Internal Document

Style Guide for Technical Publications

INTERIM DRAFT

The Open Group

Internal Document

Style Guide for Technical Publications INTERIM DRAFT

ISBN: N/A
Document Number:

Published in the U.K. by The Open Group, .

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire RG1 1AX

Or by email to:

OGEdit@opengroup.org

Contents

Part 1 Writing Style 1

Chapter 1 Document Structure 3
1.1 Front Matter 3
1.1.1 Title Page 3
1.1.2 Copyright Page 3
1.1.3 Contents 3
1.1.4 Preface 3
1.1.5 Trademarks 4
1.1.6 Acknowledgements 4
1.1.7 Referenced Documents 4
1.2 Body Text 4
1.2.1 Chapters 4
1.2.2 Reference Pages 5
1.2.3 Appendixes 5
1.2.4 Glossary 5
1.2.5 Index 6

Chapter 2 Writing Style 7
2.1 General Guidelines 7
2.2 Abbreviations, Acronyms, and Mnemonics 8
2.3 Acknowledgements 9
2.4 Alphabetical Order 9
2.5 Capitalization 9
2.6 Cautions 10
2.7 Contents 10
2.8 Copyright 10
2.9 Cross-references 10
2.10 Dates 11
2.11 Equations 11
2.12 Examples 11
2.13 Extensions 12
2.14 External References 12
2.15 Filenames, Pathnames, and URLs 12
2.16 Font Usage 13
2.17 Footnotes 13
2.18 Glossary 13

2.19 Grammar 13
2.20 Graphics 14
2.21 Hyphenation 14
2.22 Index 15
2.23 Keyboard Keys 15
2.24 Lists 16
2.25 Measurement 17
2.26 Monetary Values 17
2.27 Names 18
2.28 Notes 18
2.29 Numbers 18
2.30 Pagination 18
2.31 Punctuation 18
2.32 Special Characters 20
2.33 Spelling 21
2.34 System Items 22
2.35 Tables 22
2.36 Times 23
2.37 Titles 23
2.38 Trademarks 24
2.39 Warnings 25

Chapter 3 Reference Pages 27
3.1 Introduction 27
3.1.1 Divisions 28
3.1.2 Shadow Pages 28
3.2 Writing Style 29
3.2.1 Abbreviations, Acronyms, and Mnemonics 29
3.2.2 Cross-references 29
3.2.3 Examples 29
3.2.4 External References 30
3.2.5 Graphics 30
3.2.6 Index 30
3.2.7 Tables 30
3.2.8 Titles 30
3.3 Reference Page Sections 30
3.3.1 NAME 31
3.3.2 SYNOPSIS 32
3.3.3 DESCRIPTION 32
3.3.4 SUBCOMMANDS 33
3.3.5 OPTIONS 33
3.3.6 OPERANDS 34
3.3.7 PARAMETERS 34
3.3.8 EXTENDED DESCRIPTION 34
3.3.9 EXIT STATUS 34
3.3.10 RETURN VALUES 35
3.3.11 ERRORS 35
3.3.12 ASYNCHRONOUS EVENTS 35

3.3.13 EXAMPLES 36
3.3.14 ENVIRONMENT VARIABLES 36
3.3.15 FILES 37
3.3.16 NOTES 37
3.3.17 CAUTIONS 37
3.3.18 WARNINGS 37
3.3.19 DIAGNOSTICS 37
3.3.20 APPLICATION USAGE 37
3.3.21 FUTURE DIRECTIONS 37
3.3.22 SEE ALSO 37
3.3.23 CHANGE HISTORY 38

Chapter 4 Product Documentation 39
4.1 Introduction 39
4.2 Document Structure 40

Chapter 5 Common Product Documentation 43
5.1 Introduction 43
5.2 Language 43
5.3 Terminology 44
5.4 Equations 47
5.5 Reference Pages 47

Appendix A Terminology 51

Appendix B Extensions 63

Part 2 Tagging Document Source 65

Chapter 6 Using SGML 67
6.1 Introduction 67
6.2 Building Documents 68
6.3 SGML Coding 68
6.3.1 Front Matter 68
6.3.2 Cautions 68
6.3.3 Changebars 68
6.3.4 Comments 69
6.3.5 Cross-references 69
6.3.6 Equations 69
6.3.7 Examples 69
6.3.8 Extensions 69
6.3.9 Footnotes 69
6.3.10 Glossary 70
6.3.11 Graphics 70
6.3.12 Headings 70
6.3.13 Index 70
6.3.14 Lists 71
6.3.15 Notes 72

6.3.16 Part Pages 72
6.3.17 External References 72
6.3.18 Special Characters 72
6.3.19 System Items 72
6.3.20 Tables 74
6.3.21 Text 75
6.3.22 Warnings 76
6.4 Reference Pages 76
6.4.1 Filenaming 76
6.4.2 Structure 77
6.4.3 Identifiers 78
6.4.4 Metainformation 80
6.4.5 Shadow Pages 81
6.4.6 Cross-references 81
6.4.7 Tables 82
6.4.8 Reference Page Sections 82

Chapter 7 Using Troff 95
7.1 Introduction 95
7.2 Directory Structure 95
7.3 Building Documents 97
7.4 Troff Coding 99
7.4.1 Front Matter 99
7.4.2 Cautions 101
7.4.3 Changebars 101
7.4.4 Comments 101
7.4.5 Cross-references 101
7.4.6 Displays 102
7.4.7 Examples 103
7.4.8 Extensions 103
7.4.9 External References 104
7.4.10 Fonts 104
7.4.11 Footnotes 105
7.4.12 Glossary 106
7.4.13 Graphics 106
7.4.14 Headings 107
7.4.15 Index 109
7.4.16 Lists 109
7.4.17 Notes 110
7.4.18 Pagination 110
7.4.19 Part Pages 111
7.4.20 Point Size 111
7.4.21 Reference Pages 112
7.4.22 Special Characters 113
7.4.23 Strings 113
7.4.24 System Items 114
7.4.25 Tables 114
7.4.26 Tabs 115

7.4.27 Text 115
7.4.28 Warnings 115
7.5 Troff Coding for Popular Titles 116
7.6 Checking Source Files 116

Appendix C SUD-specific SGML 119
C.1 Introduction 119
C.2 Doctype Declaration 120
C.3 Filenaming 121
C.4 Metainformation 121
C.5 Conformance 122
C.6 Cross-references 122
C.7 Equations 123
C.8 Examples 123
C.9 Notes 123
C.10 Reference Page Sections 123
C.11 Reference Page Templates 128
C.11.1 Descriptive Reference Pages 128
C.11.2 Utility Reference Pages 130
C.11.3 Program Interface Reference Pages 134
C.11.4 Header Reference Pages 137
C.11.5 Sample Program Reference Pages 141

Appendix D SGML Examples 143
D.1 Metainformation 143
D.2 NAME 144
D.3 SYNOPSIS 144
D.3.1 Utilities 144
D.3.2 Functions and Macros 146
D.3.3 Headers 148
D.3.4 External Variables 148
D.3.5 Sample Programs 148
D.4 DESCRIPTION 148
D.5 OPTIONS 149
D.6 OPERANDS 150
D.7 PARAMETERS 151
D.8 EXTENDED DESCRIPTION 151
D.9 EXIT STATUS 152
D.10 RETURN VALUES 152
D.11 ERRORS 153
D.12 EXAMPLES 153
D.13 ENVIRONMENT VARIABLES 154
D.14 FILES 155
D.15 SEE ALSO 156

Part 3 Documentation Tools 157

Chapter 8 Documentation Tools 159
8.1 Source Control 159
8.1.1 SCCS 159
8.1.2 ODE 159
8.1.3 Clearcase 159
8.2 Bug-tracking 159
8.2.1 OT 159
8.2.2 DTTS 159
8.2.3 Corrigenda 160
8.3 Difference Marking Between Versions 161
8.4 Conversion Programs 161
8.4.1 troff-to-HTML 161
8.4.2 SGML-to-troff 161

Part 4 Presentation 163

Chapter 9 Output Formats 165

List of Tables

6-1 Structure of Identifiers 79

Preface

The Open Group

The Open Group is the leading vendor-neutral, international consortium for buyers and suppliers of technology. Its mission is to cause the development of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy-to-use as the telephone. The essential functionality embedded in this infrastructure is what we term the *IT DialTone*. The Open Group creates an environment where all elements involved in technology development can cooperate to deliver less costly and more flexible IT solutions.

Formed in 1996 by the merger of the X/Open Company Ltd. (founded in 1984) and the Open Software Foundation (founded in 1988), The Open Group is supported by most of the world's largest user organizations, information systems vendors, and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to meet its new mission, as well as to assist user organizations, vendors, and suppliers in the development and implementation of products supporting the adoption and proliferation of systems which conform to standard specifications.

With more than 200 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- Consolidating, prioritizing, and communicating customer requirements to vendors
- Conducting research and development with industry, academia, and government agencies to deliver innovation and economy through projects associated with its Research Institute
- Managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- Adopting, integrating, and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- Licensing and promoting the Open Brand, represented by the ''X'' Device, that designates vendor products which conform to Open Group Product Standards
- Promoting the benefits of the IT DialTone to customers, vendors, and the public

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development, and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trademark on behalf of the industry.

Development of Product Standards

This process includes the identification of requirements for open systems and, now, the IT DialTone, development of Technical Standards (formerly CAE and Preliminary Specifications) through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product.

The ‘‘X’’ Device is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the Open Brand Trade Mark License Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical Standards and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys, and business titles.

There are several types of specification:

- *Technical Standards* (formerly *CAE Specifications*)

  The Open Group Technical Standards form the basis for our Product Standards. These Standards are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement a Technical Standard can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand. Technical Standards are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

- *CAE Specifications*

  CAE Specifications and Developers’ Specifications published prior to January 1998 have the same status as Technical Standards (see above).

- *Preliminary Specifications*

  Preliminary Specifications have usually addressed an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is as stable as can be achieved, through applying The Open Group’s rigorous development and review procedures.

  Preliminary Specifications are analogous to the *trial-use* standards issued by formal standards organizations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a Technical Standard. While the intent is to progress Preliminary Specifications to corresponding Technical Standards, the ability to do so depends on consensus among Open Group members.

- *Consortium and Technology Specifications*

  The Open Group publishes specifications on behalf of industry consortia. For example, it publishes the NMF SPIRIT procurement specifications on behalf of the Network Management Forum. It also publishes Technology Specifications relating to OSF/1, DCE, OSF/Motif, and CDE.

  Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as Technical Standards, in which case the relevant Technology Specification is superseded by a Technical Standard.

In addition, The Open Group publishes:

- *Product Documentation*

  This includes product documentation—programmer's guides, user manuals, and so on—relating to the Pre-structured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

- *Guides*

  These provide information that is useful in the evaluation, procurement, development, or management of open systems, particularly those that relate to the Technical Standards or Preliminary Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

- *Technical Studies*

  Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

Versions and Issues of Specifications

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at **http://www.opengroup.org/corrigenda**.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at **http://www.opengroup.org/pubs**.

This Document

This document defines The Open Group house style for technical publications.

It should be followed to ensure a unified approach to documentation style, organization, terminology, and appearance.

This document is intended for anyone who drafts or writes technical documents for publication by The Open Group.

Submissions to The Open Group that conform to this house style can be processed more quickly than those that require conversion.

Readers are expected to be familiar with text editors, word processors, and the American-English language. Readers should also understand the principles of text processing using formatting commands. Detailed knowledge of text processors used by The Open Group is not required.

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

This section will be expanded by PLH to include scope and purpose, a positioning statement, and intended audience.

Trademarks

ArborText™ and ADEPT*Editor™ are trademarks of ArborText, Inc.

BSD™ is a trademark of the University of California, Berkeley, U.S.A.

Digital™ and DEC™ are trademarks of Digital Equipment Corporation.

Hewlett-Packard™ is a trademark and HP® is a registered trademark of Hewlett-Packard Company.

IBM® is a registered trademark of International Business Machines Corporation.

Microsoft® is a registered trademark of Microsoft Corporation.

Motif®, OSF/1®, and UNIX® are registered trademarks and the IT DialTone™, The Open Group™, and the ‘‘X Device’’™ are trademarks of The Open Group.

NFS® is a registered trademark and Network File System™ is a trademark of Sun Microsystems, Inc.

POSIX® is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

Postscript® is a registered trademark of Adobe Systems Incorporated.

SecureWare™ is a trademark of SecureWare, Inc.

Sun® and Sun Microsystems® are registered trademarks of Sun Microsystems, Inc.

TeX™ is a trademark of the American Mathematical Society.

X™ and X Window System™ are trademarks of the Massachusetts Institute of Technology.

XENIX® is a registered trademark of Microsoft Corporation.

Referenced Documents

Part 1: Writing Style

The following documents are referenced in Part 1:

Read Me First! A Style Guide for the Computer Industry, SunSoft Press, A Prentice Hall Title, 1996, ISBN 0-13-455347-0.

The Chicago Manual of Style, University of Chicago Press.

ISO 8859-1: 1987, Information Processing — 8-bit Single-byte Coded Graphic Character Sets — Part 1: Latin Alphabet No. 1.

Webster's Collegiate Dictionary, Merriam-Webster.

Part 2: Tagging Document Source

The following documents are referenced in Part 2:

README.1ST, SGML For Writers and Editors, Ronald C. Turner, Timothy A. Douglass, and Audrey J. Turner, Prentice Hall, 1996, ISBN 0-13-432717-9.

Developing SGML DTDs From Text to Model to Markup, Eve Maler and Jeanne El Andaloussi, Prentice Hall, 1996, ISBN 0-13-309881-8.

For more information on the DocBook DTD, which is maintained and available through the Davenport Group, see their web site at URL http://www.ora.com/davenport/docbook/.

Style Guide for Technical Publications

Part 1:
Writing Style

The Open Group

Chapter 1

Document Structure

This chapter describes the overall structure of technical documents for publication by the Open Group.

The structure of technical documents published by The Open Group is based upon the guidelines contained in *Read Me First! A Style Guide for the Computer Industry*.

The presentation of information about system items is based on the layout of UNIX reference pages.

1.1 Front Matter

1.1.1 Title Page

The type of document and its title, and the text ‘‘The Open Group.’’

