Portable Document Format Publishing with GNU Troff

Keith Marshall
<keith@address.hidden>



A GNU MANUAL

First published 2005
Revised 2009 (twice), 2010, 2012, 2013, 2014, 2018, 2021 (4 times), 2022 (4 times), 2023, 2024, 2025, 2026 (twice)

This edition published 2026
Copyright © 2005, 2009, 2010, 2012, 2013, 2014, 2018, 2021, 2022, 2023, 2024, 2025, 2026, Free Software Foundation, Inc.

Keith Marshall has asserted his moral right to be identified as the author of this work, in accordance with the UK Copyright, Designs and Patents Act, 1988.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3, (or any later version published by the Free Software Foundation), with the Front-Cover Texts being "A GNU MANUAL", with one Invariant Section, this being the content of Appendix A, and with no Back-Cover Texts.

A copy of the license is included in Appendix A, entitled "GNU Free Documentation License".

Table of Contents

1. Introduction **7**

2. Exploiting PDF Document Features **8**

2.1. The pdfmark Operator 8
2.2. Selecting an Initial Document View 8
2.3. Adding Document Identification Meta-Data 9
2.4. Creating a Document Outline 9
2.4.1. A Basic Document Outline 9
2.4.2. Hierarchical Structure in a Document Outline 10
2.4.3. Associating a Document View with an Outline Reference 10
2.4.4. Folding the Outline to Conceal Less Significant Headings 11
2.4.5. Outlines for Multipart Documents 11
2.4.6. Delegation of the Outline Definition 12
2.5. Adding Reference Marks and Links 12
2.5.1. Optional Features of the pdfhref Macro 13
2.5.2. Marking a Reference Destination 15
2.5.2.1. Mapping a Destination for Cross Referencing 15
2.5.2.2. Associating a Document View with a Reference Mark 16
2.5.3. Linking to a Marked Reference Destination 16
2.5.3.1. References within a Single PDF Document 16
2.5.3.2. References to Destinations in Other PDF Documents 17
2.5.4. Linking to Internet Resources 18
2.5.5. Establishing a Format for References 18
2.5.5.1. Using Colour to Demarcate Link Regions 18
2.5.5.2. Specifying Reference Text Explicitly 19
2.5.5.3. Using Automatically Formatted Reference Text 20
2.5.5.4. Customizing Automatically Formatted Reference Text 21
2.5.6. Problematic Links 24
2.5.6.1. Links with a Page Transition in the Active Region 24
2.6. Annotating a PDF Document using Pop-Up Notes 25
2.6.1. Controlling pdfnote Icon Placement 25
2.6.2. Options for Manipulating pdfnote Annotation Attributes 27
2.6.3. Controlling pdfnote Text Layout 28
2.7. Synchronizing Output and pdfmark Contexts 29

3. PDF Document Layout **31**

3.1. Using pdfmark Macros with the ms Macro Package 31
3.1.1. Document Structuring Considerations when using ms Macros 31
3.1.2. Using ms Section Headings in PDF Documents 32
3.1.2.1. The XH and XN Macros 32
3.1.2.2. The XH-INIT and XN-INIT Macros 32
3.1.2.3. The XH-UPDATE-TOC Macro 32
3.1.2.4. The XH-REPLACEMENT and XN-REPLACEMENT Macros 34
3.1.3. Layout Adjustment to Support Duplex Printing 34

4. The PDF Publishing Process **36**
4.1. The `pdfroff` Program 36
4.1.1. Principles of `pdfroff` Operation 37
4.1.2. How `pdfroff` Resolves Cross References 38
4.1.3. Using In-Document Hints to Control `pdfroff` Processing Options 39
4.1.4. Using a `pdfroff` Style-Sheet to Specify Document Front-Matter 40
4.1.5. How `pdfroff` Collates Tables of Contents 42
4.1.6. How `pdfroff` Formats a Document Body 44
4.1.7. How `pdfroff` Assembles a Finished Document 45
4.2. Preparing Documents for On-Screen Reading versus Hard-Copy Printing 46
4.2.1. Establishing a Page Layout for On-Screen Reading 46
4.2.2. Establishing a Page Layout for Hard-Copy Typesetting 47
4.2.3. Ensuring that Content is Printed on a Particular Side of the Page 47
4.2.3.1. Recto-Verso Page Break Handling when Using the `ms` Macros 49
4.3. Considerations for Working with Document References 55
4.3.1. Creating a Document Reference Dictionary 55
4.3.2. Deploying a Document Reference Dictionary 56
4.3.3. Using Custom Reference Formatting Keywords 56
4.3.4. Avoiding Reference Name Conflicts 57
4.4. An Alternative Technique for Generating Tables of Contents 58
4.4.1. Using the Basic `groff_toc` Macro Framework 58
4.4.2. Generating and Interpreting Table of Contents Reference Data 59
4.4.3. Binding the `groff_toc` Macro Framework with the `ms` and `pdfmark` Macros 61
4.4.3.1. Managing Side-Effects of `groff_toc` Macro Bindings 65
4.4.4. Classification and Selective Processing of Table of Contents Entries 66

Appendix A. The GNU Free Documentation License . . . . . **lxxi**
A.0. PREAMBLE lxxi
A.1. APPLICABILITY AND DEFINITIONS lxxi
A.2. VERBATIM COPYING lxxii
A.3. COPYING IN QUANTITY lxxii
A.4. MODIFICATIONS lxxiii
A.5. COMBINING DOCUMENTS lxxiv
A.6. COLLECTIONS OF DOCUMENTS lxxiv
A.7. AGGREGATION WITH INDEPENDENT WORKS lxxiv
A.8. TRANSLATION lxxiv
A.9. TERMINATION lxxiv
A.10. FUTURE REVISIONS OF THIS LICENSE lxxv
A.11. RELICENSING lxxv
ADDENDUM: How to use this License for your documents lxxv

Appendix B. Typesetting the GNU Free Documentation License ..... **lxxvii**
B.1. License Typesetting Constraints lxxvii
B.2. Choosing an Appropriate License Document Source Format lxxviii
B.3. Marking Up the Plain Text Document Source lxxviii
B.4. The GNU Free Documentation License Formatting Macros lxxxi
B.4.1. Marking Redundant Blank Input Lines — the FDL-BL Macro lxxxi
B.4.2. Introducing Numbered Section Headings — the FDL-SH and FDL-SM Macros lxxxi
B.4.3. Marking Paragraph Breaks — the FDL-LP Macro lxxxii
B.4.4. Interpreting Input Text Indentation — the FDL-TI Macro lxxxii
B.4.5. Formatting Itemized List Content — the FDL-IP Macro lxxxiii
B.4.6. Typesetting Block-Quoted License Text — the FDL-QS and FDL-QE Macros lxxxiii
B.5. Customizing the GNU Free Documentation License Formatting Process lxxxiv
B.5.1. Initial Customization — the FDL-SETUP Macro lxxxiv
B.5.2. Section Header Integration — the FDL-XH Macro lxxxvi
B.6. GNU Free Documentation License Finishing Touches lxxxviii

Appendix C. Working Macros used for Typesetting this Publication ..... **lxxxix**
C.1. Controlling the Style of Formatted Text xc
C.2. Starting a Section on a New Recto Page — the NEW-RECTO-PAGE Macro xci
C.3. Generating and Typesetting the Table of Contents xciii
C.3.1. Table of Contents Reference Data Collection — the XN-UPDATE-TOC Macro xciii
C.3.2. Establishing a Table of Contents Layout — the toc.outline Macro xcv
C.3.3. Specifying Table of Contents Entry Context — the toc.refmark Macro xcvi
C.3.4. Specifying Page Number References — the toc.pageref Macro xcvii
C.3.5. Avoiding Excessive Use of pdfhref Link Colour in Tables of Contents xcvii
C.3.6. Additional Convenience Macros for Formatting Tables of Contents xcviii
C.4. Macros for Laying Out the Content of Appendicies c
C.4.1. Specifying an Appendix Title — the XH-APPENDIX Macro ci
C.4.2. Appendix Numbering Styles — the XH-APPENDIX-NUMBER-FORMAT Macro cii
C.4.3. Appendix Subsection Numbering — the XH-APPENDIX-NH Macro ciii

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 copious 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 4.3.1, "Creating a Document Reference Dictionary"); 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 4.3.1, "Creating a Document Reference Dictionary").

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 4.3.2, "Deploying a Document Reference Dictionary"), 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 4.3.1, "Creating a Document Reference Dictionary").

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 4.3, "Considerations for Working with Document 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 4.3; 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"

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.

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
```

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 18, "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 18, 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 4.3.1, "Creating a Document Reference Dictionary".

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 18, "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 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.

---

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-pdfmark from groff-pdfmark-20230317.1 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 18, "Using Colour to Demarcate Link Regions"

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

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

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
```

might now, subject to limitations which will be discussed in section 4.3.3, "Using Custom Reference Formatting Keywords", produce a reference similar to

see chapter 2, page 18, 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

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.

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.

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

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.

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 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.

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.

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.

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] -mspdf [-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, prior to adopting an alternative table of contents generation technique, as described in section 4.4, this publication substituted 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.

3.1.3. Layout Adjustment to Support Duplex Printing

When formatting a PDF document for on-screen viewing, there is no particular need to distinguish between the layouts for even-numbered and odd-numbered pages; thus, it is common to set the page offset and line length to establish equal width pargins margins to left and right of the displayed text; for example, in ms:

```
.nr PO  2.0c
.nr LL 17.0c
```

will create two centimetre wide margins on both sides of the page, when formatting for display on the equivalent of A4 paper, in portrait orienation.

Conversely, if preparing output for a hard-copy device, which supports duplex printing, it may be desirable to reduce the effective page width by a "binding allowance", which should then be added to the left-hand page margin width, when formatting odd-numbered pages, and to the right-hand page margin width, when formatting even-numbered pages. Although ms does not provide any standard settings, for specification of alternating page offsets for odd-numbered and even-numbered pages, it *does* implement a bottom-of-page trap-invoked macro, BT, which may be exploited to achieve the desired effect. To illustrate this, the preceding example, which set equal width left-hand and right-hand page margins, of two centimetres each, when formatting for the twenty one centimetre width of A4 paper, may be extended to accommodate the addition of an optional specification on the formatter command line:

groff [-Tps|-Tpdf] -mspdf [*-options* ...] **-duplex=**<*width*> *file* ...

This additional option might then be interpreted, within the document source, such that, if unspecified, it leaves the original layout unchanged, but when specified, it changes the initial page offset setting to the value of its <*width*> argument, while leaving the line length unchanged; this modified page offset is then propagated, through the augmented BT macro, to become effective on odd-numbered pages, while an alternative page offset is calculated, (as the effective

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.

value of the residual right-hand page margin, as it will become on odd-numbered pages, deduced by subtraction of the modified initial page offset, and the specified line length, from the inferred page width), for use on even-numbered pages; a possible implementation, for `ms`, might look like this:

```
.if duplex \{\
.\" Prepare to format for duplex printing; first reset the initial
.\" value of the page offset, to specify the effective value which
.\" is to be used on odd-numbered pages.
.\"
.   if \B'\*[uplex]' .nr PO \*[uplex]
.\"
.\" Next, augment the bottom-of-page trap macro, to swap widths of
.\" left-hand and right-hand page margins, on each transition from
.\" odd-numbered to even-numbered page, and vice versa.
.\"
.   am BT
.   \" When advancing from an odd-numbered page, compute the value
.   \" of the original right-hand page margin width, which will be
.   \" used as the PO value on the following even-numbered page.
.   \"
.      ie o .nr PO 2i+\n[.l]u-\n[PO]u-\n[LL]u
.
.   \" Conversely, when advancing from an even-numbered page to an
.   \" odd-numbered page, we simply revert PO to its initial value.
.   \"
.      el .nr PO \n[PO]u
.   .
.\}
```

With code, such as the foregoing, in place *before* `ms` output begins, a `groff` invocation similar to:

`groff [-Tps|-Tpdf] -mspdf [`*`-options ...`*`] -dpaper=a4` **`-duplex=2.5c`** *`file ...`*

will set the initial `PO` value to `2.5 cm`, which, in conjunction with the initial `LL` setting of `17.0 cm`, accounts for `19.5 cm` of the `21.0 cm` page width, leaving an effective right-hand page margin of `1.5 cm`, implying that `1.0 cm` of the initial `2.5 cm` left-hand page margin represents the "binding allowance"; this will then alternate between left-hand and right-hand page margins, on odd-numbered and even-numbered pages, respectively.

To assist in understanding the foregoing duplex printing initialization code, some further explanation may be useful:

- This code is intended to be interpreted at `groff`'s outer processing level; it *must* be defined, within the input stream, to ensure that is interpreted *before* any output is generated.
- Although it may appear to be a specially defined option, `-duplex` is nothing more than an exploitation of `groff`'s standard "`-d`" option, used to define a string named "`uplex`". Similarly, "`.if duplex`" is `groff`'s "`d`" logical operator, used to determine whether, or not, this "`uplex`" register has been defined; if it has, it's content is expected to represent a numeric expression, which is evaluated either as a new absolute value for assignment as, or an increment to be added to, or subtracted from the initial `PO` register value.
- Within this code, all string and numeric references, as they are used in `PO` register assignments, are evaluated *immediately*; this is particularly important within the augmentation of the `BT` macro, where evaluation of initial values is required, and thus this evaluation is *deliberately not* deferred until this bottom-of-page trap macro is executed, (as may be more commonly expected within macro definitions).
- The derivation of the expression, used to set the new value of the `PO` register, when the trap is sprung at the bottom of an odd-numbered page, (so that it takes effect on the following even-numbered page), may not be obvious. The total page width is *not* represented *directly*, in any `groff` register; however, at start up, `groff` initializes the line length, as represented by the "`.l`" register, to a value which is two inches less than the page width, as defined in `groff`'s `papersize.tmac` file; thus, "`2i+\n[.l]u`" yields the value of the actual page width, and subsequent subtraction of both the user specified initial `PO` and `LL` values yields the effective initial width of the right-hand page margin; when this is subsequently assigned as a new `PO` value, it has the effect of interchanging the left-hand and right-hand margin widths, and thus, moves the binding allowance alternately to the left-hand side of odd-numbered pages, and to the right-hand side of even-numbered pages.

4. The PDF Publishing Process

GNU `troff`, in common with other `troff` implementations, is a *single pass* document formatter; while this may support a high level of operational performance, it does impose certain restrictions on formatting capability. In particular, when any computed content is to be interpolated into the formatted output stream, that content *must* have been computed *before* the point at which interpolation is to occur. Some examples of such computed content, which *cannot* be interpolated with only a single formatting pass, include:–

- Interpolation of "Page *n* of *nn*" annotations within page headers, or footers; the value of the *last* page number, *nn*, is unknown until the final page has been formatted, yet it is required *before* the first such annotation is to be interpolated, (typically, when formatting the *first* page). At least *two* formatting passes are required, to interpolate such annotations.
- Placement of a "Table of Contents" in its traditional location, *without* the need for manual collation, (or other post-processing operation), *after* completion of `troff` formatting. In-place formatting of a table of contents requires knowledge of the page numbers, to which the table of contents entries refer, at the point of interpolation; this requires an initial formatting pass, to collect the references into an auxiliary file, which can then be included at the appopriate location, during a further formatting pass.
- Interpolation of intra-document cross references, (especially in the case of forward references), in which the references include page numbers, or a section numbers; as in the case of in-place table of contents interpolation, this requires one (or more) initial formatting passes, in which reference data is collected into an auxiliary file, for inclusion in subsequent passes. Furthermore, when publishing a PDF document, in which cross references are to be represented as dynamic `pdfhref` links, the bounding box co-ordinates for such links *must* be computed *before* the link text is interpolated; this computation is most conveniently performed during preliminary formatting passes, captured in an auxiliary file, and subsequently reinterpreted during a final publication formatting pass.

These single pass formatting limitations *can* be mitigated, by adoption of a *multiple pass* formatting strategy. To facilitate this, for publication of PDF documents, the `groff` program suite includes the `pdfroff` program;[26] (see section 4.1, "The `pdfroff` Program"). This provides a wrapper around `groff` itself; it performs multiple preliminary formatting passes, capturing reference data by filtering it from the `stderr` output stream, and storing it to a temporary intermediate file. This intermediate file is then reinterpreted; along with the original document source, during each successive pass, either until its content stabilizes, or it becomes apparent that stability is unlikely to be achieved, before ultimate reinterpretation to produce the finished PDF document.

It may be noted that, in the absence of a mechanism for passing collected reference data from one formatting pass to the next, multiple pass processing would serve no useful purpose. Fortunately, this is not a problem, because `groff` supports two possible mechanisms for collection, and passing of reference data between passes:–

- The data may be recorded, using `groff`'s `write` request, in an intermediate file which has been *explicitly* initialized by the `open`, (or `opena`), request. This technique requires `groff` to be run in its "unsafe" mode, (enabled by the "`-U`" option), and is not supported by traditional `troff` implementations. Neither `groff`'s `pdfmark` macros, nor the `pdfroff` command, depend on the use of this mechanism; however, users may choose to adopt it for their own purposes, (e.g. in-line interpolation of a table of contents).
- The data may be written — either by use of the `tm` request, or a construct such as `groff`'s `\O` escape — to, and filtered from, the `stderr` data stream. This technique *is* used by `groff`'s `pdfmark` macros, to report `pdfhref` data, and by `pdfroff`, to make this available in subsequent formatting passes.

4.1. The `pdfroff` Program

Implemented as a Bourne shell script, and thus suitable for deployment on POSIX platforms such as GNU/Linux and contemporary Unix systems,[27] `pdfroff` serves as a multi-pass front-end driver for `groff` itself; as such, it offers mitigation of those limitations of single-pass processing which have been identified in the preceding introduction.

---

26. The `pdfroff` program was developed in tandem with the `pdfmark` macros themselves, and is the tool which has been used to format this document itself. Unavailable at the publication time of early releases of this document, later releases of `groff` include support for the `-Tpdf` post-processor, which provides similar mitigating features. Unlike `pdfroff`, which requires only a Bourne shell operating environment, the `-Tpdf` back-end is written in Perl, and thus requires an operating environment with a functional Perl interpreter; this may limit its suitability for use on some host platforms.

27. As a Bourne shell script, `pdfroff` is not *natively* supported on MS-Windows; on this platform, it may be supported by use of a third party application suite, such as Cygwin or MSYS, (or other alternative), which provides a Bourne shell command line interpreter.

Besides external dependencies on some standard POSIX utilities, including `cat`, `grep`, `sed`, `awk`, and `diff`, together with `groff`, and the Ghostscript interpreter, for final production of PDF output, the implementation of `pdfroff` assumes only standard Bourne shell interpreter syntax, (subject to a requirement that the shell itself *must* support shell functions, expressed in terms of the original *standard* Bourne shell function syntax — i.e. support for interpretation of the `function` keyword, as introduced by the Korn shell, and subsequently adopted by the GNU Bourne Again Shell, is *not* required).

Formal documentation for `pdfroff` is provided in its accompanying `pdfroff(1)` Unix manual page.[28] In common with the majority of Unix manual pages, this documentation may be found to be rather terse; thus, a more informal discussion, supported by examples relating to the publication of this document itself, may be found below.

4.1.1. Principles of `pdfroff` Operation

The operation of `pdfroff` may be characterized as a sequence of *six* distinct processing phases:–

1. Initialization: on commencement of `pdfroff` processing, the script sets up its shell environment, checks for availability of each of the required `cat`, `grep`, `sed`, `awk`, `groff`, `diff`, and Ghostscript helper programs, and then parses the command line with which it was invoked. Options which are documented, within the `pdfroff(1)` manual page, as being specific to `pdfroff`, are interpreted in place, recording their effects within the shell environment; other options, and non-option arguments are collected into a deferred options list, and an input files list, respectively, to be passed on for repeated processing by `groff`.

   In the event that standard input is *explicitly* enumerated within the list of input files, or the input files list is empty, (in which case standard input is considered to have been *implicitly* enumerated), then standard input is read by `cat`, and redirected to a temporary file, whence it my be replayed, as required, into the input stream for each subsequent `groff` processing pass.

2. Reference analysis: following initialization, and provided the `--no-reference-dictionary` option has *not* been specified, `pdfroff` enters a loop in which `groff` is executed at least twice, and at most three times, (a fourth cycle of the loop may be initiated, but `groff` will not be executed within it), to compile a reference map for the PDF document, which is to become the ultimate `pdfroff` output. During each of these reference analysis passes, the ultimate `groff` output is discarded, while the standard error stream is captured in a temporary file, whence reference data is filtered, to produce a reference map which is specific to the discarded `groff` output; this will eventually become a reference map which reflects the final state of the ultimate `pdfroff` output document.

   A further, more comprehensive, description of this phase of `pdfroff` operation may be found in the later section 4.1.2, “How `pdfroff` Resolves Cross References”.

3. Front-matter layout: executed *only* when the `--stylesheet=<filename>` option *has* been specified, and the `--no-pdf-output` option *has not* been specified, within the list of arguments passed to the `pdfroff` command, in this processing phase the specified stylesheet file, optionally augmented by additional information which is embedded within the document input file stream, is processed by `groff`, to produce a PostScript® rendition of an optional cover sheet, and additional (optional) front-matter, which is to be placed at the beginning of the eventual PDF output document.

   Once again, this phase of operation will be explored further, in the later section 4.1.4, “Using a `pdfroff` Style-Sheet to Specify Document Front-Matter”.

4. Table of contents generation: this phase is *always* executed, *unless* either the `--no-pdf-output` option, or the `--no-toc-relocation` option is specified, on the `pdfroff` command line, or as a result of evaluation of hints within the document input file stream, this phase of operation implements a rudimentary mechanism for collation of the final PDF output document, emulating the traditional `groff` technique, whereby table of contents entries are collected into a diversion, printed at the end of the document, and subsequently relocated manually, to their normal position between the front-matter, (if any), and the body of the document. Methods for controlling this phase of operation are further developed, and explained in section 4.1.5, “How `pdfroff` Collates Tables of Contents”.

5. Document body formatting: this phase is also *always* executed, *unless* the `--no-pdf-output` option is specified; it is responsible for formatting the body of the document, compiling it to PostScript® code, in preparation for combination with the front-matter, and table of contents components from the preceding two phases, to produce the final output document. Further details of this phase of `pdfroff` operation may be found in section 4.1.6, “How `pdfroff` Formats a Document Body”.

28. See https://download-mirror.savannah.nongnu.org/releases/groff-pdfmark/pdfroff.1.pdf for a PDF rendition of this manual page.

6. Final PDF document production: unless suppressed, by specification of the `--no-pdf-output` option, completion of phases 3, 4, and 5 results in the production of between one and three intermediate output files, each of which in in PostScript® format. Regardless of whether the final output is desired in PostScript® format, or is to be converted to PDF, this final processing phase uses the Ghostscript post-processor to combine[29] the intermediate files, creating a single output document file, as described in section 4.1.7, "How `pdfroff` Assembles a Finished Document".

   On completion of this processing phase, *unless* the `--keep-temporary-files` option is in effect, all intermediate files,[30] created during the earlier phases of operation, are deleted, and `pdfroff` terminates.

4.1.2. How `pdfroff` Resolves Cross References

As has already been noted, in section 4.1.1, "Principles of `pdfroff` Operation", (with the proviso that this entire phase of operation will be suppressed, if the `--no-reference-dictionary` option has been specified), `pdfroff` performs iterative resolution of cross references during the second phase of its operation; a maximum of four iterations are performed, in accordance with the following procedure:

- *Before* entering the first cycle of the iterative loop, the three internal shell variables, `WRKFILE`, `REFCOPY`, and `REFFILE`, are defined to represent the names of three working files; the first two of these represent temporary files, which will be named, and created using the best practicable mechanism afforded by the operating system, to support secure read/write access for files created, and used, by shell script processes. The third may also represent a similarly created temporary file; however, it may equally well become a permanent output file, if the `--reference-dictionary=<`*`filename`*`>` option is specified, in which case it will be named accordingly.
- Having specified appropriate working file names, the file identified by the `REFFILE` variable is created with no content, and that identified by `REFCOPY` is created with arbitrary (non-empty) content; the `pdfroff` process then enters the iterative reference resolving loop.
- At the start of each cycle of the reference resolving loop, the content of the two files identified by `REFCOPY`, and `REFFILE` is compared; if the two compare as *identical*, all references are deemed to have been resolved, and the loop is terminated. (Note that this loop termination condition *cannot* be satisfied at commencement of the first cycle of the loop, because the two files were initialized with non-identical content; thus, the first cycle *must always* be completed, and loop termination *cannot* occur before the file comparison is performed at the start of the *second* cycle).
- On commencement of a new loop cycle, when the preceding loop termination condition has *not* been satisfied, if loop execution has entered its *fourth* cycle, a warning message is written to the `stderr` stream, and the loop is terminated *without* complete resolution of references; (this is a safety measure, to prevent `pdfroff` becoming stuck in an interminable loop).
- When loop execution is allowed to continue into a new cycle, the content of the file represented by the `REFCOPY` variable, whether defined by initialization, or as carried forward from the immediately preceding cycle, is discarded, and the content of the corresponding file represented by the `REFFILE` variable is moved into its place; thus, at commencement of each new reference resolution cycle, the `REFCOPY` file represents the content of the `REFFILE` file, as it stood at the end of the *immediately preceding* cycle, (or as initialized, if executing the *first* cycle).
- Following the update of the `REFCOPY` file content, loop execution continues by running `groff`, processing all specified input files, in their specified order, to collect analytical data relating to the eventual structure of the finished document. The required analytical data is written to `groff`'s `stderr` output stream, as directed by the `pdfhref` macro, either via `tm` requests, or by exploitation of `groff`'s extended '`\O`' capability, as originally developed for use by the `grohtml` processors, to map the page co-ordinates for `pdfhref` link bounding boxes; `stderr` output is captured in the file designated by the `WRKFILE` variable, simply overwriting any content which was collected during preceding loop execution cycles; `groff`'s intermediate `stdout` stream data is discarded, *without* further processing by the `grops` post-processor.
- Still within the loop execution cycle, the file designated as `WRKFILE` is reprocessed, using a simple `awk` filter to extract pertinent reference dictionary content, redirecting it into the file named by the `REFFILE` variable.

---

29. Strictly, if there is *only one* intermediate output file, and the `--emit-ps` option is in effect, no combination is actually required; however, the single intermediate output file is reprocessed through Ghostscript, regardless.

30. If the `--reference-dictionary=<`*`filename`*`>` option has been specified, the reference dictionary ceases to be classified as an intermediate file, and is not deleted when `pdfroff` terminates.

