `pdfnote` usage gratuituously introduces some of the available options for setting `pdfnote` attributes, and the `\*[PDFNOTE.PILCROW]` control, for manipulation of text layout within the `pdfnote` pop-up window; further discussion of these may be found below, in section 2.6.2, “Options for Manipulating `pdfnote` Annotation Attributes”, and section 2.6.3, “Controlling `pdfnote` Text Layout”, respectively.

2.6.1. Controlling `pdfnote` Icon Placement

The placement of each `pdfnote` annotation, on its respective document page, is determined from its `Rect` attribute; (this is a *required* attribute, comprising an array of *four* numeric values, representing, in order, the *lower left x*, *lower left y*, *upper right x*, and *upper right y* co-ordinates of the page region in which the `pdfnote` annotation is to be placed). The `pdfnote` macro computes these four co-ordinate values, relative to the current text *output* position on the page, and specifies the required `Rect` attribute accordingly, in terms of the following numeric register, and string assignments:

`PDFNOTE.OFFSET`

A string, defined such that it may be evaluated as a numeric expression; its evaluation is interpreted as the *lower left x* ordinate, (and hence, implicitly, the *upper left x* ordinate), of the `pdfnote` placement region. By default, it is defined as

```
.ds PDFNOTE.OFFSET  "\\n[.k]+\\n[.o]+\\n[.in]\"
```

which, when evaluated, will result in placement of the *left edge* of the `pdfnote` region *immediately* to the *right* of the last running text glyph written to the output stream.

---

16. Indeed, Adobe’s own Acrobat Reader™ application may be found wanting, in this respect.

17. It may be noted that the *entire* content of any `pdfnote` *must* be entered as a *single logical* input line; this may be achieved, most effectively, and *without* necessitating an excessively long, and unwieldy, *physical* input line, by folding the `.pdfnote` call over multiple input lines, with each, *excluding* the last, terminated by a line continuation escape, (either a single “\” escape at the bitter end of each line, or a escape, followed by an optional comment).

    Furthermore, note that the continuation of the running text, following interpolation of the `.pdfnote` in this example, commences with an “`\h'5n'`” escape; this to leave sufficient space for the placement of the icon, associated with the `pdfnote`, *without* occlusion of the initial few glyphs of this continued running text.

Users may redefine PDFNOTE.OFFSET, to achieve a different left edge placement for any pdfnote annotations which follow; for example, the definition

```
.ds PDFNOTE.OFFSET  "\\n[.o]-\\n[PDFNOTE.WIDTH]-1m
```

will place pdfnote annotations into the left hand page margin, with 1em separation from the running text, as in this example:

```
.pdfnote -T "Marginal Placement Example" \
This note illustrates placement of pdfnote annotations \
in the left hand page margin, following redefinition of \
the PDFNOTE.OFFSET string.\
\*[PDFNOTE.PILCROW]\
As in the case of the previous pdfnote example, \
moving, modifying, or removing this annotation may \
invalidate the example to which it refers; please \
do not do this!
```

PDFNOTE.LEADING

The value of this numeric register is added to the value retrieved by invocation of the .mk request, to establish the vertical distance, from the top of the current document page, at which the top edge of each pdfnote icon is to be placed. By default, it is defined with a value of 0.3v, which will result in placement of pdfnote icons at 30% of the line spacing, below the *top* of the output line which is currently being composed, at the insertion point of each pdfnote annotation. This may be redefined by the user; positive values will push the icons further down the page, while negative values will pull them upwards, towards the top of the page.

PDFNOTE.HEIGHT

Combination of the effects of PDFNOTE.OFFSET and PDFNOTE.LEADING serves to specify the (x,y) page co-ordinates of the *upper left* vertex of the placement region for a pdfnote annotation; the value of the PDFNOTE.HEIGHT numeric register is added to the y ordinate of this upper left co-ordinate pair, to determine the corresponding *lower left* (x,y) co-ordinate pair, which is *required* for the specification of the Rect attribute of the pdfnote annotation pdfmark.