1.1.2 Copyright Page

Copyright information (see Section 2.8 on page 10), together with the type of document, its title, ISBN and document numbers, and contact details for problem reporting.

1.1.3 Contents

Lists parts, chapters, sections (second-level headings), subsections (third-level headings), reference pages, appendixes, glossary, and index.

1.1.4 Preface

1. Introduction

   About The Open Group and its document types.

   The same text is used in all Open Group technical documentation and is supplied by The Open Group.

2. This Document

   A brief introduction to the document and its purpose.

3. Intended Audience

   Readership and prerequisite knowledge.

4. Structure

   Brief one-line descriptions of the content of each element.

5. Applicability (Optional)

Version of software documented.

6. Typographical Conventions

   Use of fonts and special symbols.

7. History (Optional)

   How this document relates to others.

8. Problem Reporting (Optional)

   Describes how to report problems with the software documented. Provides a contact and method for problem reports or suggestions for improvement.

1.1.5 Trademarks

Full list of acknowledgements for all trademarks used in the document.

1.1.6 Acknowledgements

List of contributors.

1.1.7 Referenced Documents

List of all documents referenced as a short code and full bibliographic details. The details should include the title, version number (if applicable), author, publisher, ISBN number, document number (if applicable), and date of publication.

1.2 Body Text

Body text is made up of the following elements: chapters, reference pages, appendixes, a glossary, and an index. Each element is a separate file, although these may in turn be made up from other files.

Each element can include paragraphs, tables, figures, lists, notes, footnotes, and cross-references.

1.2.1 Chapters

Each chapter is an element of a document.

Each chapter covers one main topic. The definition of a topic varies according to the subject matter of the document. Each chapter may consist of numbered sections (second-level headings) and subsections (third-level headings). Fourth-level headings are discouraged. Fifth-level and lower headings must not be used.

The first chapter of a document should include the following:

- Objective or Purpose

  This is a brief statement.

- Overview

  This is an overview of the subject.

- Conformance

This identifies expected criteria for conformance to existing standards or the Open Brand. It is required in CAE specifications, though it is not discouraged as an indication of intent in Preliminary Specifications. In all cases, it is for guidance only; definitive conformance requirements are given in branding documentation.

- Future Directions

  This describes how the publication and related publications are expected to develop.

If there are sections or subsections, there must be more than one. A section should have subsections if the subject matter is subordinate to the main theme of the section.

If the restrictions on sections and subsections cannot easily be implemented, consider modifying the structure of the document to obtain a suitable structure within each chapter.

Unnumbered headings can be used to aid readability; they do not appear in the table of contents.

1.2.2 Reference Pages

Reference pages have a special layout (see Chapter 3 on page 27). Each page may be contained in its own file, in a chapter, or in an appendix.

Any reference page section must be preceded by a section heading and introductory text. The pages should be sorted into alphabetical order.

1.2.3 Appendixes

Each appendix is an element of a document.

An appendix contains material that is required for reference, but that would interrupt the flow of information in a chapter; for example, long tables of data. Appendixes may contain material that is normative or non-normative; in cases where it is normative, this will be clearly stated.

An appendix may contain sections and subsections, like a chapter. The guidelines about the numbers of sections and subsections are the same as for chapters.

An appendix can also contain reference pages.

Information in appendixes can include, but is not limited to, the following:

- Descriptions of data formats and file structures
- Input and output codes; for example, character conversion codes
- Global processing limitations
- Sample files, reports, or programs

1.2.4 Glossary

Brief definitions of terms used in the document. This is optional (though preferred), but is mandatory in CAE Specifications.

1.2.5 Index

This is mandatory in all documents.

Chapter 2

Writing Style

The Open Group preferred writing style is based upon the guidelines contained in *Read Me First! A Style Guide for the Computer Industry*.

The information presented here summarizes the key features of this writing style, and documents decisions made for areas where the above guide does not recommend specific editorial policy.

This chapter starts with a list of general writing style guidelines, and thereafter consists of an alphabetically ordered list of style components.

2.1 General Guidelines

These general guidelines are designed to take account of internationalization and translation considerations:

- Sensitivity to your readers' needs is important. It is important to start with a good understanding of who your readers are likely to be, and their technical expertise. This should be clearly identified in the Intended Audience section of the Preface.
- Anticipate the readers' questions.
- Provide information clearly and concisely, so that readers can find information quickly.
- Keep sentences short—preferably less than twenty five words (approximately one and one half lines)— and clear (but complete).
- Use consistent language, terminology, and typographical conventions. Try to avoid the use of synonyms.
- Make sure statements are unambiguous—do not use contractions.
- When choices exist, explain the advantages and disadvantages of the alternatives.
- Avoid humor, jargon, irony, idioms, adages, slang, sexist language, and political and religious references.

  If necessary, when using computer terms that can be interpreted as jargon, make sure that you and the reader understand the meaning of the word as used in your document. Ideally such terms should be included in the Glossary.
- Define key terms at the first mention, if the terms may be new to the reader. (And add to the glossary.)
- Avoid general modifiers, such as ‘‘nice.’’
- Avoid long strings of modifiers. For example, ‘‘The previously sent destination protocol address ...’’ would be better worded as ‘‘The destination protocol address that was previously sent ...’’

- Do not use the same word in different grammatical categories.
- Avoid using symbols to represent words, such as an ampersand (&) to represent the word ''and.''
- Avoid referring to the authors. If such a reference is essential, use ''The Open Group ...''
- Use *x* to refer to a generic letter, and *n* to refer to a generic number.
- Do not use double spaces between sentences.
- Do not refer to holidays.
- Do not use analogies and terms based on local culture.
- Do not use non-English abbreviations and terms (see Appendix A on page 51).
- Do not use terms that attribute human characteristics to software, including gender.
- Do not use words that do not appear in the dictionary (see Section 2.33 on page 21).

2.2 Abbreviations, Acronyms, and Mnemonics

Where possible, keep to commonly used abbreviations, acronyms, and mnemonics and avoid inventing new ones. Once introduced, use them consistently.

An abbreviation is a shortened form of a term. An acronym is a word formed from the initial letters or parts of compound terms. A mnemonic is a memory aid.

For clarity, define abbreviations, acronyms, and mnemonics the first time you use them and add them to the glossary. In such cases, show the spelled-out version followed by the abbreviation, acronym, or mnemonic in parentheses. For example:

the X/Open Transport Interface (XTI)

Use uppercase letters for acronyms and mnemonics. Use uppercase initials for spelled-out versions where this is helpful. Acronyms can include lowercase letters if they are trademarks.

The derivation of an abbreviation, acronym, or mnemonic may be of little importance to the reader if it is commonly used; for example, BASIC or FORTRAN. In this case, omit the spelled-out version.

If the pronunciation of an acronym may not be obvious to the user, provide the pronunciation.

The preceding indefinite article (*a* or *an*) depends on the way the abbreviation is commonly pronounced; *an* is used where the abbreviation begins with a vowel sound, otherwise use *a*. For example, ''a mOSI'' (pronounced ''mosee''), ''a LAN,'' ''an NDR.''

Do not use an apostrophe to form the plural of an abbreviation, acronym, or mnemonic. Add a lowercase ''s;'' for example, OEMs.

Avoid beginning sentences with abbreviations, acronyms, or mnemonics, unless they have been fully explained in preceding text.

Do not use periods or intervening spaces with abbreviations, acronyms, and mnemonics unless the abbreviation can be confused with an actual word; for example, ''no.'' for number, or ''in.'' for inch.

Abbreviations may be used in tables, examples, figures, and footnotes.

Use periods in the acronyms of country names; for example, U.K. and U.S.A.

2.3 Acknowledgements

Create a list of contributors, and indicate the nature of their contribution, such as author of source material, provider of source material, drafting, reviewing, and so on.

If applicable, you may include a list of the members of the Working Group involved in the development of the document.

2.4 Alphabetical Order

In general, any list of items which does not have a sequence or priority should be arranged in alphabetical order.

Use the order produced by the *sort* command, except that all special characters should be grouped together.

The following list (read from left to right, from the top down) shows the correct alphabetical sequence:

```
!  ‘‘  #  $  %  &  ’  (  )  *  +  ’  -  .  /  :
;  <  =  >  ?  @  [  \  ]  ^  _  ‘  {  |  }  ~
0  1  2  3  4  5  6  7  8  9
A  a  B  b  C  c
 .
 .
 .
X  x  Y  y  Z  z
```

Alphabetize acronyms according to their shortened form.

Use locale-specific collating order for non-English languages.

2.5 Capitalization

Use initial capitals for nouns that refer to a specific person, title, or object.

Do not use initial capitals for nouns that refer to generic or non-specific people, titles, or objects; for example, system manager, computer, program.

Use initial capitals for product names.

Use initial capitals for the names of objects capitalized on the screen; for example, ‘‘The Clear menu item ...’’.

Use initial capitals for the following words when followed by a letter or number: Chapter, Appendix, Section, Version, Release, Guideline, Table, and Figure.

Do not use initial capitals for the words step, line, or column when they are followed by a number.

See also Section 2.37 on page 23.

2.6 Cautions

Use a caution to draw the reader's attention to information which can have a significant or serious impact on the subject being described. See also Section 2.28 on page 18 and Section 2.39 on page 25.

2.7 Contents

The table of contents is produced automatically when a document is built.

2.8 Copyright

All Open Group documents are copyrighted, and this information must be included in all documentation.

Copyright notices must include the word *Copyright* and the copyright symbol (©), the copyright date (month, year), and the copyright owner. For example, ''Copyright © July 1997, The Open Group.''

Follow this with the words ''All rights reserved.'' on a separate line.

Include The Open Group standard statement concerning restricted rights.

A statement of the country of origin should be added, such as ''Published in the U.K, by The Open Group, July 1997.''

All copyright information will be verified by The Open Group.

2.9 Cross-references

Cross-references can be made to complete elements (such as chapters and appendices), sections, figures, tables, and examples.

Refer to all figures, tables, and examples *before* they appear.

If a table or figure has no identifier, you can refer to it is as ''the following ...'', provided this text is immediately before the referenced object.

Do not hard-code page numbers.

All internal cross-references should be as precise as possible, to ensure rapid location of the required material. Note that a reference to Section 7.1 is an instruction to the reader to read all of Section 7.1, not merely the introductory paragraph.

Brief cross-references should be placed in parentheses within sentences. Longer cross-references should be placed in a separate sentence.

2.10 Dates

Dates should be written out in full. The month should be spelled out and the year should not be abbreviated, even when used in a range. For example, ‘‘January 5, 1998.’’

If dates are used in examples, include a comment that explains the purpose, so that it can be translated correctly.

2.11 Equations

Equations should be used with care—they may not display adequately in all output formats.

2.12 Examples

Consider adding examples to illustrate the following:

- Clarification of a point in the text
- Typical usage of a system item (if not obvious)
- Illustration of the differences between related system items
- Complex usage of a system item

Do not use artificial names such as ‘‘foo’’ and ‘‘bar’’ in examples. Try to use meaningful names for file names and other arguments used in examples.

Vary the use of names for people — Anglo-Saxon and non-Anglo-Saxon, male and female. A telephone directory is a useful source of ideas.

In location examples, use cities that are recognizable without the state or country.

Never include names of actual system accounts and passwords. Remember to edit system and user information out of screen-captures.

Program examples should include extensive comments as part of the program text.

Avoid the use of editorial ‘‘we’’, ‘‘our’’, and ‘‘let’s’’ in text surrounding examples.

Avoid examples that require an alphabetically ordered list of abbreviations, acronyms, or mnemonics to convey meaning. This can cause translation difficulties because the translated list will probably not be in alphabetical order. If you cannot avoid using an alphabetically ordered list, include a comment in the source file indicating the purpose of the example so that translators can design an example that is appropriate.

Example titles should use the conventions described in Section 2.37 on page 23.

If the example shows an action performed by a system item, use the gerund form of the action in the example title (for example, ‘‘Sorting a File’’). Do not use gerunds in titles of descriptive examples (such as showing a sample file or presenting a table of useful expressions).

2.13 Extensions

Extensions identify features of particular interest, such as extensions to existing standards or known problems.

The definition of each type of extension is given in Section 2.13.

2.14 External References

The preferred method for referring to any document is to use standard text, such as a string, citation, or text entity.

You can use an abbreviated document title within the text of a document (for example, ‘‘see the ANSI COBOL standard’’), but you should include the full title and bibliographic details in the Referenced Documents section.

When referring to a book produced by The Open Group, use the book’s abbreviated title (for example, ‘‘see the XA Specification’’).

Do not refer to specific elements (such as chapters or sections) of another document.

All bibliographic details should be verified using the following URLs:

Open Group documents	http://www.opengroup.org/public/pubs/catalog
Standards Documents	http://www.opengroup.org/public/togaf/section5.htm
	http://www.iso.ch

The correct information should become part of the document source so that it can be reproduced at any point in the future.

Brief cross-references should be placed in parentheses within sentences. Longer cross-references should be placed in a separate sentence.

2.15 Filenames, Pathnames, and URLs

Use initial periods when referring to a file type. For example, ‘‘a .c program,’’ ‘‘the .LIS file.’’

If punctuation characters are part of a filename, pathname, or URL, set it off from the text when its appearance in a sentence might be confusing.

2.16 Font Usage

Refer to Parts 2 and 3 for font usage.

2.17 Footnotes

Use footnotes with care to avoid introducing distracting information which can hinder rather than help the reader. If an explanation is required that would interrupt the flow of information in the text, use a footnote.

2.18 Glossary

A glossary should include specialized words, abbreviations, acronyms, and mnemonics used in the body of the document.

Each glossary entry should consist of the term itself, followed by a brief definition. A brief expanded definition of the term may also be added.

Synonyms of words used in the text should be included with a cross-reference to the main glossary entry.

A central Open Group look-up will be made available in due course.

2.19 Grammar

Where no particular rule is specified and if there is a difference between usage in Britain and the U.S., the U.S. version is used.

Use simple syntax, present tense, and active voice whenever possible.

Use imperative verbs to tell the user what to do.

An infinitive (to *verb*) must be treated as one word; do not split the infinitive by inserting another word between the *to* and *verb*.

A sentence must not end with a preposition.

Avoid using irregular perfect participles such as *learnt* and *spelt*, using instead *learned* and *spelled*.