- Although *not* strictly necessary for reference resolution,[31] if executing the *first* cycle — and *not* repeated in any subsequent cycle — of the reference resolving loop, the working file designated by `WRKFILE` is further reprocessed, to facilitate extraction, and evaluation, of optional `pdfroff` processing hints, as described in section 4.1.3, "Using In-Document Hints to Control `pdfroff` Processing Options".
- As the final step, within each execution cycle of the reference resolving loop, the content of the `WRKFILE` is reprocessed one final time,[32] extracting dynamically propagated document content, and redirecting it into designated files, as identified by hints from the preceding step. On completion of this final step, execution of the reference resolving loop continues with the commencement of a new cycle.

On normal termination of the preceding loop, one further processing step is required to complete the resolution of *internal* cross references, and to compile the final reference dictionary:

- The `WRKFILE` is processed one final time, using a further `awk` filter to extract any `grohtml` records, which have been generated due to the placement of zero-width markers, inserted by the `pdfhref` macro, to mark the position of link "hot-spots", within the document; the `awk` filter extracts the page number, and page co-ordinate references from these records, and reformats them as '`pdfhref Z`' records, which are then appended to the `REFCOPY` file, for subsequent use during the later phases of final document production.

Of the three working files, created during this phase of `pdfroff` processing, the `WRKFILE` is not required in any later processing phase; the `REFFILE` *may* be exported as a permanent *external* reference dictionary, otherwise it too is of no further use; only the `REFCOPY` file, which incorporates *both* the *external*, and the *internal* constituents of the reference dictionary, is reused in later phases of the publishing process. Nonetheless, all three remain in place until the `pdfroff` process itself terminates, when *all* temporary files, which the process has created, are normally[33] deleted.

4.1.3. Using In-Document Hints to Control `pdfroff` Processing Options

Although it has been informally supported since the release of `groff-1.22.3`, when the `spdf.tmac` binding macros for `ms` added the request:

```
.tm pdfroff-option:set toc_relocation=enabled
```

within the implementation of their `TC` macro, and `pdfroff` added code to retrieve the resultant output from the reference resolving `WRKFILE`, to interpret the implied hint, such that the effect of the `--no-toc-relocation` option is assumed, *unless* the associated hint is actually present in the `WRKFILE` data stream, this feature was not formally implemented until the `groff-pdfmark-20230317.1` release of `pdfroff`.

The formal implementation, of this feature now depends on the use of the new `pdfroff` macro,[34] with the hint in `spdf.tmac` now being specified as:

```
.if d pdfroff .pdfroff option toc_relocation=enabled
```

More generally, usage of the `pdfroff` macro, to specify optional processing hints, takes the form:

```
.pdfroff option <variable-name>=<value>
```

with <*variable-name*> (currently) being restricted[35] to either of `toc_file`, or `toc_relocation`, the effects of which will be considered further, in section 4.1.5, "How `pdfroff` Collates Tables of Contents", or alternatively, the variable `preserve_blank_pages`, which accepts a value of `toc`, `body`, or `all`, to control how *entirely* blank pages are processed during collation of tables of contents, within the body of the document, or in both of these contexts, respectively, thus providing an in-document alternative to the use of the `--no-kill-null-pages` option; (see section 4.1.7, "How `pdfroff` Assembles a Finished Document").

---

31. Releases of `pdfroff`, pre-dating `groff-pdfmark-20230317.1`, performed this hint evaluation *after* completion of the reference resolution loop; however, to the extent that such hints *may* result in propagation of dynamically generated document content through the `WRKFILE`, which *may* impact the reference resolution process, (e.g. due to references embedded in a dynamically generated table of contents), the effect of this early evaluation may become significant.
32. This final step, within the reference resolution loop, was not performed in any release of `pdfroff` pre-dating `groff-pdfmark-20230317.1`; in earlier releases, a new cycle of the loop was initiated *immediately* following the update of `REFFILE` content.
33. Temporary files, created by `pdfroff`, are normally deleted on process termination, *unless* the `--keep-temporary-files` option has been specified.
34. The `pdfroff` macro is defined in the new macro file, `pdfroff.tmac`; this is loaded each time `pdfroff` invokes `groff`, and is not intended to be used in any other context; doing so may produce unpredictable results.
35. The `pdfroff` implementations, released with `groff-1.22.3` (and later), and in `groff-pdfmark` up to, and including, `groff-pdfmark-20230317.1`, did *not* impose this restriction; consequently, these earlier `pdfroff` releases may be vulnerable to an arbitrary code execution attack, when processing untrusted document mark-up.

4.1.4. Using a `pdfroff` Style-Sheet to Specify Document Front-Matter

Of the six `pdfroff` processing phases, identified in section 4.1.1, "Principles of `pdfroff` Operation", *three* perform document formatting, producing three separate output document components in PostScript® format, in preparation for collation, and final assembly of the finished document, either as a finished PostScript® document, or, more commonly, as a finished PDF document. In the first of these, which section 4.1.1 identifies as phase no. 3 in the enumeration of processing phases, `pdfroff` applies a specified style-sheet, in conjunction with meta-data abstracted from the primary document source file, or files,[36] to format the document front-matter; this is saved, in its own individual (temporary) PostScript® component file, to be collated, and subsequently assembled into the finished document, becoming the *first* component of the finished document output file.

As already noted, in section 4.1.1, style-sheet processing is performed only if `pdfroff` is invoked with a command, such as that which may have been used to format this document itself, which includes an *explicit* formal specification for the "`--stylesheet=<`*`filename`*`>`" option, in the form:

```
pdfroff -mspdf --stylesheet=cover.ms pdfmark.ms > pdfmark.pdf
```

This causes `pdfroff` to perform a *single* `groff` formatting pass, in which the input file "`cover.ms`" is read, in its *entirety*, followed by front-matter specific meta-data extracted from "`pdfmark.ms`", to produce an intermediate PostScript® front-matter component file, which is saved only until it has been collated into the finished document, as described in section 4.1.7, "How `pdfroff` Assembles a Finished Document".

The input file, which is nominated as the "`<`*`filename`*`>`" argument of the "`--stylesheet=<`*`filename`*`>`" option, ("`cover.ms`" in the example above), *must* be provided by the document author. In the simplest practicable scenario, this could be a basic `groff` input file specifying the content for the front-matter section of the single document, which is the designated output of a single particular invocation of `pdfroff`; such a "style-sheet" file is simple, and requires no additional meta-data input from the primary document source files, ("`pdfmark.ms`" in the preceding example), but it does suffer from the disadvantage that it is specific to just *one* document, (and thus, barely merits description as a front-matter "style-sheet").

Although the simple front-matter formatting technique, alluded to in the previous paragraph, is compatible with the operation of `pdfroff`, a more sophisticated, generic style-sheet handling capability is also supported, and may be preferred; its principal advantage is that a single, generic front-matter style-sheet, may be suitable for use with more than one document, with document-specific content being specified within, and conveyed from, a meta-data section within the primary document source files. This generic style-sheet technique has been adopted for formatting of the front-matter of this document, and usage examples may be drawn from its accompanying `cover.ms` style-sheet file, and the meta-data specification within its primary `pdfmark.ms` source files.

When designing a generic front-matter style-sheet, careful consideration should be given to the interaction between the style-sheet itself, and the meta-data section, or sections, which are extracted from the primary input files; in particular, it should be noted that the style-sheet will have been read, in its entirety, before *any* meta-data is encountered. Thus, while it is reasonable that the style-sheet should specify any "boiler-plate" text, which is to be reproduced within the front-matter of any dependent document, within the style-sheet itself, such "boiler-plate" text should normally be encapsulated within macro, or string definitions, so that its eventual output may be deferred until called out, on request from within the document meta-data.

The meta-data, which specifies the document-specific variant content of the front-matter, and directs the formatting activity of the style-sheet, is *always* read from the primary document source files; it is identified by its placement between a pair of macro calls, to the nominally named[37] macros, `CS` at the start of each meta-data section,[38] and `CE` at the end, thus:

```
.CS
.\" ... document-specific meta-data appears here ...
.CE
```

36. In this context, "primary document source files" refers to the aggregate of all input files, which are *explicitly* specified on the `pdfroff` command line, read in the order in which they are so specified.

37. The macro names, `CS` and `CE`, are the defaults assumed by `pdfroff`, to mark the start, and the end of a meta-data section, respectively. These defaults may be overridden, by assignment of alternative macro names to the `CS_MACRO` and `CE_MACRO` environment variables respectively; however, unless there is some particularly compelling reason for it, such reassignment of the macro names is strongly discouraged.

38. It is permissable for the primary document source files to specify more than one meta-data section, and `pdfroff` will interpret them all; however, the processing of multiple meta-data sections, and in particular the execution of more than one instance of the `CE` macro, introduces additional complexity to the design of the style-sheet, so it is recommended that *no more than one* such section should be specified.

It may be observed that CS and CE are not defined as standard groff macros; thus the onus is placed on the document author, and the front-matter style-sheet designer, to ensure that appropriate definitions are provided;[39] furthermore, *different* definitions of each macro will normally be required, when processing a style-sheet for formatting front-matter, and when formatting normal document content.

From the foregoing, it may be inferred that the style-sheet should be implemented as a collection of macro, string, and possibly numeric register definitions, including, as a bare minimum, implementations of the CS and CE macros, which will drive the formatting of the document front-matter, while the document source should arrange for provision of alternative definitions for this pair of macros, to handle embedded meta-data sections appropriately, while formatting the remainder of the document.

In the case of normal document formatting, other than within the front-matter context, appropriate handling of meta-data may be as simple as ignoring it. For users of groff's "ms" macros, when these are used in conjunction with groff-pdfmark's spdf.tmac binding macros, as previously noted in footnote,[39] suitable definitions for CS and CE, to achieve this behaviour, are provided, *without* the need for any specific provision by the document author; for those who do not wish to, or simply cannot, use spdf.tmac, equivalent behaviour — without error handling — may be achieved by providing macro definitions similar to:

```
.de CS
.   ig CE
..
.de CE
..
```

Conversely, the front-matter style-sheet *must* implement alternative definitions for *both* CS and CE, together with definitions for any other macros which are intended to be called out from the document's meta-data section, (or sections); the aggregate effect of calling such style-sheet macros, from the document's meta-data sections, beginning with the first CS call, and ending with the *last*[40] CE call, and ignoring all other content of the document source files, should result in formatting, and output of the front-matter component of the finished document. Typically, the style-sheet should define the CS macro, initially, to set up the page layout controls for formatting the cover sheet, (if any), and any such controls which may also apply throughout the document's front-matter; for example, a style-sheet for use in conjunction with groff's ms macros — based on the implementation of the style-sheet for this document itself — might define the CS macro to be something like:

```
.de CS
.   nr HM  0
.   nr PO  2.1c
.   nr LL 17.1c
.   nr HY  0
.   nr PS 24
.   nr VS 30
.   nop
.   sp |5.9c
.   CD
.   fam T
..
```

anticipating that the first meta-data section encountered will commence with a specification of text, which is to be set as a centred 24pt title block, in Times-Roman font, and which is to be placed 5.9cm below the top edge of the first page of the front-matter, (which will take the form of a cover sheet).

To complement the CS macro definition, a definition for the CE macro is also required. Continuing the preceding example, and again with reference to the usage within this document itself, (which uses only one embedded meta-data

---

39. The spdf.tmac macro package *does* provide definitions of CS and CE, with CS having an effect equivalent to that of ".ig CE", and CE serving as a do-nothing macro, (albeit with the addition of diagnostic checks in both, to ensure that CS and CE are correctly paired at point of use), for marking the end of the ignored block. The effect of these definitions is that meta-data sections will not be interpreted, in any way, during normal document processing, which may be suitable for many documents; however, alternative definitions *will* surely be required, within any front-matter style-sheet.

40. While the first CS call is trivially easy to identify, it is the difficulty of recognizing the last CE call which complicates the handling multiple meta-data sections, and hence, why use of multiple such sections is not recommended.

section), the CE macro picks up the front-matter formatting towards the bottom of the cover sheet, adding an image, the stipulated front-cover text, and ultimately, proceeding to incorporate a copyright assignment page:

```
.de CE
.   DE
.   sp |17.5c
.   PSPIC gnu.eps
.   nr PS 19
.   CD
.   fam H
.   tkf HR 10z 2p 20z 4p
.   nop \H'-4z'A GNU MANUAL\H'0'
.   DE
.   \" ... additional macro code follows here ...
.   \" ... this may, for example, add a copyright assignment page ...
.   \" ... or any other appropriate front-matter content ...
..
```

Notice that, in this particular example, the CS macro ends, leaving an open CD display block, (i.e. a standard ms centred display); the complementary CE macro assumes that this will have remained open, and immediately closes it, *before* proceeding with the image output. Any meta-data content, which has been specified between the opening CS call, and its corresponding CE, will be processed *after* completion of the CS call, and *before* commencement of the CE. This may include directly specified text, to be formatted within the open display block, or other macro calls, which will be executed as encountered; if any of this meta-data content causes the initial centred display block to be closed, then it is assumed that a new display block — not necessarily centred — will have been opened, *before* control passes to CE; any meta-data content, which is to be formatted *after* control has been passed to CE, *must* be saved — in string space, for example — so that it may be reinterpreted *during* execution of CE.

An examination of the source mark-up for this document, which is provided in the accompanying pdfmark.ms (primary source), and cover.ms (style-sheet) example files, will reveal that the embedded meta-data does, indeed, depend on additional macros, beyond the required CS and CE implementations. All of these additional macros, (including some which replace standard ms implementations), are defined within cover.ms; it may be observed that all, *both* in implementation *and* in usage, comply with the requirements laid out in the preceding paragraphs.

4.1.5. How pdfroff Collates Tables of Contents

When formatting documents with troff, and directing output to a hard-copy typesetting device, a traditional method of generating tables of contents is to collect copies of the section headings, and their corresponding page numbers, in a diversion, which is printed at the *end* of the document, whence it is then physically separated, and *manually* moved to its natural position, immediately following the front-matter. This technique can be readily supported by groff, and remains useful for generation of tables of contents; although awkward to automate, and alternative techniques, such as those which will be described in section 4.4, "An Alternative Technique for Generating Tables of Contents", may offer better performance, it *does* serve as the default basis for table of contents collation, used by pdfroff.

To facilitate separation of the formatted table of contents from the formatted document body, pdfroff invokes groff *twice more*, after completion of the reference resolution phase, (see section 4.1.2, "How pdfroff Resolves Cross References"), to format, and temporarily save, *two* intermediate PostScript® copies of the complete document; the first of these will ultimately become the table of contents component, to be assembled together with, and preceding the second, which will become the main document body, with both preceded by the front-matter (if any), to create the finished document, (see section 4.1.7, "How pdfroff Assembles a Finished Document").

As previously noted, in section 4.1.1, "Principles of pdfroff Operation", where it is enumerated as phase no. 4 in the sequence of operations, pdfroff's default table of contents generation procedure is *automatically* executed, *unless* steps are taken to disable it. It may be disabled:

- *Explicitly,* by specifying the --no-toc-relocation option, when the pdfroff command is invoked.
- *Implicitly*,[41] if no toc_relocation=enabled hint is detected, when resolving references; (however, it is implicitly assumed that this hint is present, if the --no-reference-dictionary option is specified, in which case, no reference resolution is performed).

41. Implicit control of toc_relocation was first introduced for the groff-1.22.3 release; it is unsupported in earlier releases.

When pdfroff runs groff, to generate a separate table of contents component, which will eventually be combined with the front-matter component (if any), and the document body component, to assemble the document in its finished form, it indicates the intent of this phase of operation by passing a "-rPHASE=1" register assignment option. The effect of running groff, in the absence of any special consideration of this PHASE assignment, might be expected to be the production of a formatted copy of the complete document, with the table of contents placed *at the end*, whereas, what is required for final document assembly, is to discard the entire document body, which precedes the table of contents in this formatted component, leaving *only* the formatted table of contents, as its effective residual substance.

To facilitate the eventual removal of document body content, from the table of contents component, pdfroff expects the document author to make arrangements to place groff in its "pen-up" output state, (selected by placing a \O[0] escape in the input stream), *before* the body content, and to restore the "pen-down" state, (by complementary placement of a \O[1] escape), at the start of the table of contents, when the PHASE register is defined, with a value of one. Such arrangements may be made, conveniently, within a macro package which controls the overall document format, (for example, spdf.tmac handles the arrangements automatically, without any requirement for further intervention by the document author, when the table of contents entries are specified using the XS, XA, and XE macros, and the table of contents, itself, is eventually output using the TC macro).

Of course, pdfroff does *not* impose a requirement for the exclusive selection of spdf.tmac as the primary macro package for document formatting. It may be practicable to adapt *any* primary macro package, of the document author's choice, to emulate spdf.tmac's behaviour; alternatively, if the document author deems it impractical to adapt the chosen macro package, the required emulation may be achieved *directly* within any document's input data stream. In either case, some basic ground rules *must* be respected:

- Neither any macro package, nor any document's input data stream, is permitted to interfere with pdfroff's assignment of the PHASE register; it should *never* be set, or modified in any way, other than as a result of direct assignment by pdfroff itself.
- Initialization of groff's output state controls *must* be completed, *before* commencement of the output of the first page of the formatted document; this initialization may be achieved, most conveniently, by inclusion of mark-up[42] similar to:

  ```
  .mso opmode.tmac
  .
  .nr PDF-TOC-ONLY   1
  .nr PDF-BODY-TEXT  2
  .
  .OP \n[PDF-BODY-TEXT]
  ```

  early in the document input data stream, or better still, within a macro package, such that it will be executed soon after the start of groff processing, and in particular, *before* any output is generated; this ensures that groff starts in the correct output state, with respect to pdfroff's PHASE register, for collection of table of contents data, within its own diversion, based on mark-up within the body of the input data stream.
- When the input data stream has been fully processed, the table of contents diversion should be closed, and, in preparation for flushing it to the output, a new page should be started, and the output state should be adjusted, by execution of:

  ```
  .OP \n[PDF-TOC-ONLY]
  ```

  after which, the substance of the table of contents diversion should be written, together with any desired page headings, and footers, to the document output stream. For convenience, consideration should be given to encapsulation of this entire sequence of steps, which is required to prepare for, and to complete the output of the table of contents, in a macro which is analogous to spdf.tmac's TC; this may then simply be invoked at the end of the input data stream. Such consideration is particularly recommended when writing a primary macro package, or a binding macro package, which is intended to be used in conjunction with pdfroff.

The effect of the OP macro, as used in the foregoing, is to insert a \O[1] escape into the input stream, when the value of its single argument is equal to the value of pdfroff's PHASE register,[43] and a \O[0] escape otherwise, while recording the effective output state in a register called OPMODE. Thus, within pdfroff's table of contents collation phase, the initial OP invocation, with an argument value which is equivalent to 2, generates the required \O[0] escape — which does not actually eliminate the document body, but causes groff to emit one *entirely* blank page for each page of body content — while the second invocation, with an argument value equivalent to 1, inserts a \O[1]

42. This example mark-up depends on the opmode.tmac helper macro package, which is distributed as an integral component of groff-pdfmark-20230317.1, and later releases; it is backwardly compatible with earlier groff releases of pdfroff.

43. If the PHASE register is not defined, the OP macro simply records an effective output state of 1, in OPMODE, but *does not* insert any \O escape sequence, into the input data stream, and thus, does not change groff's actual output state.

escape, resulting in output of the formatted table of contents; the initial blank pages are subsequently removed during final document assembly, (see section 4.1.7, "How pdfroff Assembles a Finished Document").

4.1.6. How pdfroff Formats a Document Body

To format the body of any document, pdfroff repeats the default groff process that it uses for collation of a table of contents, (as described in section 4.1.5, "How pdfroff Collates Tables of Contents"), *except* that the PHASE register is set to a value of two, (rather than the value of one, which is used when generating a table of contents), by the assignment "-rPHASE=2", which is passed as a command line argument when pdfroff invokes groff, and the ensuing groff output is written to a differently named intermediate output file. The effect of changing the PHASE register assignment, assuming that the OPMODE controls are implemented, and managed as previously described in section 4.1.5, is to place groff in its "pen-down" output state when formatting the document body, and then to switch to the "pen-up" state, only if processing ultimately progresses to the output of an appended table of contents.

An important consideration, for authors writing documents to be formatted by pdfroff, or for those implementing macro packages to facilitate this, is that the default formatting process expects a single input data stream, in which the document body will be processed *first*, and a table of contents will be appended, *at the end*. This input data stream will be read *twice*, to produce *two* separate intermediate output files, which will eventually be conjoined, (see section 4.1.7, "How pdfroff Assembles a Finished Document"), to produce *a single* final output document file.

If the document author, or the macro package implementor, neglects the interpretation of pdfroff's PHASE register assignment, or the implementation of the associated OPMODE handling described in section 4.1.5, the two intermediate output files will exhibit *identical* content, and their conjunction will result in a finished document which contains *two* copies of the formatted document body, each of which will be followed by a copy of the formatted table of contents.

Conversely, when pdfroff's PHASE register interpretation, and OPMODE handling *have* been appropriately addressed, the intermediate output with PHASE=1, as already noted in section 4.1.5, will comprise a sequence of entirely blank pages, followed by the formatted table contents, while that for PHASE=2 will contain the desired formatted document body, which will then be followed by a further sequence of blank pages, with one corresponding to each and every non-blank page of the table of contents. Conjunction of this pair of intermediate files will result in the desired content, in the final document file, but it will be padded by a (possibly very large) number of blank pages; these blank pages are normally unwanted; they will be removed in the final document assembly process, (see section 4.1.7, "How pdfroff Assembles a Finished Document").

An interesting possibility, when formatting a document body, is that, even when the input stream has been configured to generate the document body output, with appended table of contents, and a sequence of blank pages, corresponding to the pages of the document body, *is* required at the beginning of the table of contents intermediate file, it is *not* actually necessary to emit the sequence of appended blank pages which would correspond to the pages of the table of contents, when generating the document body intermediate file. In fact, it is fairly straightforward to test the effective value of the PHASE register, and to omit the formatting of the appended table of contents, when the register value is two, or greater. This optimization[44] may be performed, either within, and at the end of, the document source:

```
.ie d pdfroff \{\
.   ie \n[PHASE]>1 .nr DO-TOC 1
.   el .nr DO-TOC 0
.\}
.el .nr DO-TOC 1
.if \n[DO-TOC] \{\
.\" ... code to emit formatted table of contents goes here ...
.\}
```

or, perhaps more conveniently, within a table of contents formatting macro, such as, for example, in spdf.tmac's implementation of its TC macro:

```
.de TC
.if d pdfroff .if \\n[PHASE]>1 .return
.\" ... code to emit formatted table of contents goes here ...
..
```

which then requires only that the document source ends by calling this TC macro.

---

44. These examples assume that the table of contents should *always* be output, *except* within the scope of pdfroff's document body formatting procedure, when they may be optimized out. Notice that there is no check for existence of the PHASE register, before testing its value, (as might be considered desirable to avoid groff warnings); no such check is necessary, because pdfroff *guarantees* that the register will have been defined, in *any context* in which a user-visible warning could be raised, so checking for pdfroff alone is sufficient.

4.1.7. How `pdfroff` Assembles a Finished Document

After `pdfroff` has completed each of the processing phases, enumerated as phase no. 1 to phase no. 5 in section 4.1.1, "Principles of `pdfroff` Operation", *at least one*, and *at most three* intermediate PostScript® output files will have been created, (in addition to the temporary files designated by the `WRKFILE`, `REFFILE`, and `REFCOPY` shell variables). If only one intermediate output file is created, it *must* be the document body component; if more than one such file is created, one *must* be the document body component, while any others may be *either one* of, or *both* of a front-matter component, and a table of contents component. In this final phase of `pdfroff` processing, the entire complement of created intermediate output files is conjoined, in the order:

- Front-matter component (if present);
- Table of contents component (if present);
- Document body component

to assemble the finished output document, either merging all component content into a single PostScript® document, (if the `--emit-ps` option has been specified), or normally, merging all content, and simultaneously converting to PDF.

Depending on the particular complement of intermediate output files which have been created, document assembly proceeds as follows:

- If a front-matter intermediate output file has been created, its content is simply copied, without change, to the beginning of the final output file.
- As has been noted previously, in section 4.1.5, "How `pdfroff` Collates Tables of Contents", if a table of contents intermediate output file has been created, it *will* contain the formatted table of contents, but this will be preceded by a sequence of blank pages, with one blank page for each page in the document body; these blank pages will (usually) not be wanted in the finished document, so, before merging this intermediate file content, it is passed through a `sed` filter, to remove them; the residual comtent is then copied into the final output file, either following the front-matter, if present, or otherwise, at its beginning.
- Finally, the content of the document body is read from its respective intermediate output file, and merged into the final output file, following any content which had previously been merged from front-matter and table of contents intermediate output files; as in the case of table of contents intermediate output, this document body output is also passed, by default, through the same blank page removal `sed` filter.

It may be observed that, by default, `pdfroff` will filter *both* the table of contents intermediate output, if any, and the document body intermediate output, to remove blank pages. While the intent of this is to discard *phantom* blank pages, which precede the actual table of contents, and which follow the document body, it must be understood that it will actually remove *all* blank pages — *both* those which promote this intent, *and* any others which may appear *within* the table of contents, and the document body. This may, or may not, be desirable behaviour — it probably *is*, within the table of contents, but is less so within the document body, especially if the trailing blank pages, resulting from "pen-up" formatting of an appended table of contents, have been optimized out, as suggested in section 4.1.6, "How `pdfroff` Formats a Document Body".

Some control of blank page removal may be achieved by specification of the `--no-kill-null-pages` option — or `--no-kill-null-pages[=<whence>]`, from the `groff-pdfmark-20230406.1` release onwards — on the `pdfroff` command line, or alternatively, and once again, only from the release of `groff-pdfmark-20230406.1` onwards, by use of the `preserve_blank_pages=<whence>` in-document hint. When supported, the `<whence>` argument may take a value of "`toc`", "`body`", or "`all`", with "`all`" being the default, and equivalent to the original behaviour; its effect is to suppress removal of *all* blank pages originating from the table of contents intermediate output file, *and* from the document body intermediate output, which is probably not the desired effect. More useful behaviour may be to retain the default removal of blank pages originating from the table of contents intermediate output, while suppressing it for those originating within the document body; this may be achieved by invoking `pdfroff` with a command such as:

```
pdfroff --no-kill-blank-pages=body ...
```

or by including the equivalent in-document hint:

```
.pdfroff option preserve_blank_pages=body
```

within the document source file, (perhaps making it conditional on some appropriate condition, such as selection of a duplex printing configuration, for example).