The default value specified for PDFNOTE.HEIGHT is 9 millimetres; this corresponds, approximately, to the height of "Text" annotation icons in many PDF viewer applications. The user *may* choose to define an alternative value; however, the usefulness of doing so may be questionable.

PDFNOTE.WIDTH

As in the case of addition of the value of PDFNOTE.HEIGHT to the y ordinate of the upper left pdfnote placement co-ordinate pair, to compute the *lower* left co-ordinate pair, the value of the PDFNOTE.WIDTH numeric register is added to the upper left x ordinate, to compute the corresponding *upper right* (x,y) co-ordinate pair; this is *required* to complete the Rect attribute specification for the annotation pdfmark.

The default value specified for PDFNOTE.WIDTH is 8 millimetres; this corresponds, approximately, to the width of "Text" annotation icons in many PDF viewer applications. The user *may* choose to define an alternative value; however, as in the case of PDFNOTE.HEIGHT, the usefulness of such an alternative definition may be questionable.

It may be worthy of note that the Adobe® PDF Specification is rather vague, with respect to how the Rect attribute of "Text" annotations should be interpreted, (simply stating that this attribute specifies the placement of such annotations on their respective pages), and there is substantial inconsistency among PDF viewer applications, in their respective interpretations. Whereas the Adobe® "pdfmark Reference Manual" states that the Rect attribute specifies the vertex co-ordinates "of the rectangle defining the open note window", (which might be construed as referring to the pop-up window in its open state), it appears that few — if indeed any — of of the currently available PDF viewer applications have adopted this interpretation. All *do* appear to agree that the *upper left corner* of the annotation *icon* should be placed at the page co-ordinates which are derived by combination of the *lower left x* ordinate, and the *upper right y* ordinate, as specified for the Rect attribute; there is significantly less agreement on what effect, if any, the width, and height of the rectangle, which may be deduced from the Rect attribute specification, should have. All viewers appear to use a fixed size icon, and an arbitrarily chosen size, and placement, for the associated pop-up window; at least one viewer *does* appear to interpret the derived annotation width, and height, as a specification of the extent to which the effective clickable region covers, or extends beyond, the region occupied by the icon itself, but most appear to ignore them altogether,

2.6.2. Options for Manipulating `pdfnote` Annotation Attributes

To the extent to which various PDF viewing applications may support them, the `pdfnote` macro will interpret the following optional argments, (which *must* be placed *before* any text specifying the content of the annotation), to affect the style of `pdfnote` annotations:

`-O` Select "*open*" as the preferred initial state for the associated `pdfnote` pop-up window; no additional arguments are parsed, beyond `-O` itself, when interpreting this option.

This option sets the `Open` attribute for the associated `pdfnote` annotation to `true`; some PDF viewer applications may not reliably interpret this attribute. The example to the left is specified thus:

```
.pdfnote -O -T "A Pop-Up Note in Initially Open State" \
This note should be displayed in the open state, when the \
document itself is opened, if the PDF viewer supports \
this capability.
```

it should be displayed in the initially open state, when this document is opened in a PDF viewer application which *does* correctly interpret the attribute.

`-T "`*`Title Bar Text ...`*`"`
Define text to be displayed within the title bar of the pop-up window which is associated with a `pdfnote` annotation; requires *exactly one* following argument, in addition to the `-T` itself; this argument should be a text string, and should be enclosed in programming quotes (ASCII 34), if spaces are to be included. Any of the preceding `pdfnote` annotation examples illustrate how this option is used.

This option causes its text string argument to be passed as the value of the `Title` attribute, when invoking the `pdfmark` macro to create the associated `pdfnote` annotation; this appears to enjoy better support than the `Open` attribute, among PDF viewer applications, but support is by no means universal.