Do not leave out articles, such as ‘‘a’’, ‘‘an,’’ and ‘‘the’’. This style of writing can lead to misinterpretation and incorrect translation. (This rule may be waived where space is limited, such as in a table, figure, or example.)

Make sure the noun to which a pronoun refers is clear. If necessary, repeat the noun.

The following pronouns are always singular: another, either, each, neither, every, one, any, some, anybody, everybody, everything, someone, nobody, nothing, no-one. The following pronouns are always plural: both, others, few, several, many.

Avoid using system items as verbs; for example, use ‘‘use *grep* to find the string’’, rather than ‘‘grep for the string ...’’.

2.20 Graphics

Use an initial capital for the first letter of every word of text used in the graphic, with the exception of flowcharts and data structures (where just the first word is capitalized), and case-sensitive system item names.

Figure titles should use the conventions described in Section 2.37 on page 23.

2.21 Hyphenation

Use the hyphen for the following prefixes:

all- cross- half- multi- non- post- quasi- self-

Do not use the hyphen for the following prefixes:

anti bi co dis extra infra inter intra macro meta micro
mid mis multi non over pre pseudo re sub super ultra un

However, when any prefix is followed by a word beginning with the same letter, add a hyphen.

Prefixes should always include a hyphen if the root word is all uppercase, has an initial capital, a number expressed as a figure, or if the root element is a hyphenated compound.

Use a hyphen if adding a prefix results in unclear meaning. For example, re-collect/recollect, re-cover/recover, re-solve/resolve, and so on.

The words in a noun phrase, when the phrase is functioning as an adjective or modifier, are joined by a hyphen (even though they might not be otherwise). For example, a process initiated by the user is a user-initiated process.

The following suffixes can be added to nouns:

-based -defined -dependent -independent -oriented -specific

The compound adjectives so formed must be used with care. Do not adopt this practice where the sentence is simpler and clearer with a different construction; for example, ‘‘the transaction-manager-dependent issue’’ is clearer as ‘‘the issue dependent on the transaction manager.’’

Adverbial phrases are not hyphenated; for example, ‘‘externally developed program.’’

Hyphenate a modifying phrase when it precedes the noun it modifies. For example, ‘‘state-of-the-art design.’’

2.22 Index

An index is produced automatically when the document is built, provided index terms have been added.

The index should consist of important words and symbols from the text, together with concepts, synonyms, or paraphrases that are related to the main index entries.

Be consistent in spelling index entries and avoid using plurals and initial capitals wherever possible, so that all entries on one topic can be collected together correctly.

Do not use automatic indexing tools. The results are typically too long and do not adequately pinpoint information for the reader.

2.23 Keyboard Keys

When referring to the name of a control key on a keyboard (such as *Control* or *Enter*), use an initial capital letter.

When referring to the name of a key which is not on a keyboard, use a lowercase initial; for example, the space bar, the comma key.

Use *x* to refer to a generic letter key, and *n* to refer to a generic number key.

Refer to a control sequence in text as follows: Ctrl-*x*. That is, use an initial capital for the word ‘‘Ctrl’’, followed by a hyphen, followed by the lowercase letter. Use ^*x* to refer to a control sequence when describing system output.

Place angle brackets around key names that are labeled on the keyboard; for example, <Return>.

Use <Return> to refer to the key used to enter commands.

Use the verb *press* when referring to keys.

Refer to *keys* on the keyboard, not *buttons*.

Do not use the name of a key as an adjective. For example, use ‘‘Press <Return>’’ and not ‘‘Press the Return key.’’

F: Functions may be bound to different keys due to localization. Follow these guidelines when documenting keyboards:

- Select a default keyboard, and develop a method for providing information about alternative keyboards.
- Put as much keyboard information online as possible.
- Use *function names* rather than *key cap names* in documenting software applications.

2.24 Lists

Lists are a good way to break up long sentences or paragraphs and to clarify choices and steps, but it is important lists are introduced with a clear (but brief) lead-in phrase. Too many lists without clear lead-in phrases interrupt the flow of the text, and can be distracting to the reader.

Use a colon after a lead-in phrase that is a complete statement; for example, ‘‘The values of *local* are defined as follows:’’.

The type of list you use depends on the type of items contained in the list:

- Use an unnumbered (or unordered) list to show choices or groups of items for which there is no sequence.
- Use a numbered (or ordered) list for items that occur in a specific sequence (such as a procedure or a list of results for some action), for a list that shows the precedence of items, or a hierarchy.
- Use a definition (or variable) list for items that have a term and a corresponding definition or description.

Use the following rules when constructing lists. These rules apply to all types of lists, unless otherwise indicated.

- End each list item with a period if the list item contains a complete sentence. Do not use end punctuation for list items that do not contain a complete sentence.
- Avoid mixing complete sentences and sentence fragments as items in the same list. If you are unable to avoid mixing sentences and sentence fragments, use a period after each list item.
- Start each list item with the same part of speech if possible.
- Capitalize the first word of every list item, whether the item is a complete sentence or not, except for literal names that are lowercase. If possible, reword the list item to avoid this.
- Avoid putting items of obviously dissimilar values in the same list.
- Use parallel construction for all items within a list. For example, lists of options or parameters in reference pages always use the name of each option or parameter as the term of an entry in a definition list. The definition of the term always begins with a verb in the present tense, so that the term and definition, when read together, form a complete sentence. For example:

  –a Lists all information.

  –b Lists only basic information.

- In a procedure, use only complete sentences for each item. Begin each step with a verb in the imperative form, and tell the reader what will happen as a result of each step. Avoid using cross-references in an ordered list. It is better to give all the needed information within a step, rather than referring the reader elsewhere for information needed to complete a task.

Nested lists can be used as follows:

- Unnumbered lists within variable or numbered lists
- Numbered lists within variable lists
- Numbered lists within numbered lists
- Unnumbered lists within unnumbered lists

If a list item runs to several paragraphs, or includes complex material, you should consider using unnumbered headings instead of a list.

For lists in text use commas to separate items in a series of three or more words, phrases, or clauses, including the last item before a conjunction. For example, ‘‘a, b, and c.’’

2.25 Measurement

Provide measurements as imperial unit symbols, followed by the metric equivalent in parentheses.

Make sure that the precision of a converted measurement reflects the precision of the original measurement.

Place the abbreviation for a unit of measurement at the end of a series of two or more items; for example, ‘‘1200, 1400, or 1600MHz.’’

Repeat the abbreviation for a unit of measurement in a series of two items; for example, ‘‘10 ft to 12 ft.’’

Plural and singular abbreviations of units of measurement are the same; for example, ‘‘1 lb’’ and ‘‘10 lb.’’

Insert a space between a number and the unit of measurement it modifies, except for KB, MB, kHz, MHz, GHz and K. For example, ‘‘6 ft’’, and ‘‘6MB’’.

The context should enable differentiation between the two meanings of *K*: binary thousand and Kelvin.

Avoid using the number sign (#), single quote (’), and double quotes (") symbols to indicate the pound, foot, and inch units of measurement.

Comment source files to indicate which units of measurement are used as an aid to translators.

2.26 Monetary Values

Avoid monetary values since they are country-specific.

Monetary values may be included in examples, but there must be a comment in the source file indicating their purpose as an aid to translators.

2.27 Names

Use initial capitals for names.

The name of an organization is treated as a singular noun. For example, ‘‘The Open Group publishes documents.’’

2.28 Notes

Notes should be used to call attention to important information that the reader should not overlook. They should be used sparingly. See also Section 2.6 on page 10 and Section 2.39 on page 25.

2.29 Numbers

As a general rule, use numerals rather than spelled-out numbers, and be consistent in their use.

Avoid beginning sentences or headings with numerals.

Use *to* (and not *through*) for inclusive ranges. In tables and graphics, use a dash.

Hyphenate numbers or numerals in compound modifiers (500,000-byte file), but not in single modifiers (500,000 bytes).

Use a comma in numerals of more than four digits (for example, 10,000). Do not use commas in binary, octal, or hexadecimal numbers.

Do not use an apostrophe to form the plural of a number. Instead, add a lowercase s; for example, 4s, 1920s.

Insert a space between a number and the unit symbol it modifies, except for degrees, minutes, and seconds of an angle.

2.30 Pagination

Refer to Parts 2 and 3 for pagination.

2.31 Punctuation

( ) Parentheses enclose qualifying or explanatory material that is included in a sentence or paragraph.

If the text inside parentheses is a complete sentence, put the terminating period inside the parentheses and terminate the preceding sentence. (This is an example.)

If the text inside parentheses is not a complete sentence, embed it in a sentence, so that the normal period or comma appears on the outside of the parentheses (like this).

If required by the context, place a comma after the closing bracket.

Try to avoid nested parentheses in text; nested tangential thoughts can be difficult to follow. Use several sentences inside the parentheses if necessary.

[ ] Brackets (do not use ‘‘square’’) are not used in text. They are used literally in syntax to indicate bracket characters. Each document should describe how it uses brackets in a

section on ‘‘Typographical Conventions.’’

{ } Braces (do not use ‘‘curly’’) are not used in text. They are used literally in syntax to indicate brace characters. Each document should describe how it uses braces in a section on ‘‘Typographical Conventions.’’

< > Angle brackets are not used in text. They are used literally in some cases where, for example, C programs use them to indicate a filename. In arithmetic, text may refer to these characters as *less than* and *greater than*, respectively.

They should be placed around key names that are labeled on the keyboard; for example, <Return>.

’ An apostrophe indicates possession or a missing character. It is never used to form a plural.

Its is the possessive pronoun of it. *It is* should be used in preference to its contracted form *it’s*.

Use an apostrophe and a lowercase ‘‘s’’ to form the plural of a lowercase letter used as a noun; for example, a’s, y’s.

This character is also a single quote. Use 66-99 quotes in preference to single quotes. However, the single quote may be used literally in syntax.

: A colon directs the reader’s attention to whatever follows it.

Use a colon when you use *for example*, *the following*, *follows*, or *as follows* to lead in to an object beginning on the line below.

Use a colon at the end of lead-in phrases for lists, tables, figures, and so on.

; Use a semicolon to join closely related independent sentences or clauses.

When the elements in a series are long and complex, or involve internal punctuation, they should be separated by semicolons.

(Double quote) Use literally in syntax if required. See 66-99 below.

‘ (Back quote) Use literally in syntax if required.

‘‘ ’’ (66-99) In text, quote using 66-99 (two back quotes and two single quotes). Only use quotes when the item quoted would look awkward, incomplete, or ambiguous without them, or when the reader would have trouble parsing the sentence without them. Do not use them when defining a new term.

Only use quotes as a way to emphasize a word or phrase when that word or phrase is unique in some context or technical sense.

Use quotes to indicate material quoted from another source, or to enclose single letters.

Do not use quotes where they may be interpreted as part of the syntax. If in doubt, place the text on its own line.

Do not put quotes around items designated as screen objects.

Closing quotation marks (’’) appear after:

- Most adjacent punctuation marks (commas and periods)
- Question marks and exclamation points when the question marks and exclamation points are part of the quoted material

Closing quotation marks appear before:

- Colons
- Semicolons
- Question marks and exclamation points, when the question marks and exclamation points are not part of the quoted material

, (Comma) Place a comma before the conjunction in a compound sentence (consisting of two or more independent clauses), unless the clauses are short and closely related. For example, ‘‘The system prints an error message, but you can continue processing the file.’’

Use commas to set off a non-restrictive modifier which provides additional information but does not affect the meaning of the words it modifies. For example, ‘‘A symbol value may be an absolute constant, expressed as a 32-bit integer, or a relocatable value.’’ Conversely, do not use commas to set off a restrictive clause which does affect the meaning of the word it modifies. For example, ‘‘Table 6-1 describes the hardware that you need to complete your system.’’

Use commas to set off contrasting and opposing expressions within sentences. For example, ‘‘He changed the software, not the hardware.’’

Place a comma after an introductory clause or long introductory phrase. For example, ‘‘To specify an output device, enter a name in the command line.’’

2.32 Special Characters

The standard character reference used by The Open Group is the ISO 8859-1 Latin-1 character set.

When referring to a printable character for the first time, use the spelled-out name of the character first, followed by the character in parentheses (except in examples, where you should use the character literally). You can add the word ‘‘character’’ if necessary (for example, ‘‘the backslash character’’). After the first occurrence, you can use the name of the character without showing the character itself.

A special character has no plural form. Spell out the name; for example, ‘‘Enter three backslashes (\\\).’’

When referring to non-printable characters, use the name of the character alone in text. For example, ‘‘You must separate arguments with a space character.’’ If the character must be shown without spaces between it and an adjacent character, enclose the name of the nonprintable character in angle brackets. For example:

```
<Tab><Tab>field1
```

The special characters you are most likely to use are:

\# The international number symbol; referred to as the *hash sign*.
£ The UK pound sign (UKP). It should be referred to as the Sterling sign to avoid ambiguity.
@ The at sign.
& The ampersand.
* The asterisk.
\ The backslash.
! The exclamation mark.
$ The dollar sign.
% The percent sign.

+ The plus sign.

/ The slash. Use to separate components of a pathname. Do not end pathnames with a slash. Do not use a slash before a single directory or filename unless it refers to the system root. Do not use the standalone slash to refer to the root directory; use *root*.

= The equals sign.

? The question mark.

ˆ The circumflex.

_ The underscore.

` The grave accent.

| The vertical bar or pipe symbol.

˜ The tilde.

— The em-dash. Use to set off an appositive series from the rest of a sentence, to show an abrupt change in thought, and to set off material for emphasis. With the correct use of commas and parentheses, the em-dash is nearly redundant. Do not place spaces around the em-dash. The exception to this rule is in the NAME section of a reference page:

ls — list contents of directory

- The en-dash. Use to indicate a range; for example, 00-61.

– The minus sign. Use to indicate negative numbers. If the minus sign appears in a program display, use the <-> key.

... Ellipses (horizontal or vertical) indicate the omission of information. In syntax, ellipses indicate an item that repeats. If an ellipsis is part of a screen object, include it in the name; for example, ‘‘The Open... menu item.’’ When using an ellipsis in text, put a space before it and use three periods without spaces.

2.33 Spelling

Use American English spelling.