4.2. Preparing Documents for On-Screen Reading versus Hard-Copy Printing

When preparing a PDF document, which is to be optimized for reading on a video display screen, it is reasonable to make formatting choices such as:

- Set up of the page layout, such that the left-hand and right-hand margins are of equal width, and remain unchanged between formatting of recto (odd-numbered) pages, and verso (even-numbered) pages; such a layout may be conveniently, and simply, achieved by assignment of suitable, and invariant, values for `groff`'s page offset, and line length settings.
- Elimination of *entirely* blank pages. These, (especially in the case of blank verso pages), may be inserted when formatting for printing on a duplex-capable hard-copy output device, to ensure that new chapters, or major sections, commence on a new recto page; however, they serve little purpose, and can be distracting, when reading a document on-screen. The elimination of such blank pages is performed *automatically*, by `pdfroff`, *unless* this capability is disabled *explicitly*, by the user.

Conversely, when preparing a PDF document which is suitable for subsequent printing on a hard-copy typesetting device, different formatting choices may be more appropriate; for example:

- A wider page margin may be desired, on whichever side of each page will lie adjacent to the spine, to provide a "binding allowance", when the document is to be bound; any such "binding allowance" should be *added to* the nominal page offset, on recto pages, and *deducted from* it, on verso pages.
- When the typesetting device supports *duplex* printing, (i.e. printing on *both* recto and verso pages), *and* this mode of printing is to be used, then any blank (normally verso) pages which have been inserted, to force the following content to appear on a particular side of the printed page, *must* be preserved within the PDF document structure. When using `pdfroff`, to format the PDF document, such blank pages will normally be eliminated; the `--no-kill-null-pages` option, or its corresponding `preserve_blank_pages` in-document hint must be specified, to override `pdfroff`'s default behaviour.

4.2.1. Establishing a Page Layout for On-Screen Reading

When preparing a PDF document, which is ultimately intended for reading on a video display screen, the page layout will normally be characterized by arrangement of text between equal width margins, on *both* left-hand and right-hand sides of the page; these margins will, typically, be defined at the start of document processing, and, aside from local internal variations in indentation, will remain unchanged throughout the document, without regard to whether any individual page would be printed as a recto (odd-numbered) page, or a verso (even-numbered) page.

The mechanics of setting the left-hand and right-hand page margins depend on the user's choice, if any, of document formatting macro package. Fundamentally, the left-hand margin is equivalent to whatever page offset may have been set, by invocation of `troff`'s "`.po`" request, the text width is set by invocation of `troff`'s "`.ll`" request, (with an alternative width for three-part titles, set by the "`.lt`" request), and the right-hand margin is simply the remnant of the physical page width,[45] after deduction of the page offset and text width. Typically, users do not invoke these fundamental `troff` requests directly, but rely on the features of a higher level macro package to invoke them when appropriate, to apply user-defined settings, which are established in numeric registers; for example, the `ms` macro package will invoke the "`.po`" request, to achieve the effect of

```
.po \n(POu
```

at the start of every new page, thus setting the effective left-hand margin for the page, to whatever value the user has assigned to the `PO` register; similarly, it will invoke the "`.ll`" request, with the effect of

```
.ll \n(LLu
```

on each paragraph transition, (in conjunction with other local adjustments, based on the settings of other layout control registers),[46] to maintain the fundamental text width at whatever value the user has assigned to the `LL` register.

It may be observed, from the foregoing, that setting up an overall page layout, which is suitable for on-screen reading, is normally a one-time process; once established, at the start of the document formatting process, other than when making localized temporary indentation adjustments, there is usually no need to change it.

---

45. Traditional `troff` provides no user-visible indication of the physical page width. GNU `troff` *does* define a page width, at start-up, based on standard paper size specifications, but it does not make this *directly* visible to the user; it may be *indirectly* inferred, *immediately* after start-up, *before* any subsequent "`.ll`" request has been invoked, by adding two inches to the initial value stored in the "`.l`" register.

46. Details of all page layout control registers, as used by the `ms` macros, may be found in the `groff_ms(7)` manual page. Similar details, relating to other macro packages, should be available in their respective documentation.

4.2.2. Establishing a Page Layout for Hard-Copy Typesetting

For the most part, when it is ultimately intended that a PDF document will be printed, on some hard-copy typesetting device, the mechanics of establishing the overall page layout are identical to those which have been described already, in section 4.2.1, "Establishing a Page Layout for On-Screen Reading"; a fundamental difference arises, only in the case where it is desired to add a "binding allowance" to either the left-hand, or the right-hand page margin.

When such a "binding allowance" is desired, the adjustment to the set-up of the page layout may be trivial, or relatively more complex, depending on whether:

- The ultimate output is to be printed one-sided, (typically as recto-only pages, each with blank verso): in this case, the "binding allowance" will be added to the left-hand page margin *only*, requiring no more than a trivial increment, equivalent to the width of the "binding allowance", in the initial assignment of the page offset, (which *automatically* results in a corresponding reduction in the width of the right-hand page margin); as in the case of the on-screen layout, this one-sided printing layout requires no subsequent adjustment.
- The ultimate output is destined for two-sided printing, (on *both* recto *and* verso pages): in this case, the set-up of the page layout becomes relatively more complex, because the "binding allowance" must be *added to* the underlying page offset, to set a wider left-hand margin on recto pages, but must be *deducted from* it, to set a correspondingly narrower left-hand margin on verso pages; consequently, the page offset must be adjusted, at the start of *every* page, to achieve the desired alternation of left-hand and right-hand margin widths.

  Neither traditional `troff`, nor `groff`, provide a standard method for configuration of the required margin width alternation; it is achievable, however, through page traps, which must be appropriately specified by the user. Section 3.1.3 "Layout Adjustment to Support Duplex Printing" provides an illustration of how this may be implemented, when using the `ms` macros, by exploiting the existing bottom-of-page trap to adjust the `PO` register setting at the *bottom* of every page, such that it will take effect, to adjust the page offset as required, at the start of the *following* new page; this technique should be adaptable for use with other macro packages, or even within a user-defined trap, should the user choose to rely on basic `groff` requests alone.

4.2.3. Ensuring that Content is Printed on a Particular Side of the Page

When printing a document, in a duplex format, style conventions may dictate that certain sections, such as tables of contents, body content, and also any appendices, any collected bibliographic references, and any index entries, which may be present, should *always* begin on a particular side of the page, (usually the recto); such conventions may, or may not, be extended to require that indiviual chapter headings, or major section headings, should also be placed on a particular side (again, usually the recto) of a new page. When such conventions are applied, the usual practice is to insert a page break *immediately* before the content which is to be so placed; however, if the content which immediately precedes this page break is already being printed on the side of the page, on which it is desired to place the following content, then one page break will not suffice; it will be necessary to add a second, so as to insert a blank page, and thus ensure that printing resumes on the appropriate side of the next available new page.

As we've already seen, in section 3.1.3, "Layout Adjustment to Support Duplex Printing", `groff`'s "`.if o`" request may be used to detect when document output is currently being directed to a recto page, (and conversely, the "`.if e`" request may be used to detect output to a verso page); thus, we may define specialized page break macros similar to:

```
.de NEW-RECTO-PAGE
.\" Insert a page break, resuming output at the top of the
.\" next available new recto (i.e. odd numbered) page.
.\"
.   ADVANCE-TO-NEW-PAGE o
..
```

and:

```
.de NEW-VERSO-PAGE
.\" Insert a page break, resuming output at the top of the
.\" next available new verso (i.e. even numbered) page.
.\"
.   ADVANCE-TO-NEW-PAGE e
..
```

Each of these page break macros simply delegates its operation to the generalized "`ADVANCE-TO-NEW-PAGE`" helper macro, passing either `groff`'s "`o`", or "`e`" page number property comparison operator, as argument, to specify whether output should resume on a new recto page, (odd numbered page), or a new verso page, (even numbered page),

respectively. A tentative, and perhaps a naïvely simplistic, implementation for such a generalized helper macro might be defined as simply as:

```
.de ADVANCE-TO-NEW-PAGE
.\" Insert page breaks, as required, to resume output at the
.\" top of the next available new recto page, or new verso page,
.\" as determined by the passed argument, (which MUST be either
.\" of groff's conditional operators, "o" or "e").
.\"
.\" Usage: .ADVANCE-TO-NEW-PAGE o  \" NEW-RECTO-PAGE
.\"        .ADVANCE-TO-NEW-PAGE e  \" NEW-VERSO-PAGE
.\"
.   if \\$1 .bp  \" need to skip an entire page
.   bp           \" advance to desired new page
..
```

Depending of the behaviour of any suite of macros, (whether standard, or user-defined), which has been chosen to control the document layout, this simplistic "ADVANCE-TO-NEW-PAGE" macro implementation may require some refinement. For example, when the ms macros are being used, this naïvely simplistic implementation will *not* result in the intervening verso page being skipped over, when the "NEW-RECTO-PAGE" macro is invoked while groff is already processing output which is to be printed, or otherwise displayed, on a recto page.

It may seem surprising that the preceding trivial implementation of the "ADVANCE-TO-NEW-PAGE" macro should fail, in the manner described, when used in conjunction with the ms macros, under the circumstances as described; however, given a basic understanding of the operation of page transition traps, in ms, the failure may be anticipated, and moreover, it is readily explained:

- When the "NEW-RECTO-PAGE" macro is invoked, during processing of a recto page, interpretation is redirected to the "ADVANCE-TO-NEW-PAGE" macro, with the "o" argument specified.
- On entering the "ADVANCE-TO-NEW-PAGE" macro, the expression ".if \\$1 .bp" is interpreted as ".if o .bp"; since this is being evaluated within the processing context of a recto page, (which has an odd page number), the ".if o" condition evaluates as "true", so the conditional ".bp" request *is* invoked.
- Invocation of the conditional ".bp" request causes the output position to advance to the bottom of the current recto page, (thus invoking any intervening traps, from the initial output position down to, and including, the bottom-of-page trap), and thence onwards to the top of the following verso page, whence ms arranges for activation of groff's "no-space" mode.
- Following invocation of the conditional ".bp" request, the "ADVANCE-TO-NEW-PAGE" macro then issues a further *unconditional* ".bp" request, with the intent of advancing the output position further towards the bottom of the new verso page, and beyond, ultimately placing it at the top of the immediately following recto page. Unfortunately, by the time this unconditional ".bp" request is issued, "no-space" mode has already been activated, and consequently, since this ".bp" request is issued *without* any explicitly specified new page number argument, it is ignored,[47] and the output position remains at the top of the verso page, whence normal output will eventually resume.

Having established that the preceding, naïvely simplistic implementation of an "ADVANCE-TO-NEW-PAGE" macro may be vulnerable to failure, when "no-space" mode may become active during its execution, (and furthermore, having established that this vulnerability is *not* specific to use in conjunction with the ms macros, or indeed with *any* particular macro suite), it behooves us to refine the "ADVANCE-TO-NEW-PAGE" macro implementation, in order to eliminate the vulnerability. To achieve this, we might consider refinements such as:

- Following invocation of the conditional ".bp" request, which implements the first of two page advances, in any case when two are required, insert an explicit ".rs" request, to explicitly deactivate "no-space" mode *before* the second ".bp" request is invoked, thus ensuring that this is *not* ignored.
- As an alternative to the preceding option, *implicitly* deactivate "no-space" mode, following the first of the two ".bp" request invocations, and prior to the second, by writing some arbitrary, but invisible output to the intermediate output page; implicitly, this has the same effect as insertion of the ".rs" request, but it is less

47. This is *not* a defect in the ms macros; rather, it is a natural consequence of the design decision to activate "no-space" mode at the top of each new page — a perfectly legitimate design choice, which is intended to eliminate the output of any block of redundant vertical space, immediately following a transition to a new page. Furthermore, although this issue has been identified in the context of interoperation of the "ADVANCE-TO-NEW-PAGE" macro, as defined, and the ms macros, similar behaviour will be observed in *any* context in which "no-space" mode is activated, following page transitions.

elegant, more cumbersome to implement, and its intent is less obvious, so it may be a less favourable method of achieving the desired effect.

- Rather than attempting to manipulate "no-space" mode, as both of the preceding options do, modify the form of the second, (i.e. the *unconditional*, or maybe even both), of the internal "`.bp`" requests, such that the form becomes "`.bp <page-number>`"; unlike the form of the "`.bp`" request without arguments, this form, with the "`<page-number>`" argument, is *not* ignored, whether "no-space" mode is active, or not.

Any one of these modifications will offer an effective solution to the identified "no-space" mode failure of our original "`ADVANCE-TO-NEW-PAGE`" macro implementation; of the three, the first is the most convenient, and perhaps also the most obvious to adopt; the second would require a more cumbersome, yet a less obvious implementation, but it offers no particular advantage over the first, so it probably merits no further consideration. Conversely, although it will necessitate a significantly more complex implementation — keeping track of suitable page numbers, for use as arguments to the "`.bp <page-number>`" requests, may present something of a challenge — the third option for refinement of the "`ADVANCE-TO-NEW-PAGE`" macro implementation may offer some behavioural advantages, over the features accorded by the simpler first option; we will explore some of these potential advantages, with particular reference to document formatting using the `ms` macros, in section 4.2.3.1, "Recto-Verso Page Break Handling when Using the `ms` Macros".

Notwithstanding that adoption of the third of the preceding "`ADVANCE-TO-NEW-PAGE`" macro refinement options may be advantageous, the simplicity and elegance of the first option may still offer a compelling reason for considering it. The refined implementation is straightforward:

```
.de ADVANCE-TO-NEW-PAGE
.\" Insert page breaks, as required, to resume output at the
.\" top of the next available new recto page, or new verso page,
.\" as determined by the passed argument, (which MUST be either
.\" of groff's conditional operators, "o" or "e").
.\"
.\" Usage: .ADVANCE-TO-NEW-PAGE o  \" NEW-RECTO-PAGE
.\"        .ADVANCE-TO-NEW-PAGE e  \" NEW-VERSO-PAGE
.\"
.   if \\$1 \{\
.   \" Current page faces as does the desired output target,
.   \" so we need to skip an entire opposing page.
.   \"
.      bp  \" advance to new opposing page
.      rs  \" get out of "no-space" mode
.   \}
.   bp     \" advance to desired new page
..
```

and, with this modified implementation in place, the original "`NEW-RECTO-PAGE`" and "`NEW-VERSO-PAGE`" macros will each introduce either one or two page breaks, as required, to resume output on the respectively appropriate side of the next available, and suitably facing, new page.

4.2.3.1. Recto-Verso Page Break Handling when Using the `ms` Macros

Provided that the "`ADVANCE-TO-NEW-PAGE`" macro refinement, to correctly handle "no-space" mode effects, has been incorporated, the simple "`NEW-RECTO-PAGE`" and "`NEW-VERSO-PAGE`" macros, as developed in the preceding section, *will* correctly introduce the appropriate number of page breaks, such that output resumes on the next available new recto page, or new verso page, respectively; why then, might we wish to consider adoption of a more complex technique for introduction of such page breaks?

One reason why we may wish to adopt the more complex technique is that, when two page breaks are required to advance to the appropiately facing page, any traps which are specified on the intervening page *will* be processed, as the output position advances down that page; when such traps result in the output of page headers, or footers, or both, as will be the norm when formatting with a macro suite such as `ms`, these headers, or footers, or both *will* be printed on the otherwise blank page. Thus, this intervening page will *not* be *completely* blank, so will *not* be identified as a candidate for `pdfroff`'s blank page removal procedure; it may become a distraction when formatting for on-screen viewing.

By default, `ms` does *not* specify page footers; it *does* specify a single-line page header, displaying the page number at the centre-top of each page, *except* that this header is omitted, *in its entirety*, on any page having a page number of one, (unless such header omission is *explicitly* overridden). This suggests a possible technique, suitable for use with `ms`

when its default page header and footer policy is in effect,[48] by manipulating the *effective* page numbers around the page break, to ensure that any intervening blank page remains *entirely blank*; this might be achieved, within a modified variant of the "ADVANCE-TO-NEW-PAGE" macro, by initially saving the *actual* page number prior to the page break, followed by a temporary change of *effective* page number, to one, when issuing the first (conditional) page break request, and ultimately, restoring the original *actual* page numbering sequence, when issuing the second (unconditional) page break request:

```
.de ADVANCE-TO-NEW-PAGE
.\" Insert page breaks, as required, to resume output at the
.\" top of the next available new recto page, or new verso page,
.\" as determined by the passed argument, (which MUST be either
.\" of groff's conditional operators, "o" or "e").
.\"
.\" Usage: .ADVANCE-TO-NEW-PAGE o  \" NEW-RECTO-PAGE
.\"        .ADVANCE-TO-NEW-PAGE e  \" NEW-VERSO-PAGE
.\"
.   nr \\$0.% \\n%    \" save current page number
.   if \\$1 \{\
.   \" Current page faces as does the desired output target,
.   \" so we need to skip an entire opposing page.
.   \"
.      nr \\$0.% +1   \" update to skipped page number
.      bp 1           \" skip, numbering as page one
.   \}
.   bp \\n[\\$0.%]+1  \" advance, restoring page number
.   rr \\$0.%         \" clear saved page number
..
```

Notice that, with this modification, it is unnecessary to explicitly cancel "no-space" mode after the conditional page break, because the following unconditional "`.bp \\n+[\\$0.%]`" request will cause a further page break, even when "no-space" mode is in effect. However, there *is* a potential pitfall with this modification: it *will not work* if the effective format for the page number register has been assigned as anything other than decimal numerals! To avoid this pitfall, it is necessary to temporarily force the page number register to exhibit a decimal numeric format, within the scope of execution of the "ADVANCE-TO-NEW-PAGE" macro, for example, by encapsulating the simple request:

```
.   nr \\$0.% \\n%    \" save current page number
```

(which itself requires the page number to be expressed in decimal numeric format), within an extended sequence of requests, such as:

```
.   af \\$0.% \\g%    \" save page number format
.   af % 0            \" interpret as decimal numeric...
.   nr \\$0.% \\n%    \" to save its current value
.   af % \\g[\\$0.%]  \" restore its original format
```

Furthermore, when the subsequent request:

```
.   bp \\n[\\$0.%]+1  \" advance, restoring page number
```

is eventually interpreted, the *saved* page number *must* be expressed in decimal numeric format; thus, it is convenient to further extend the request sequence, for saving the original page number format and value, completing it by appending the additional request:

```
.   af \\$0.% 0       \" keep saved value as decimal
```

48. The default page header and footer policy ceases to be in effect, if the (effectively irreversible) "`.P1`" macro has been called, thus overriding the omission of page one headers, and so causing the page header to be printed on any subsequent page numbered one, (which would include those introduced by the conditional page break, within the "ADVANCE-TO-NEW-PAGE" macro), or if any of the page header or page footer trap macros have been redefined, or if any page footer text has been defined. If any such policy changes *are* in effect, the modified "ADVANCE-TO-NEW-PAGE" macro would need to take steps to nullify them; such steps could significantly add to the required complexity of the macro. Details of the additional complexity, which would be necessary, depend on the precise nature of the departure from the default policy, within each individual document, and thus are left to the ingenuity of the publisher of the document to devise.

Thus, the modified form of the "ADVANCE-TO-NEW-PAGE" macro becomes:

```
.de ADVANCE-TO-NEW-PAGE
.\" Insert page breaks, as required, to resume output at the
.\" top of the next available new recto page, or new verso page,
.\" as determined by the passed argument, (which MUST be either
.\" of groff's conditional operators, "o" or "e").
.\"
.\" Usage: .ADVANCE-TO-NEW-PAGE o  \" NEW-RECTO-PAGE
.\"        .ADVANCE-TO-NEW-PAGE e  \" NEW-VERSO-PAGE
.\"
.   af \\$0.% \\g%    \" save page number format
.   af % 0            \" interpret as decimal numeric...
.   nr \\$0.% \\n%    \" to save its current value
.   af % \\g[\\$0.%]  \" restore its original format
.   af \\$0.% 0       \" keep saved value as decimal
.   if \\$1 \{\
.   \" Current page faces as does the desired output target,
.   \" so we need to skip an entire opposing page.
.   \"
.      nr \\$0.% +1   \" update to skipped page number
.      bp 1           \" skip, numbering as page one
.   \}
.   bp \\n[\\$0.%]+1  \" advance, restoring page number
.   rr \\$0.%         \" clear saved page number
..
```

As it now stands, when used with the ms macros, with their default page header and footer policies in effect, this implementatation of the "ADVANCE-TO-NEW-PAGE" macro will advance the output position to the top of the next available, appropriately facing new page; if an additional intervening page is inserted, it will remain *completely* blank, and page numbering will resume on the new output page.

A further option, which is not supported by the "ADVANCE-TO-NEW-PAGE" macro, as it now stands, may be worthy of consideration: instead of inserting a page header, and thus resuming page numbering immediately, on the page where output itself resumes following "ADVANCE-TO-NEW-PAGE", also omit the header of this page, then reinstate it to resume numbering only on the next following page. Once again, assuming that ms is being used, with its default page header and footer policies in effect, this additional feature may be readily supported, in a similar manner to the suppression of *all* output on intervening pages, by replacing the unconditional statement:

```
.   bp \\n[\\$0.%]+1  \" advance, restoring page number
```

within the current "ADVANCE-TO-NEW-PAGE" implementation, with the alternative unconditional statements:

```
.   bp 1              \" advance, without numbering the page
.   pn \\n[\\$0.%]+1  \" restore numbering on following page
```

or make it conditional, for example on having passed[49] a second argument of "no" (say), to suppress immediate resumption of page numbering, otherwise, resume it immediately:

```
.   ie '\\$2'no' \{\
.   \" Page numbering is to be suppressed, on the first page
.   \" on which output is resumed.
.   \"
.      bp 1               \" advance, without numbering the page
.      pn \\n[\\$0.%]+1   \" restore numbering on following page
.   \}
.   \" Otherwise, page numbering is to be resumed immediately.
.   \"
.   el .bp \\n[\\$0.%]+1  \" advance, restoring page number
```

49. To achieve this, the "NEW-RECTO-PAGE" and "NEW-VERSO-PAGE" macros would also require modification, to pass their own arguments on to "ADVANCE-TO-NEW-PAGE", following the "o" and "e" arguments, which they already pass, respectively.

Thus, the refined set of page break macros, for advancing the output position to a new recto page, or to a new verso page, becomes the complementary pair:

```
.de NEW-RECTO-PAGE
.\" Insert a page break, resuming output at the top of the
.\" next available new recto (i.e. odd numbered) page.
.\"
.   ADVANCE-TO-NEW-PAGE o \\$@
..
```

and its complement:

```
.de NEW-VERSO-PAGE
.\" Insert a page break, resuming output at the top of the
.\" next available new verso (i.e. even numbered) page.
.\"
.   ADVANCE-TO-NEW-PAGE e \\$@
..
```

together with their common helper macro:

```
.de ADVANCE-TO-NEW-PAGE
.\" Insert page breaks, as required, to resume output at the
.\" top of the next available new recto page, or new verso page,
.\" as determined by the first passed argument, (which MUST be
.\" either of groff's conditional operators, "o" or "e").
.\"
.\" Usage: .ADVANCE-TO-NEW-PAGE o [no]  \" NEW-RECTO-PAGE
.\"        .ADVANCE-TO-NEW-PAGE e [no]  \" NEW-VERSO-PAGE
.\"
.\" The second argument is optional; if specified as "no", the
.\" ms page header, (incorporating the page number), will not
.\" be printed on the first page, on which output is resumed.
.\"
.   af \\$0.% \\g%          \" save page number format
.   af % 0                 \" interpret as decimal numeric...
.   nr \\$0.% \\n%         \" to save its current value
.   af % \\g[\\$0.%]       \" restore its original format
.   af \\$0.% 0            \" keep saved value as decimal
.   if \\$1 \{\
.   \" Current page faces as does the desired output target,
.   \" so we need to skip an entire opposing page.
.   \"
.      nr \\$0.% +1        \" update to skipped page number
.      bp 1                \" skip, numbering as page one
.   \}
.   ie '\\$2'no' \{\
.   \" Page numbering is also to be suppressed, on the first page
.   \" on which output is resumed.
.   \"
.      bp 1                \" advance, without numbering the page
.      pn \\n[\\$0.%]+1    \" restore numbering on following page
.   \}
.   \" Otherwise, page numbering is to be resumed immediately.
.   \"
.   el .bp \\n[\\$0.%]+1   \" advance, restoring page number
.   rr \\$0.%              \" clear saved page number
..
```

Another optional refinement, which may be considered, could be to assign the page number format, for use on pages following any page break which is introduced by either "NEW-RECTO-PAGE", or "NEW-VERSO-PAGE", by passing the desired formatting code as a macro argument, rather than preserving the prevailing format internally, within the "ADVANCE-TO-NEW-PAGE" macro. A possible implementation of such a refinement may be achieved by defining a new internal-use macro:

```
.de ADVANCE-TO-NEW-PAGE.af
.\" Helper macro, to be called ONLY by ADVANCE-TO-NEW-PAGE;
.\" assign page number format for use on subsequent pages, as
.\" specified by passed argument, or default to decimal.
.\"
.   if '\\$2'no' .shift  \" ignore "no" argument
.   af % \\$2 0          \" assign as specified, or default
..
```

This new internal-use macro would then be called, normally exclusively in practice, by a further modified variant of the "ADVANCE-TO-NEW-PAGE" macro:

```
.de ADVANCE-TO-NEW-PAGE
.\" Insert page breaks, as required, to resume output at the
.\" top of the next available new recto page, or new verso page,
.\" as determined by the first passed argument, (which MUST be
.\" either of groff's conditional operators, "o" or "e").
.\"
.\" Usage: .ADVANCE-TO-NEW-PAGE o [<arg> ...]  \" NEW-RECTO-PAGE
.\"        .ADVANCE-TO-NEW-PAGE e [<arg> ...]  \" NEW-VERSO-PAGE
.\"
.\" The second, and subsequent arguments are optional; if the
.\" second is specified as "no", the ms page header, (in which
.\" the page number is normally included), will not be printed
.\" on the first page, on which output is resumed, and a third
.\" argument, if present, will be interpreted as specifying a
.\" page number format for use on subsequent pages.
.\"
.\" Otherwise, if a second argument is specified, and it is
.\" not "no", it will be interpreted as the specification of
.\" the page number format for use on subsequent pages.
.\"
.   af % 0                \" make page number decimal...
.   nr \\$0.% \\n%        \" to save its current value
.   \\$0.af \\$@          \" select, and apply new format...
.   af \\$0.% 0           \" keeping saved value as decimal
.   if \\$1 \{\
.   \" Current page faces as does the desired output target,
.   \" so we need to skip an entire opposing page.
.   \"
.      nr \\$0.% +1       \" update to skipped page number
.      bp 1               \" skip, numbering as page one
.   \}
.   ie '\\$2'no' \{\
.   \" Page numbering is also to be suppressed, on the first page
.   \" on which output is resumed.
.   \"
.      bp 1               \" advance, without numbering the page
.      pn \\n[\\$0.%]+1   \" restore numbering on following page
.   \}
.   \" Otherwise, page numbering is to be resumed immediately.
.   \"
.   el .bp \\n[\\$0.%]+1  \" advance, restoring page number
.   rr \\$0.%             \" clear saved page number
..
```

It is worth noting that each of the page break macros, developed above, is implemented *exclusively* using fundamental `groff` requests; none of them is *explicitly* dependent on `ms`. However, they *do* exhibit an *implicit* dependency on default `ms` behavioural traits — specifically that no page headers are printed on any page with a page number of one, and page footers are normally *entirely* blank. Thus, they should work equally effectively with *any* macro suite which mimics, or can be made to mimic, these `ms` behavioural traits.