`-C <`*`red-value`*`> <`*`green-value`*`> <`*`blue-value`*`>`
Specifies the background colour, which is to be used for the `pdfnote` annotation's icon, and also, if supported by the PDF viewer application, for the frame, and title bar, of the associated `pdfnote` pop-up window. This option requires *exactly three* additional arguments, following the `-C` itself; each of these *must* be a decimal number, in the range `0.0 ... 1.0`, representing the intensity, in RGB colour space, for each of the red, green, and blue components of the desired colour, respectively.

If this option is not specified, the PDF viewer application will assign a default colour, for both the `pdfnote` icon background, and, if supported, for the pop-up window's frame.

Specification of this option causes a `Color` attribute assignment to be included within the `pdfmark` invocation, which is used to place the associated `pdfnote` annotation. Differing PDF viewer applications vary in the extent to which they support this attribute. The example to the left has been specified thus:

```
.pdfnote -C 0.7 1.0 0.7 \
-T "Icon Background Colour Example" \
This example specifies a pale green colour, for the icon \
background and pop-up window frame, and serves to illustrate \
the extent to which text annotation colours are supported by \
the current PDF viewer application.
```

which may serve as an illustration of the current PDF viewer application's level of support for colour attributes, when applied to "`Text`" annotations defined using the `pdfnote` macro.

`-I <`*`icon-name`*`>`
Assigns an alternative icon, to indicate placement of a `pdfnote` annotation; requires one additional argument following the `-I`, indicating the style of icon which is to be assigned; the selected style is assigned, via the `pdfmark` macro, to the `Name` attribute of the annotation.

Icon styles are identified by name. The particular set of named icons, which are available, depends on the PDF viewer application which is in use; however, regardless of any non-standard choices, which a particular viewer might support, the Adobe® PDF Specification requires, as minimum, that icons named `Note`, `Comment`, `Help`, `Insert`, `Key`, `NewParagraph`, and `Paragraph` should be available. If no explicit icon style is selected, the `Note` style is used, by default.

As an example of how an alternative icon style might be used,[18] a keynote annotation may be placed thus:

```
.als "" PDFNOTE.QUOTED
...
.pdfnote -I Key -T "An Example Keynote Annotation" \
This is an example of a \*[""  keynote annotation], which has been \
defined using the pdfnote macro, using its optional \*[""  Key] \
icon selection.
```

-PD <***line-count***>
: Set the number of blank lines which should be inserted, to serve as paragraph separators within `pdfnote` content, following an end-of-paragraph `PDFNOTE.PILCROW` mark, (see section 2.6.3, "Controlling `pdfnote` Text Layout"), within the content of `pdfnote` annotations. Requires *exactly one* additional argument, following `-PD` itself; this should be an *integer numeric* value, indicating the number of *additional* newlines which should be inserted, *following* the one which is normally placed at the end-of-paragraph mark.

  Unlike each of the preceding `pdfnote` options, (each of which assigns annotation attributes, and applies only to the individual `pdfnote` instance for which it is specified), the `-PD` option — so named by analogy with the similarly named `ms` macro, which has a similar effect — *does not* assign annotation attributes; rather, it sets a count initializer, which is internal to the `pdfnote` macro itself. Its effect is "sticky": that is, it applies not only to the `pdfnote` instance which specifies it, but also to any `pdfnote` instances which follow it, unless it is specified again, with a different — or even (albeit redundantly) with the same — `line-count` value, for any such following instance.

-- Suppresses interpretation of any further `pdfnote` macro arguments as options. This is *not*, strictly, an option *per se*, but may be required in any case where the following argument is intended to begin the annotation content, when it could be mistaken for an optional feature specification.

2.6.3. Controlling `pdfnote` Text Layout

The Adobe® PDF and pdfmark specifications make very little provision for control of the layout of the content of pop-up windows which are associated with "`Text`" annotations, stipulating only that the size and font should be chosen by the PDF viewer application, which usually offers little, or no opportunity for user participation in these choices.