As an authority for spellings not specifically recommended in this guide, see *Webster’s New Collegiate Dictionary*. An on-line version of this Dictionary is available as follows:

,LI Hypertext Webster Interface
Webster’s Dictionary online with hypertext connections to related words within each definition (Size 4.1K):

http://c.gp.cs.cmu.edu:5103/prog/webster

- Merriam-Webster WWWebster Dictionary
[Help] | [New Search] Type in your word or phrase and press ENTER/RETURN. © 1996, Merriam-Webster, Inc (Size 1.5K):

http://www.m-w.com/netdict.htm

2.34 System Items

Use the construction ‘‘the *name* item’’ when identifying an existing system item such as a command, function, parameter, and so on.

Reverse the construction when identifying a hypothetical system item or something the user must create; for example, ‘‘the file **/etc/OLDpasswd**.’’

Do not rely solely on typographic conventions to identify a system item.

Capitalize system item names as they appear on the system. All system items which are case-sensitive must be presented correctly.

Do not place quotation marks around system item names.

Do not hyphenate variable names used with commands.

Use system item names only as nouns or adjectives; do not use them as verbs.

Avoid starting sentences with the name of a system item or other name that begins with a lowercase letter. However, if you must begin a sentence with such a name, do not change the way it is capitalized.

The following system items should be semantically identified in source files:

arguments	group names
array names	headers
bits	keyboard legends
character classes	macros
constant expressions	modifiers
constants	operands
data structure fields or members	options
data structure names	parameters
environment variables	return values
error values	signals
external variables	symbolic limits
file names	tokens
flags	user-defined structure names
functions	utilities
global variables	

2.35 Tables

Tables should be kept as simple as possible to enable conversion to other formats and display on electronic media with display limitations (such as when using the *man* command on a dumb terminal).

Introduce tables with a complete sentence, followed by a colon when the sentence immediately precedes the table. For example, ‘‘The ASCII codes for these functions are shown in the following table:’’ If you cannot avoid intervening text between the lead-in sentence and the table, use a period at the end of both the lead-in sentence and the intervening text.

Capitalize the first word in each table cell, regardless of whether the cell contains a complete sentence (the only exception is when the word is a literal term that must begin with a lowercase letter). Do not capitalize any other word within a table cell, unless the word is a proper noun, begins a sentence, or is a literal term that must be capitalized.

Abbreviations can be used in table cells where space is limited, but they must be defined. Notes can be used in tables to define abbreviations. Use superscript numbers for notes.

When a common unit of measure applies to all entries in a column, abbreviate it or spell it enclosed in parentheses after the column head. When units of measure are not common to all entries in a column, include the appropriate unit of measure with each entry in the column.

Align columns of decimal numbers on the decimal point, adding trailing zeros for consistency if necessary.

Use parallel construction in table text.

Table titles should use the conventions described in Section 2.37.

2.36 Times

Write specific times using the abbreviations a.m. (ante meridian, morning) and p.m. (post meridian, afternoon). (There is no space after the first period.)

If a time could be misinterpreted (for example, when discussing noon or midnight), you can also add the 24-hour equivalent. For example, ‘‘1:00 p.m. (13:00).’’

Midnight is defined as 12:00 a.m. (00:00).

Noon is defined as 12:00 p.m. (12:00).

Use the full name of time zones rather than their mnemonic codes. For example, Central European Time, Eastern Standard Time.

Do not refer to the date of changes to or from Daylight Saving Time, since this differs depending on the country.

Avoid using the words o’clock, noon, or midnight.

2.37 Titles

In general, make sure all titles are a reasonable length; ideally less than 60 characters.

Titles are used to start elements (such as chapters and appendices) sections, and subsections, and to label figures, tables, and examples.

Titles should summarize content.

Use titles levels 1, 2, 3, and unnumbered. Use of level 4 is discouraged, and use of lower levels is disallowed.

Use initial capitals for the following words in titles:

- The first and last words (always)
- Nouns, verbs, pronouns, adjectives, and adverbs
- Prepositions that contain four or more letters
- The second word in a hyphenated-compound (except articles, coordinating conjunctions, and system items that are case-sensitive)
- Abbreviations, acronyms, mnemonics, and keywords that are normally written in uppercase or lowercase. (If used, remember to define them in the following text.)

Do not use initial capitals for the following words in titles:

- The word ‘‘to’’ in infinitives
- Articles (a, an, and the)
- Conjunctions (and, but, as, and because)
- Case-sensitive system items

Avoid using an article as the first word in a title.

Do not begin a title with a technical term that must begin with a lowercase letter, such as the name of a function or utility.

Do not use typographic conventions in titles.

When a section describes an action to be performed (as in an example or procedure), you can begin the title with a gerund (for example, ‘‘Copying a Directory’’). Sections that are more descriptive can use a noun phrase in the title (for example, ‘‘Output Formats’’).

2.38 Trademarks

Trademarks are names, symbols, or other devices that identify products that are legally restricted to the use of the owner or manufacturer.

Trademarks should be spelled out and capitalized correctly.

Trademarked terms are not marked within text.

All documents for publication, whether in printed or online form, should include a list of trademark acknowledgements in the front matter.

Arrange trademark acknowledgements in alphabetical order.

Use trademarked names only as proper adjectives. For example, the word ‘‘UNIX’’ must be followed by the word ‘‘system.’’ Trademarks can therefore never be possessive or pluralized.

Do not use a trademark as part of a hyphenated compound.

The registered trademark UNIX must not be used to refer to the broader range of UNIX operating systems and their derivatives offered by other companies. UNIX is *now* a brand applied to systems that are branded to the appropriate Open Group specification, and *not* to product implementations.

If an abbreviation or acronym is also a trademarked term, do not spell it out.

It is the responsibility of the author of the document to provide a list of all the trademarks mentioned, although these will also be verified by Open Group editors.

2.39 Warnings

Use a warning to draw the reader's attention to information which, if ignored, can have a critical impact. See also Section 2.28 on page 18 and Section 2.6 on page 10.

Chapter 3

Reference Pages

3.1 Introduction

A *reference manual* is a document with a highly structured design that makes it easy for users to find information on a particular component or topic. The basic unit of information in this documentation is a *reference page*. This guide uses the term reference page to identify an entry in a reference manual.

A reference page may document a single system item, a descriptive topic, a sample program, or some other system feature. This guide uses the term *item* to describe the system component that is being documented in a reference page, and the term *reference page name* for the title of the reference page on which the item is documented. (The reference page name is derived from the name for the item, but the two are not always identical.)

On UNIX systems, the system reference manual is always available online, and users can access the manual from the command line by using the *man* utility. (The name of this utility is an abbreviation of the word *manual*.) For this reason, each reference page ideally should be a separate file, with a name that can be associated with the item being described.

Try to use the name of the system element being documented as the filename. This name cannot include a slash (/), but can include underscores (_). For this reason, the name may differ from the actual system name of the item. If a suffix is part of the actual system name, include it in the filename.

For example, the reference page for the **<sys/time.h>** header is **systime.h**.

If source files are stored on systems that do not support long file names or more than one suffix on a file name, be careful not to create duplicate file names when file names are truncated. In this situation, you must also supply a clear mapping of abbreviated file names to full file names (and reference page names).

Each reference page corresponds to an entry for the *man* utility. (The terms *manual page*, *man page*, and *manpage* are all synonyms for the term *reference page*.) These terms make sense as long as you understand that *page* refers to the unit of reference information, and does not necessarily correspond to a page in a printed book.

Do not refer to the subject of a reference page as ‘‘this *name*’’. Generally, you should refer to the subject of a reference page by name at least the first time it appears in each paragraph; for example, ‘‘The *mknod*( ) function.’’

3.1.1 Divisions

Reference pages may be combined into collections of information, either for external publication or to manage information. For this reason, reference pages are classified into divisions.

A reference page belongs to one of the following divisions, depending on the system item that the reference page describes:

User Utilities
: Describes utilities that are available to all system users.

System Administration Utilities
: Describes utilities that are used to manage systems and networks. System and network administrators read reference pages in this division.

System Calls
: Describes system calls which are functions that are entry points into the operating system kernel. Application developers read reference pages in this division.

Library Routines
: Describes library routines which are functions or macros that are included in libraries. Application developers read reference pages in this division.

File Formats
: Describes formats of headers (include files), program input and output files, and some system files. Application developers and system and network administrators read reference pages in this division.

Special Files
: Describes device special files, related drivers, and networking support available on the system. Application developers and system and network administrators read reference pages in this division.

Miscellaneous and Descriptive Topics
: Provides descriptive information on miscellaneous topics, or describes macro packages used for text processing. All users read reference pages in this division. Introductory pages typically fall into this division.

Examples and Demos
: Describes online examples, sample programs, demos, and games available on the system. All users read reference pages in this division.

3.1.2 Shadow Pages

For a page that documents multiple system items, you should include shadow pages for each system item and external variable (if applicable).

Shadow pages are reference pages that do not have content of their own. Instead, a shadow page points to another page. For example, the reference page for the *write*( ) function also includes documentation for the *writev*( ) function. The name of that reference page is *write*; but you can create a shadow page for a reference page named *writev*.

Shadow files are also used to provide documentation for external variables that are documented on the reference page for a function or header.

3.2 Writing Style

The guidelines documented in Chapter 2 on page 7 should be followed for reference pages. However, this section documents exceptions and additions to those guidelines which are specific to reference pages.

3.2.1 Abbreviations, Acronyms, and Mnemonics

Abbreviations, acronyms, and mnemonics should be defined (using the full name with the short form in parentheses) the first time the term is used in *each* reference page.

3.2.2 Cross-references

Cross-references in reference pages should be limited to the following:

- Within a reference page, you can cross-refer to another section, or to an example, within the same reference page.
- You can refer to another reference page.

All references to other references pages and documents should be listed in the SEE ALSO section.

Avoid referring to a specific section in another reference page. If required, use a descriptive phrase instead (for example, ‘‘see the information on portable file-name characters in ...’’).

To refer to other sections within the same reference page, do not precede the section name with *the*; for example, ‘‘see EXTENDED DESCRIPTION’’.

3.2.3 Examples

Reference pages should include examples for both naive and experienced users.

Short examples can be included to illustrate specific points in the descriptive sections of a reference page. More complete examples can be included in the EXAMPLES section. In the EXAMPLES section, each example should have a title so that it can be referred to from elsewhere in the reference page.

Examples of programming interfaces in a reference page are typically fragments of programs, but separate reference pages containing complete sample programs can be used to show programming calls in context. These complete sample programs can be used as source for example fragments, or you can simply refer the reader to the sample program reference page for examples.

Complete program examples should include extensive comments as part of the program text.

Consider adding examples for the following situations:

- Typical uses of a function or command (if not obvious)
- Usage suggested in Application Usage text
- Clarification of a point in the text
- Illustration of the differences between related functions or commands
- Complex usage of a function or command

3.2.4 External References

If you need to refer to an external document, use an abbreviated document title within the body of the reference page and include the full title in the SEE ALSO section. (As before, the full bibliographic details should be included in the Referenced Documents section.)

3.2.5 Graphics

Reference pages should not include graphics, because pictorial information cannot be included when the pages are displayed using the *man* utility.

3.2.6 Index

The minimum level of indexing for reference pages is a single primary index entry for the name of the item documented on the page.

For reference pages that describe more than one item, include an index entry for each one.

Further index entries may be added as required.

3.2.7 Tables

When creating tables for reference pages, keep in mind that they may need to be displayed in different ways, including on character displays without proportional fonts (as when the *man* command is used on a dumb terminal). For this reason, tables should be kept as simple as possible.

Tables in reference pages should not have captions.

3.2.8 Titles

First-level sections in reference pages have standard titles.

Standard headings that are worded as plurals—such as ERRORS or EXAMPLES—should not be changed to singular when there is only one item in the section.

Subheadings can be used within these standard first-level sections, when it is necessary to subdivide a section. If subheadings are necessary, try to give at least two headings at each level (this is not an absolute requirement for reference pages, but it is the preferred style).

Do not use more than three levels of heading in a reference page.

3.3 Reference Page Sections

Reference page sections use standard names and are ordered in a standard way. Inclusion of the various sections is dependent on the document type, as shown in the following table:

Reference Page Section	Product Documentation*	Shadow Pages	Specifications	Shadow Pages
NAME	M	M	M	M
SYNOPSIS	M[1]	M	M	M
DESCRIPTION	M	M	M	M
SUBCOMMANDS	X	X	O	X
OPTIONS	M (U)	X	M(U)	X
OPERANDS	O (U)	X	O(U)	X
PARAMETERS	O (I)	X	O(I)	X
EXTENDED DESCRIPTION	O	X	O	X
EXIT STATUS	M (U)	X	M(U)	X
RETURN VALUES	M (I)	X	M(I)	X
ERRORS	M (I)	X	M(I)[2]	X
EXAMPLES	O	X	O	X
ENVIRONMENT VARIABLES	M[3]	X	M[3]	X
ASYNCHRONOUS EVENTS	X	X	O (U)	X
FILES	O	X	O	X
NOTES	X	X	O	X
CAUTIONS	X	X	O	X
WARNINGS	X	X	O	X
DIAGNOSTICS	X	X	O	X
APPLICATION USAGE	X	X	O	X
FUTURE DIRECTIONS	X	X	O	X
SEE ALSO	O	X	O	X
CHANGE HISTORY	X	X	M[4]	M[4]

M Mandatory
O Optional
X Exclude
U For utilities.
I For interfaces.
* Includes Common Product Documentation.
1 Except in descriptive pages.
2 This information may be included in RETURN VALUES.
3 If environment variables affect the item.
4 If the item was described in a previous Open Group document.

3.3.1 NAME

Lists the exact names of one or more items discussed in the reference page and provides a very brief description of what each item does.

If more than one item is documented on a reference page, the item that is listed first is used as the title of the reference page. In some cases (such as the *exec*( ) functions), a *group name* is used to represent all of the items on the page.

The description for the page is a short phrase that describes the item documented on the page:

- For functions, macros, and utilities, begin the description with an imperative verb (for example, ‘‘list directory entries’’).
- For descriptive reference pages, use a noun phrase (such as ‘‘conformance’’).
- For headers, begin the description with the phrase ‘‘include definitions for ...’’.

- For sample programs, begin the description with ‘‘sample program for ...’’.

The purpose should include articles as needed for readability. For example, ‘‘compress a file’’ is preferable to ‘‘compress file.’’ Do not use end punctuation or tag individual words within the purpose.