There is at least one potentially detrimental consequence of depending on these `ms` behavioural traits, which will become apparent in any reference to the "`\n%`" page number register on any page on which an effective page number of one has been substituted for the real page number, to suppress printing of the page header on the page where output is resumed, following a page break. The detrimental effect of page number substitution will be particularly noticeable when collecting of references for inclusion in a table of contents; such references would be expected to reflect the real page number, but the "`\n%`" register will reflect only the effective page number of one. This effect, and a mechanism for neutralizing it, will be explored in section 4.4, "An Alternative Technique for Generating Tables of Contents".

As a final observation, on the handling of recto-verso page breaks: the "NEW-RECTO-PAGE", "NEW-VERSO-PAGE", and their supporting "ADVANCE-TO-NEW-PAGE" and "ADVANCE-TO-NEW-PAGE.af" macro variants, as they have been developed above, are intended only as examples; depending on their individual requirements, users are invited to adapt, and consolidate the techniques which they illustrate, as may be deemed appropriate. A practical illustration may be found in the pdfmark.ms source file, for this document; this never uses the "NEW-VERSO-PAGE" macro, and it *always* suppresses the printing of page headers on the first output page following any use of its "NEW-RECTO-PAGE" macro; thus, rather than implementing them separately, as illustrated above, it consolidates the implementations of both the "ADVANCE-TO-NEW-PAGE", and the "ADVANCE-TO-NEW-PAGE.af" macros, with *unconditional* page header suppression, into a free-standing "NEW-RECTO-PAGE" macro implementation, within which it also incorporates a mechanism for propagation of *real* page numbers, when constructing table of contents references,

4.3. Considerations for Working with Document References

The provisions made by the pdfmark macros, for creation of, and linking to, in-document reference marks is described in section 2.5, "Adding Reference Marks and Links".

In general, when producing any PDF document using the pdfroff program, adoption of the techniques described in section 2.5 is sufficient for creation of, and linking to, document reference marks, *without* the need for any further user intervention, *provided* that *all* of the reference marks, and *all* references to them, are encapsulated within the one document which is being produced. However, if it is desired that any created reference mark should be accessible for referencing from other documents, or any reference is made to a reference mark within another document, then it may be helpful to specify such reference marks in association with one or more pdfmark *reference dictionaries*.

4.3.1. Creating a Document Reference Dictionary

When any PDF document is produced by pdfroff, a reference dictionary is created *automatically*; however, this is normally created within a temporary file, which is deleted when pdfroff completes processing of each particular document, so it will not be readily available for exposure of any reference marks which it specifies, for subsequent use within other documents. This limitation may be overcome, by specifying the "--reference-dictionary" option when invoking pdfroff, in conformance with the synoptic model:

```
pdfroff --reference-dictionary=filename.ref [option ...] \
   input-file ... > filename.pdf
```

which instructs pdfroff to save, to the user nominated file, "*filename*.ref", any reference dictionary content which it generates, rather than deleting it on completion of processing, as it normally would.

In addition to adoption of the foregoing recommendation, to ensure that a reference dictionary is saved, consideration should also be given to the creation of individual reference dictionary records, and the scope of the reference context information which is stored in each.

Each individual reference dictionary record takes the form of a pdfhref macro call:

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

and is associated with *exactly one* named reference mark; each such record may be added to the dictionary, at the time when the reference mark itself is created, by use of a defining macro call in the form:[50]

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

When the ".pdfhref M ..." macro is invoked thus, with the "-X" option in effect, if the "-N <*name*>" option is *explicitly* specified, either in this, or in its equivalent "-D <*name*>" form,[51] then a corresponding reference dictionary record is instantiated, with a verbatim copy of the specified "-N <*name*>" option incorporated, *immediately* following the ".pdfmark D" preamble. Conversely, if *neither* the "-N <*name*>" option, *nor* the equivalent "-D <*name*>" option is specified, then *at least one* space-delimited word of the otherwise optional "<*text*> ..." sequence of arguments *must* be specified; from this, a "-N <*name*>" option is *implicitly* derived, for incorporation into the instantiated reference dictionary entry. In either case, the remaining "[<*keyword*> <*value*>] ..." and "<*text*> ..." components of this dictionary entry are determined in accordance with the currently active specification of the "PDFHREF.INFO" template string, as defined by the user.

50. The equivalent form, ".pdfhref M **-X [-D** <*name*>] [[-E] <*text*>]", may be used, if preferred; the "-D <*name*>" option is provided for consistency with its use in the ".pdfhref L ..." macro form, while the "-N <*name*>" option is defined for consistency with ".pdfhref D ..." syntax, and exhibits *identical* behaviour in ".pdfhref M ..." usage.

51. If the equivalent "-D <*name*>" form of this option is specified, it is internally convert to the "-N <*name*>" form, and explicitly interpreted as such.

In general, it is better to specify "PDFHREF.INFO" such that the reference dictionary will record *more* context than is deemed to be strictly necessary; unwanted context can simply be ignored, at the ultimate point of use, but anything which has not been recorded *cannot* be inferred on subsequent demand. Typical context information, which may be recorded, includes:

- The reference mark name; (this is *always* recorded, due to the incorporation of the "-N <*name*>" option, whether *explicitly* assigned, or *implicitly* inferred from the "<*text*> ..." reference mark description).
- A file name reference; this is processed according to the "PDFHREF.FILEREF" formatting specification, when a "file <*filename*>" tuple is included within the "[[<*keyword*> <*value*>] ...]" region of the "PDFHREF.INFO" template specification, and any link to the associated reference mark is subsequently interpolated.
- A page number reference; ultimately processed in accordance with the "PDFHREF.PAGEREF" formatting specification, when subsequently interpolating any link to the associated reference mark, this will be recorded when a "page \\n%" tuple is included within the "[[<*keyword*> <*value*>] ...]" region of the "PDFHREF.INFO" template specification.
- A section number reference, (if applicable); when recorded, by inclusion of a "section *(*SN*" tuple within the "[[<*keyword*> <*value*>] ...]" region of the "PDFHREF.INFO" template specification, this will be processed in accordance with the "PDFHREF.SECTREF" formatting specification, when any link to the associated reference mark is subsequently interpolated.
- Any reference data which may be associated with user-specified keywords; such keywords *must* be defined in each individual document (file) processing context in which they may be referenced, wherein they will be processed in accordance with the formatting specification which the user has associated with each particular keyword. Considerations for the use of such formatting specifications will be discussed further, in section 4.3.3, "Using Custom Reference Formatting Keywords".

All such data, which has been recorded within a saved reference dictionary, will be available for interpolation into inter-document references, formatted as described in section 2.5.5, "Establishing a Format for References", within any document in which this reference dictionary is deployed, as described in the following section.

4.3.2. Deploying a Document Reference Dictionary

When pdfroff is used to drive the groff PDF formatting process, for any given document, then the reference dictionary for that document, itself, will be *automatically* incorporated into the document input stream; this is required to accommodate the resolution of *intra-document*references, and occurs *irrespective* of whether the reference dictionary is designated to be committed to persistent file storage, as described in the preceding section, or is to persist only *temporarily*, and will be deleted on completion of the pdfroff processing run.

Conversely, if any *inter-document* references, to locations within any other, *external* PDF files are specified, the corresponding reference dictionaries will *not* be automatically incorporated into the pdfroff input stream. The onus for incorporation of such reference dictionaries rests *entirely* with the document author; this may be accomplished, most readily, by inclusion of a ".so filename.ref" request, within the referring document source.

Additionally, when such reference dictionaries are included, the document author should be aware of the possibility that these may introduce conflicting reference names, and should assume responsibility for resolving any such conflicts which may arise; techniques for doing so will be introduced in section 4.3.4, "Avoiding Reference Name Conflicts".

4.3.3. Using Custom Reference Formatting Keywords

When a reference dictionary comprises records which incorporate *exclusively* default pdfmark.tmac formatting keywords, then it may be safely assumed that these keywords will have been defined, by the simple expedient of including pdfmark.tmac within the pdfroff input data stream. However, no such assumption may be made, in the case of any reference dictionary in which user-defined custom formatting keywords may be present, and thus, the document author must accept the responsibility of ensuring that any such custom keywords have been defined, *before* referring to any named location for which the reference dictionary entry includes them.

For use within *intra-document* references, any custom formatting keywords which are to be incorporated should be defined in the document source stream, using the ".pdfhref K ..." capability, (see section 2.5.5.4, "Customizing Automatically Formatted Reference Text"), *before* placement of any reference which may require them.[52] Additional

52. When using an implementation of pdfmark.tmac from the 25.04, or any later release of groff-pdfmark, the ".pdfhref K ..." definitions will be duplicated in the generated reference dictionary, thus ensuring early definition in each pdfroff processing pass, with the exception of the first pre-processing pass.

custom keyword definitions, as may be required to support *inter-document* references, may also be defined within the document source, alongside *intra-document* keywords; however, it may be more convenient to emulate the behaviour of `pdfmark.tmac` release 25.04 and later, manually editing any reference dictionaries which may have been generated using an earlier `pdfmark.tmac` release, to include any custom keyword definitions which they may require.

It may be noted that provision of a "`.pdfhref K ...`" definition alone is insufficient to ensure appropriate handling of custom formatting keywords; the onus remains on the document author, to ensure that each such definition is accompanied by a suitable template string definition for the designated, named format string itself.[53]

4.3.4. Avoiding Reference Name Conflicts

Within any single PDF document, each `pdfhref` reference destination *must* be *uniquely* named. The onus for ensuring this lies with the document author; to facilitate honouring the obligation, it is recommended that:

- *Every* reference destination is *explicitly* named, by specifying the "`-N <name>`" option, (as described in section 2.5.2, "Marking a Reference Destination"), when each destination reference mark is placed, by use of the "`.pdfhref M ...`" macro call.
- Additionally, to help keep track of reference names which have already been allocated, and the destinations to which they refer, it is suggested that the "`-X`" option should be applied, for each invocation of the "`.pdfhref M ...`" macro, and the "`--reference-dictionary=<filename>`" option should be specified when running the "`pdfroff`" command; this allows for consultation of the nominated reference dictionary file, both as a reminder of allocated reference destination names, and the document locations to which they refer, (which may be helpful when constructing subsequent reference links).

Unfortunately, while the foregoing advice may help the author to avoid reference name conflicts within *any single* PDF document, it cannot guard against the possibility of such conflicts arising when referring to locations within *external* documents, (*especially* when such external documents originate from a *different* author, and the chosen reference names are beyond the control of the referring document author).

A technique which could be considered, to mitigate potential reference name conflicts across document boundaries, might be to assign a distinct *local* "namespace" for each individual document. Such a technique is *not*, currently,[54] supported by `pdfmark.tmac`, so document authors are free to implement a suitable mechanism, at their own discretion; a suitable strategy may be to:

- Edit the reference dictionary files, which are associated with each external file to be referenced, (using a stream editor such as `sed`, for example), to prefix a suitably chosen *distinct* "namespace" identifier to each reference name which is defined.
- Within the local document source, register each chosen "namespace", associating each with the appropriate set of `pdfhref` external filename reference options, which should be included in any "`pdfhref L ...`"; macro call, to resolve references within the designated "namespace".
- Also within the local document input data stream, implement a wrapper macro, which is to be used in place of "`pdfhref L ...`"; this wrapper macro should examine each specified destination name, *remove* any registered "namespace" prefix, then redirect the call to "`pdfhref L ...`" itself, while adding the appropriate external file name reference options, within the scope of the redirected call.

Until such time as capabilities, similar to those described above, are formally integrated into `groff-pdfmark`, details of a suitable substitute implementation are left to the discretion, and ingenuity, of the document author.

---

53. Prior to `pdfmark.tmac` release 25.04, this onus rests *entirely* on the document author; in the absence of any pre-existing format string definition, the "`.pdfhref K ...`" implementation within `pdfmark.tmac` release 25.04, and later, will supply a default definition, in the form "`.ds <format-name> "<keyword> \\$*\"`", which the document author may choose to adopt, or to override, as preferred.

54. Although the current `pdfmark.tmac` implementation does not, directly, support any technique, such as that described here, for avoiding reference name collisions, it is anticipated that built-in support may be incorporated in a near-future release of `groff-pdfmark`.

4.4. An Alternative Technique for Generating Tables of Contents

Although pdfroff offers built-in support for the traditional troff technique of printing tables of contents *after* the body of each document, and subsequently relocating them, *manually*, to their normal position *before* the document body, this technique is not particularly well suited to the publication of PDF documents, and the implementation is decidedly kludgey. In particular, when employing the traditional ms macro technique of collecting table of contents entries using the **.XS, .XA,** and **.XE** macros, and subsequently using the **.TC** macro to print the collected table of contents, for relocation by pdfroff, it has been found to be extremely challenging, if not actually impossible, to specify pdfhref links between the table of contents entries and corresponding reference marks within the document body. Consequently, while early editions of this publication *did* use the traditional ms technique for collation of the table of contents, this technique has now been abandoned, in favour of a more streamlined, and flexible collation technique, based on the groff_toc macro framework, which is provided as a supplementary component of the groff-pdfmark distribution.

4.4.1. Using the Basic groff_toc Macro Framework

The basic groff_toc macro framework is provided by the groff macro file, toc-base.tmac; this defines the **.toc** macro, with built-in support for the following set of basic operations:

.toc file [*filename*]
: Specify the path name for a file, into which collected table of contents data is to be written. Any previously active data collection stream is closed, before opening the specified *filename* stream.

  Omission of the optional *filename* argument causes subsequently collected table of contents data to be directed to the stderr output stream.

  Specification of any named data collection stream, other than stderr, may require groff to be invoked in "unsafe" mode, (i.e. with the "**-U**" option specified). This limitation *does not* apply, in the case of document processing by pdfroff.

.toc put *opcode* [*data ...*]
: Write a request, of the form "**.toc** *opcode* [*data ...*]",[55] to the currently active table of contents data collection stream, as specified by a prior call to "**.toc file** [*filename*]", or in the absence of any such prior call, to the stderr output stream.

  "**.toc put** *opcode* [*data ...*]" macro calls may be freely interspersed, throughout any document source, to export reference data, in the form of further "**.toc** *opcode* [*data ...*]" macro calls, whence a table of contents itself may be generated, when the resultant recorded "**.toc** *opcode* [*data ...*]" macro calls are imported, and interpreted at the point where the table of contents should appear.

.toc error *diagnostic message ...*
: Write the text of "*diagnostic message ...*" to the stderr output stream, identifying it as originating from within the groff_toc macro framework.

Since the groff_toc macro framework is implemented within a groff macro file, which is expected to reside within the default macro file path, (possibly augmented by assignment of the **GROFF_TMAC_PATH** environment variable), it *may* be incorporated into the document publication process by adding the "**-mtoc-base**" option to the groff, or the pdfroff command line. However, since it will usually be necessary to furnish a collection of binding macros, it may be more convenient to load the framework using the "**.mso toc-base.tmac**" request, and to delegate this request to a suitable binding macro package, (for example, see section 4.4.3, "Binding the groff_toc Macro Framework with the ms and pdfmark Macros").

55. Strictly, the request is written in the form "***[TOC.REQUEST]** *opcode* [*data ...*]", with the default assignment for TOC.REQUEST being specified as ".toc"; in normal use, changing this default assignment for TOC.REQUEST is *not* recommended; thus the default form of the request, as written, becomes "**.toc** *opcode* [*data ...*]", as stated.

Following the incorporation of the groff_toc framework into the document publishing process, *and* the definition of any supplementary macros which may be required to service the specified "**.toc** *opcode* [*data ...*]" requests, the table of contents itself may be incorporated into the document, by interpretation of groff markup code similar to:

```
.ce 1
.\" Place a centred one-line heading over the table of contents;
.\" default is 'Table of Contents', but allow for prior definition,
.\" e.g. to support translation to alternative languages.
.\"
.if !dTOC .ds TOC "Table of Contents\"
.pdfhref O 1 \*[TOC]
\fB\s'+2z'\[TOC]\s'-2z'\fP
.sp 2v
.
.\" Import the reference data, interpret it, and format the table
.\" of contents itself; notice that, only AFTER importing the data,
.\" is it safe to reset the reference data collector, because the
.\" action of resetting the file stream, (e.g. refdata.toc here),
.\" will obliterate its data content, which we wish to import.
.\"
.so refdata.toc
.toc file refdata.toc
```

whereafter, "**.toc put** *opcode* [*data ...*]" macro calls may be inserted into the document source, as required, to specify table of contents page references, and their associated descriptions.

4.4.2. Generating and Interpreting Table of Contents Reference Data

In the preceding example, as presented in section 4.4.1, it should be implicitly apparent that multiple groff processing passes will be required, for generation, and for interpretation of the table of contents reference data:

- One initial pass will be required to establish the early content of the refdata.toc file, (as it is named in the example); this may not even exist at the outset, and thus it will not be available for import, much less for interpretation, during this initial processing pass, but it will be created, if necessary, during this pass, and its content will then become available for import, and refinement during subsequent passes.
- During subsequent passes, the refdata.toc file will be imported, and its content will be interpreted, to tentatively format the table of contents; its content will then be cleared, and then rewritten, for refined reinterpretation during further subsequent passes.

Such multiple pass processing is an inherent feature of the pdfroff program, which will, thus, be particularly well suited to this task.[56]

It should be noted that the choice of reference data file name, "refdata.toc", as illustrated in the example presented in section 4.4.1, is entirely arbitrary; users are free to choose any alternative name, which may be better suited to their publishing conventions.

A further observation might be that section 4.4.1 offers absolutely no insight regarding particulars for the specification of any "*opcode*" symbolic names, nor for any of their associated "*data ...*" arguments, which may be used within "**.toc put** *opcode* [*data ...*]" macro calls. This apparent omission is, in fact, intentional: the groff_toc macro framework does not stipulate any such "*opcode*" symbolic names; nor does it impose any restrictions on the syntax, or on the content of any associated "*data ...*" arguments. Thus, users are free to make their own choices regarding the representation, and the ultimate interpretation of table of contents reference data; such choices are subject only to a few restrictions, namely:

- Any number of user-defined "*opcode*" names may be specified; at least one is required.
- Each user-defined "*opcode*" *must* be represented by a valid groff symbolic name.
- The "*opcode*" names, "**file**", "**put**", and "**error**", are *reserved*, and therefore, may *not* be chosen as user-defined "*opcode*" names; any attempt to do so may compromise the integrity of the groff_toc macro framework.

56. An additional, possibly less obvious, advantage of the choice of pdfroff for this task is that any groff error message, resulting from an attempt to import a non-existent reference data file, will be filtered from the stderr data stream, captured by pdfroff, within its temporary working data set, subsequently discarded, and consequently hidden from the user.

- Each user-defined "*opcode*" name *must* be supported by an implementation, in the form of a user-defined macro named "**toc.***opcode*", to correspond to the chosen "*opcode*" name. This macro will become integrated into the `groff_toc` macro framework, within which it will serve as the handler for table of contents reference data generated by macro calls of the form: "**.toc put** *opcode* [*data* ...]", with matching "*opcode*" name.

While it may be evident that the implementation of the `groff_toc` macro framework is fundamentally incomplete, and the burden of completion is placed on the user, the extensible design does accord, to the user, considerable latitude in the choice of exactly how table of contents reference data should be recorded, and ultimately presented, in any given publication; essentially, this may be viewed as a choice between adoption of a comparatively trivial implementation, or the pursuit of a more sophisticated solution, such as an adaptation of the extended framework which has been developed for use in this publication itself, is discussed further in section 4.4.3, "Binding the `groff_toc` Macro Framework with the `ms` and `pdfmark` Macros", and ultimately, has been implemented as documented in Appendix C, section C.3. , "Generating and Typesetting the Table of Contents".

Should the user choose to adopt a trivial implementation, this may be as simple as provision of a macro similar to:

```
.de toc.pageref
.\" Format and emit a single table of contents entry, which has
.\" previously been recorded by a macro call of the form:
.\"
.\"   .toc put pageref \n% <reference text> ...
.\"
.\" \$1 is expected to represent a page number reference; capture
.\" it, together with leader and tab, for relocation to the end of
.\" the formatted entry, then move it out of the way.
.\"
.   ds \\$0-% "\h'\w!00!u'\a\t\\$1\"
.   shift
.
.\" Emit the remaining reference text, followed by the captured
.\" page number, and then, clean up temporary working storage.
.\"
.   nop \\$*\\*[\\$0-%]
.   rm \\$0-%
..
```

With a macro such as this in place, beforehand, the example illustrated in section 4.4.1, for incorporation of the table of contents into the published document, should be adapted, (omitting embedded comments), to something like:

```
.ce 1
.ll 17.0c
.ta (u;\n[.l]-\w'00000')R \n[.l]uR
.if !dTOC .ds TOC "Table of Contents\"
.pdfhref O 1 \*[TOC]
\fB\s'+2z'\[TOC]\s'-2z'\fP
.sp 2v
.
.so refdata.toc
.toc file refdata.toc
```

while the actual content, complete with associated page number references, which is to be incorporated as table of contents entries, would be specified by inserting reference marks, in the form of macro calls of the form:

```
.toc put pageref \n% <text for table of contents entry>
```

at strategic points within the marked-up document source, (possibly by encapsulation of such calls within a further user-defined macro, to facilitate reproduction of the same reference text as a heading within the document body, in addition to incorporating it into the table of contents; a technique for implementing this, and also for reproducing the same reference text as a PDF document outline link, will be introduced in section 4.4.3).

4.4.3. Binding the `groff_toc` Macro Framework with the `ms` and `pdfmark` Macros

As noted in section 3, "PDF Document Layout", the formatting of this document is directed by the `groff ms` macros; its early editions used the built-in table of contents generation macros, with the assistance of the associated collation features of `pdfroff`. This *did* create a visually acceptable table of contents; however, it was found to be functionally lacking, insofar as it did not exhibit any capability for linking to the associated reference locations, and the challenge of providing this capability seemed insurmountable. Hence, the decision to develop, and deploy the `groff_toc` macro framework, as described here, was taken; the outcome has proven to be very successful.

Development of the `groff_toc` macro framework itself was quite straightforward, and effective bindings between the `ms` and `pdfmark` macros already exist, courtesy of the supplementary macro package `spdf.tmac`; thus, development of further bindings, between `groff_toc` and `spdf.tmac` alone, should suffice to bind all of `ms`, `pdfmark`, and `groff_toc`, to form a logically integrated macro suite.

A primary objective, in the development of a binding macro package, is that it should modify the behaviour of the underlying implementation, upon which it depends, but it should remain separate from that implementation, and should not *physically* alter it *in any way*. With this in mind, in this section we will explore the principles of development of a binding package for the `ms`, `pdfmark`, and `groff_toc` macros, while keeping the bindings separate, by placing them in the additional `spdf-toc.tmac` binding macro file.[57] Moreover, we will focus, primarily, on meeting only the requirements of this document, so the collection of binding macros, as discussed here, may require further development to complete their implementation for more generalized use.

To begin construction of the `spdf-toc.tmac` binding macro file, we will insert conventional file header commentary, together with a guard test[58] to avoid unintentional repeated interpretation:

```
.\" spdf-toc.tmac
.\"
.\" Binding macros for use of the groff_toc macro framework,
.\" in conjunction with the ms, and pdfmark macros.
.\"
.if d toc.end .nx  \" load this macro file only once
.ig
... specify copyright and licensing terms here ...
..
```

Following this file header, and its repeat inclusion guard, we may wish to test that certain prerequisite document processing conditions will be satisfied. For example, it is common practice for `groff` macro files to check that `groff` is actually used as the document processing engine;[59] however, since we intend to use `pdfroff` as the `groff` driver, we *may*[60] wish to enforce this:

```
.if !d pdfroff .ab aborting: please use pdfroff to format this document
```

After completing any such desired prerequisite condition checks, the next step is to ensure that the macro packages, to which `spdf-toc.tmac` specifies bindings, are loaded:

```
.mso spdf.tmac       \" incorporate the "ms" and "pdfmark" macros
.mso toc-base.tmac   \" augmented by the "groff_toc" framework
```

before proceeding to implement the required binding macros.

Considering the particular requirements for this document alone, it may be noted that all of its table of contents entries are derived from numbered section headings, each of which is defined by calling `spdf.tmac`'s **`XN`** macro. Thus, to provide a minimal implementation which will meet our immediate requirements, it should be sufficient to modify the behaviour of this particular macro.

---

57. This binding macro file is included with the `groff-pdfmark` distribution, but it does not promise to deliver a fully developed implementation; it should be considered as an example, which users may wish to adapt to suit their own requirements.

58. Guard tests are typically included *early* in macro files, to avoid the overhead of reading them more than once, in the event of them being included from more than one location. Such tests are predicated on the prior existence of some definition which is expected to be unique to the included file; we have chosen to test for a definition of the `end.toc` macro, which is *not* required by the `groff_toc` macro framework, but which we plan to implement, as part of our binding macro set.

59. This particular test will be performed, in any case, when we include `spdf.tmac`, so we may simply delegate it thence.

60. While the choice to use `pdfroff` is a *sufficient* condition to ensure that the document processing engine will be `groff`, this may not actually be a *necessary* prerequisite for using our `spdf-toc.tmac` binding macros; the specification of this prerequisite test should be considered as illustrative, and may require review.