Generally, PDF viewer applications will open annotation pop-up windows when required, each with default width and height as specified by the viewer application itself. The annotation content is displayed in a font which is also specified by the viewer application; this is usually a proportionally spaced font, and there is no mechanism for choosing an alternative. The content is nominally interpreted as a single-line of text, which flows to fit the width of the window; text flow is facilitated by insertion of "soft" line breaks, coincident with white space, resulting is a flush left, ragged right layout. The extent to which the author of the annotation may influence this layout is limited to insertion of "hard" line breaks; these will always be rendered as such, when the text is displayed in the pop-up window, producing the effect of a paragraph break.

When placing an annotation, using the `pdfnote` macro, if the author wishes to affect the text layout by inserting a hard line break, this *must* be represented by the literal "`\n`" character sequence. Unfortunately, simply specifying this character sequence within any argument to the `pdfnote` macro, (as is necessary to include it within the annotation content), presents a non-trivial challenge to the author: the "`\`" character introduces a `troff` escape, and when it is followed by the "`n`" character, the escape is interpreted as a reference to a numeric register, which is resolved according to whatever follows. Simply escaping the "`\`" character itself, at the point of the `pdfnote` macro call, does *not* present a satisfactory solution to this challenge, since *multiple levels* of escaping are required, to survive interpretation through an indeterminate number of internal macro call levels. Thus, to circumvent this challenge, and to robustly facilitate inclusion of the literal "`\n`" character sequence within the `pdfmark` output stream, the `pdfmark` macros define the following named strings:

PDFNOTE.NEWLINE
: A string representation of the "`\n`" character sequence, which is encoded in a manner which, when interpreted within the immediate arguments to the `pdfnote` macro, re-encodes the sequence such that its ultimate interpretation is deferred, until it is eventually written, as a literal representation of a *single* "`\n`" character sequence, to the `pdfmark` output stream. Use of `PDFNOTE.NEWLINE` is analogous to that of

18. While this example serves, primarily, to illustrate the the selection of the "`Key`" icon style, for the associated `pdfnote` annotation, it also illustrates the use of `PDFNOTE.QUOTED` interpolation, (with aliasing to *["" ...text...] as document-local shorthand), to introduce double quoted text within the content of the annotation.

PDFNOTE.PILCROW, which is described below, and is illustrated in previous examples within section 2.6, "Annotating a PDF Document using Pop-Up Notes".

PDFNOTE.PILCROW

So named for its association with the typographer's pilcrow mark, when interpreted within the immediate arguments to the pdfnote macro, this marks the end of a logical paragraph, and is re-encoded as a (possibly recurring) sequence of *[PDFNOTE.NEWLINE] re-encodings. At least one such re-encoding is *always* inserted; this is then repeated as many times as specified by the <line-count> argument to the last-specified, if any, -PD option — see section 2.6.2, "Options for Manipulating pdfnote Annotation Attributes" — to the immediate, or any preceding, instance of pdfnote macro use. The effect is to introduce a new logical paragraph, within the content of the pdfnote annotation, separated from the preceding paragraph, of which the end is indicated by the *[PDFNOTE.PILCROW] mark, by <line-count> blank lines.

In the absence of any preceding -PD option specification, the effect of *[PDFNOTE.PILCROW] becomes *identical* to that of a single instance of *[PDFNOTE.NEWLINE].[19]

2.7. Synchronizing Output and pdfmark Contexts

It has been noted previously, that the pdfview macro, (see section 2.2, "Selecting an Initial Document View"), the pdfinfo macro, (see section 2.3, "Adding Document Identification Meta-Data"), and the pdfhref macro, when used to create a document outline, (see section 2.4, "Creating a Document Outline"), do not immediately write their pdfmark output to the PostScript® data stream; instead, they cache their output, in a groff diversion, in the case of the pdfview and pdfinfo macros, or in an ordered collection of strings and numeric registers, in the case of the document outline, until a more appropriate time for copying it out. In the case of pdfview and pdfinfo "meta-data", this "more appropriate time" is explicitly chosen by the user; in the case of document outline data, *some* cached data may be implicitly written out as the document outline is compiled, but there will *always* be some remaining data, which must be explicitly flushed out, before the groff formatting process is allowed to complete.