Do not capitalize the first word of the purpose unless it is a proper noun or a literal term that must be capitalized.

3.3.2 SYNOPSIS

The SYNOPSIS summarizes the syntax for an item.

For utilities, this section shows the command-line syntax. It should provide complete reference information, including all options, option-arguments, and operands. To ensure that the SYNOPSIS is useful as a quick reference, avoid using general terms to represent a group of options or operands.

First, list the options that do not take arguments. Group required and optional options separately. Do not group options that are mutually-exclusive—instead, separate them using vertical bars.

List options in alphabetical order. When options are listed in separate groupings, list them in alphabetical order within each grouping.

For example:

```
utility-name [-AaBcXxyz] [-p option-argument]
```

Mutually exclusive forms of a utility, or distinct ways of using a utility, should each be shown.

For functions or macros, this section shows the include statement and the form used for program calls; it may also show external variable declarations. It should provide complete reference information, including all parameters.

If multiple interfaces are documented on the same reference page, each should have its own entry.

Put mandatory parameters before optional parameters, unless the item requires otherwise. Indicate parameters of indeterminate number with an ellipsis following the parameter name. Do not use plural parameter names.

For headers, this section shows the include statement for the header. It may also show external variable declarations.

For sample programs, this section shows the compile line for the program.

For descriptive reference pages, this section is omitted.

3.3.3 DESCRIPTION

The DESCRIPTION provides a summary description of the item being documented.

For a function, macro, or utility, this section explains how the item is used. The description should be detailed enough so the reader can understand the standard use of the item, but should be kept quite short. (More detailed information should be presented in the EXTENDED DESCRIPTION section.)

A good summary should provide the following information:

- Purpose of the item (in general terms).
- How to control the item to perform different tasks. In explaining different uses, you can mention specific options, operands, or parameters, but should not discuss them in depth. For a utility, you should also explain how the utility operates when no options are specified.
- Common applications of the item.
- Input and output for the item.

If more than one item is documented on the page, this section should give some idea of the differences among them.

Unless all of the descriptive information for the item can be contained within 8 to 10 lines of text, avoid explaining the internal operation of a function, macro, or utility; discussing differences among implementations; mentioning specialized uses; or presenting output formats. Such information should instead be included in the EXTENDED DESCRIPTION section.

This section should use the same names for option-arguments, operands, and parameters as those used in the SYNOPSIS section.

Reference pages for descriptive topics do not include EXTENDED DESCRIPTION sections. Instead, they use subheads within the DESCRIPTION section. The same approach is used for reference pages that document files and sample programs.

Shadow pages should use this section to cross-refer to another reference page.

3.3.4 SUBCOMMANDS

The SUBCOMMANDS section describes a utility's subcommands, if any.

3.3.5 OPTIONS

The OPTIONS section lists all the options for a utility and provides a description of each option. The options are described in a definition list that is introduced by a lead-in sentence. Begin each description with a verb in the present tense, beginning with a capital letter (such as ‘‘Specifies’’ or ‘‘Indicates’’).

Information that applies to all options should precede the list, so that readers will not overlook it.

List options in alphabetical order.

For each option, specify whether it is mandatory or optional, or is mutually-exclusive with another option.

If no options are included with a utility, use standard text to state this fact.

Do not start a sentence with an option; use ‘‘The –z option ...’’

Arguments that are listed in the SYNOPSIS section, but are not associated with a particular option, are *operands*. Operands should be described in the OPERANDS section.

3.3.6 OPERANDS

This section provides a list of the arguments that are supplied directly to a utility, rather than to one of its options. Operands generally follow the options on the command line.

The operands are described in a definition list that is introduced by a lead-in sentence. Begin each description with a verb in the present tense, beginning with a capital letter (such as ‘‘Specifies’’ or ‘‘Identifies’’).

List operands in alphabetical order.

3.3.7 PARAMETERS

This section describes the arguments supplied to a programming interface (system call or library routine).

The parameters are described in a definition list that is introduced by a lead-in sentence. Begin each description with a verb in the present tense, beginning with a capital letter (such as ‘‘Specifies’’ or, for a pointer, ‘‘Points to’’). Do not include asterisks for the pointer, either on the term or within the definition.

List parameters in alphabetical order.

3.3.8 EXTENDED DESCRIPTION

This section gives more detailed information about the use of the item being documented, and about its behavior and features. It expands on subjects discussed in the DESCRIPTION section, but with more detail and background information. Use this section to discuss the internal operation of a function, macro, or utility; to point out differences among implementations; to mention specialized uses; and to present output formats.

This section should use the same names for option-arguments, operands, and parameters as those used in the SYNOPSIS section.

Implementation-dependent information within the EXTENDED DESCRIPTION section should be discussed at the end of a paragraph (see Chapter 5 on page 43).

The EXTENDED DESCRIPTION section may use short examples to clarify a point being made. If an extensive example is needed, include the example in the EXAMPLES section, and cross-refer to it.

Information within the EXTENDED DESCRIPTION section can be organized under subheadings if the information is extensive and their addition improves readability. As far as possible, you should use standard titles for these subheadings, such as Standard Input, Standard Output, Standard Error, Input Files, Output Files, Consequences of Errors.

3.3.9 EXIT STATUS

This section describes the values returned by a utility after completion. These values differ from those in the RETURN VALUES section, which are used for programming interfaces.

The exit status values are described in a definition list that is introduced by a lead-in sentence (use standard text for the lead-in sentence).

List the exit status for successful completion first. If appropriate (based on the source information), begin the description with the phrase ‘‘Successful completion.’’ If additional information follows, use a colon (:) after the initial phrase, followed by a sentence beginning with a lowercase letter.

Next, present the exit codes for errors in numerical order. If appropriate (based on the source information), begin the description with the phrase ‘‘An error occurred.’’ If additional information follows, use a colon (:) after the initial phrase, followed by a sentence beginning with a lowercase letter.

3.3.10 RETURN VALUES

This section lists the returned values or codes for a programming interface (system call or library routine), followed by a description for each. (If appropriate, a description may say that a return value is unused or reserved for future use.)

Use a definition list for return values. (This may not be the best approach when the return values are not literals.) If more than one function or macro is included on a reference page, you may need to include more than one list of return values. Each list is introduced by a lead-in sentence that indicates the name of the functions to which each group of return values applies.

The return values for success and failure are clearly identified using standard phrases: ‘‘Success’’ or ‘‘Failure.’’ If additional explanation follows, the phrase is ended with a colon, and the explanation follows in a sentence that begins with a lowercase letter.

3.3.11 ERRORS

This section documents *errno* values set by a programming interface (system call or library routine).

Usually an interface returns a numeric value to indicate failure, and it also sets *errno* to specify the reason. The structure of this section allows quick access to the error information.

Use the definition list format for errors.

Errors should be categorized depending on whether they are required or optional for a conforming system. Use standard text for the lead-in sentences for each such grouping of errors.

If multiple interfaces are documented on the same reference page, the errors should be divided by interface (unless grouping can eliminate duplication without sacrificing clarity). If the *errno* values differ in the description of the condition, the *errno* values should be separated for each interface.

List error codes in alphabetical order within each grouping.

If no errors are defined, or if the specification indicates that only the errors listed in the specification can occur, include standard text to state that fact.

3.3.12 ASYNCHRONOUS EVENTS

This section lists how a utility reacts to such events as signals and which signals are caught.

3.3.13 EXAMPLES

This section should be included whenever possible, to show how functions or utilities can be used. Although tutorial information does not appear in reference pages, examples are often useful to help readers understand how to use a function or utility. Examples should be sufficiently complete real-world examples of actual user tasks, unlike the code fragment examples in the EXTENDED DESCRIPTION section, which simply clarify a point.

Do not manually number examples; this should be handled by the output format.

Command and utility examples should show how to use the utility to accomplish a specific task.

Interface examples should show how to use the interface in a program. The example need not be a complete, compilable program. It can be a code fragment. If the example is a complete program, the example should be tested to be certain it will compile and link on a target system. An introductory sentence should indicate that it is a compilable program.

Complete sample programs may be contained in separate reference pages, and can be used as a source of examples or referred to from other reference pages.

If the SYNOPSIS tells you everything you need to know about a utility or function, no examples are needed.

The title for an example should describe what the example is demonstrating. When creating a title for an example that shows how to use a utility or function to perform a particular task, use a gerund (that is, a verb ending in ‘‘-ing’’) to indicate that an action is occurring. In addition, the title should usually describe the task in general terms, identifying a particular technique or task, rather than simply listing the constructs that are used in the example. For example, an example for the *tar* utility might be entitled ‘‘Creating an Archive.’’ This title is preferable to one such as ‘‘Using the –c Option.’’

For examples that do not show how to perform specific tasks, such as a sample file or a list of expressions, use a noun form such as ‘‘Sample File Access Permissions.’’

3.3.14 ENVIRONMENT VARIABLES

This section lists and describes the environment variables that affect the system item. It appears in most reference pages for utilities, but is rarely used in reference pages for system calls or library routines.

Environment variables are always spelled using all uppercase letters.

The environment variables are described in a definition list that is introduced by a lead-in sentence. Begin each description with a verb in the present tense, beginning with a capital letter.

Environment variables sometimes have the same names as locale categories. Be careful not to confuse them. You do not need to list or describe locale categories in the ENVIRONMENT VARIABLES section.

In general text, where only a name (such as LC_CTYPE) is used in the source, you should add text to indicate whether an environment variable or a locale category is being discussed.

List environment variables in alphabetical order.

3.3.15 FILES

This section lists all of the files or directories of files, other than those having user-supplied names, that are read, created, modified, or otherwise touched by an interface or utility. List all such files in a definition list, and give a brief description of each file.

Include files such as system-supplied configuration, initialization, or log files in this section. Do not include user-supplied input files or output files.

List files in alphabetical order.

3.3.16 NOTES

This section includes any supplementary information that is peripheral to the actual operation of the system item. Use this section instead of individual notes in other sections.

3.3.17 CAUTIONS

This section includes information on possible system damage or data corruption that may occur as a result of using the system item in a specific implementation.

3.3.18 WARNINGS

This section includes information which, if ignored, can have a critical impact.

3.3.19 DIAGNOSTICS

This section provides information useful for diagnosing errors that may result when the system item is used.

3.3.20 APPLICATION USAGE

This section includes information by which the user can avoid misuse of the system item.

3.3.21 FUTURE DIRECTIONS

Include this section if there are firm plans to change the item in the future or to introduce new features that address a specific limitation, or if there is a known uncertainty which is expected to be resolved.

3.3.22 SEE ALSO

This section lists other reference pages that are referred to in the reference page, along with any others that are relevant. References to standards documentation may be made here, in a separate paragraph.

The list of reference pages is grouped by reference manual divisions, and alphabetized within each group (ignoring case). The groups are listed in the following order:

- Utilities (user command and system administration divisions, ordered as a single division)
- Programming Interfaces (system call and library routine divisions, ordered as a single division)
- File Format
- Miscellaneous (descriptive pages)

- Device
- Example

Where referring to a system item that is not the main item on another reference page, use the main reference page name.

After listing the reference pages, you can list any other document references (using complete titles) that are appropriate in a separate paragraph, also in alphabetical order.

Do not include the word *and* before the last entry, or type a period after the last entry, in either of these paragraphs. Separate the entries with a comma.

3.3.23 CHANGE HISTORY

This section indicates the first document in which the system item appeared, and briefly summarizes all changes applied since the previous issue, if any.

Chapter 4

Product Documentation

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

This chapter needs to be developed.

4.1 Introduction

Documentation plays an important part in how well a product is received by its users.

Most readers approach new product documentation with similar attitudes:

- They are eager for action.
- They are primed to do work with the product.
- They are short of time.
- They are motivated to succeed.

You can help your readers learn quickly by writing for action. Where appropriate, introduce simple tasks or procedures very early, and make sure users are certain of success in performing those procedures.

Research shows that readers approach learning about new products in two ways, with two different learning styles.

Experimenters want to try things right away instead of taking the time to read the instructions. Experimenters share these characteristics:

- They read written procedures only as a last resort.
- They believe instructions are probably incomplete.
- They rely heavily on tables of contents, section titles, and indexes when searching for information.
- They consult examples more often than text for information.

Directed learners read before they act. Directed learners share these characteristics:

- They read procedures and explanations carefully.
- They believe the documentation is correct.
- They pause over tiny discrepancies.
- They consult examples to confirm the correctness of their actions.

Most documentation should provide information that works for both the experimenter and the directed learner. Where appropriate, you should include step-by-step teaching exercises (tutorials) for directed learners and easily accessible reference and procedural information for

experimenters.

Experimenters make heavy use of the table of contents, section titles, and the index to find information. Directed learners often read straight through a document and later return to the same elements for review.

A comprehensive index is the most useful aid you can provide a searching reader; a clear table of contents is almost as important. When you write section titles, make them lively and direct so that readers can tell exactly what a section is about. The table of contents then becomes much more useful as a pointer to information.

Your knowledge of the product—its uses, benefits, and idiosyncrasies—is critical to developing useful documentation. Occasionally, writers try to develop documentation based solely on the developers' specifications. Even when the developers are very capable writers, the documentation suffers.

Get your hands on the product and use it as much as time allows. When you work with the product you learn to use it effectively, and you can pass along to readers many tips and shortcuts you could not otherwise have provided.

As you think about potential users of your documentation, keep in mind the product's market—what kind of individuals or businesses will use it, how they will use it, and what benefits the product can provide. Consider the internationalization and localization issues that must be addressed as you plan the development of your document. Questions like the following can be useful when examining potential users of your document:

- Who are the users?
- What does the product do for the users?
- What are the product's benefits?
- Where will the product be used?

4.2 Document Structure

User Documentation

The requirements in this section apply to documentation that describes how to install, operate, and manage software. The following table shows the components that are required in all software user documentation and the order in which they appear. Most components are required in all volumes of a multivolume documentation set.