As we've already seen, in section 3.1.2.1, "The XH and XN Macros", the XN macro, (and its companion, the XH macro), each performs *three* separate, but related functions:

- Each specifies text, which is to be incorporated into the document body, as a section heading.
- The text of each such section heading is inserted into the PDF document outline, as a pdfhref linked reference to the location, within the document body, of the section heading itself.
- The text of each section heading is also made available for inclusion into a table of contents, under the control of the (possibly) user-defined XH-UPDATE-TOC callback macro.

Of these three functions, we wish to modify *only* the last. Fortunately, due to the design of the XH, and the XN macros, this may be readily accomplished, by providing an appropriate replacement for spdf.tmac's implementation of the XH-UPDATE-TOC macro; noting that, when it is called, the first argument passed to this macro will *always* represent a document outline nesting level, while the second,[61] and all remaining arguments, represent the substance of the ensuing table of contents entry, a tentative — albeit naïve — replacement implementation might look something like:

```
.de XH-UPDATE-TOC
.\" Record data for construction of a table of contents entry
.\"
.   toc put outline \\$1  \" record outline nesting level ...
.   shift                 \" ... and move it out of the way
.   toc put refmark \\$*  \" record reference context ...
.   toc put pageref \\n%  \" ... and location
..
```

It may be noted that, unlike the simplistic example presented in section 4.4.2, this tentative implementation of XH-UPDATE-TOC requires *three* supplementary user-defined macros, namely "toc.outline", "toc.refmark", and "toc.pageref", serving as extensions to the groff_toc framework; the implementation of these will be discussed later, with a separate section devoted to each. However, the naïvety of this tentative implementation does raise a number of issues:

- If the proposed prerequisite for use of pdfroff is honoured, then the included ".toc put ..." macro calls will produce a potentially considerable quantity of visible, and not particularly useful, noise on the stdout diagnostic stream, during pdfroff's final document publication phase; this may be avoided by checking for the existence of pdfroff's "**PHASE**" register, when XH-UPDATE-TOC is first invoked, and when this register has been defined, with any positive non-zero value, redefine XH-UPDATE-TOC itself, to do nothing on subsequent invocations, and then immediately exit, without doing anything more; this may be achieved, for example, by modifying the preceding tentative definition of XH-UPDATE-TOC as follows:

```
.de XH-UPDATE-TOC
.\" Record data for construction of a table of contents entry
.\"
.\" If pdfroff's PHASE register is defined, with any positive
.\" non-zero value, this macro has nothing to do, for this or
.\" any subsequent call; redefine it as empty, for subsequent
.\" calls, then immediately exit.
.\"
.   if r PHASE \{.if \\n[PHASE] \{.de \\$0 return
.      return
.   \}\}
.
.\" If we get this far, record the requisite reference data.
.\"
.   toc put outline \\$1  \" record outline nesting level ...
.   shift                 \" ... and move it out of the way
.   toc put refmark \\$*  \" record reference context ...
.   toc put pageref \\n%  \" ... and location
..
```

61. In addition to representing some part of the substance of a table of contents entry, in the particular case of XH-UPDATE-TOC having been called by the XN macro, (as is *always* the case, when formatting *this* document), the second argument represents a section heading number; as such, it may merit special handling within the substantive scope of the table of contents entry.

- The tentative implementation provides no mechanism for exploiting the `pdfhref` feature of the `pdfmark` macros, to create any links between the table of contents and the material to which it refers. To rectify this omission, if we arrange for our "`toc.refmark`", and "`toc.pageref`" macro calls to assemble, and ultimately emit the constructed table of contents entry, in the form of a

  ```
  .pdfhref L -D <collected-reference-data>
  ```

  macro call, and recognizing that, within the implementation of the `XN` macro, as provided by `spdf.tmac`, the `XH-UPDATE-TOC` macro call is *always* preceded by a "`.pdfhref O ...`" call, then we might choose to replace the

  ```
  .toc put refmark \\$*  \" record reference context ...
  .toc put pageref \\n%  \" ... and location
  ```

  macro calls, within our tentative `XH-UPDATE-TOC` implementation, with something like:

  ```
  .\" Record the table of contents reference context ...
  .\"
  .   toc put refmark \\*[PDFBOOKMARK.NAME] -- \\$*
  .
  .\" ... and corresponding page number reference.
  .\"
  .   toc put pageref \\n%
  ```

  Alternatively, we may wish to incorporate some structural layout controls, as employed in this document, into the development of our `XH-UPDATE-TOC` replacement,[62] and accompanying `groff_toc` extension macros. For example, to emulate the indentation and vertical spacing style of the table of contents in this document, in which *all* entries are specified by calling the `XN` macro, and all are linked to the material to which they refer, we might rewrite the `XH-UPDATE-TOC` macro, beginning with the `stdout` noise suppression preamble, as suggested above:

  ```
  .de XH-UPDATE-TOC
  .\" Record data for construction of a table of contents entry
  .\"
  .\" If pdfroff's PHASE register is defined, with any positive
  .\" non-zero value, this macro has nothing to do, for this or
  .\" any subsequent call; redefine it as empty, for subsequent
  .\" calls, then immediately exit.
  .\"
  .   if r PHASE \{.if \\n[PHASE] \{.de \\$0 return
  .      return
  .   \}\}
  .
  ```

  We then follow this with a modified implementation of the reference data recording sequence; the initial step, which is responsible for recording the outline nesting level, remains functionally unchanged, *except* that we *do not* immediately discard the associated macro argument:

  ```
  .\" If we get this far, record the requisite reference data,
  .\" beginning with the outline nesting level ...
  .\"
  .   toc put outline \\$1
  .
  ```

62. Notice that, in this discussion of `XH-UPDATE-TOC` macro replacements, it is implicitly assumed that *all* table of contents entries are to be derived from numbered section headings, each of which has been specified using the `XN` macro, in conjunction with the `NH` macro; this limitation may be particularly relevant, in the case of the implementation which is currently under discussion.

In the next step, we see the predominant functional changes; we need to embed not only the associated reference name, but also font size, font style, and spacing controls into the assembled reference text, which will ultimately be entered into the table of contents, using a "`pdfhref L -D ...`" macro call:

```
.\" ... following up with the outline reference name, and
.\" document section number, as established by a prior .NH
.\" macro call ...
.\"
.   ds toc@refmark.text "\\*[PDFBOOKMARK.NAME] -- \"
.   ie \\$1=1 \{\
.   \" ... with font size increment, emboldening, and spacing
.   \" adjustment, as appropriate for top level headings ...
.   \"
.      as toc@refmark.text "\s'+1z'\fB\Z'\\$2'\fP\s'-1z'\"
.      as toc@refmark.text "\h'1.5n+\w!0.!u'\s'+1z'\fB\"
.   \}
.   \" ... while maintaining normal font size, and style, for
.   \" all levels of sub-headings.
.   \"
.   el .as toc@refmark.text "\\$2\h'1.5n'\"
.
.\" Discard the two arguments, which have been interpreted
.\" already, append all remaining arguments to the assembled
.\" reference data record, export it, and clean up.
.\"
.   shift 2
.   toc put refmark \*[toc@refmark.text]\$*
.   rm toc@refmark.text
.
```

Finally, we complete the amended `XH-UPDATE-TOC` macro definition, with one more functionally unchanged step, to record the page number reference:

```
.\" Finally, append a page number reference, to wrap up
.\" this table of contents entry record.
.\"
.   toc put pageref \\n%
..
```

- In every variant of the tentative `XH-UPDATE-TOC` replacement macro, which has been discussed up to this point, it has been assumed that the page number, to be included within the recorded table of contents reference data, is *always* represented by the value of the "`\n%`" register. While this may be a reasonable assumption, in a majority of cases, it *cannot be guaranteed* to be valid in *every* case. For example, in this document the value of the "`\n%`" register, as set by a "`.pn`", or a "`.bp`" request, is occasionally, and artificially reset to one, while tracking the progression of the *true* page number in a supplementary — but not persistently updated — "`\n[%%]`" register, in order to take advantage of an `ms` feature, whereby page headers are not *normally*[63] printed on any page which has an *effective* page number — as represented by the "`\n%`" register — of one. However, since this document's "`\n[%%]`" register is defined *only* when the *true* and *effective* page numbers differ, the required behaviour may be readily accommodated by adaptation of the final step in the `XH-UPDATE-TOC` replacement macro:

```
.\" Finally, append a page number reference, to wrap up
.\" this table of contents entry record.
.\"
.   ie r %% .toc put pageref \\n(%%
.   el .toc put pageref \\n%
..
```

63. This "normal" behaviour of `ms` printing control may be disabled, by invocation of the `.P1` macro; unfortunately, the effect of this is irreversible — without delving into the undocumented internals of the `ms` macros.

- Notwithstanding that the replacement XH-UPDATE-TOC macro implementation, as discussed up to this point, should deliver satisfactory results when invoked via any call to the XN macro, it must be recognized that this *same* macro can also be invoked via calls to the XH macro, in the event of which its behaviour may be less satisfactory.

  Adaptation of the replacement implementation, as illustrated thus far, to deliver satisfactory behaviour when invoked via XH macro calls, *exclusively*, should be fairly straightforward, and is left as an exercise for the reader. Adaptation to interoperate with *both* XH and XN macro calls *interchangeably*, while delivering a consistent layout, is a work in progress; in the meantime, this too is left as an exercise for the reader.[64]

- The replacement XH-UPDATE-TOC macro, as illustrated, neither uses, nor does it offer any mechanism for supporting, the XS, XA, or XE macros, which are central to the traditional ms table of contents collation technique; neither does this groff_toc based implementation offer any method for specifying table of contents entries, other than those which are derived from section headings which have been interpolated by the GNU-specific XN macro, or — to some extent — by the similarly GNU-specific XH macro. Thus, users who require such capabilities would need to extend the implemention, to satisfy their own needs; once again, development of such an extended implementation is left as an exercise for the reader.

Based on a combination of appropriate macro fragments, abstracted from the preceding discussion, and with the addition of some logic to control its interaction with groff diversions, a listing of the complete implementation for a replacement XH-UPDATE-TOC macro, as used in the production of this publication, may be found in Appendix C, section C.3.1. , "Table of Contents Reference Data Collection — the XN-UPDATE-TOC Macro".

4.4.3.1. Managing Side-Effects of groff_toc Macro Bindings

In general, any side-effects exhibited by a suite of binding macros, which have been developed along the lines discussed in section 4.4.3, will be innocuous. There is, however, one noteworthy exception: the use of the ".pdfhref L ..." macro call, to format each, and every page reference entry, will result in the entire text of the table of contents — with the exception of any initial heading — being rendered in whatever colour is specified for PDFHREF.TEXT.COLOUR.

While some authors may be willing to accept this text colour effect,[65] many will find it undesirable, and distracting. Fortunately, the effect may be readily mitigated, by temporary reassignment of the PDFHREF.TEXT.COLOUR specification,[66] *before* scheduling the output the table of contents:

```
.\" Break PDFHREF.TEXT.COLOUR and PDFHREF.TEXT.COLOR equivalence,
.\" then reassign PDFHREF.TEXT.COLOUR, (which is given precedence),
.\" to match the normal running text colour.
.\"
.rm PDFHREF.TEXT.COLOUR
.ds PDFHREF.TEXT.COLOUR "\n(.m\"
.\"
.\" Insert instructions for incorporation of the TOC, here ...
```

and subsequently, delete this temporarily assigned specification, allowing the next invocation of ".pdfhref L ..." to *automatically* reinstate the default PDFHREF.TEXT.COLOR equivalence:

```
.\" Discard our overriding PDFHREF.TEXT.COLOUR assignment, thus
.\" scheduling automatic reinstatement of the normal equivalence
.\" of PDFHREF.TEXT.COLOUR and PDFHREF.TEXT.COLOR by pdfhref
.\"
.rm PDFHREF.TEXT.COLOUR
```

Bracketting of the instructions, for formatting of the table of contents, between a pair of request fragments such as those illustrated above, will produce page references in the default body text colour; for convenience, the functionality which

64. The appendices to this document do, in fact, use the XH macro to introduce section headings, in a manner which *appears* to be compatible with this implementation of XH-UPDATE-TOC; this is made possible *only* by ensuring that, when XH is called, the argument which is specified immediately following its <*outline-level*> argument, resembles a section number, and thus appears, to XH-UPDATE-TOC, as if it had been called by XN.

65. Note that this colour effect is caused by the use of the ".pdfhref L ..." macro, which is introduced by binding with the pdfmark macro package. Thus, it will be observed for any bindings which incorporate this package; it is *not* restricted to bindings with the ms macro package, as illustrated in section 4.4.3.

66. Note that, *before* simply reassigning the value of PDFHREF.TEXT.COLOUR, its default assignment *must* be removed; this is necessary, to avoid implicit reassignment of its associated PDFHREF.TEXT.COLOR alias, in which it is desired to retain the default setting.

is provided by these two request fragments may be encapsulated within a further binding macro, such as that suggested in Appendix C, section C.3.5. , "Avoiding Excessive Use of `pdfhref` Link Colour in Tables of Contents", which may be incorporated into the user-provided binding macro package.

4.4.4. Classification and Selective Processing of Table of Contents Entries

While many documents will require only a single table of contents listing, usually placed before the running text of the document body, and the capabilities of the `toc-base.tmac` macro package are generally sufficient to produce such a listing, some documents — notably those of a technical nature — may benefit from the inclusion of more than one such listing — all similar in style, but each logically distinct in terms of context. For example, in addition to the normal table of contents, it may be desirable to include a separate list of figures, or a list of equations, list of data tables, or other similar entities, or indeed, any combination of such entities.

In any case where only a single table of contents listing is required, then this may be produced by adopting the techniques described in earlier subsections of section 4.4; for those cases where the production of additional listings, in the style of a table of contents but logically distinct in context, is required, then the extended capabilities of the `toc-class.tmac` variant of the `groff_toc` macro framework may be found to be useful.[67]

The `toc-class.tmac` extension to the `groff_toc` macro framework may be activated by specification of the "**`-m toc-class`**" command line option, when invoking `groff`, (either directly, or via a driver program such as `pdfroff`). Alternatively — perhaps more conveniently — the same effect may be achieved, by including the request:

```
.mso toc-class.tmac
```

in a binding macro file, which is provided by the user. In either case, as a side effect, the `toc-base.tmac` macro framework file will also be activated — it is unnecessary to load it separately.

In addition to the basic features supported by the `toc-base.tmac` macro framework, as described in section 4.4.1, activation of the `toc-class.tmac` extension makes the following **`.toc`** macro operations available:

`.toc activate` [*`class ...`*]
: Typically used *only* internally, during table of contents reference data collection, to propagate table of contents entry classification requests, from any document input data stream, through the collected data interpolation procedure, to the resultant document output data stream.

`.toc classify` [*`class ...`*]
: Insert a "**`.toc activate`** [*`class ...`*]" request into the table of contents reference data collection stream, specifying that each subsequently recorded reference is to be classified such that it becomes associated with each of the specified, arbitrarily named `class` arguments, (if any).

  If "**`.toc classify`**" is invoked *without* specifying any `class` arguments, any subsequently recorded references will be considered to be *unclassified*, (as are any which are recorded before the first of any invocations of "**`.toc classify`** *`class ...`*", which *does* specify `class` arguments); this unclassified state prevails until any subsequent invocation of "**`.toc classify`** *`class ...`*", *with* `class` arguments.

  Successive invocations of "**`.toc classify`** *`class ...`*", each of which specifies one, or more `class` arguments, will result in a complete *replacement* of the `class` associations for any subsequently recorded table of contents references; the classification effect is *not* cumulative.

`.toc classified` *`opcode ...`*
: Attach a classification filter to each of the "**`.toc.`***`opcode`*" service macros, associated with each specified *`opcode`* argument; each of these associated service macros *must* have been fully defined, *before* calling "**`.toc classified`**" with its *`opcode`* as an argument.

  Internally, each associated "**`.toc.`***`opcode`*" macro is renamed as "**`.toc.`***`opcode`***`.classified`**", and "**`.toc.`***`opcode`*" is redefined as an alias for an internal classification filter macro; this filter intercepts calls to the original "**`.toc.`***`opcode`*" macro, and forwards them to "**`.toc.`***`opcode`***`.classified`**", *only* when interpolating a table of contents reference which is *not* classified as being a member of an actively excluded reference class, and then is *either* a member of an actively selected reference class, *or* there is no actively selected reference class.

`.toc exclude` [*`class ...`*]
: Specify zero, or more table of contents reference classes, the member records of each of which are to be actively excluded from the ensuing output data stream, when table of contents reference data is interpolated.

67. Although this document, itself, does not incorporate any such contextually distinct listings, in the style of a table of contents, it *does* use this extended variant of the `groff_toc` framework, to isolate selected subdivisions of the table of contents itself, and to reproduce them within its appendices.

Each successive call to "**.toc exclude** *class ...*" *replaces* any previously active list of excluded classes; if called *without* `class` arguments, *all* active exclusions are *cancelled*.

.toc select [*class ...*]
Specify zero, or more table of contents reference classes, the member records of each of which are to be actively *included* in the ensuing output data stream, when table of contents reference data is interpolated.

Each successive call to "**.toc select** *class ...*" *replaces* any previously active class selections; if called *without* `class` arguments, *all* active selections are *cancelled*.

If the collected table of contents reference data includes *unclassified* records, these will be included in the output data stream, *only* if there is no active class selection, when the reference data is interpolated.

Having adopted the `toc-class.tmac` extension of the `groff_toc` macro framework, in order to use its extended features *effectively*, calls to any user-defined macro which participates in the ultimate interpolation of table of contents reference data, into the output data stream, should be redirected through the classification filter. In the case of the trivial example, as illustrated in section 4.4.2, "Generating and Interpreting Table of Contents Reference Data", this may be accomplished by invoking:

```
.toc classified pageref
```

after the "`toc.pageref`" macro has been defined, whereas, in the more complex example illustrated in section 4.4.3, "Binding the `groff_toc` Macro Framework with the `ms` and `pdfmark` Macros", it becomes necessary to invoke:

```
.toc classified outline refmark pageref
```

after *all* of the "`toc.outline`", "`toc.refmark`", and "`toc.pageref`" macros have been defined. Then, to ensure that each recorded table of contents reference is appropriately classified, interpose:

```
.toc classify class ...
```

or:

```
.toc classify
```

macro calls, at appropriate locations throughout the document input file(s), to activate, or respectively — assuming that unclassified records are used to define the basic table of contents — to cancel classification, as may be necessary.

After completion of the foregoing steps, to classify the recorded table of contents reference data, and to ensure that the classified entries can be appropriately filtered, the collected reference data may be *selectively* interpolated into the document output stream, *before* completing the:

```
.toc file refdata.toc
```

assignment. For example, if the objective is to produce a conventional table of contents, followed by a separate list of figures, the conventional table of contents references may be collected *without* classification, while those references which are to be incorporated into the list of figures may be identified by association with a class name such as "`figrefs`", which may be assigned by the use of code similar to:

```
.toc classify figrefs
.\" Code to generate figure goes here...
.toc put pageref \n% "Caption for figure ..."
.toc classify
```

to encapsulate each figure, at the point where it appears within the document input file(s).

Having specified the *unclassified* table of contents entries, and appropriately *classified* list of figures entries, within the collected `refdata.toc` file,[68] as described above, the content of this file may be interpreted *twice* — or even more often, if more than two distinct tables of content types are required — subject to distinct constraints in each case, in order to interpolate, for example, a table of contents, followed by a *separate* list of figures; a typical interpolation sequence might be:

1. Apply mitigation for any undesired text colour effects, as previously discussed in section 4.4.3.1, "Managing Side-Effects of `groff_toc` Macro Bindings".

68. Note that this concept is *not* restricted to segregation of just the two tables of content reference categories; *any* arbitrary number of additional reference classes may be specified, and reference entries may then be distributed amongst them, to achieve any desired segregation of reference categories into separate arbitrarily identified tables.

2. Ensuring that the following output will commence at the top of a new page — typically, this would be a new recto page — emit a "Table of Contents" heading, specify a "`toc class ...`" exclusion list, for entries of *all* reference categories which are *not* intended to appear within the primary table of contents — for example, the "`figrefs`" class — and process the `refdata.toc` file, to interpolate those *unclassified* reference entries which comprise the table of contents:

```
.\" Assume that the output position is at the top of page;
.\" disable filling, set tabs to mark the beginning, and the
.\" end of the page number field, emit a centred heading for
.\" the "Table of Contents", and interpolate the associated
.\" references, excluding those for the "List of Figures".
.nf
.ta (u;\n[.l]-\w'00000') \n[.l]uR
.ce 1
.nop \fB\s'+2z'Table of Contents\s'-2z'\fP
.sp 2v
.toc exclude figrefs
.so refdata.toc
```

and then, finish this step in the sequence, by cancelling the "`toc class ...`" exclusion list, which was responsible for selection of *only* those reference entries which *do* belong in the table of contents:

```
.\" Cancel exclusion of all reference classes which
.\" were not to be included in the table of contents;
.\" in the example above, this was just the figrefs
.\" class.
.\"
.toc exclude
```

3. Move forward to another new page, emit a suitable heading for an alternative class of content category references — for example, a "List of Figures" — specifying a corresponding "`toc class ...`" selection list, and process the `refdata.toc` file one further time, to interpolate the reference table for the corresponding content category:

```
.\" Advance to another new page, emit a centred heading for
.\" the "List of Figures", and interpolate the reference data
.\" once again, selecting ONLY those references which are to
.\" be included within the list of figures.
.bp
.ce 1
.nop \fB\s'+2z'List of Figures\s'-2z'\fP
.sp 2v
.toc select figrefs
.so refdata.toc
```

4. Repeat step (3) as often as may be required, substituting appropriate alternative headings, and corresponding "`toc class ...`" selection list specifications, in order to interpolate any other desired tables of classified content references, such as lists of equations, lists of data tables, etc.

5. When all of the desired tables of content categories have been interpolated, the reference data interpolation context may be cleaned up, any mitigation for undesirable text colour effects may be cancelled, and the reference data collector may be initialized:

```
.\" Clean up table of contents interpolation context,
.\" cancelling all currently active toc class selections,
.\" and any active text colour effect mitigation, then
.\" initialize the reference data collection stream.
.\"
.toc select
.rm PDFHREF.TEXT.COLOUR
.toc file refdata.toc
```

6. Finally, prepare for formatting of the document body; typically, this would also commence on yet another new page,[69] without any residual need for the tabulation settings of the table of contents, and with text filling enabled:

```
.\" Finally, advance to another new page, where output of
.\" the document body content is to begin, reset tabs, and
.\" enable text filling.
.bp
.ta
.fi
```

In practice, the preceding sequence of operations would be incorporated into the document source file,[70] *after* any desired front-matter content has been specified,[71] but *before* commencing the specification of the document body content.[72]

69. Normal printing conventions dictate that the start of the document body should be placed on a new *recto* page; a technique for ensuring this, which may be adopted in place of the simple `.bp` request of the above example, is discussed in section 4.2.3, "Ensuring that Content is Printed on a Particular Side of the Page".

70. Notice that this strategy results in attempted interpolation of the content of the "*refdata.toc*" file, *before* its content has been collected. Since this content is collected during processing by `groff`, at least one preliminary pass — and preferably two, since page numbers may be offset from their ultimately correct values, after just one pass — through "`groff -z`", to prepare the "*refdata.toc*" file for ultimate `groff` output processing. All such necessary preliminary processing passes will be performed automatically, if `pdfroff` is chosen as the `groff` process driver program.

71. In the unlikely event that any table of contents references are to be incorporated into the front-matter, it will be necessary to implement some strategy to defer recording of the associated reference data, until after the "`.toc file` *refdata.toc*" assignment has been completed.

72. The "`.toc file` *refdata.toc*" assignment *must* have been completed *before* the first of any table of contents reference data collection requests appears in the document body specification; since execution of the "`.toc file` *refdata.toc*" assignment effectively clobbers all previously existing content in the "*refdata.toc*" file, the effect of any attempt to interpolate the table of contents reference data again, *after* this file assignment has been completed, is undefined; however, this limitation will *not* be observed, if `pdfroff` is employed as the `groff` process driver program.

Appendix A

GNU Free Documentation License
Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

A.0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

A.2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher

that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

A.6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

A.8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

A.9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

A.10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

A.11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B

Typesetting the GNU Free Documentation License

This appendix discusses, and documents the procedure which has been adopted to typeset the GNU Free Documentation License, as it is presented in Appendix A; the discussion is developed under the headings:

B.1. License Typesetting Constraints lxxvii
B.2. Choosing an Appropriate License Document Source Format lxxviii
B.3. Marking Up the Plain Text Document Source lxxviii
B.4. The GNU Free Documentation License Formatting Macros lxxxi
B.4.1. Marking Redundant Blank Input Lines — the FDL-BL Macro lxxxi
B.4.2. Introducing Numbered Section Headings — the FDL-SH and FDL-SM Macros lxxxi
B.4.3. Marking Paragraph Breaks — the FDL-LP Macro lxxxii
B.4.4. Interpreting Input Text Indentation — the FDL-TI Macro lxxxii
B.4.5. Formatting Itemized List Content — the FDL-IP Macro lxxxiii
B.4.6. Typesetting Block-Quoted License Text — the FDL-QS and FDL-QE Macros lxxxiii
B.5. Customizing the GNU Free Documentation License Formatting Process lxxxiv
B.5.1. Initial Customization — the FDL-SETUP Macro lxxxiv
B.5.2. Section Header Integration — the FDL-XH Macro lxxxvi
B.6. GNU Free Documentation License Finishing Touches lxxxviii

B.1. License Typesetting Constraints

The GNU Free Documentation License is a legal document, originally published by the Free Software Foundation in the United States of America. As such, it is written in the English language, conforming to the idiomatic style, and spelling conventions which prevail throughout the USA, and its dependencies; while its presentational style, and layout may be adjusted, to match the conventions of the containing document, any modification of its textual content, and paragraph structure, is strictly prohibited.

Notice that the language of *this* containing document is also English, but its idiomatic and spelling conventions are those which are prescribed by the Oxford English Dictionary, and generally adopted throughout the English speaking world, where the dominant influence is that of the British Commonwealth; these conventions, and in particular those related to spelling, often exhibit differences — frequently significant — from the conventions which prevail in the USA. However, in the spirit of conformity with the foregoing constraints, the idiomatic and spelling conventions of the GNU Free Documentation License have been preserved, verbatim, not only throughout its encapsulating appendix,[73] but also within all references thereto.

73. One minor presentational departure — which could potentially be misinterpreted as a content change — may be observed: each section number, as it appears within its respective numbered section heading, has been qualified by a prefix, representing the encapsulating appendix number; thus, a verbatim representation of each original numbered heading may be reproduced, by the simple expedient of removing this appendix number prefix.