To allow the user to choose when cached pdfmark data is to be flushed to the output stream, the pdfmark macro package provides the pdfsync macro, (to synchronize the cache and output states). In its simplest form, it is invoked without arguments, i.e.

```
.pdfsync
```

This form of invocation ensures that *both* the "meta-data cache", containing pdfview and pdfinfo data, *and* the "outline cache", containing any previously uncommitted document outline data, are flushed; ideally, this should be included in a groff "end macro", to ensure that *both* caches are flushed, before groff terminates.

Occasionally, it may be desirable to flush either the "meta-data cache", without affecting the "outline cache", or vice-versa, at a user specified time, prior to reaching the end of the document. This may be accomplished, by invoking the pdfsync macro with an argument, i.e.

```
.pdfsync M
```

to flush only the "meta-data cache", or

```
.pdfsync O
```

to flush only the "outline cache".

The "meta-data cache" can normally be safely flushed in this manner, at any time *after* output of the first page has started; (it may cause formatting problems, most notably the appearance of unwanted white space, if flushed earlier, or indeed, if flushed immediately after a page transition, but before the output of the content on the new page has commenced). Caution is required, however, when explicitly flushing the "outline cache", since if the outline is to be subsequently extended, then the first outline entry after flushing *must* be specified at level 1. Nevertheless, such explicit flushing may occasionally be necessary; for example, the TC macro in the spdf.tmac package, (see section 3.1, "Using pdfmark Macros with the ms Macro Package"), invokes ".pdfsync O" to ensure that the outline for the "body" section of the document is terminated, *before* it commences the formatting of the table of contents section.

19. Neither PDFNOTE.NEWLINE, nor PDFNOTE.PILCROW were provided in any version of the pdfmark macros, which was published before Feb-2023. Earlier versions provided PDFLB, (for PDF line-break), as an alternative; it offered a similar capability to PDFNOTE.NEWLINE, but its implementation was flawed, and was not robust. The flawed implementation of PDFLB is still supported, but it is now considered to be deprecated, and using it is not recommended; either PDFNOTE.NEWLINE, or PDFNOTE.PILCROW should be used instead.

3. PDF Document Layout

The pdfmark macros described in the preceding section, (see section 2, "Exploiting PDF Document Features"), provide no inherent document formatting capability of their own. However, they may be used in conjunction with any other groff macro package of the user's choice,[20] to add such capability.

In preparing this document, the standard ms macro package, supplied as a component of the GNU Troff distribution, has been employed. To facilitate the use of the pdfmark macros with the ms macros, a binding macro package, spdf.tmac, has been created. The use of this binding macro package is described in the following section, (see section 3.1, "Using pdfmark Macros with the ms Macro Package"); it may also serve as an example to users of other standard groff macro packages, as to how the pdfmark macros may be employed with their chosen primary macro package.

3.1. Using pdfmark Macros with the ms Macro Package

The use of the binding macro package, spdf.tmac, allows for the use of the pdfmark macros in conjunction with the ms macros, simply by issuing a groff command of the form[21]

```
groff [-Tps|-Tpdf] [-m spdf [-options ...] file ...
```

When using the spdf.tmac package, the groff input files may be marked up using any of the standard ms macros to specify document formatting, while PDF features may be added, using any of the pdfmark macros described previously, (see section 2, "Exploiting PDF Document Features"). Additionally, spdf.tmac defines a number of convenient extensions to the ms macro set, to better accommodate the use of PDF features within the ms formatting framework, and to address a number of ms document layout issues, which require special handling when producing PDF documents. These additional macros, and the issues they are intended to address, are described below.