Component	First or Only Volume	Other Volume
Title Page	M	M
Copyright Page	M	M
Restrictions	M	M
Warranties*	R	R
Acknowledgements	M	M
Contents	M	M
List of Illustrations	M	M
List of Tables	M	M
Preface	M	R
Audience	M	M
Applicability**	M	M
Purpose	M	R
Document Usage	M	R
Related Documents	R[1]	R
Typographic and Keying Conventions	M	M
Problem Reporting	M	R
Document Body	X	X
Appendixes	O	O
Glossary	M	M[2]
Index	M	M[2]

Notes:

M Must be included when the information exists
O Optional.
R Reference; include within the volume or reference another document that contains the information.
X Dependent on document type.
1. Addresses relationship to other volumes.
2. Must be included in at least one volume; use references to the component in other volumes.
* Warranty information varies by country, and is often legally required in the local language. Warranty information should therefore be included in an addendum.
** Applicability is a new section which is added specifically for Product Documentation. It identifies the software product (including version number) to which the document applies. It includes, if relevant, information about the software and hardware environment required to use this software.

Development Documentation

The requirements in this section apply to documentation that describes how to develop, modify, and port software to specific platforms. Examples are coding style guides, programmer's reference manuals, and software porting guides.

Chapter 5

Common Product Documentation

5.1 Introduction

Common product documentation is based on approved Open Group specifications and is designed to:

- Include all pertinent information in a way that is understandable and meaningful to all users
- Be consistent in organization and style
- Provide placeholders for integration of vendor-specific information

The inclusion of vendor-specific information enables the clear distinction between that which applies to all conforming systems and that which is vendor-specific. It also ensures that vendor-specific information is included in a way that is consistent across all vendor systems.

Common product documentation presents information in a form that is suitable for users, programmers, and system administrators. It provides clear explanations of system features, examples of their use, and complete reference information. It also distinguishes between common features that are available across all conforming systems and extensions that may be supported only on systems from a particular system vendor.

Language in specifications—because it often has a special meaning, and because it is addressed to system implementors—often needs to be revised for product documentation.

Specifications also include conformance information for branding purposes. This information is not necessarily used in product documentation, but the information should be retained in the document source.

5.2 Language

The language used in specifications is for implementors. As such, it often uses precisely defined terms that will be unclear to users of a system. To create common documentation it is often necessary to revise text to clarify the meaning for a reader who is more interested in how a particular system behaves, or how much they can rely on a particular behavior across systems that conform to the specification, than in how the specification defines a particular behavior.

When features or behavior may vary among different systems, you should add a *vendor placeholder* that can be used to present system-specific information, so that system vendors can document the differences on their systems. In addition, when you create a vendor placeholder, you should include a comment to the vendor (which does not appear in formatted text) indicating the information that should be added. Information that is optional for vendors to add should be marked with the phrase ‘‘VENDOR ADDITION:’’ within the comment. Information that vendors must add for conformance to the specification should be marked with the phrase ‘‘VENDOR REQUIREMENT:’’ within the comment.

In addition, specifications often abbreviate the information for some items, by referring to other volumes of the specifications, by referring to a glossary entry, or by specifying the behavior of a system item in relation to another system item, instead of spelling out exactly how it works from a user's perspective.

The audience for common product documentation is interested in a clear answer to the following questions:

- Will all systems that conform to the specification include this feature or behavior? If not, reword and add a vendor placeholder.
- More specifically, if I am writing an application or script that must be portable, can my script or application depend on this feature or behavior? If not, reword and add a vendor placeholder.

When vendor-specific information is integrated, users also need a clear answer to the following question:

- What are the specifics for the product I am using now? Add a vendor placeholder when the specification does not define a particular feature or behavior, so that vendors can explain how it works on their systems.

The reader must also be able to distinguish clearly between base information (which applies to all conforming systems) and vendor-specific information that might be added in a vendor placeholder (which applies only to a specific system). Further, the presentation must make sense both when vendor-specific information is omitted (because vendors are not required to add information for every vendor placeholder) and when vendor-specific information is added.

5.3 Terminology

The following terms are indicators that you may need to add a placeholder for vendor-specific material, or otherwise adapt text from the specification:

Can ????

Implementation-dependent

In a specification, this term refers to values or behavior not defined by the specification. Users are likely to need, and therefore vendors should supply, implementation-dependent information in order to write and debug programs or scripts. The source code should supply a vendor placeholder.

The term ‘‘implementation-dependent’’ can be ambiguous when it is used in product documentation, because the term can refer to variations among a vendor's product lines, as well as to each vendor's implementation of a feature according to a standard. Therefore, reword the sentence to explicitly note any portability implications and to ensure graceful integration of product information by vendor writers. For example:

In a specification:

> The behavior of *fseek*( ) on devices that are incapable of seeking is implementation-dependent.

In common documentation:

When the *fseek*( ) function is used for devices that are incapable of seeking, the results may vary among systems that conform to the specification.

[vendor placeholder]

The following example shows a case in which the specification requires that vendor-specific information be added:

In a specification:

An implementation will document any condition not specified by this document under which the implementation generates signals.

In common documentation:

```
VENDOR REQUIREMENT:
The specification requires a system vendor to document any
conditions that cause signal generation and that are not
described in the specification. The following placeholder
is provided to enable vendors to add any appropriate
conditions.

[vendor placeholder]
```

In this example, no text actually appears in the formatted document unless the vendor adds information for the placeholder; but the information is required for conformance to the specification.

May

The term ‘‘may’’ indicates optional behavior that may vary among conforming systems, so you should state that point directly and add a vendor placeholder:

Because the term ‘‘may’’ implies variation among different systems, you should include a vendor placeholder when this word is used in a SYNOPSIS, DESCRIPTION, RETURN VALUES, or ERRORS section.

For example, in a specification:

This function may return an error code of –1 if an error occurs.

is changed to:

On some conforming systems, this function returns an error code of –1 if an error occurs.

In an Errors section:

The *fopen*( ) function may fail if: [list of errnos and conditions]

is changed to:

The *fopen*( ) function can set *errno* to one of the following if the corresponding error condition occurs, but systems that conform to the specification are not required to detect these conditions: [list of errnos in the specification’s ‘‘may’’ list and vendor placeholder]

Obsolescent

Some text in the specification may be marked as obsolescent using the marginal tag OB, with shading of the text that applies to the obsolescent feature. Remove these markings.

In addition, you should add text such as the following at the end of the DESCRIPTION section:

The *grep* –**F** utility is the recommended alternative to the *fgrep* utility, which may be withdrawn in a future version of the specification.

Should ????

To be withdrawn

Reword this terminology to avoid misunderstanding.

For example, in a specification:

cc — a C-language compilation system (TO BE WITHDRAWN)

is changed to:

cc - a C-language compilation system

In addition, add text such as the following at the end of the DESCRIPTION section:

The *cc* utility is scheduled to be withdrawn from a future version of the specification. The *c89* utility is the recommended replacement.

Undefined

You should spell out what these words mean because vendor writers may have to describe error detection and behaviors that the specification says are undefined and unspecified. It is confusing for a reader to read that something is unspecified in one section of a reference page and then read the behavior specification somewhere else. Mark-up techniques do not entirely eliminate this confusion, especially when readers are searching the reference page to find specific strings.

Here are some examples of how to reword this terminology in reference pages.

In a specification:

It is unspecified whether writes to the same portion of the file prior to the *msync*( ) call are visible by read references to the memory region.

In common documentation:

On some systems that conform to the Single UNIX Specification, read references to a specific memory region may not be able to access the results of write operations that were made to the same portion of the file before the current *msync*( ) call.

This rewording allows the vendor writer to incorporate, if necessary, a paragraph like the following one without altering the text supplied by the specification:

On *name* systems, read references can access write operations made to the file prior to the call.

In a specification:

If an insufficient number of arguments is passed to accept all the values that the regular expression returns, the behavior is undefined.

In common documentation:

If the call does not pass a sufficient number of arguments to accept all the values that the regular expression returns, the results may vary among systems that conform to the Single UNIX Specification.

[vendor placeholder]

Unspecified

Refer to Undefined.

Will When the term ‘‘will’’ is used in a SYNOPSIS, DESCRIPTION, RETURN VALUES, or ERRORS section, you can reword the sentence to simply describe the feature or behavior as the way the item works (on all systems that conform to the specification).

Depending on context, the verb can be eliminated entirely, or may need different wording. For example, in the following text the word *will* indicates the required behavior of the function, so the sentence can be rephrased to use the present tense:

In a specification:

> This function will return an error code of –1 if an error occurs.

In common documentation:

> This function returns an error code of –1 if an error occurs.

The following examples shows the use of ‘‘will’’ in an ERRORS section:

In a specification:

> The *fopen*( ) function will fail if: [list of errnos and conditions]

In common documentation:

> On all systems that conform to the specification, the *fopen*( ) function sets *errno* as listed for the following conditions: [list of the errnos in the specification’s ‘‘will’’ list]

5.4 Equations

Equations should be followed by a vendor placeholder that can be used to include an alternate version of the equation.

5.5 Reference Pages

Copyright

Each reference page contains a copyright notice as part of the metainformation for the page. This copyright statement, added as a customizable text entity, assigns the copyright to The Open Group. When vendors integrate reference pages into their own product documentation, they should add their own copyright statements to the file as a separate line of metainformation.

SYNOPSIS

Do not change the order of parameters in a function synopsis; use the same sequence as in the specifications.

OPTIONS

System vendors can define additional options for a utility, so you should include vendor placeholders before and after the option list. The placeholder before the option list can be used to indicate that additional options are defined. The placeholder following the list can be used to list the additional options.

OPERANDS

System vendors can define additional operands for a utility, so you should include vendor placeholders before and after the operand list. The placeholder before the operand list can be used to indicate that additional operands are defined. The placeholder following the list can be used to list the additional operands.

EXTENDED DESCRIPTION

Implementation-dependent information within the EXTENDED DESCRIPTION section should be discussed at the end of a paragraph, so that system vendors can add information following the paragraph, in a modular fashion.

There are several specification section headings that should not appear as section headings in reference pages. They are:

- STDIN (Standard Input)
- STDOUT (Standard Output)
- STDERR (Standard Error)
- INPUT FILES
- OUTPUT FILES
- CONSEQUENCES OF ERRORS

The information contained in these sections should be added to the reference page EXTENDED DESCRIPTION section. Do not use these section headings in the reference pages unless the information they contain is extensive or significant, and their addition improves readability. In this case, use the specification's main headings as subheadings in the EXTENDED DESCRIPTION section.

If they are used as headings, spell out the headings Standard Input, Standard Output, and Standard Error.

You may also include nonstandard subheadings (such as ''Output Format'' and ''Interactions Among Options'') if their inclusion makes the subject matter easier to scan.

In reference pages that document multiple programming interfaces, the information for each interface should be kept as a block of text that is identified by a subheading (unless the information is very brief). The subheading should use the interface name as the title of the section.

Some specification section headings contain information that may be included in the EXTENDED DESCRIPTION section. Include the information, but do not use these titles as subsection headings. They are:

- APPLICATION USAGE
- ASYNCHRONOUS EVENTS

Some specification section headings contain information that should not be included in the reference page. Do not include information from those sections. They are:

- CHANGE HISTORY
- FUTURE DIRECTIONS

RETURN VALUES

System vendors can define additional return values for a function or macro, so you should include vendor placeholders before and after the list of return values. The placeholder before the list can be used to indicate that additional return values are defined; the placeholder following the list can be used to list the additional return values.

ERRORS

In most cases, system vendors can define additional errors for a function or macro, so you should include vendor placeholders before and after each list of errors. The placeholder before the list can be used to indicate that additional errors are defined; the placeholder following the list can be used to list the additional errors.

EXAMPLES

One of the primary jobs of a reference page writer is to consider whether additional examples are needed for a particular reference page. If the specifications do not include examples, or if the examples are not helpful for the user of a command or for a programmer using a function, you should consider adding examples.

FUTURE DIRECTIONS

Not included.

CHANGE HISTORY

Not included.

Appendix A

Terminology

This appendix gives selected terms, their meaning, and preferred spelling and alternatives, if applicable.

Open Group documentation is likely to be translated. Remember to convey important distinctions by using the right term.

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

This is a starting point only.

ANSI
American National Standards Institute.

ASCII
American National Standard Code for Information Interchange.

above
Do not use to refer to another location in a document.

alphanumeric
When used as a modifier.

and/or
Avoid using this term. Rewrite the sentence, or use the two options, followed by ‘‘or both.’’

appendixes
Do not use *appendices*.

argument
Value provided to a function or utility (such as the value of a parameter, operand, or option-argument).

async
Use *asynchronous*.

B
Byte.

Boolean
Note capitalization.

BSD
Berkeley Software Distribution.

b
Bit.

backslash
When used as a modifier or noun.

backspace
When used as a modifier, noun, or verb.

baud rate
Often incorrectly assumed to indicate the number of bits per second (bps) transmitted. Baud rate actually measures the number of events, or signal changes, that occur in one second. In most instances when baud rate is used, the correct term is ‘‘bps.’’ Check your source material before using the term baud rate.

behavior
Do not use ‘‘behaviour.’’

below
Do not use to refer to another location in a document.

boot
Start up a system.

built-in
When used as a modifier.

built-in utility
Special utility implemented within the shell. Also see *shell*.

can
Describes options, requirements, and recommendations that apply to all portable applications and scripts.

The term *can* describes a feature that must be implemented on a system that conforms to the specification, but that is optional for a user or application.

cf
Expand to *compare*.

choose
Use this verb when picking an operation from a menu.

circa
Expand to *about*.

client-server
When used as a modifier.

commands
Call to the shell to perform a specific task. A string entered on the command line or in a script. Contrast with *utility* which is the name of an executable.

command-line
When used as a modifier.

cross-reference
When used as a noun.

DBMS
Database management system.

DCE
Data communications equipment.

DCE
Distributed Computing Environment.

DES
Data Encryption Standard.

DIF
Data interchange format.

DSR
Data set ready.

DTD
Document Type Definition.

DTE
Data terminal equipment.

DTR
Data terminal ready.

data
Use as a singular noun. Do not use *datum*.

database
When used as a modifier or noun.

default
Value or behavior provided by the system.

directory name
When used as a noun.

disk
Use for any disk other than an optical disc.

diskette
State the size (3.5 or 5.25 inches), and do not use the modifier *floppy*.

display
Use the verb *display* rather than *appear*. For example, ‘‘The prompt is displayed on the screen.’’ Avoid using *displays* without an object. For example, ‘‘The system response displays on the screen.’’ (Use of the noun *appearance* is acceptable.)

dump file
When used as a noun.