B.2. Choosing an Appropriate License Document Source Format

The GNU Free Documentation License, as published by the Free Software Foundation, is available in several distinct document type formats; unfortunately, none of these is immediately suitable as input for *direct* processing by `groff`, (or by `pdfroff`), irrespective of any particular choice of primary typesetting macro package. Thus, in order to include the GNU Free Documentation License as Appendix A of this publication, it is necessary to choose one of the existing, published forms of the GNU Free Documentation License document, and adapt it for formatting by `groff`, while honouring the requirement that its text remains unchanged; this entails removal of any existing presentational mark-up code, and insertion of appropriate `groff` mark-up code, while preserving *all* existing document text, *exactly* as it is.

Choosing from the collection of various GNU Free Documentation License document formats, which are available from the Free Software Foundation's web-site, the most likely candidates for adaptation to formatting by `groff` would appear to be either the plain text variant, "`fdl-1.3.txt`", or the Texinfo variant, "`fdl-1.3.texi`". Of these two, the Texinfo variant includes existing formatting mark-up, which *could*, potentially, be converted — although perhaps not so readily by use of an automated procedure — to `groff` mark-up, whereas the plain text variant includes no such mark-up; however, structural analysis of the plain text document *does* suggest that automated insertion of suitable `groff` mark-up may be practicable. Thus, after consideration of the properties of these two original documents, the plain text variant, "`fdl-1.3.txt`", has been adopted as the original source for the representation of the GNU Free Documentation License, as it appears within this publication.

B.3. Marking Up the Plain Text Document Source

An unmodified copy of the file, "`fdl-1.3.txt`", as downloaded from https://www.gnu.org/licenses/fdl-1.3.txt, has been imported into the source tree for this publication, under the alternative name of "`fdl/fdl-v1.3.txt`". While formatting the encapsulating document, this file is processed as directed by the "`fdl/Makefile.sub`" specification, merging it with the "`fdl/fdl-v1.3.ms.in`" template, and adding appropriate formatting macro call mark-up, as may be required, to produce the "`fdl-v1.3.ms`" intermediate input file; this is then interpolated into the input stream for the encapsulating document, to produce the formatted content of the GNU Free Documentation License appendix, as it appears within the finished publication.

Processing of the "`fdl/fdl-v1.3.ms.in`" and "`fdl/fdl-v1.3.txt`" files, in this order, is performed by an `awk` script, (which has been encoded as an embedded script, within "`fdl/Makefile.sub`"). This script begins by initializing a pair of internal tags:

```
BEGIN { bl_tag = qs_tag = "BL"; }
```

This initialization is followed by the statement:

```
FILENAME ~ /.in$/{ print; next; }
```

which causes the "`fdl/fdl-v1.3.ms.in`" file — and indeed, any other input file with a "`.in`" final extension, which may have been specified — to be copied in its entirety, line by line, to the "`fdl-v1.3.ms`" intermediate input file, *without* any further processing by the script; thus, since "`fdl/fdl-v1.3.ms.in`" is the first input file to be processed, a verbatim copy of its content, which defines a collection of `groff(7)` macros, as specified in Appendix B, section B.4. , "The GNU Free Documentation License Formatting Macros", will be propagated to the beginning of the resulting "`fdl-v1.3.ms`" intermediate input file.

Having provided this mechanism for verbatim copying of any specified "`*.in`" input files, the remainder of the script handles the transcription of the "`fdl/fdl-v1.3.txt`" input file, inserting formatting mark-up, based on the macros which are defined in "`fdl/fdl-v1.3.ms.in`", in the process. The first step which is performed, in this stage of the transcription process, is:

```
/^[ \t]*$/{ bl_count += 1; next; }
```

which filters out entirely blank input lines, while accumulating a tally[74] of such lines in any one contiguous group.

Each non-blank input line — i.e. each of those which passes through the preceding blank line filter — is classified, on the basis of its content, and assigned to one of the following context groups:

- Indented text, identified by matching the beginning of the entire input line to the regular expression:

  ```
  /^ /
  ```

  (thus requiring at least one space to precede the non-blank content of the input line).

74. Note that the `bl_count` variable, in which this tally is accumulated, is not *explicitly* initialized at the start of the script; rather, it `is` explicitly (re)set, to zero, during processing of any non-blank line, while relying on `awk`'s *implicit* initialization of previously unreferenced variables, to zero, in the event that any blank line is processed, *before* any such reset is actioned.

- Numbered section headings; these are identified as any line which matches the regular expression:

  ```
  $1 ~ /^([0-9]+\.)+$/
  ```

  This expression is evaluated *after* the test for indented content; thus, it is subject to the constraint that the input line exhibits *no* leading space, preceding the section number field, which *must*, itself, be placed at the beginning of the input line.

- List items; these are identified by an initial line which matches the regular expression:

  ```
  $1 ~ /^[A-Z]+\.$/
  ```

  (once again, as in the case of the section number field of a numbered section heading, *without any* white space preceding the upper case alphabetic item label field).

  In cases where the content of any single list item spans more than one input line, the additional input lines, within the context of each item, are identified by matching the indented text expression.

- Addendum headings; identified by an *exact* match at the beginning of the input line:

  ```
  $1 == "ADDENDUM:"
  ```

  (once again, with *no* white space preceding the "`ADDENDUM:`" label); such input lines are interpreted similarly to those which specify numbered section headings, *except* that the associated heading will be *unnumbered*, with the prefix, "**ADDENDUM:**" replacing the section number.

- Any non-blank input line, which does not match any of the preceding classification criteria, is regarded as a *normal* input line, and remains otherwise *unclassified*.

Regardless of classification, any non-blank input line which is not, itself, classified as a numbered section heading, and is processed *before* the *first* line to be so classified, is deemed to represent a constituent of the license document front-matter; within this section of the license document, any occurrence of either of the three character strings, "`(C)`", or "`(c)`", is converted, by application of the `awk` substitution:[75]

```
qs_tag == "BL" { sub( /\([Cc]\)/, "\\(co" ); }
```

to the equivalent typographic copyright symbol, as represented by `troff`'s "`\(co`" special character code.

In addition to the preceding typographic substitution, which affects *only* the representation of the copyright symbol in the front-matter section of the license document, several other typographic substitutions are performed on *every* non-blank input line, regardless of any pending classification; first, the substitution:

```
sub( /[ \t]+$/, "" );
```

removes any *insignificant* trailing white-space — i.e. space and horizontal tab — characters from the end of any input line which exhibits them. Following this, the substitution:

```
gsub( /--/, "\\(em" );
```

inserts a typographic em-dash, in place of any contiguous pair of ASCII hyphen/minus characters, the substitution sequence:

```
if( gsub( /"[^"]*"/, "\\%lq&\\%rq" ) > 0 )
{ gsub( /\\%lq"/, "\\(lq" ); gsub( /"\\%rq/, "\\(rq" );
}
```

inserts a balanced pair of typographic double quotation marks, around any character sequence within any single input line, which is enclosed between ASCII double quotation marks, in place of those ASCII quotation marks, and a final substitution sequence:

```
if( gsub( /[^- ]+-/, "\\%&" ) > 0 )
{ gsub( /-\\%/, "-" );
  while( gsub( /\\%[^- ]+-/, "&%!%hy" ) > 0 )
  { gsub( /-%!%hy/, "\\(hy" );
  }
}
```

identifies any hyphenated string, in which each hyphen is represented by an ASCII hyphen/minus character, inserts typographic hyphens, as represented by `troff`'s "`\(hy`" special character code, in place of each corresponding ASCII

75. It may be noted that identification of license document front-matter context is predicated on the truth of the condition:

    ```
    qs_tag == "BL"
    ```

    The state of this condition is *explicitly* made `true`, during `awk` script initialization, but it becomes `false`, when the first — and indeed, any subsequent — input line is classified as a numbered section heading; thus, the license front-matter is *implicitly* terminated, on identification of the first numbered section heading input line.

character, and prefaces the entire hyphenated string with the "`\%`" control code, to suppress line breaking of the typeset text, within any such hypenated string.

Each of these typographic substitutions, where appropriate, together with input line classification, is performed on each input line, as it is read; based on classification, markup to invoke any of the formatting macros, as documented in the following section, is added, and the resulting, marked up input line is copied to the intermediate `troff` input file.

To facilitate the introduction of macro call mark-up, the `awk` function:

```
function macro( tag ){ return "."TXT2ROFF_PREFIX"-"tag; }
```

is defined. Typically, having arranged for the `awk` variable, "`TXT2ROFF_PREFIX`", to represent the string, "`FDL`", this function is called, using a statement of the form:

```
print macro( tag ), <arguments> ...;
```

to insert marked-up content into the `troff` intermediate input file.

One further `awk` function:

```
function bl_insert( tag )
{ if( bl_count > 0 ){ print macro( tag ), bl_count; bl_count = 0; }
}
```

is also defined. This facilitates context-sensitive interpretation of the effect of immediately preceding blank lines, when classifying input lines for marking-up; it inserts additional mark-up, appropriate to the identified context, when a preceding blank line effect is pending, and clears the pending blank line indicator. With the assistance of this pair of functions, the mark-up of classified input lines is performed, in the following order, by a series of `awk` code sequences; the first of these adds appropriate mark-up for lines with explicit indentation:

```
/^ /{ bl_insert( qs_tag );
      ct = length( $0 ); sub( /^ +/, "" );
      print macro( "TI" ), ct - length( $0 ), $0;
      next;
    }
```

which uses the "`FDL-TI`" macro to mark up *all* indented input lines, noting that immediately preceding blank line interpretation is predicated on the content of the "`bl_tag`" string variable, in order to differentiate indentation of block-quoted content, from indented continuation of list items, and from indentation which is specified to centre content within the license front-matter. Following this, numbered section headings are marked up, by the code:

```
$1 ~ /^([0-9]+\.)+$/{
      bl_insert( "BL" );
      bl_tag = "LP"; qs_tag = "QS"; print macro( "SH" ), $0;
      next;
}
```

which *always* marks up immediately preceding blank lines using the "`FDL-BL`" macro, while arranging to use the "`FDL-LP`" macro in place of blank lines which precede *unclassified* input lines, and the "`FDL-QS`" macro in place of blank lines which precede indented input lines, *after* the *first* numbered heading has been processed, and marking up each numbered heading, itself, using the "`FDL-SH`" macro. Additionally, the code:

```
$1 == "ADDENDUM:" { bl_insert( "BL" ); print macro( "SH" ), $0; next; }
```

will mark up any addendum heading, such that it will be formatted similarly to a numbered heading, substituting the "**ADDENDUM:**" label in place of a section number. This is then followed, in turn, by:

```
$1 ~ /^[A-Z]+\.$/{
      bl_insert( "BL" ); print macro( "IP" ), $0;
      next;
}
```

thus marking up the initial line of any list item, using the "`FDL-IP`" macro, (assuming that any continuation lines will be identified as indented lines, and will be marked up accordingly).

Finally, the `awk` code:

```
{ bl_insert( bl_tag ); print }
```

which is executed in respect of every *unclassified* input line, copies all such lines to the `troff` intermediate input file, *without* supplementary mark-up, while ensuring any such line, which is immediately preceded by one or more blank input lines, will be preceded by either a "`.FDL-BL <n>`" mark-up line, or a "`.FDL-LP <n>`" mark-up line, in the

troff intermediate input file, depending on whether it appears *before*, or *after* encountering the first numbered section heading, respectively.

B.4. The GNU Free Documentation License Formatting Macros

With the exception of the typographic substitutions,[76] the formatting effects of the mark-up, which is introduced as described in the preceding section, is achieved *entirely* through calls to the various macros which are defined in the "fdl/fdl-v1.3.ms.in" file. The implementation, and usage of each of these macros, together with an explanation of the context in which each is used, is documented below.

B.4.1. Marking Redundant Blank Input Lines — the FDL-BL Macro

With an effective implementation defined as:

```
.de FDL-BL
..
```

the FDL-BL macro does nothing when called. It is inserted, into the troff intermediate input file, in the form:

```
.FDL-BL <n>
```

to mark any point at which a contiguous sequence of <n> input lines has been discarded, by the blank line filter, (*except* in any context in which the blank lines serve as a new paragraph marker). Since this macro has no formatting effect, its principal value is to facilitate reconstruction of the original input file, from the content of the generated troff intermediate input file; since it does nothing, when called, it is also used internally, as an alias for other macros, in any context in which their normal effect is to be suppressed.

B.4.2. Introducing Numbered Section Headings — the FDL-SH and FDL-SM Macros

Introduced into the troff intermediate input file, in the form:

```
.FDL-SH <input-line-content>
```

when the input line under consideration has been classified as *either* a numbered section heading, *or* an addendum heading, (either of which will, typically, be preceded by one or more blank input lines, the presence of which will *always* be indicated by the use of the FDL-BL macro), the implementation of the FDL-SH macro is:

```
.de FDL-SH
.   FDL-QE
.   SH 2
.   if dFDL-XH .FDL-XH \\$@
.   ds \\$0-1 "\\$1\"
.   if \B'\\$1' \{\
.      if r XH-APPENDIX-NUMBER .ds \\$0-1 "\\n[XH-APPENDIX-NUMBER].\\$1\"
.   \}
.   shift
.   nop \\*[FDL-SM \\*[\\$0-1] 1] \\$*
.   rm \\$0-1
..
```

When invoked, with its argument list representing a verbatim copy of the input line from whence its use has been advocated, the FDL-SH macro will first terminate any block-quoted context which may be active, (by executing the FDL-QE macro). It will then prepare to print a heading, at a type size corresponding to the containing document's level two numbered headings, (reserving level one for the appendix title block), and, if the optional, user-defined FDL-XH hook macro has been provided, it will be called, to propagate the heading text to the table of contents, or to a document outline, (or to both).

Next, the the first argument is copied to temporary, local string storage, whence it will be interpreted as a heading label, which will be prefaced by the associated appendix number, *if* it is identified as being numerically valued; the original specification of this label is then discarded from the argument list.

76. It should be noted that, while such substitutions *do* change the *physical* content of the input lines, their effect is restricted to typographic presentational style — primarily of punctuation — *only*; none affect the textual content of the license document.

Finally, the resultant heading is printed, with the initial label being set at a reduced type size, by expansion of the variant string macro:

```
.ds FDL-SM "\s'-\\$2z'\\$1\s'+\\$2z'\"
```

and the temporarily stored label is discarded.

B.4.3. Marking Paragraph Breaks — the `FDL-LP` Macro

With its implementation defined as:

```
.de FDL-LP
.   FDL-QE
.   LP
..
```

the `FDL-LP` macro is inserted into the `troff` intermediate input file, in the form:

```
.FDL-LP <n>
```

to mark any point at which a contiguous sequence of `<n>` input lines has been discarded, by the blank line filter, *after* the first numbered section heading has been processed, *and* the immediately following input line is *unclassified;* (this represents the context in which the discarded blank input lines signify a paragraph break, and thus, in which the use of the `FDL-BL` macro is *not* appropriate).

While the `FDL-LP` macro may be considered as a synonym for the `FDL-BL` macro, for the purpose of reconstructing the original input file from the generated `troff` intermediate input file, (for which purpose the `<n>` argument is significant), in normal use, when the `troff` intermediate input file is interpreted, the `<n>` argument is ignored, any active block-quoted context is implicitly closed, (by execution of the `FDL-QE` macro), and a new paragraph is initiated, by execution of the `ms LP` macro.

B.4.4. Interpreting Input Text Indentation — the `FDL-TI` Macro

Within the "`fdl/fdl-v1.3.txt`" input file, text indentation is indicated by an arbitrary number of spaces, at the beginning of the indented input line; within the generated `troff` intermediate input file, it will be interpreted on a basis which is determined by context:

- *Before* any numbered section heading has been identified, the indentation is assumed to introduce centred content, within the license document's front-matter; this content, *without* the initial spaces, will be formatted under the aegis of a `troff ce` request.
- *After* any numbered section heading has been identified, and when *immediately* preceded by the initial line of an itemized list entry, (i.e. an immediately preceding line which has been marked up with a call to the `FDL-IP` macro), the indentation is interpreted as indicating continuation of the list item content; it will be formatted, (once again, ignoring *all* initial spaces), in accordance with the conventions pertaining to indented paragraphs, as established by calling that macro.
- In any other context, the indentation will be interpreted as an offset, which is applied to "block-quoted" text, (which, assuming that its initial input line was preceded by at least one blank input line, should have been marked up with a preceding call to the `FDL-QS` macro); formatting, (once again, ignoring all initial spaces), will be performed in accordance with the conventions established by calling that macro.

Regardless of which of these contexts may apply, *all* indented input lines, (after removal of their initial spaces), are marked up by a macro call of the form:

```
.FDL-TI <n> <input-line-content>
```

in which the `<n>` argument represents the number of indenting spaces which were removed, while the aggregate of all remaining arguments, specified in place of `<input-line-content>`, represent the residual content of the input line, *after* removal of the initial spaces.

The implementation of the `FDL-TI` macro is defined as:

```
.de FDL-TI
.   shift
.   nop \\$*
..
```

which, when invoked, effectively ignores its `<n>` argument, and passes the content of the original input line, excluding any extraneous spaces, as represented by `<input-line-content>`, to the formatter.

B.4.5. Formatting Itemized List Content — the `FDL-IP` Macro

The GNU Free Documentation License includes just one itemized list; each of its constituent items is labeled with an alphabetically serialized upper-case letter.

Identified by matching the `awk` regular expression:

```
$1 ~ /^[A-Z]+\.$/
```

the first input line of each constituent items content is copied to the `troff` intermediate input file, with the addition of mark-up in the form:

```
.FDL-IP <input-line-content>
```

in which the `<input-line-content>` argument list represents the entire content of the input line, *including* the initial, alphabetically serialized, upper-case item label.

The implementation of the `FDL-IP` macro, itself, is defined as:

```
.de FDL-IP
.   if '\\$1'A.' \{\
.      af fdl:item A
.      nr fdl:item 0 1
.      FDL-QPUSH
.   \}
.   shift
.   IP \\*[FDL-SM \\n+[fdl:item] 1.25]. \w'W.n'u
.   nop \\$*
..
```

When this is executed, its first action is to examine its first argument, which represents the the serialized alphabetic item label, extracted from the `<input-line-content>` argument list. If this is found to be *identically* equivalent to "`A.`", then the associated input line is deemed to represent the beginning of the content of the first entry in a new item list; the internal register, "`fdl:item`", is initialized as an enumeration counter, with upper-case alphabetic presentation format, which will be used as the item label for the first, and all subsequent, items in the ensuing list. Furthermore, in addition to initialization of the enumeration counter, this item list initialization procedure calls the `FDL-QPUSH` macro, (which is internally aliased to the `FDL-QS` macro), thus opening an indented section, (emulated as a block-quote section), within which the entire content of the item list will be encapsulated; (this will be closed *implicitly*, when the next section heading, or normally formatted paragraph is encountered).

It may be noted that the preceding initialization is performed *only* when processing the *first item* in any item list; it will be bypassed, when processing any subsequent item.

Regardless of whether item list initialization is performed, or bypassed, execution of the `FDL-IP` macro concludes by discarding the representation of the item label, as propagated from the original input line, and then invokes the `IP` macro, from the standard `ms` macro package, deriving the label from the auto-incrementing "`fdl:item`" enumeration counter,[77] (which is set in a reduced type size, using the same `FDL-SM` string interpolation macro as is used by the the `FDL-SH` macro), allocating sufficient paragraph indentation to accommodate the widest possible label, which is derived from this counter, and ultimately, passing the remaining content from the original input line to the formatter, as the initial content for the ensuing indented paragraph.

B.4.6. Typesetting Block-Quoted License Text — the `FDL-QS` and `FDL-QE` Macros

Any sequence of input lines, which have been marked up in the form of calls to the `FDL-TI` macro, and which are *immediately* preceded by one or more blank input lines, will be classified, as noted in the preceding discussion of the `FDL-TI` macro, as block-quoted text; in this case, the preceding sequence of blank input lines will be marked up as a call to the `FDL-QS` macro, thus opening a block-quoted document section. This section will, ultimately, be closed when the complementary `FDL-QE` macro is called, *implicitly*, during the next following execution of either the `FDL-SH` macro, or the the `FDL-LP` macro.

Internally, the `FDL-QS`, and the `FDL-QE` macros are implemented by mapping to alternatively named macros, `FDL-QPUSH` and `FDL-QPOP` respectively; the `FDL-QS` macro is *persistently* aliased to the `FDL-QPUSH` macro, whereas the `FDL-QE` macro is aliased to the `FDL-QPOP` macro, *only* when a block-quoted section is open, and to the `FDL-BS` macro, otherwise.

---

77. The "`fdl:item`" enumeration counter is *not* expunged, on completion of any `FDL-IP` macro call; it *must* persist, for use in any subsequent `FDL-IP` call, in which initialization is *not* performed.

The implementation of the FDL-QPUSH macro, (and thus, also of the aliased FDL-QS macro), is defined as:

```
.de FDL-QPUSH
.   als FDL-QE FDL-QPOP
.   QS
..
```

while that of the complementary FDL-QPOP macro (but *not* initially, as an alias for the FDL-QE macro), is:

```
.de FDL-QPOP
.   als FDL-QE FDL-BL
.   QE
..
```

Initially, the FDL-QE macro is implemented as an alias for FDL-BL; thus, when called implicitly by either the FDL-SH macro, or the FDL-LP macro, and there is no block-quoted section, which has been opened by a prior invocation of the FDL-QS macro, the implicit FDL-QE call will exhibit its default FDL-BL personality, with its "no-op" behaviour.

Conversely, a prior call of the FDL-QS macro will, through its aliased FDL-QPUSH implementation, remap FDL-QE to its alternative FDL-QPOP persoality, and will establish a set of formatting conventions as prescribed by invocation of the standard ms macro package's QS macro. This state will persist until the next subsequent invocation of the FDL-QE macro, (whether implicit, or explicit),[78] which, now exhibiting its FDL-QPOP personality, will remap itself, so that it reverts to its default FDL-BL personality, (and thus, to its default "no-op" behaviour, for any further subsequent calls which are *not* preceded by another FDL-QS call), and invokes the standard ms macro package's QE macro, to restore the formatting conventions to those which were in effect *before* the most recently preceding QS macro call, (which is assumed, nominally, to have occurred as a result of calling the FDL-QS macro).

B.5. Customizing the GNU Free Documentation License Formatting Process

The collection of macros, implemented as described in Appendix B, section B.4. , "The GNU Free Documentation License Formatting Macros", provides a sufficient mark-up framework for typesetting of the license using groff; however, when incorporated as an appendix, some customization may be required, in order to adapt to the formatting conventions of the containing document. To facilitate this, the mark-up macro framework provides a pair of hooks, namely FDL-SETUP, and FDL-XH, by means of which the default behaviour may be augmented, by provision of correspondingly named, user-defined macros.

B.5.1. Initial Customization — the FDL-SETUP Macro

Invoked by the *conditional* macro call:

```
.if dFDL-SETUP .FDL-SETUP
```

the FDL-SETUP macro is invoked, *after* the collection of mark-up macros described in section B.4 have been defined, and *before* proceeding with the formatting of the marked up content of the "fdl/fdl-v1.3.txt" file, *only if* the user has provided a definition of it, *before* the content of the "fdl-v1.3.ms" intermediate input file is interpreted.

For the purpose of incorporating a copy of the GNU Free Documentation License into this publication, customization of the basic FDL macros is performed by inclusion of the package-local "fdl-setup.tmac" macro file; this defines the FDL-SETUP macro as:

```
.de FDL-SETUP
.\" Locally initialize txt2roff behaviour, for interpretation
.\" of generated mark-up within fdl-v1.3.ms
.\"
.   ds fdl:replacement.tag START
.   FDL-BACKUP-AND-REPLACE FDL-SH FDL-TI
.   rm FDL-BACKUP-AND-REPLACE fdl:replacement.tag
..
```

78. In its present "fdl-v1.3.txt" incarnation, the GNU Free Documentation License does not require explict closure of any FDL-QS block-quoted context, and the marked up "fdl-v1.3.ms" intermediate input file never requests it; for a more robust implementation of the automated mark up procedure, consider adding an

    ```
    END { print ".FDL-QE" }
    ```

    statement, to the current implementation of the awk processing script.

Since the FDL-SETUP macro, itself, is invoked *without* passing *any* arguments, the foregoing implementation serves, primarily, as a wrapper for the FDL-BACKUP-AND-REPLACE helper macro, which *does* accept arguments:

```
.de FDL-BACKUP-AND-REPLACE
.\" Backup, and define temporary overrides for specified macros.
.\"
.   while \\n(.$ \{\
.      rn \\$1 \\$1-BACKUP
.      rn \\$1-\\*[fdl:replacement.tag] \\$1
.      shift
.   \}
..
```

When invoked, as it is, with "FDL-SH", and "FDL-TI" as its arguments, this backs up, (by renaming the pair as "FDL-SH-BACKUP", and "FDL-TI-BACKUP", respectively), and temporarily replaces their initial implementations, by substitution of alternatives, originally named "FDL-SH-START", and "FDL-TI-START", respectively.

The combined effect of this pair of temporarily redefined macros, (which persists *only* until the first instance of the redefined FDL-SH macro is processed), results in interpretation of any input lines, which have been marked up with the FDL-TI macro, to be interpreted as specifying the content of a "title block", introducing the content of the GNU Free Documentation License itself; the redefined FDL-TI macro, which performs this function, is:

```
.de FDL-TI-START
.\" Early-use override for FDL-TI; this remains in effect until
.\" the first use of (overridden) FDL-SH, taking control of the
.\" FDL title block.
.\"
.   ie !r fdl:title.lines \{\
.   \" On first time of use, initialize to accept five lines of
.   \" title block, with the first two set at the size of level
.   \" two headings, and centred.
.   \"
.      nr fdl:title.lines 5 1
.      SH 2
.      ce 2
.   \}
.   el \{ .ie \\n-[fdl:title.lines]=3 \{\
.   \" Set the third and fourth lines at a point size reduced
.   \" from normal paragraph text, and still centred.
.   \"
.      LP
.      SM
.      vs -3p
.      sp 1.5v
.      ce 2
.   \}
.   el .if \\n[fdl:title.lines]=1 \{\
.   \" Collect any further title lines into a filled diversion,
.   \" to be flushed out, and centred, when FDL-SH is called for
.   \" the first time.
.   \"
.      di fdl:title.block
.   \}\}
.   \\$0-BACKUP \\$@
..
```

Invocation of this redefined FDL-TI macro results in up to *four* lines of input text being *immediately* typeset as a centred title, (with the first *two* lines at an increased type size, and the following two at reduced size); any further input lines, beyond the *fourth*, are then collected into a diversion, which will, ultimately, be typeset when this specialized interpretation is terminated.