3.1.1. Document Structuring Considerations when using ms Macros

Every published document *must* incorporate, as a minimum, a document body; additionally, many documents may include *front-matter*, which precedes the body, and *end-matter*, which follows the body. Additionally, when publishing as a PDF document, it may be desired to incorporate a document outline, referring to chapter, or section headings, within the document body.

Conventionally, when a document is to include a *table of contents*, this should be placed at the end of the *front-matter*.

Traditional AT&T implementations of ms provide a number of macros to control front-matter style, (of which only the "released paper" style, selected by use of the RP macro, is supported by groff ms), accompanied by several macros to specify front-matter content, (also supported by groff ms). Both traditional, and groff ms implementations also provide a small set of macros to facilitate compilation of a table of contents; they do not, however, offer any standard facilities for creation of a corresponding document outline.

Unfortunately, the traditional ms method of compiling the table of contents results in it being printed at the end of the document, rather than in its normal position, at the end of the front-matter. Traditionally, this unusual placement of the table of contents would be corrected, by manual collation, after printing; emulation of this mechanical collation technique presents a challenge, when the document is to be published in PDF format.

Taking up the challenge of collating the various document sections into the correct order, when producing any PDF document, will necessitate special consideration during the PDF publication process; this will be discussed in greater depth, in section 4, "The PDF Publishing Process". To accommodate any specialized processing which may be required, spdf.tmac provides:–

- Macros to isolate the *front-matter*, (excluding the *table of contents*), from the body of the document.
- Further macros to compile a table of contents, and a corresponding PDF document outline, deriving both from section headings, (see section 3.1.2.1, "The XH and XN Macros").
- A redefined implementation of the TC macro, (to be invoked at the end of the document, as in traditional ms usage); this isolates the table of contents from its preceding front-matter (if any), and from the document body, to facilitate the collation process.

---

20. Any of the standard groff "full-service" macro packages, me, mm, mom, or ms, or indeed, any "home-brew" macro package provided by the user, should be suitable for the purpose; regardless of the chosen "full-service" macro package, it is likely that a binding package, specific to this choice, will be required.

21. Once again, as noted in footnote[6] to section 2, "Exploiting PDF Document Features", do not specify any -T*dev* option, other than -Tps, or -Tpdf; specify -Tpdf, if you wish to avoid the conversion of PostScript® output to PDF, which will be required if you specify -Tps, or if you omit the -T*dev* option entirely.

3.1.2. Using `ms` Section Headings in PDF Documents

Traditionally, ms provides the NH and SH macros to introduce section headings. However, in traditional ms implementations, there is no standard mechanism for generating a table of contents entry based on the text of the section heading; neither is there any recognized standard method for establishing a cross reference link, or a document outline reference, to the section.

To address this limitation of traditional ms implementations, the spdf.tmac binding macro package provides the XH and XN macros,[22] (see section 3.1.2.1, "The XH and XN Macros"), to be used in conjunction with the SH and NH macros respectively; each of these identifies, by specification of appropriate arguments, text which is to be incorporated into the section heading, duplicated within the PDF document outline, and in the table of contents.

3.1.2.1. The `XH` and `XN` Macros

Formalized from the release of groff-1.23 onwards,[23] and nominally intended to be used following SH and NH respectively, the calling syntax for this pair of spdf.tmac macros is specified as:–

```
.SH
.XH [-N <name>] [-S] [-X] <outline-level> <heading-text> ...

.NH <outline-level>
.XN [-N <name>] [-S] [-X] <heading-text> ...
```

In either case, the <heading-text>... arguments are incorporated into the document body, formatted as section heading text. Additionally, these same <heading-text>... arguments, (prefixed by the content of the SN string, in the XN case), are incorporated into the PDF document outline, at the level specified by the <outline-level> argument, and they are made available to the user-definable XH-UPDATE-TOC call-back macro, (see section 3.1.2.3, "The XH-UPDATE-TOC Macro"), to support creation of a corresponding entry in the document's table of contents.