EBCDIC
Extended Binary-coded Decimal Interchange Code.

EIA
Electronics Industry Association.

EISA
Extended Industry Standard Architecture.

EOF
End of file.

EOT
End of tape.

EOT
End of transmission.

Ethernet
Note capitalization.

e.g.
Expand to ‘‘for example,’’ (note that addition of a comma).

et al
Expand to ‘‘and others.’’

email
When used as a modifier or noun.

end user
Person using a system feature.

end-user
When used as a modifier.

enter
To submit input to the system. Also see *type*.

Do not use *enter* to indicate the start-up of an application.

entry level
When used as a noun.

etc.
Expand to ‘‘and so forth’’.

execute
To run a command using the current execution environment. Also see *invoke*.

FIFO
First in, first out.

FILO
First in, last out.

FIPS
Federal Information Processing Standard.

Fortran
Formula translation.

FTP
File Transfer Protocol.

fewer
Use to refer to countable items; for example, ‘‘you will find fewer errors’’.

file name
When used as a noun.

filename
When used as a variable in syntax or examples.

file sharing
When used as a noun.

file-sharing
When used as a modifier.

file system
When used as a noun.

fixed length
When used as a noun.

floating point
When used as a noun.

floating-point
When used as a modifier.

G
Giga (prefix).

Gbyte, GB
Gigabyte.

GID
Group identification.

general-purpose
When used as a modifier.

hard copy
When used as a noun.

hard-copy
When used as a modifier.

hexadecimal
When used as a modifier.

IEEE
Institute of Electrical and Electronics Engineers.

I/O
Input/output.

IRQ
Interrupt request.

ISA
Industry Standard Architecture.

ISA
Instruction-set architecture.

ISDN
Integrated services digital network.

ISO
International Organization for Standardization.

i.e.
Expand to ‘‘that is,’’ (note that addition of a comma).

if
Use when an event is conditional.

implementation
Refers to the way a function or utility works on a particular operating system.

implementation-dependent
Describes a value or behavior that may vary among conforming systems. An application that relies on such values or behavior is not portable across all conforming systems.

Each implementation shall provide documentation of its behavior.

invoke
Run a command with suppression of searching for shell functions and special built-in utilities.

initialize
Not ‘‘initialise’’.

in-line
When used as a modifier.

input
Use as a noun only, not as a verb.

interconnect
When used as a modifier, noun, or verb.

interface
When used as a noun or modifier, not as a verb.

interprocess
When used as a modifier or noun.

Kb
Kilobit.

Kbyte, KB, or K
Kilobyte.

Korn shell
Not *KornShell*.

k
Kilo (prefix).

keyboard
When used as a modifier or noun.

LP
Line printer.

left-justified
When used as a modifier.

legacy
Describes a feature that is being retained for compatibility with older applications, but have limitations which make them inappropriate for developing portable applications. New applications should use alternative means of obtaining equivalent functionality.

less
Use to refer to non-countable items or when discussing something in terms of size or degree; for example, ‘‘this is less complicated’’.

log file
When used as a noun.

log in
When used as a verb.

login
When used as a modifier or noun.

log out
When used as a verb.

logout
When used as a modifier or noun.

lowercase
When used as a modifier or noun.

MAC
Medium access control.

MAC
Memory access controller.

Mb
Megabit.

Mbps
One million bits per second.

Mbyte, MB
Megabyte.

MSB
Most significant bit.

mailbox
When used as a noun.

may
Indicates a feature or behavior that is not required on conforming systems. (‘‘Need not’’ is the opposite of ‘‘may.’’) An application that relies on such features or behavior is not portable across all conforming systems.

media
Use as a singular noun.

modem
Modulator.

mount
Make available to the system.

mouse
Use to refer to any pointing device, screen button, or menu operation. (Remember to define this usage.)

mouse button
Use to refer to a button on a mouse. Avoid the generic *button*. Use the verbs *click*, *double-click*, *drag*, *press*, *hold*, and *release* to refer to a mouse button.

ms
Millisecond.

multiplexer
When used as a modifier or noun.

multitasking
When used as a modifier.

multiuser
When used as a modifier.

must
Describes options, requirements, and recommendations that apply to all portable applications and scripts. The term *must* describes a requirement for a user or application.

NaN
Not a number.

need not
The negative of *may*. Used in preference to *may not* to avoid ambiguity.

newline
When used as a modifier or noun.

nonzero
When used as a modifier or noun.

OR
Do not use as a verb. For example, instead of saying ''OR-ing the bits'' say ''a logical bitwise OR of the bits''.

OS
Operating system.

OSI
Open Systems Interconnection.

obsolescent
Describes a feature that may be considered for withdrawal in a future version of the specification. Such features are retained because of their widespread use, but are not recommended for new applications. Vendors may continue to support such features, even after they are withdrawn from the standard.

offline
When used as a modifier.

online
When used as a modifier.

open systems
(Or XSI-conformant systems) Use generically. If you need to specify the subset of open systems that are based on the UNIX operating system, write: ''UNIX operating systems and their derivatives.''

option
Argument to a command that (typically) changes the default behavior of the command.

option-argument
Argument to an option.

output
When used as a noun only, not as a verb.

PID
Process identifier.

POSIX
Portable Operating System Interface for Computer Environments.

path name
When used as a noun.

pathname
When used as a variable in syntax examples.

path-name
When used as a modifier.

per

press
Used to indicate the action of pressing a key that does not echo to the screen; the Control key is one such example.

previous
Do not use to refer to another location by position.

remove
Use this verb to refer to a dialog box. For example, ‘‘The dialog box is removed from the screen.’’

runtime
When used as a modifier or noun.

SCCS
Source Code Control System.

SGML
Standard Generalized Markup Language.

s
Second.

screen button
Use to refer to a button on a screen. Use the verb *click on* for controls on the screen.

screen object
Anything that appears on a screen; for example, box, menu, icon, and so on. Do not use the names of screen objects as verbs.

sec
Second.

select
Use this verb to designate information that will be the subject of a subsequent operation.

shall
The feature must be implemented; applications can rely on its existence.

shell, the
Change to ‘‘the shell as documented in the *sh*( ) reference page’’ when referring to the default shell provided by systems that conform to the Single UNIX Specification.

should
This term describes options, requirements, and recommendations that apply to all portable applications and scripts.

The term *should* describes features of an implementation that are recommended but not required. An application that relies on such features is not portable across all conforming systems.

When referring to a user or application, this term describes recommended practice that is suggested for maximum portability.

If possible, reword the sentence; for example, ‘‘To ensure portability, applications ...’’

shut down
When used as a verb.

shutdown
When used as a modifier.

string
Contiguous sequence of bytes, terminated by and including the first null byte.

subdirectory
When used as a noun.

superuser
When used as a noun.

sync
Synchronous.

TCP
Transmission Control Protocol.

TCP/IP
Transmission Control Protocol/Internet Protocol.

tab stop
When used as a noun.

text-only
When used as a modifier.

that
When used as a restrictive pronoun. For example, ‘‘... the subset of open systems that are based on ...’’

time zone
When used as a noun.

type
Used to indicate the entering of information. For example, ‘‘Type the following command.’’

UID
User identification.

UNIX
Note capitalization.

U.S.
United States.

UUCP
UNIX-to-UNIX Copy.

undefined
Describes a value or behavior that may occur in response to an error, but that is not defined by the specification and may vary among conforming systems. An application that relies on such values or behavior is not portable across all conforming systems.

unspecified
Describes a value or behavior that may occur in response to correct usage, but that is not specified by the specification and may vary among conforming systems. An application that relies on such values or behavior is not portable across all conforming systems.

uppercase
When used as a modifier or noun.

user ID
When used as a noun.

user name
When used as a noun.

utility
Executable file that can be called by name from a shell (not including *built-in utilities*). See also *command*.

versus

via
Change to ‘‘through’’ or ‘‘by means of.’’

vice versa

viz
Expand to ‘‘namely.’’

when
Use if an event is inevitable. Do not use to mean in contrast/comparison to.

where
Do not use to mean in contrast/comparison to.

which
When used as a nonrestrictive pronoun, and preceded with a comma. For example, ‘‘... the X/Open Portability Guide, which contains ...’’

while
Do not use to mean in contrast/comparison to.

will
Indicates a behavior that is required on conforming systems. This means that a user or application can depend on the feature or behavior across all systems that conform to the specification.

windows
Use the verbs *open* and *close* to refer to windows.

zeros
Not zeroes.

Appendix B

Extensions

This appendix defines the extensions in use at the time of publication.

The short code in parentheses should be displayed in the output.

Extension (EX)
: The feature described is an extension to ISO POSIX-1 and ISO POSIX-2. Application writers may confidently make use of an extension as it will be supported on all XSI-conformant systems.

FIPS Requirements (FIPS)
: The **Federal Information Processing Standards (FIPS)** are a series of U.S. Government Procurement Standards managed and maintained on behalf of the U.S. Department of Commerce by the National Institute of Standards and Technology (NIST). The feature described has been restricted in order to align with the FIPS requirements.

Job Control Extension (JC)
: Job control is an optional feature in the operating system described by ISO POSIX-1, but it is supported by all XSI-conformant systems.

Obsolescent (OB)
: The feature described is obsolescent. It is fully portable to all current XSI-conformant systems, but may be withdrawn in future issues.

Output format incompletely specified (OF)
: The format of the output produced by the feature is not fully specified. It is therefore not possible to post-process this output in a consistent fashion. Typical problems include unknown length of strings and unspecified field delimiters.

Optional header (OH)
: This indicates that the marked header is not required on XSI-conformant systems.

Dependent on optional service in XSI (OP)
: Typical implementations depend on an optional service and the functionality affected need not be present if the optional service is not supported.

The behavior cannot be guaranteed to be consistent (PI)
: It is not possible to guarantee that the feature behaves in the same way on all XSI-conformant systems.

Realtime (RT)
: The feature described is part of the Realtime Feature Group.

Realtime Threads (RTT)
: The feature described is part of the Realtime Threads Feature Group.

Possibly unsupportable feature (UN)
: It need not be possible to implement the required functionality (as defined) on all XSI-conformant systems and the functionality need not be present.

Style Guide for Technical Publications

Part 2:

Tagging Document Source

The Open Group

Chapter 6

Using SGML

This chapter describes how to code document elements in SGML, using the DocBook DTD.

6.1 Introduction

Standardized General Markup Language (SGML) is a method of markup that identifies elements by *content*. For example, a parameter to a programming call is tagged as a <parameter> element, rather than being tagged for a format (such as *italics*).

The writer (and editor) should be concerned primarily with the correct tagging of information. Keep in mind that content and formatting are two different things in SGML.

The tags used for an SGML document instance are defined by a *document type definition* (DTD), which defines a set of *elements* and the context in which they can be used. Section- or paragraph-level elements are called *block elements*, while elements that can be used for words or phrases within paragraphs are called *inline elements*.

SGML elements may have *attributes* that provide additional information about the element.

The Open Group uses elements defined for the DocBook DTD, Version 2.4.1 (1995). DocBook is produced and maintained by HaL Computer Systems, Inc., O'Reilly & Associates, Inc., and ArborText, Inc.

SGML documents can also declare and use *entities*, which enable a document to identify a file, a passage of text, or other information that can be used repeatedly in the document. The following types of entities can be used:

- Parameter entities to control the inclusion or exclusion of information
- File entities
- Text entities that define standard wording for repetitive text elements

SGML file names are of the form **primaryname.sgm**.

Identifiers (or IDs) are values for the id attribute of a DocBook element.

6.2 Building Documents

This section describes how to use AdeptPublisher to create postscript files from SGML.

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

To follow.

6.3 SGML Coding

This section describes document components, starting with the front matter, followed by other components arranged in alphabetical order.

6.3.1 Front Matter

Title and Copyright Pages

TBD.

Preface

TBD.

Trademarks

TBD.

Referenced Documents

TBD.

6.3.2 Cautions

The <caution> tag is used to label information that cautions the user against potential damage to software or data. The <caution> tag should include a <title> tag, and one or more <para> tags.

A caution is tagged as follows:

```
<caution>
<title>Caution</title>
<para>Text of note.</para>
</caution>
```

6.3.3 Changebars

TBD.

6.3.4 Comments

TBD.

6.3.5 Cross-references

The <xref> tag is used to identify cross-references. The <xref> tag has a linkend attribute that must match the ID of the element to which it is referring. The <xref> tag is an empty tag; it does not contain any content or require an end tag.

6.3.6 Equations

DocBook does not include elements for tagging equations, but simple tagging can be done using the <superscript> tag for superscript text, the <subscript> tag for subscript text, and <replaceable> or <emphasis> tags.

The DocBook tagging for equations may not be supported for all display environments. For this reason, equations should be followed by a placeholder that can be used to include an alternative version of the equation. If an equation is given within a line of text, include the placeholder immediately following the equation (inside the paragraph). If an equation is in a paragraph-level element (such as <informalexample>), include the placeholder immediately after that element.

6.3.7 Examples

The <example> tag contains a formal, numbered example. The tag also has a cross-reference identifier, and includes a <title> tag which may be followed by a <para>. The <example> tag may contain a <programlisting> for examples of a program or script, or a <screen> tag that contains a <userinput> tag for examples of commands. The <userinput> tag contains the actual command to be typed by the user. Use the <computeroutput> tag within the <screen> tag to show the output of a command, or a prompt.

The <informalexample> tag contains an informal example that occurs in general text. The <informalexample> tag can contain a <programlisting> tag for programming examples or examples showing the output format for a utility, or a <screen> tag including a <userinput> tag for examples of user commands. Unlike the <example> tag, this element cannot have an identifier or title.

The <userinput> tag can be used for in-line examples of complete commands.

6.3.8 Extensions

Extensions are coded using the conformance attribute, which is part of the Effectivity group of attributes. These are attributes of most elements of Version 3.0 of the DocBook DTD. The conformance attribute is an extension to Version 2.4.1 of the DocBook DTD.

6.3.9 Footnotes

TBD.

6.3.10 Glossary

The <glosslist> tag contains a list of glossary terms. Within this tag, the <glossterm> and <glossdef> tags are used. The <glossterm> contains a glossary term. The <glossdef> tag contains the definition of a glossary term.

6.3.11 Graphics

TBD.

Figure Titles

TBD.