Regardless of the ultimate disposition of any input lines, which are processed while this redefined implementation of the FDL-TI macro remains in effect, its actual typesetting effect is delegated to the original FDL-TI implementation, which has been temporarily designated as FDL-TI-BACKUP.

Cancellation of this modified FDL-TI macro behaviour occurs, eventually, when *any* instance of the FDL-SH macro is encountered, in *its* modified guise:

```
.de FDL-SH-START
.\" First-use override for FDL-SH; it completes formatting and
.\" output of the title block, cleans up the entire FDL override
.\" context, restoring normal behaviour for output of the first
.\" section heading, and the remainder of the fdl-v1.3 text.
.\"
.   if '\\n(.z'fdl:title.block' \{\
.      br
.      di
.      unformat fdl:title.block
.      sp 1.5v
.      ce 10
.      fdl:title.block
.      ce 0
.   \}
.   sp 1i
.   FDL-RESTORE \\$0 FDL-TI
.   rm FDL-RESTORE fdl:title.block
.   rr fdl:title.lines
.   \\$0 \\$@
..
```

When this variant of the FDL-SH macro is invoked, its first action is to reprocess, and typeset, any residual title block content, which has been deferred by storing within a local diversion, created by the preceding variant of the FDL-TI macro. It then reverts *both* the FDL-SH macro, and the FDL-TI macro, to their original, default implementations, thus restoring their default behaviours, for any subsequent invocations. Finally, it cleans up all temporary local storage, which is associated *exclusively* with its modified initial behaviour, and then, it passes its own section heading arguments to its reinstated original implementation, for typesetting.

To facilitate reinstatement of the original, default implementations of the FDL-SH, and FDL-TI macros, the modified variant of FDL-SH calls a helper macro, FDL-RESTORE, which is implemented as:

```
.de FDL-RESTORE
.\" Restore original behaviour of overridden macros.
.\"
.   while \\n(.$ \{\
.      rn \\$1-BACKUP \\$1
.      shift
.   \}
..
```

This helper macro is subsequently deleted, during the clean-up of temporary storage, by the initially modified variant of the FDL-SH macro.

B.5.2. Section Header Integration — the FDL-XH Macro

The second of the hooks, which is exposed by the FDL macros, namely the FDL-XH macro, is invoked *exclusively* by the FDL-SH macro, using a *conditional* call of the form:

```
.if dFDL-XH .FDL-XH \\$@
```

passing quoted arguments, which *exactly* reproduce those passed in the referring FDL-SH macro call itself; the intent is to provide a mechanism for propagation of the section heading specification, beyond the immediate internal scope of this referring FDL-SH macro call.

While the FDL-XH macro will be called, *only if* the user has actually provided an implementation, the package-local "fdl-setup.tmac" macro file, as used in the production of this publication, *does* provide such an implementation, which is defined as:

```
.de FDL-XH
.\" Propagate FDL section headings to the document outline,
.\" and to the table of contents.
.\"
.   ie \B'\\$1' \{\
.      pdfhref O 2 \\n[XH-APPENDIX-NUMBER].\\$*
.      \\$0-UPDATE-TOC \\$@
.   \}
.   el \{\
.      pdfhref O 2 \\$*
.      XH-UPDATE-TOC 3 \h'\w!0.!u'\\$*
.   \}
..
```

When called by FDL-SH, this FDL-XH macro examines its copy of the FDL-SH arguments, in order to discriminate between numbered sections, and "addendum" sections; in the former case, it adds the appendix number, (expressed in its designated upper-case alphabetic format), as a dotted prefix to the assigned section number, and enters the resultant numbered section heading into the PDF document outline, before going on to call a further custom macro, named FDL-XH-UPDATE-TOC, to create a corresponding table of contents entry; in the latter case, it simply enters the given section heading, without change, into the PDF outline, and calls the default XH-UPDATE-TOC macro, to propagate this to the table of contents.

The implementation of the custom FDL-XH-UPDATE-TOC macro, which is called by the preceding FDL-XH macro when processing numbered section headings, is defined as:

```
.de FDL-XH-UPDATE-TOC
.\" Helper for FDL-XH, to propagate table of contents entries,
.\" with text alignment for heading levels less than ten, with
.\" those for levels of ten or more.
.\"
.   ie \w'\\$1'<\w'10.' \{\
.   \" Heading level is less than ten, so append space equal
.   \" to the width of one digit, to achieve alignment, then
.   \" recurse to emit the table of contents entry.
.   \"
.      ds \\$0.$1 "\\$1\0\"
.      shift
.      \\$0 "\\*[\\$0.$1]" \\$@
.      rm \\$0.$1
.   \}
.   \" When heading level is ten or more, or alignment space
.   \" has already been added, emit the entry.
.   \"
.   el .XH-UPDATE-TOC 2 \\n[XH-APPENDIX-NUMBER].\\$*
..
```

The effect of calling this macro is to adjust the spacing between section numbers and the following heading text, when constructing the associated table of contents entries, to keep the heading text vertically left-aligned, within the table of contents, regardless of whether the preceding section number occupies one, or two digit spaces.

B.6. GNU Free Documentation License Finishing Touches

Although the automated process, which has been adopted for marking up the "`fdl/fdl-v1.3.txt`" file, such that it is made suitable for formatting with `groff`, can achieve reasonable results, it *does* leave some scope for improvement through certain manual adjustments. In particular:

- The automated process makes no attempt to address the avoidance of widow and orphan lines; this is best handled by judicious human intervention, and, in the case of typesetting for inclusion in this publication, `troff` conditional "`.ne 2v`" pagination requests have been manually inserted, at appropriate points in the generated "`fdl-v1.3.ms`" intermediate input file, *after* a critical inspection of the initial typeset result.
- The HTTP references, which may be found within the "`fdl/fdl-v1.3.txt`" file, are *not* automatically converted to live URL references. Adaptation of the automation script, to achieve this objective, seemed to pose rather more of a challenge than the simple expedient of manually adding the necessary mark-up, *after* generation of the "`fdl-v1.3.ms`" intermediate input file; thus, for the purposes of this publication, suitable "`.pdfhref W ...`" mark-up has been manually inserted into the automatically generated file.

The application of these finishing touches, together with the automatically inserted mark-up, as described in Appendix B, section B.4. , "The GNU Free Documentation License Formatting Macros", and the supplementary macros described in Appendix B, section B.5. , "Customizing the GNU Free Documentation License Formatting Process", produces the typeset rendition of the GNU Free Documentation License which is incorporated into Appendix A of this publication.

Appendix C

Working Macros used for Typesetting this Publication

This appendix provides reference listings for each of those macros which have been developed, specifically to facilitate the production of this publication. Implemented on the basis that the document is to be typeset using the `pdfroff` program, in conjunction with the `ms` and `pdfmark` macros, (as they have been integrated, and augmented, within the `spdf.tmac` implementation), these document-local macros may be categorized by section headings:

C.1. Controlling the Style of Formatted Text xc
C.2. Starting a Section on a New Recto Page — the `NEW-RECTO-PAGE` Macro xci
C.3. Generating and Typesetting the Table of Contents xciii
C.3.1. Table of Contents Reference Data Collection — the `XN-UPDATE-TOC` Macro xciii
C.3.2. Establishing a Table of Contents Layout — the `toc.outline` Macro xcv
C.3.3. Specifying Table of Contents Entry Context — the `toc.refmark` Macro xcvi
C.3.4. Specifying Page Number References — the `toc.pageref` Macro xcvii
C.3.5. Avoiding Excessive Use of `pdfhref` Link Colour in Tables of Contents xcvii
C.3.6. Additional Convenience Macros for Formatting Tables of Contents xcviii
C.4. Macros for Laying Out the Content of Appendicies c
C.4.1. Specifying an Appendix Title — the `XH-APPENDIX` Macro ci
C.4.2. Appendix Numbering Styles — the `XH-APPENDIX-NUMBER-FORMAT` Macro cii
C.4.3. Appendix Subsection Numbering — the `XH-APPENDIX-NH` Macro ciii

C.1. Controlling the Style of Formatted Text

In addition to the standard `ms` font styling macros, this document uses a number of locally defined variants. First, for emphasis, the text may be italicized, with a 0.3 point increment in size, using this variant of the "`I`" macro:

```
.de EM
.   I "\s'+0.3z'\\$1\s'-0.3z'" "\\$2" "\\$3"
..
```

Furthermore, each of the following three variants of the "`CW`" macro:

```
.de CWB
.   nop \\$5\f(CR\\$3\fP\f(CB\\$1\fP\f(CR\\$2\fP\\$4
..
.de CWI
.   nop \\$5\f(CR\\$3\fP\f(CI\\$1\fP\f(CR\\$2\fP\\$4
..
.de CWBI
.   nop \\$5\f(CR\\$3\fP\f[CBI]\\$1\fP\f(CR\\$2\fP\\$4
..
```

will format the text of their first argument in *constant width bold*, *constant width italic*, and *constant width bold-italic*, respectively; in each case, the text of the second and third arguments will be formatted in the regular *constant width* font, as suffix and prefix respectively, bracketting the text of the first argument, *without* intervening space, and the entire construct will be bracketted by the text of the fourth and fifth arguments, in the font of the running text, again *without* intervening space, and again, as suffix and prefix respectively.

Finally, the variant string:

```
.ds = \f(CB\\$1\f(CR\\$4\f[CBI]\\$2\f(CR\\$3
```

offers a convenient mechanism for interpolating the "<*tag*>" argument to a "`.IP`" macro call, setting the text of the first string argument in the *constant width bold* font, followed by the second, if specified, in the *constant width bold-italic* font, bracketted by the third argument as suffix, and the fourth argument as prefix, each in the regular *constant width* font, and *without* intervening space between *any* of these interpolated string arguments.

C.2. Starting a Section on a New Recto Page — the NEW-RECTO-PAGE Macro

Occasionally, publishing convention will demand that a document section should commence on the right hand (recto) page, when the publication is printed, and bound; the NEW-RECTO-PAGE macro, which relies on features of the ms macros, and is implemented as defined below, facilitates conformance with this convention:

```
.de NEW-RECTO-PAGE
.\" Usage: .NEW-RECTO-PAGE <register-name> [<page-number-format>]
.\"
.\" Advance to the next available odd-numbered page, record
.\" its page number in <register-name>, set its effective page
.\" number to one, and apply any <page-number-format> which
.\" may have been specified, on this and subsequent pages.
.\"
.\" Initially, to facilitate comparison of page numbers, arabic
.\" numeral formatting is required; assignment of an alternative
.\" format, if specified by the optional second argument, must be
.\" deferred until later.
.\"
.   af % 0
.   af \\$1 0
.   ie \\n[\\$1] .nr \\$1 +1
.   el .nr \\$1 \\n%+1
.   if o \{\
.   \" The current page number is odd, so advancing one page will
.   \" make the page number even.  However, we want to advance to
.   \" to the next available odd numbered page, so we must insert
.   \" an additional blank page; we set this artificially to page
.   \" number one, so that ms will not print the page number, but
.   \" we do increment the true recorded number, so that this may
.   \" be correctly restored later.
.   \"
.      nr \\$1 +1
.      bp 1
.   \}
.\" Having guaranteed that we will land on an odd numbered page,
.\" advance one page, once again setting the effective landing page
.\" number to one, to suppress printing of the page number, while
.\" arranging for correct numbering to resume on the immediately
.\" following page.
.\"
.   bp 1
.   pn 1+\\n[\\$1]
.
.\" Only now, may an alternative page number format be assigned.
.\"
.   af \\$1 \\$2 0
.   af % \\$2 0
..
```

It may be observed that, as defined, this implementation of the NEW-RECTO-PAGE macro represents a consolidated adaptation of the concepts developed in section 4.2.3, "Ensuring that Content is Printed on a Particular Side of the Page", (and in particular, section 4.2.3.1, "Recto-Verso Page Break Handling when Using the ms Macros"). It does *not* support the complementary NEW-VERSO-PAGE capability, (although it could be adapted to do so); it *does* set the *effective* page number, on the page to which it advances, to *one*, to take advantage of the ms macro feature which suppresses the printing of page headers on any page which is so numbered, while using a user-nominated backup register, to record the *real* page number — the user *must* specify a suitable alternative (i.e. *not* "%") page number register name for the <*register-name*> argument, (such as the "%%" register which is used for this purpose, in this publication), to fulfil this requirement.

A practical *disadvantage*, which may arise from the use of a backup page number register to record the real page number, is that this register *must* be updated for *every* change of page number, *if* it is to remain valid as a basis for page number references, (such as those which are required when constructing a table of contents). To avoid the burden of maintaining the progression of the "%%" register, in this publication, it is defined *only* when the "%" has been assigned an artificial value of one, and is subsequently destroyed when normal progression of the "%" register is restored. The "%%" register is used as the basis for table of contents references, when it exists; its destruction is accomplished by adaptation of the HD macro:

```
.am HD
.   if r%% \{\
.   \" The "%%" register exists; it needs to remain, while the "%"
.   \" register is artifically set to zero, or one, and be removed
.   \" only after "%" has resumed its normal sequential progression;
.   \" as within NEW-RECTO-PAGE, we need "%" to exhibit the arabic
.   \" numeral format, to facilitate comparison, but must revert
.   \" to any alternative which is borne by the "%%" register.
.   \"
.      af % 0
.      if \\n%>1 \{\
.         af % \\g(%%
.         rr %%
.      \}
.   \}
..
```

which is invoked by an ms page position trap, at the start of *every* new page; after such destruction of the "%%" register, the basis for table of contents page number references reverts to the "%" register.

C.3. Generating and Typesetting the Table of Contents

The macros presented below offer a (mostly) transparent replacement for the standard capabilities of the ms macros, for the collection, and subsequent collation of table of contents reference data, which has been specified using either the XH, or the XN macro; (it has been designed, primarily, to support the latter).

When this implementation is deployed, it will *automatically* assume responsibility for collection of table of contents reference data, *without* any user intervention; however, it neither relies on, nor supports, *any* use of the XS, XA, or XE macros, which are traditionally used in ms, for collation of tables of contents. Furthermore, it is incompatible with the traditional ms technique of printing the table of contents *after* the document body, and relocating it later; thus, instead of using the TC macro for this purpose, the document source should incorporate code, (illustrated with reference to the actual file name used for *this* publication), similar to:

```
.ce 1
\fB\s'+2z'Table of Contents\s'-2z'\fP
.sp 1.5v
.so pdfmark.toc
.toc file pdfmark.toc
.toc end
```

at the point where the table of contents is to appear, in the ultimate document output.

C.3.1. Table of Contents Reference Data Collection — the XN-UPDATE-TOC Macro

Operating in conjunction with the ms macros, this replacement implementation of the XH-UPDATE-TOC macro intercepts table of contents data collection calls, originating from the XH, or the XN macro calls, thus overriding the standard ms table of contents handling mechanism, by redirecting the effects of such calls to corresponding handlers, which have been implemented in terms of the groff_toc macro framework.

The actual implementation of the XH-UPDATE-TOC macro, as it appears within the example spdf-toc.tmac file, distributed with groff-pdfmark and deployed in the production of *this* publication, is defined in two distinct parts; the first of these, which is a self-replacing stub, and is invoked *at most* once, the first time XH-UPDATE-TOC is called, whether by XH or by XN, is:

```
.de XH-UPDATE-TOC
.\" Record data for construction of a table of contents entry
.\"
.\" If pdfroff's PHASE register is defined, with any positive
.\" non-zero value, this macro has nothing to do, for this or
.\" any subsequent call; redefine it as empty, discarding its
.\" ultimate replacement, for all subsequent calls, and then
.\" immediately exit ...
.\"
.   if r PHASE \{.if \\n[PHASE] \{.rm \\$0-ULTIMATE
.      de \\$0 return
.      return
.   \}\}
.
.\" ... but, when the PHASE register is either not defined, or
.\" if it is defined, but it does not have a positive non-zero
.\" value, then install the ultimate replacement for this stub
.\" implementation of the XH-UPDATE-TOC macro, then continue
.\" execution, by passing control on to it.
.\"
.   rn \\$0-ULTIMATE \\$0
.   \\$0 \\$@
..
```

It may be noted that this initial implementation of the XH-UPDATE-TOC macro does not actually provide any data collection capability. Rather, it assumes that the formatting process is controlled by pdfroff, (or a compatible process, which makes similar use of a register named PHASE), and evaluates the state of this register, to determine whether table of contents data collection is necessary, or not; if not, it redefines itself to do nothing, when it is called

subsequently; otherwise, it installs, and ultimately replaces itself with, an alternative `XH-UPDATE-TOC` data collection handler, which is implemented as follows:

```
.de XH-UPDATE-TOC-ULTIMATE
.\" If we get this far, record the requisite reference data,
.\" ensuring that the actual data is captured at the point of
.\" ultimate output, (not when recording a diversion) ...
.\"
.   if '\\n(.z'' \{\
.   \" ... beginning with the outline nesting level ...
.   \"
.      toc put outline \\$1
.
.   \" ... following up with the outline reference name, and
.   \" document section number, as established by a prior .NH
.   \" macro call ...
.   \"
.      ds \\$0.text "\\*[PDFBOOKMARK.NAME] -- \"
.      ie \\$1=1 \{\
.      \" ... with font size increment, emboldening, and spacing
.      \" adjustment, as appropriate for top level headings ...
.      \"
.         as \\$0.text "\s'+1z'\fB\Z'\\$2'\fP\s'-1z'\"
.         as \\$0.text "\h'\\*[TC-HS0]+\w!0.!u'\s'+1z'\fB\"
.      \}
.      \" ... while maintaining normal font size, and style, for
.      \" all levels of sub-headings.
.      \"
.      el .as \\$0.text "\\$2\h'\\*[TC-HS1]'\"
.
.   \" Discard the two arguments, which have been interpreted
.   \" already, append all remaining arguments to the assembled
.   \" reference data record, export it, and clean up.
.   \"
.      shift 2
.      toc put refmark \\*[\\$0.text]\\$*\h'\\*[TC-HS2]'
.      rm \\$0.text
.
.   \" Finally, append a page number reference, to wrap up
.   \" this table of contents entry record.
.   \"
.      ie r %% .toc put pageref \\n(%%
.      el .toc put pageref \\n%
.   \}
.
.\" ... but, if a diversion is being recorded, reschedule
.\" this macro call for reprocessing, when the diversion is
.\" eventually flushed to the output stream.
.\"
.   el \!.\\$0 \\$@
..
```

After this ultimate table of contents data collector macro *has* been installed, by the initialization stub, it will be called *immediately*, to complete the handling of the initial `XH-UPDATE-TOC` macro call, which caused it to be installed; it will also be called *directly*, to handle each subsequent `XH-UPDATE-TOC` macro call.

A detailed inspection of this ultimate `XH-UPDATE-TOC` data collection implementation will reveal that it introduces dependencies on *three* supplementary `groff_toc` framework extension macros. Each of these must be supplied by the user; suitable implementations are presented in the following sections of this appendix.

C.3.2. Establishing a Table of Contents Layout — the `toc.outline` Macro

When table of contents data collection is performed by a `XH-UPDATE-TOC` macro, which has been implemented as shown in Appendix C, section C.3.1. , "Table of Contents Reference Data Collection — the `XN-UPDATE-TOC` Macro", the *first* of the supplementary macros, which will be called when each table of contents entry is to be output, is *always* invoked by a call of the form "`.toc outline <level>`"; each such call is serviced by the `toc.outline` macro, which is implemented as follows:

```
.de toc.end de
.de toc.outline
.   ie \\$1>1 \{\
.   \" Entries at outline-levels greater than one are indented
.   \" by an adjustable computed offset for each level; this is
.   \" controlled by assignment of a value to the "TC-HS1" string,
.   \" which is evaluated when executing the "XH-UPDATE-TOC" macro.
.   \" Additionally, on any increase of indentation level from that
.   \" of the preceding entry, which has been recorded in the local
.   \" register, "toc#outline.current", additional vertical space
.   \" equal to "TC-VS2" is inserted, while on decreasing to any
.   \" level which remains greater than one, similar additional
.   \" vertical space equal to "TC-VS3" is inserted.
.   \"
.      nr toc#indent (\w'0.'u+1.5n)*(\\$1-1)
.      if \\$1>2 .nr toc#indent +\w'0.'u*(\\$1-1)*(\\$1-2)/2
.      if !r toc#outline.current .nr toc#outline.current \\$1
.      ie \\$1>\\n[toc#outline.current] .sp \\*[TC-VS2]
.      el .if \\n[toc#outline.current]>\\$1 .sp \\*[TC-VS3]
.      nop \h'\\n[toc#indent]u'\c
.
.   \" After emitting the appropriate indentation space, the
.   \" tab-stop settings must be adjusted to compensate.
.   \"
.      nr toc#indent \\n[.l]-\\n[.i]-\\n[toc#indent]
.      ta (u;\\n[toc#indent]-\\n[TC-MARGIN]) (u;\\n[toc#indent])R
.      rr toc#indent
.   \}
.   el \{\
.   \" Entries at outline-level one are not indented, but each
.   \" except the first, (identified by not yet having defined
.   \" the "toc#outline.current" register), will be preceded by
.   \" vertical space equal to "TC-VS1"; in any case, any open
.   \" "KS" block is closed, (by calling "toc.end"), to permit
.   \" insertion of a conditional page break prior to the next
.   \" level one TOC entry, tab-stops are reset to comply with
.   \" "ms" convention, and a further "KS" block is opened, in
.   \" preparation for output of any level one TOC entry.
.   \"
.      toc.end
.      if r toc#outline.current .sp \\*[TC-VS1]
.      ta (u;\\n[.l]-\\n[.i]-\\n[TC-MARGIN]) (u;\\n[.l]-\\n[.i])R
.      als toc.end KE
.      KS
.   \}
.\" After initialization for the first TOC entry, tracking of the
.\" current outline-level is persistently maintained, by recording
.\" of the "toc#outline.current" register.
.\"
.   nr toc#outline.current \\$1
..
```

Irrespective of the particular text, which is to appear within each table of contents entry, this `toc.outline` macro is responsible for controlling the layout of all such entries, on the table of contents output page. Notice that, in addition to the user-redefinable layout specification strings, "`TC-HS0`", "`TC-HS1`", and "`TC-HS2`", (which are actually interpreted by, and the effect of which is propagated from, the `XH-UPDATE-TOC` macro), together with the further layout specification strings, namely "`TC-VS1`", "`TC-VS2`", and "`TC-VS3`", (each of which is interpreted by the `toc.outline` macro itself), this implementation of the `toc.outline` macro introduces a further dependency, on the supplementary `toc.end` macro; initially defined as an empty macro, and subsequently redefined as an alias for `KE`, this *should* be called by the user, (in the form of a "`.toc end`" macro call, as illustrated in the introduction to section 4.1.5, "How `pdfroff` Collates Tables of Contents"), *after* any inclusion of a collected table of contents specification file.

For convenience, and to ensure that there is no infiltration of extraneous trailing space into their respective definitions, the `TC-DEFINE` macro, which is defined as:

```
.\" The following TC-DEFINE macro provides a convenient mechanism
.\" for defining strings, WITHOUT propagating trailing spaces into
.\" the value assigned to the "TC-" prefixed named string.
.\"
.de TC-DEFINE
.   ds TC-\\$1 "\\$2\"
..
```

may be used to define, or to redefine, each of the "`TC-HS0`", "`TC-HS1`", "`TC-HS2`", "`TC-VS1`", "`TC-VS2`", and "`TC-VS3`" layout specification strings; for example, their initial definitions are:

```
.TC-DEFINE VS1 1.30v   \" leading for top level
.TC-DEFINE VS2 0.15v   \" leading at nesting level increment
.TC-DEFINE VS3 0.25v   \" leading following nested group
.
.TC-DEFINE HS0 1.50n   \" indentation adjustment at top level
.TC-DEFINE HS1 1.50n   \" space between section number and text
.TC-DEFINE HS2 0.50n   \" space between text and following leader
```

With the exception of "`TC-HS1`", each of these layout control strings retains its initial setting, throughout the processing of this publication; as the sole exception, the setting of "`TC-HS1`" is changed to:

```
.TC-DEFINE HS1 "1.5n-\w!.\ !u"
```

after formatting of the main document body, and Appendix A, "The GNU Free Documentation License", but before, and throughout the formatting of all of the remaining appendices, to compensate for a change in the method which has been adopted to generate section numbers within these appendices, and the manner in which space is allocated, on the output line, for incorporation of such section numbers into their respective table of contents entries.

C.3.3. Specifying Table of Contents Entry Context — the `toc.refmark` Macro

Following each invocation of "`.toc outline` <*level*>", as described in the preceding section, Appendix C, section C.3.2. , "Establishing a Table of Contents Layout — the `toc.outline` Macro", the contextual content of the entry is specified, by calling "`.toc refmark` <*refname*> `--` <*section*> <*context*> `...`"; this call is serviced by the `toc.refmark` macro, which is implemented as follows:

```
.\" Called immediately after toc.outline ...
.\"
.de toc.refmark
.\" ... this simply stores the destination tag name and the text
.\" for the TOC pageref entry, within the toc@refmark.tag and the
.\" toc@refmark.text strings respectively, whence they are ...
.\"
.   ds toc@refmark.tag \\$1
.   ie '\\$2'--' .shift 2
.   el .shift
.   ds toc@refmark.text \\$*
..
```

The effect of calling `toc.refmark` is to collect the section number reference, and its associated contextual information, conditioned by some layout control information which has been propagated from the initiating invocation of `XH-UPDATE-TOC`, into local string storage buffers, in preparation for eventual completion, formatting, and writing to the document output stream, as explained in the following section.