In both cases, the supported macro options[24] are:–

-N <name>
: Create a pdfhref destination, with the specified <name>, and associate it with the corresponding section heading, as designated by <heading-text>.

-S Strip any font-family selection escape sequences, which may have been specified, from a copy of <heading-text>, before incorporating this into the document outline; (this is necessary when such escape sequences are present, to avoid verbatim rendition of the escape sequences themselves, within the text of the document outline).

-X Ensure that any pdfhref destination name, specified by the **-N** <name> option, is included within the document's cross-reference dictionary.

3.1.2.2. The `XH-INIT` and `XN-INIT` Macros

This pair of macros serve as context initialization hooks; called by the default implementations of the XH and XN macros respectively, without arguments, *before* XH-UPDATE-TOC is called. By default, both return immediately, *without* performing *any* action. However, users may override either, or both, to perform any desired activity ... e.g. to save context for subsequent use by any user-defined macro, which may have been provided to override the default implementation of XH-UPDATE-TOC.

3.1.2.3. The `XH-UPDATE-TOC` Macro

This macro is called by both XH and XN, (there is no corresponding XN-UPDATE-TOC equivalent, since none is required to support the default XH and XN implementations), to propagate content from the specified section heading arguments to the document's table of contents. From groff-1.23 onwards, a rudimentary default implementation of

22. On a technical note, since groff-1.23, the groff implementation of ms itself has incorporated basic infrastructure providing XH and XN macros, to facilitate duplication of section heading text into the table of contents; spdf.tmac builds on top of this infrastructure, *indirectly* redefining XH and XN, by provision of macros XH-REPLACEMENT and XN-REPLACEMENT respectively, to accommodate the duplication of section heading text into the PDF document outline, in addition to the table of contents. Use of this indirect technique is recommended, whenever redefinition of XH, or XN, is desired.

23. Prior to the release of groff-1.23, a prototypical implementation of spdf.tmac was introduced with groff-1.19.2; this prototype included an implementation of the XN macro, but it did *not* provide XH, nor did it support the XH-INIT, XN-INIT, and XH-UPDATE-TOC call-back features, nor the XH-REPLACEMENT, and XN-REPLACEMENT capabilities.

24. *None* of these options are supported by the underlying ms implementations of XH or XN, as implemented from groff-1.23 onwards. Prior to groff-1.23, only the -N <name> and -X options are supported by the prototypical spdf.tmac implementation of XN, as provided from groff-1.19.2 onwards.

XH-UPDATE-TOC is provided within the standard ms macro suite; however, it is anticipated that the user will override this default implementation, in order to achieve more effective control of table of contents formatting.

When writing a replacement for the XH-UPDATE-TOC macro, it should be implemented such that it will interpret arguments as specified in the prototype

.XH-UPDATE-TOC *<outline-level>* [*<section-number>*] *<heading-text>* ...

in which the *<outline-level>* and *<heading-text>* arguments are the same as those specified for the XH, or the NH/XN call sequence, from which XH-UPDATE-TOC itself is called; the *<section-number>* argument is *always* specified, when XH-UPDATE-TOC is called by XN, (and *never* when called by XH); when present, it represents the value of the SN string, which prevails at the time of the invoking XN call, and is simply processed as a prefix to the *<heading-text>* argument.