C.3.4. Specifying Page Number References — the `toc.pageref` Macro

Although the preceding sequence of two macro calls, the first invoking "`.toc outline <level>`", and the second invoking "`.toc refmark <refname> -- <section> <context> ...`", may have established the page layout, and the contextual content, for any individual table of contents entry, this pair of macro calls, alone, is insufficient to completely prepare that entry for output; it still lacks a page number reference! Thus, to complete the sequence, a further macro call, of the form "`.toc pageref <pagenum>`", is required; this is serviced by the `toc.pageref` macro, for which the implementation is:

```
.\" Completing a sequence of "toc.outline" and "toc.refmark"
.\" macro calls, a call to the following "toc.pageref" macro
.\" causes the prepared table of contents entry layout, and
.\" associated contextual information ...
.\"
.de toc.pageref
.\" ... to be incorporated into an active pdfhref link, along
.\" with the appropriate page number reference, and emitted to
.\" the document output stream.
.\"
.   ie \\n[toc#outline.current]>1 .as toc@refmark.text "\a\t\\$*\"
.   el .as toc@refmark.text "\f[]\s'-1z'\a\s'+1z'\fB\t\\$1\f[]\s'-1z'\"
.   pdfhref L -D \\*[toc@refmark.tag] -- "\\*[toc@refmark.text]"
.   rm toc@refmark.tag toc@refmark.text
.   br
..
```

C.3.5. Avoiding Excessive Use of `pdfhref` Link Colour in Tables of Contents

As has been noted, in section 4.4.3.1, given the implementation as detailed in the preceding sections of this appendix, with the exception of the heading on its initial page, the table of contents will comprise *exclusively* `pdfhref` links. If the default style is applied to these, the table of contents, almost in its entirety, will be rendered it the `pdfhref` link colour; this may be perceived to exhibit an unnecessarily distracting appearance! This potential distraction may be avoided by *manually* overriding the default `PDFHREF.TEXT.COLOUR` style, *immediately* before scheduling output of the table of contents:

```
.\" Break PDFHREF.TEXT.COLOUR and PDFHREF.TEXT.COLOR equivalence,
.\" then reassign PDFHREF.TEXT.COLOUR, (which is given precedence),
.\" to match the normal running text colour.
.\"
.rm PDFHREF.TEXT.COLOUR
.ds PDFHREF.TEXT.COLOUR "\n(.m\"
```

and subsequently reinstate it, *either* by *explicitly* restoring the original equivalence:

```
.\" Restore normal PDFHREF.TEXT.COLOUR handling, by reinstating
.\" PDFHREF.TEXT.COLOUR and PDFHREF.TEXT.COLOR equivalence.
.\"
.als PDFHREF.TEXT.COLOUR PDFHREF.TEXT.COLOR
```

or by delegating *automatic* reinstatement to `pdfhref`:

```
.\" Discard our overriding PDFHREF.TEXT.COLOUR assignment, thus
.\" scheduling automatic reinstatement of the normal equivalence
.\" of PDFHREF.TEXT.COLOUR and PDFHREF.TEXT.COLOR by pdfhref
.\"
.rm PDFHREF.TEXT.COLOUR
```

after completion of table of contents output.

While it may be reasonable to *manually* override PDFHREF.TEXT.COLOUR, for "one-shot" table of contents output, it may be found to be more convenient to encapsulate this override capability within a macro, as part of a groff_toc framework implementation; a suitable macro implementation might be:

```
.de toc.color als
.als toc.colour toc.color
.\" This implementation is common to both ".toc colour" usage,
.\" and ".toc color" usage; either may be used, interchangeably,
.\" according to individual user preference.
.am toc.colour
.\" Usage:
.\"   .toc colour [<colour-specification>]
.\"   .toc color [<colour-specification>]
.\"
.\" Break PDFHREF.TEXT.COLOUR and PDFHREF.TEXT.COLOR equivalence;
.\" it will be restored automatically, by pdfhref, if no altenative
.\" <colour-specification> argument is specified ...
.\"
.   rm PDFHREF.TEXT.COLOUR
.\"
.\" ... but, when such an argument IS specified, then assign it, so
.\" that it may take precedence over PDFHREF.TEXT.COLOR
.\"
.   if \\n(.$ .ds PDFHREF.TEXT.COLOUR "\\$1\"
..
```

This "convenience" macro may then be called, immediately *before* table of contents output, to substitute the normal running text colour as an override for the default PDFHREF.TEXT.COLOUR assignment:

```
.\" Override the default PDFHREF.TEXT.COLOUR assignment, with
.\" the normal running text colour.
.\"
.toc colour \n(.m
.so pdfmark.toc
```

and then again, *after* completion of this table of contents output, to schedule automatic, "on-demand" reinstatement of the default PDFHREF.TEXT.COLOUR interpretation:

```
.\" Delegate reinstatement of default PDFHREF.TEXT.COLOUR handling
.\" to pdfhref, on the next occasion of it being required.
.\"
.toc colour
```

C.3.6. Additional Convenience Macros for Formatting Tables of Contents

Although the macros, as presented in the preceding sections of this appendix, are sufficient to prepare, and to emit, a table of contents, some additional macros may be helpful; the added convenience will become particularly apparent, when the collected table of contents reference data is to be interpreted selectively, usually more than once, as described in section 4.4.4, "Classification and Selective Processing of Table of Contents Entries".

Some additional "convenience" functionality, which may be found to be useful, might include:

- Output of a suitably formatted table of contents heading, optionally with variants to place this at the top of a new page, and possibly to ensure that this will be a recto page.
- Restoration of layout controls to their default initial state, prior to reinterpretation of the collected table of contents reference data, (perhaps with a modified set of selection criteria).
- Reading in, and interpretation of the table of contents reference data file, with integral reinitialization of the layout control parameters, and management of the recommended PDFHREF.TEXT.COLOUR override, (see Appendix C, section C.3.5. , "Avoiding Excessive Use of pdfhref Link Colour in Tables of Contents").

Of these three suggested additional "convenience" features, the current publication does not implement the first; rather, it simply uses in-line requests to set up a new page, and emit a centred "Table of Contents" heading. If it is desired to encapsulate this functionality within macros, a suitable implementation might look something like:

```
.de toc.heading
.\" Usage: .toc heading [<heading text> ...]
.\"
.\" Print an emboldened, centred "Table of Contents" heading.
.\"
.ie \\n(.$ \{.ce 1
.   ft B
.   nop \s'+2z'\\$*\s'-2z'
.   ft
.   sp 2v
.\}
.el .\\$0 \\*[TOC]
..
.if !dTOC .ds TOC "Table of Contents\"
```

Notice that the preceding macro will simply place the "Table of Contents" heading at the current output position, on the current output page; it will *not* ensure that this placement is at the top of a page. In practice, such placement at the top of a page will be desirable; this may be achieved, if either a recto, or a verso page is acceptable, by simply preceding a "`.toc heading ...`" macro call with a "`.bp`" request:

```
.bp [<page-number>]
.toc heading [heading text ...]
```

or, (as will often be the case), to ensure that the table of contents starts on a recto page: [79]

```
.NEW-RECTO-PAGE %%
.toc heading [heading text ...]
```

The principal advantage of implementing *any* of the recommended "convenience" macro features is that they facilitate repeated interpretation of the collected table of contents reference data, (for example, to compile separate tables of contents, lists of figures, lists of equations, etc.); before any such reinterpretation of the table of contents reference data is performed, the internal state of the "`.toc outline`" layout controls must be restored to their initial condition; this may be facilitated by adoption of the second recommendation, implemented[80] as the convenience macro:

```
.de toc.reset
.\" Usage: .toc reset
.\"
.\" Restore the toc.end macro to its initial "do-nothing" state,
.\" by first defining a new empty macro, with a temporary name, and
.\" then renaming this, to replace any current implementation of
.\" the toc.end macro itself.
.\"
.   de \\$0.tmp rn
.   rn \\$0.tmp toc.end
.
.\" Furthermore, when beginning to format any new table of contents,
.\" the internal toc#outline.current register should be undefined.
.\"
.   rr toc#outline.current
..
```

which may then be invoked by macro calls of the form: "`.toc reset`".

When the table of contents reference data *is* to be interpreted more than once, then it is likely that, not only will it be necessary to restore the initial "`.toc outline`" state, but the author may also favour application of the recommended

79. See Appendix C, section C.2. , "Starting a Section on a New Recto Page — the `NEW-RECTO-PAGE` Macro".

80. Assuming that the preceding implementations of the `XH-UPDATE-TOC`, `toc.outline`, `toc.refmark`, `toc.pageref`, and `toc.end` macros have been adopted.

PDFHREF.TEXT.COLOUR override during each successive interpretation, as suggested in the third convenience recommendation; a suitable macro, for implementation of this recommendation, may be:

```
.de toc.load
.\" Usage: .toc load <file-name>
.\"
.\" Load, and interpret the named table of contents reference
.\" data file, after reinitialization of the .toc outline state,
.\" and overriding PDFHREF.TEXT.COLOUR, to match the colour of
.\" the running document text, (as set at the time when this
.\" macro is DEFINED, and NOT when it is EVALUATED).
.\"
.   toc reset
.   toc colour \n(.m
.   so  \\$1
.
.\" Restore normal PDFHREF.TEXT.COLOUR handling, and wrap up.
.\"
.   toc colour
.   toc end
..
```

C.4. Macros for Laying Out the Content of Appendicies

Although the standard page layout, and paragraph styling macros, which are provided by the author's choice of basic document formatting macro package, will be found to be generally sufficient for laying out the content of appendices, some additional macros, which are specifically tailored for the purpose, may be helpful for the specification of appendix titles, and appendix section headings. In the case of this publication, which has been formatted using groff's ms macro suite, these capabilities are supported by the provision of *three* additional macros, namely XH-APPENDIX, XH-APPENDIX-NUMBER-FORMAT, and XH-APPENDIX-NH, the implementations of which may be found within the supplementary appendix.tmac file; formulated as a complement to the spdf.tmac extended variant of groff's ms macros, the respective implementation, and usage, of each of these macros is discussed in the following subsections of this appendix.

When using this trio of macros, to lay out the content of appendicies, the XH-APPENDIX-NUMBER-FORMAT macro is, typically, called first, and just once, *immediately* before the first use of the XH-APPENDIX macro, to begin the first appendix, (which will be serially identified as appendix number *one*, although this may commonly be represented as an element of an alphabetic sequence). Thereafter, each subsequent invocation of XH-APPENDIX, if any, will begin a further (new) appendix, with serially incrementing identification numbers, at an incremental interval of one.

Explict use of the XH-APPENDIX-NUMBER-FORMAT macro is *optional*; however, *if* it is *not* called *before* the first use of the XH-APPENDIX macro, then it will be called *implicitly*, and the serialization of appendix numbers, together with page numbers within the appendices, will be repersented by arabic numerals, which may not be the desired effect. This is discussed further, in section C.4.2. , "Appendix Numbering Styles — the XH-APPENDIX-NUMBER-FORMAT Macro".

Following any specific invocation of the XH-APPENDIX macro, the effective scope of the call extends[81] *either* until the XH-APPENDIX macro is called a subsequent time, *or* to the end of the document. Within this extended scope of each XH-APPENDIX call, numbered subsection headings, each of which is specific to the containing appendix, may be introduced by calling the XH-APPENDIX-NH macro, which is discussed in section C.4.3. , "Appendix Subsection Numbering — the XH-APPENDIX-NH Macro".

A discussion of the implementation, and the usage, of the macro itself, may be found in section C.4.1. , "Specifying an Appendix Title — the XH-APPENDIX Macro".

81. This concept, of the extended scope of any XH-APPENDIX macro invocation, is significant *only* insofar as it affects the operation of the XH-APPENDIX-NH macro; in all other respects, the content of each appendix is formatted in accordance with the convertions which apply throughout the publication.

C.4.1. Specifying an Appendix Title — the `XH-APPENDIX` Macro

The XH-APPENDIX macro may be used to introduce each of any arbitrary number of individual, serially numbered, appendicies to the publication; with its implementation defined as:

```
.de XH-APPENDIX
.\" Usage: .XH-APPENDIX <title-text> ...
.\"
.\" Start a document appendix, numbering sequentially from Appendix 1,
.\" with numbering style, applied by 'af', as specified by a prior call
.\" to XH-APPENDIX-NUMBER-FORMAT, (defaults to arabic numerals).
.\"
.   af % 0
.   if !r XH-APPENDIX-NUMBER .XH-APPENDIX-NUMBER-FORMAT
.   ie r\\$0.PSINCR .nr PSINCR \\n[\\$0.PSINCR]
.   el .nr \\$0.PSINCR \\n[PSINCR]
.\"
.\" Printing of each appendix should start on a new recto page.
.\"
.   NEW-RECTO-PAGE %% \\g[appendix.page.number.format]
.\"
.\" Print a centred level 1 "Appendix #" page heading, with associated
.\" PDF outline, and table of contents entries.
.\"
.   SH 1
.   ce 999
.   ds \\$0.title "Appendix \\n+[XH-APPENDIX-NUMBER]\"
.   pdfhref O 1 \\*[\\$0.title]. \\$*
.   XH-UPDATE-TOC 1 \& \h'-(\w!0.!u+1.5n)'\\*[\\$0.title].\h'1.5n'\\$*
.   nop \\*[\\$0.title]
.\"
.\" Further heading output will be suppressed, if the XH-APPENDIX-SP
.\" register is not defined...
.\"
.   if r XH-APPENDIX-SP \{\
.   \" ...but when not suppressed, print the macro arguments as a
.   \" centred, possibly multi-line, level 2 heading, followed by
.   \" downward space equivalent to the value of XH-APPENDIX-SP.
.   \"
.      SH 2
.      ce 999
.      nop \\$*
.      sp \\n[XH-APPENDIX-SP]u
.   \}
.\" Reset section numbering, so that it restarts at one, for section
.\" headings at the top level, in each successive appendix.
.\"
.   rr \\$0-NH-h1
.\"
.\" Adjust type size scaling, for subsequent subheadings, clean up
.\" temporary storage, reset the pending centred line count, and we
.\" are done.
.\"
.   nr PSINCR \\n[PSINCR]*2/3
.   rm \\$0.title
.   ce
..
```

it begins by ensuring that the appendix numbering sequence has been properly initialized, with assigned formats for the appendix number itself, and also for any page numbers which are to be displayed within the appendices, (by calling the XH-APPENDIX-NUMBER-FORMAT macro, if necessary); it then synchronizes the incremental point size, which is to

be used for typesetting of the appendix title block, with that of top level section headings within the document body, and advances to the next available recto page, which will become the first page of the appendix.

After initialization, and advancing to the new page, an appendix identifier of the form "Appendix #" — in which the "#" represents the next sequentially available appendix serial number, (which, following assignment, becomes available in the XH-APPENDIX-NUMBER register), beginning with *one* for the first appendix, and incrementing by *one* for each successive XH-APPENDIX macro call — is generated, and then typeset as a centred page heading, in a type size which corresponds to that of a level one section heading, within the body of the containing publication, and with the appendix serial number expressed in the format specified by a preceding call — whether *explicit*, or *implicit* — of the XH-APPENDIX-NUMBER-FORMAT macro; this appendix identifier is also incorporated, together with with any specified <title-text> arguments, into a corresponding entry within the document's table of contents, in addition to the PDF document outline.

In addition to typesetting the appendix identifier, as a level one centred page heading, *if* the author has defined a register named XH-APPENDIX-SP,[82] the aggregate content of any specified <title-text> arguments is typeset, in a type size which corresponds to that of a level two heading within the document body, as a further centred page heading, followed by vertical space equivalent to the value, in groff's basic measurement units, which has been assigned to XH-APPENDIX-SP, offsetting the body of the appendix from the page heading.

Finally, in preparation for typesetting the body of the appendix, the subheading numbering sequence is reset to one, at heading level one, with the heading type size increment scaled to two thirds of that which applies within the document body, (and when typesetting the appendix page heading), before cleaning up all temporary storage which is used locally, within the scope of execution of the XH-APPENDIX macro.

C.4.2. Appendix Numbering Styles — the XH-APPENDIX-NUMBER-FORMAT Macro

With the intention that it should be called *before* the *first use* of the XH-APPENDIX macro, *if* expression of appendix serial numbers, and/or page numbers within the subsequent appendicies, is desired to exhibit any style other than arabic numerals, the implementation of the XH-APPENDIX-NUMBER-FORMAT is defined as:

```
.de XH-APPENDIX-NUMBER-FORMAT
.\" .XH-APPENDIX-NUMBER-FORMAT <appendix-num-fmt> <page-num-fmt>
.\"
.\" Establish the format for appendix numbers, and optionally, the
.\" style to be used for page numbers within appendices.
.\"
.   ie \\n(.$>1 \{\
.   \" Two arguments are required, for assignment of the appendix
.   \" number style (\\$1), and page number style (\\$2).  Moreover,
.   \" the appendix number register must exist, with initial value
.   \" of zero, and auto-increment of one.
.   \"
.      af appendix.page.number.format \\$2
.      if !r XH-APPENDIX-NUMBER .nr XH-APPENDIX-NUMBER 0 1
.      af XH-APPENDIX-NUMBER \\$1
.   \}
.\" If fewer than two arguments are specified, recurse to provide
.\" arabic numerals as defaults for those which are missing.
.\"
.   el .\\$0 \\$* 1 1
..
```

Within the current publication, this is invoked by the call:

```
.XH-APPENDIX-NUMBER-FORMAT A i
```

thus selecting a progression of upper-case alphabetic characters as the desired style for expression of appendix serial numbers, and lower-case roman numerals for numbering their associated content pages.

---

82. The XH-APPENDIX-SP register is *not* defined by default; the effect of leaving it unspecified is to omit typesetting of any default second level appendix page heading. This is designed to accommodate the typesetting of the GNU Free Documentation License, (see Appendix B, "Typesetting the GNU Free Documentation License"), during which an alternative strategy has been adopted for specification of a second level appendix page heading.

C.4.3. Appendix Subsection Numbering — the `XH-APPENDIX-NH` Macro

When typesetting a document, using the `ms` macros, *unnumbered* section headings, within appendices, may be specified using the `SH` macro, just as they would be throughout the document body. However, if *numbered* section headings, in which the appendix number is included as a section number prefix, are to be specified, then any use of the standard `NH` macro may be found to be *much* less convenient; thus, the `XH-APPENDIX-NH` macro is provided, in order to mitigate the potential attendant inconvenience.

The implementation of the `XH-APPENDIX-NH` macro is somewhat lengthy; thus, it will be presented, and discussed, as an incremental series of separate[83] fragments; the first of these is defined as:

```
.de XH-APPENDIX-NH
.\" Usage: .XH-APPENDIX-NH <level> [<dest-name> <heading-text> ...]
.\"
.\" Modelled on the behaviour of ms' "NH" macro, assign a section
.\" number, at any particular heading level, for use in appendices;
.\" the generated section number comprises the appendix number, in
.\" its specified format, followed by a sequence of arabic numeral
.\" section numbers, in the same style as an "NH" section number,
.\" appropriate to the specified heading level, then terminated
.\" by the character sequence, ".\&\ ".
.\"
.\" Unlike ms' "NH" macro, this DOES NOT support the "S <n> ..."
.\" style for specifying heading levels; it DOES, however, support
.\" the specification, as additional arguments, of a ".pdfhref L"
.\" destination name, followed by text which is to be assigned as
.\" the numbered heading text, and also propagated to a document
.\" table of contents, and to a PDF document outline.
.\"
.\" A persistent register, named \\$0-hl, is used to track the
.\" most recently assigned heading level, while temporary register,
.\" \\$0-hl-local, is used in internal manipulations based on this,
.\" in association with \\$1, which may leave it unchanged, or may
.\" increase it in intremental steps of one, or decrease it by any
.\" arbitrary decremental step size, (subject to a limitation that
.\" it cannot be reduced to less than a value of one).
.\"
.   af \\$0-hl 0
.   if \\$1-\\n[\\$0-hl]-1 \{\
.   \" The heading level, specified by \\$1, is greater than, but
.   \" not contiguous with the level currently assigned to \\$0-hl;
.   \" this creates an unwelcome gap in the level progression.
.   \"
.      tmc troff:\\n(.F:\\n(.c:\\$0: warning: \"
.      tm  \\$0 heading level \\$1 follows level \\n[\\$0-hl]
.   \}
..
```

Following its introductory header comment block, this initial fragment performs a sanity check on the value of the mandatory first, "*<level>*", argument; if this exceeds any currently assigned heading level, by an increment of more than one, then a warning diagnostic message is displayed, advising of a discontinuity in heading level progression.

83. Although presented here, in the style of an incrementally defined macro, the actual implementation of the `XH-APPENDIX-NH` macro is defined as a single integrated whole.

Following the initial sanity check, for continuity of incremental heading level progression, the next fragment of the XH-APPENDIX-NH macro implementation is defined as:

```
.am XH-APPENDIX-NH
.\" Usage: .XH-APPENDIX-NH <level> [<dest-name> <heading-text> ...]
.\"
.\" If \\$1 is less than \\$0-hl, then all sub-section numbers,
.\" for heading levels greater than \\$1, become defunct; discard
.\" each of these in turn.
.\"
.   nr \\$0-hl-local 1+\\n[\\$0-hl] 1
.   while \\n-[\\$0-hl-local]-\\$1 .rr \\$0-h\\n[\\$0-hl-local]
..
```

This has no effect, *if* the specified <*level*> argument has the *same value* as, *or* a *greater value* than, the currently active heading level; conversely, if the specified <*level*> argument has a *lesser value* than the currently active heading level, then the `while` loop will take effect, resulting in the deletion, and consequent reset, of all numeric registers which are associated with sub-section numbers, at greater depth than that specified by the <*level*> argument.

Note that, the effective *minimum* value which is permitted for the <*level*> argument is *one*; specification of any lesser value will *not* be diagnosed, but the *effective* value will be silently adjusted to this permitted minimum.

After validation of the <*level*> argument, and cleaning up of any redundant internal section number counters, the XH-APPENDIX-NH macro progresses to construction of a section number record relating to the ensuing numbered section heading; the implementation for this phase of execution is defined as:

```
.am XH-APPENDIX-NH
.\" Usage: .XH-APPENDIX-NH <level> [<dest-name> <heading-text> ...]
.\"
.\" Increment the sub-section number at the heading level which
.\" is specified by \\$1; (if this is a newly started level, then
.\" the starting point is initialized to a default value of zero,
.\" which is then incremented to one).
.\"
.   af \\$0-h\\$1 0
.   nr \\$0-h\\$1 +1
.\"
.\" Construct a string representation of the corresponding
.\" sub-section number, for use as a prefix to the eventual text
.\" of the sub-section heading, then simulate the effect of .NH,
.\" using an equivalent level .SH call.
.\"
.   nr \\$0-hl-local 0 1
.   ds XH-APPENDIX-SN-NO-DOT "\\n[XH-APPENDIX-NUMBER]\"
.   while \\$1-\\n[\\$0-hl-local] \{\
.      as XH-APPENDIX-SN-NO-DOT ".\\n[\\$0-h\\n+[\\$0-hl-local]]\"
.   \}
.   nr \\$0-HL 1+\\$1
.   SH \\n[\\$0-HL]
.\"
.\" Update \\$0-hl for use during the next, if any, invocation
.\" of this macro, append a "." to the constructed sub-section
.\" number, and clean up temporary internal storage.
.\"
.   nr \\$0-hl \\$1
.   ds XH-APPENDIX-SN-DOT "\\*[XH-APPENDIX-SN-NO-DOT].\"
.   rr \\$0-hl-local
..
```

which begins by incrementing the internal record of the particular constituent element of the sub-section number, which is associated with the current heading <*level*>, before assembling it, together with the appendix number, and each of the lower level section number elements, in sequence, to create a pair of publicly visible string representations of the composite section number — analogous to, but not bound to, the SN-DOT and SN-NO-DOT strings, which are defined

when the NH macro is called. In addition, it computes a suitable, publicly visible numeric register value — with no NH analogue — which is suitable for use as the heading level argument value, in the GNU extended variant of the SH macro call; it also invokes this call, in preparation for output of the ensuing section heading, and it also records the value of its own <*level*> argument, in a persistent internal-use register, for subsequent use in the sanity checking phase of the *next* invocation — if any — of the XH-APPENDIX-NH macro.

It may be observed that, on the basis of just the fragments which have been presented thus far, this implementation of the XH-APPENDIX-NH macro provides a fairly complete analogue for the NH macro; however, it is *not* compatible with the subsequent use of the XN macro,[84] which may be used following NH, to specify the text of the section heading, such that it will also be incorporated into the table of contents, and possibly a PDF document outline. In order to work around this incompatibility, rather than definining a complementary XH-APPENDIX-XN macro, it has been chosen to add a further fragment to the implementation of the XH-APPENDIX-NH macro, such that it will process additional optional <*dest-name*> and <*heading-text*> arguments, to achieve an effect which is analogous to the sequence:

```
.NH <level>
.XN -S -N <dest-name> <heading-text> ...
```

The implementation of this final, optional argument processing fragment of the XH-APPENDIX-NH macro, using XH to emulate an effect which is analogous to that of XN,[85] is defined as:

```
.am XH-APPENDIX-NH
.\" Usage: .XH-APPENDIX-NH <level> [<dest-name> <heading-text> ...]
.\"
.\" Check if the optional extra arguments have been specified...
.\"
.   if \\n(.$-2 \{\
.   \" ...and, when they have, assemble them, together with the
.   \" assigned section number, into a form which may be processed
.   \" through the "XH" macro...
.   \"
.      ne \\n[PD]u+3v
.      ds \\$0-ARGV "\\$2 \\*[\\$0-HL] \\*[XH-APPENDIX-SN-DOT]\"
.      shift 2
.      XH -S -N \\*[\\$0-ARGV]\&\ \& \\$@
.   \"
.   \" ...and ultimately, clean up the assembled argument list,
.   \" (which is no longer required).
.   \"
.      rm \\$0-ARGV
.   \}
..
```

Regardless of whether it has been called *with*, or *without* the optional <*dest-name*> and <*heading-level*> arguments, the XH-APPENDIX-NH macro will *always* leave the publicly visible XH-APPENDIX-NH-SN-DOT, and XH-APPENDIX-NH-SN-NO-DOT string representations of the resultant section number — analogous to the NH macro's SN-DOT and SN-NO-DOT strings, respectively — for any possible subsequent use at the discretion of the document's author; it also exposes the XH-APPENDIX-NH-HL numeric register — for which the NH macro offers no *publicly visible*[86] analogue — representing the effective value of the heading level argument, as passed internally, to the SH macro, in order to control the type size used for the ensuing heading text.

84. The XH macro *may* be used, in place of XN, following any invocation of XH-APPENDIX-NH, but the required argument values, which must be specified, may not bear an entirely obvious relationship to the XH-APPENDIX-NH call itself.

85. It should be noted that this analogue, as illustrated here, is based on the spdf.tmac implementations of the XN, and XH macros; it is dependent on use of the -S, and -N <*dest-name*> options, *neither* of which is supported by the default implementation of either macro, as defined in s.tmac.

86. The NH macro *does* have an internal-use analogue for the XH-APPENDIX-NH-HL register; however, this is *undocumented*, and thus, *cannot* be considered to be publicly visible.