The default implementation of XH-UPDATE-TOC offers only rudimentary formatting of the resultant table of contents entry; the *<outline-level>* argument is simply ignored, and the remaining arguments are passed to the standard ms table of contents generating capability, in a form which is equivalent to

```
.XS
\&[<section-number> ]<heading-text> ...
.XE
```

As an example (with abridged comments) of how XH-UPDATE-TOC may be redefined, to achieve more creative formatting of a table of contents, this publication substitutes the following document-local implementation:

```
.ds XNVS1 0.50v  \" leading for top level
.ds XNVS2 0.15v  \" leading at nesting level increment
.ds XNVS3 0.30v  \" leading following nested group
.
.de XH-UPDATE-TOC
.   XS
.      if r tc*hl \{\
.            \" Compute additional leading at <outline-level> change
.            \"
.            ie \\$1>1 \{\
.                  ie \\$1>\\n[tc*hl] .sp \\*[XNVS2]
.                  el .if \\n[tc*hl]>\\$1 .sp \\*[XNVS3]
.               \}
.            el .sp \\*[XNVS1]
.         \}
.
.      \" Record <outline-level> of this entry, to compare with next
.      \"
.      ie \\$1 .nr tc*hl \\$1
.      el .nr tc*hl 1
.
.      \" Set indentation, and insert <section-number> for this entry
.      \"
.      nop \h'\\n[tc*hl]-1m'\\$2\c
.
.      \" Append <heading-text> for this entry
.      \"
.      shift 2
.      nop \h'1.5n'\\$*\h'0.5n'
.   XE
..
```

Used in conjunction with NH and XN, this uses document-local register tc*hl to track, group, and indent the table of contents entries for this document, on the basis of their specified *<outline-level>* specifications, separating *<outline-level>* groups by additional line spacing, (having an effect similar to that of increased leading), as controlled by the XNVS1, XNVS2, and XNVS3 document-local strings, at each change in *<outline-level>*.

3.1.2.4. The XH-REPLACEMENT and XN-REPLACEMENT Macros

The default XH and XN macro implementations *reserve* this pair of macro names, to facilitate *redefinition* of XH and XN behaviour respectively, while retaining the ability to take advantage of first-time-of-use infrastructure initialization logic, which is incorporated within the respective default implementations.

It is important to understand that, in conventional usage, neither of these macros should ever be called directly. Rather, either one, or both, should be defined, *after* loading s.tmac, and *before* calling either XH, or XN for the first time; the defined implementations are then invoked when XH, or XN are called, respectively.

User-written XH-REPLACEMENT and XN-REPLACEMENT macros may implement *any* desired functionality. They are not constrained to emulation of the default XH and XN capabilities; however, it is *strongly* recommended that they do so, while adding any required extended features. For example, spdf.tmac defines both replacement macros thus:[25]

```
.de XH-REPLACEMENT als
.als XN-REPLACEMENT XH-REPLACEMENT
.am XH-REPLACEMENT
.   \\$0-INIT
.   rm spdf:refname
.   als spdf:bm.define spdf:bm.basic
.   while d spdf:XH\\$1 \{\
.         spdf:XH\\$1 \\$*
.         shift \\n[spdf:argc]
.      \}
.   rr spdf:argc
.   if '\\$1'--' .shift
.   spdf:\\$0.format \\$@
..
```

with macros XH-N, XH-S, and XH-X defined locally, extending the default behaviour, such that the non-default -N, -S, and -X option flags are interpreted, (and register spdf:argc is set, to control the while loop which does so); it further extends the default behaviour, by using locally defined macros, spdf:XH.format, and spdf:XN.format, (dynamically modified by spdf:bm.basic, spdf:bm.define, and spdf:refname), to propagate the specified section heading text to the PDF document outline, in addition to reproducing the default propagation to the document's table of contents, by calling XH-UPDATE-TOC.

25. An important consideration, in the design of such replacement macros, is that they will ultimately be invoked as XH, and XN respectively; thus, they *must* interpret their arguments *exactly* as they would be passed to XH and XN, and within the macro bodies, \\$0 will be interpreted as XH or XN, as appropriate.

4. The PDF Publishing Process

4.1. Resolving Cross References

4.1.1. Creating a Document Reference Map

4.1.2. Deploying a Document Reference Map