6.3.12 Headings

Chapters

The <refsect1> tag is used to identify a first-level section of text. This tag has a cross-reference identifier, and includes a <title> tag.

Appendixes

The <refsect1> tag is used to identify a first-level section of text. This tag has a cross-reference identifier, and includes a <title> tag.

Sections

The <refsect2> tag identifies a section heading used to define separate specific topics of discussion within a <refsect1> tag. This tag has a cross-reference identifier, and includes a <title> tag.

Subsections

The <refsect3> tag identifies a subsection heading used to define separate specific topics of discussion within a <refsect2> tag. This tag has a cross-reference identifier, and includes a <title> tag.

Lower Levels

TBD.

Unnumbered

TBD.

6.3.13 Index

The <indexterm> tag is used to identify an index entry. An <indexterm> tag contains at least a <primary> tag, and may include <secondary> and <tertiary> tags for secondary and tertiary index entries. In reference pages, include at least one index entry for each <refname> or <refdescriptor> in the NAME section.

6.3.14 Lists

Unordered Lists

The <itemizedlist> tag produces an unnumbered (unordered) list of elements, which may be words, symbols, or paragraphs of text. Within the <itemizedlist> tag, each list element is tagged with a <listitem> tag. The <listitem> tag must include a <para> tag.

This type of list is tagged as follows:

```
<itemizedlist>
<listitem><para>First item</para></listitem>
<listitem><para>Second item</para></listitem>
</itemizedlist>
```

Ordered Lists

The <orderedlist> tag produces a numbered or lettered list, depending on the attribute settings. Within the <orderedlist> tag, each list element is tagged with a <listitem> tag. The <listitem> tag must include a <para> tag.

This type of list of tagged as follows:

```
<orderedlist>
<listitem><para>First item</para></listitem>
<listitem><para>Second item</para></listitem>
</orderedlist>
```

Variable Lists

The <variablelist> tag produces a list where each list entry includes a term and a definition of that term. Within the <variablelist> tag, each list element contains <varlistentry> tags for each item, which must include a <term> tag for the item being defined (or described), and a <listitem> tag for the definition. The <listitem> tag must include a <para> tag.

This type of list of tagged as follows:

```
<variablelist>
<varlistentry>
<term>A</term>
<listitem><para>First item</para></listitem>
</varlistentry>
<varlistentry>
<term>B</term>
<listitem><para>Second item</para></listitem>
</varlistentry>
</variablelist>
```

Avoid use of the attribute termlength, which can be used to specify the width of the first column, since this is defined in the style sheet.

6.3.15 Notes

The <note> tag is used to label information that is a note. The <note> tag should include a <title> tag, and one or more <para> tags, or a list.

A note is tagged as follows:

```
<note>
<title>Note</title>
<para>Text of note.</para>
</note>
```

Invisible Notes

TBD.

6.3.16 Part Pages

TBD.

6.3.17 External References

The <citetitle> tag identifies a document title.

6.3.18 Special Characters

Use the DocBook <literal> tag for special characters. For example: ‘‘Use the backslash (<literal>\<literal>) to escape special characters.’’

In examples, use the DocBook <replaceable> tag for a name that represents a character.

Use the — character entity for em-dashes.

Use the – character entity for en-dashes.

Use the − character entity for negative numbers.

Use either keyboard characters or the character entities “ and ” for quotation marks. Do not use two apostrophes or the <quote> tag to represent a quotation mark.

Use hyphens instead of underscores in multiword variable names (including parameters, operands, and option-arguments), such as *target-file*. However, do not change the underscores in literal names, such as **wchar_t**.

6.3.19 System Items

Arguments

The <replaceable> tag is used for arguments.

Commands

The <command> tag is used for a utility name that appears within general text or a <cmdsynopsis> tag.

Constants

The <systemitem class="constant"> tag is used for system-defined constants, including symbolic limits and signals.

Data Structures

The <structname> tag is used for the names of data structures.

Environment Variables

The <systemitem class="environvar"> tag is used for environment variables. The <symbol> tag is used for external or global variables, such as *errno*.

Errors

The <systemitem role="errno"> tag is used for error values, such as [EDOM]. (This is not the global variable *errno*, but the value it holds.)

Fields

The <structfield> tag is used for the names of members or fields within a data structure.

Filenames

The <filename> tag is used for system file names, such as **/etc/passwd**, and directories.

Functions

The <function> tag is used, both in function synopses and in general text, for the names of system calls and library routines. When a user-defined function name is used as the parameter to another function, code it as <replaceable>.

Headers

The <filename class="headerfile"> tag is used for header file names.

Macros

The <systemitem class="macro"> tag is used for system macros and constant expressions. Macros with arguments should be coded as follows:

```
<function><systemitem class="macro">macro_name</systemitem></function>
```

Operands

The <replaceable> tag is used for operands.

Options

The <option> tag is used for utility options. Use one of the following attributes: role="dash", role="nodash", or role="plus".

Parameters

The <parameter> tag is used to tag the name of parameters, array names, and user-defined structure names.

Return Values

The <returnvalue> tag is used for literal return values. Do not tag descriptive phrases such as ‘‘nonzero’’.

6.3.20 Tables

<informaltable>

The <informaltable> tag is used to contain a table. This tag cannot include an identifier or title, and cannot be used as the target of a cross-reference. The <thead> and <tbody> tags contain <row> tags; a <row> tag contains <entry> tags for each column in the row.

The recommended style is to include a rule above and below the table (by using the frame="topbot" attribute on the <informaltable> tag), and a rule below the row that contains the table header (by using the rowsep="1" attribute). Further horizontal rules within the <tbody> are optional. This attribute can be used on the <tgroup>, <colspec>, <row>, or <entry> tags. To omit a rule following an element, use the rowsep="0" attribute. An <entry> tag can have text entered directly, or it can contain a <para> and other paragraph-level tags. Do not use vertical rules in tables.

Use relative, and not absolute, width specifications for columns.

A simple table is tagged as follows:

```
<informaltable frame="topbot">
<tgroup cols="2" colsep="0" rowsep="1">
<colspec colwidth="264*">
<colspec colwidth="264*">
<thead>
<row>
<entry align="left" valign="top">Heading 1</entry>
<entry align="left" valign="top">Heading 2</entry>
</row></thead>
<tbody>
<row rowsep="0">
<entry align="left" valign="top">Text 1 for column 1</entry>
<entry align="left" valign="top">
<para>First line of text for column 2</para>
<para>Second line of text for column 2</para>
</entry></row>
</tbody></tgroup></informaltable>
```

<table>

TBD.

Multi-page Tables

TBD.

Table Titles

TBD.

6.3.21 Text

A <para> tag is used for paragraphs. The <para> tag may contain general text, text entities, or inline tags (such as <replaceable>).

The <emphasis> tag is used for text that requires emphasis, such as *must*. It is *not* used for the introduction of special terms. Instead, use the <firstterm> tag.

The <literal> tag is used for any system-defined name that represents actual text typed into a program, or for any fixed name when a more specific tag is not available. Examples are flags, bits, user file names, program names, character classes, modifiers, and tokens. This tag is also used in-line for brief programming examples or fragments of commands.

The <phrase> tag is used to identify a section of text. When used with the class attribute, it associates text with a particular conformance value.

The <replaceable> tag is used for utility variable values when a more specific tag is not available.

6.3.22 Warnings

TBD.

6.4 Reference Pages

Reference pages use the following identifier for the DTD:

"-//The Open Group//DTD DocBook V2.4.1-Based Extension SUD V1.0//EN"

[TO BE AGREED]

The following DOCTYPE statement must be at the beginning of each file:

<!DOCTYPE DOCBOOK PUBLIC "-//The Open Group//DTD DocBook V2.4.1-Based Extension SUD V1.0//EN" [entity-definitions-specific-to-this-reference-page]

[TO BE AGREED]

Within the DOCTYPE statement, local entity declarations can be included. For reference pages, these entities include the following:

- Parameter entities to define marked sections for vendor placeholders.
- File entities for any vendor placeholders.

6.4.1 Filenaming

Each reference page is stored as a separate file. The filename is of the form *primaryname.sgm*.

The placeholder *primaryname* is used to represent the name of the item being documented on a reference page. This name is also used as the entry for the <refentrytitle> tag. The terms *primaryname* and *reference page name* are used interchangeably in this document.

To determine the *primaryname* for a reference page, do the following:

1. If more than one item is documented on a reference page, use the first name listed in the NAME section as the basis for the *primaryname*. The names of items are listed within the <refnamediv> tag in reference pages; the first name can be either a group name (<refdescriptor>) that represents *all* of the items on the page, or (if no group name is used) the first <refname>.
2. If the name of the first item includes any slash characters (/), leave those characters out to create the *primaryname*.

The reference page name *can* include underscores (_).

The file name *cannot* include slashes (/).

If a suffix is part of the *primaryname*, include it in the file name; for example, the file name for the reference page named **<systime.h>** is *systime.h.sgm*.

Note: If source files are stored on systems that do not support long file names or more than one suffix on a file name, be careful not to create duplicate file names when file names are truncated. In this situation, you must supply a clear mapping of abbreviated file names to full file names (and reference page names). For example, the file name for the reference page named **<systime.h>** might be *hsystime.sgm*.

For each item documented on a reference page that does not correspond to the *primaryname* for the page, you should create a shadow page. See ‘‘Shadow Pages’’.

6.4.2 Structure

The top-level DocBook tag used with reference pages is the <refentry> tag. All other tags are contained within that element.

The following template shows the overall structure of a reference page. This example includes all elements that are required for every type of reference page.

```
<refentry id="refentryid-divid">
<refmeta>metainformation</refmeta>
<refnamediv id="refentryid-divid-name">
<refname></refname>
<refpurpose></refpurpose>
</refnamediv>

[synopsis - if required]

<refsect1 id="refentryid-divid-desc">
<title></title>
</refsect1>

[additional sections]

</refentry>
```

Individual reference pages can be collected into a larger document by including them within the DocBook <reference> tag, which contains related reference entries. The <reference> tag can, in turn, be part of a higher-level document element such as a <chapter> or <book>.

For example, the following wrapper file shows one method for organizing reference pages into a manual. This organization assumes that each file to be included is declared as a file entity, and then referred to from within the wrapper file.

```
DocType declaration
<book>
<reference>
<title>Reference-Manual-Section-Name</title>
&file-name-1;
&file-name-2;
...
</reference>
...
</book>
```

Section headings within a reference page should be ordered consistently and are all coded using <refsect1>. The only sections which can include <refsect2> tags are the Synopsis and the Extended Description. The following table lists the standard sections of a reference page that are required or permitted.

Section Headings	Reference Manual Divisions						
	user	admn	sysc	file	devs	misc	exmp
NAME	M	M	M	M	M	M	M
SYNOPSIS	M	M	M	O	O	O	M
DESCRIPTION	M	M	M	M	M	M	M
OPTIONS	M	M	—	—	—	O	—
OPERANDS (if any)	M	M	—	—	—	O	—
PARAMETERS (if any)	—	—	M	—	—	O	—
EXTENDED DESCRIPTION	O	O	O	—	—	O	—
EXIT STATUS (if any)	M	M	—	—	—	O	—
RETURN VALUES	—	—	M	—	—	O	—
ERRORS	—	—	M	—	—	O	—
EXAMPLES	O	O	O	O	O	O	—
ENVIRONMENT VARIABLES (if any)	M	M	M	O	O	O	—
FILES (if any)	M	M	M	M	M	M	M
SEE ALSO (if any)	M	M	M	M	M	M	M

M The heading is mandatory (if there is source information for the topic).

O The heading is optional.

— The heading is not allowed.

6.4.3 Identifiers

Identifiers for reference pages have a standardized format. Identifiers are required for the following elements:

- <refentry>

  The *refentryid*, also referred to as the *reference page ID*, is based upon the *primaryname*. Like the *primaryname*, this identifier is based on the name of the item being documented on the page, but an identifier cannot contain slashes (/). In addition, identifiers cannot contain underscores (_). To form a valid identifier, replace underscores in a *primaryname* with hyphens (-) unless the underscore occurs at the beginning of the *primaryname*; at the beginning of a *primaryname*, replace an underscore with an uppercase M (which stands for ‘‘Macro’’).

  Examples: <refentry id="grep-user">, <refentry id="Mlongjmp-libr">

- <refnamediv>

  The *sect1id* represents an identifier for a first-level section of a reference page. First-level sections are those tagged with <refsect1>, <refnamediv>, and <refsynopsisdiv> tags.

  The *divid* represents a reference page division. The identifiers for reference page divisions are as follows:

  user User utilities. If the reference page describes a utility for all users.

  admn System administration utilities. If the reference page describes a utility for system or network administrators.

  sysc System calls. If the reference page describes a system call programming interface.

  libr Library routines. If the reference page describes a library routine programming interface.

file File formats. If the reference page describes a file format or a header file.

devs Special files. If the reference page describes a device special file or network-related driver.

misc Miscellaneous and descriptive topics. If the reference page describes a macro package or provides general information on a topic, such as regular expressions or environment variables.

exmp Examples and demos. If the reference page describes an online example or demo program.

Example: <refnamediv id="alarm-sysc-name">

- <refsynopsisdiv>

  As above.

- <refsect1>

  As above.

- <refsect2>

  The *sect2id* represents an identifier for a second-level section of a reference page. Second-level sections are those tagged with <refsect2> or <example> tags. (The <example> tag is not by definition a second-level tag, but it is only used as a second-level tag in reference pages.)

- <example>

  As above.

  Example: <refentry id="df-admn-exam-3">

  Note: Once assigned, do not change the example number, even if the examples are reordered.

The value entered for an identifier can be used in a link (to refer to a reference page from another reference page), or in a cross-reference (to refer to a section or example within the same reference page).

The identifier is optional for <refsect3> tags. Identifiers can also be used on other elements, but you should only create cross-references to sections and examples in reference pages, not to any other elements.

The following table shows the building blocks that can be used to construct identifiers. The component for which you are creating an ID is shown in the leftmost column. You can determine the ID by reading across the table. The *refentryid* is the first component of each ID, and additional ID components follow after a hyphen (-). If no value should be used in a particular column, the column shows the abbreviation N/A (not applicable).

Table 6-1 Structure of Identifiers

Component *Identifier*	**Reference Page ID** *refentryid*	**Division ID** *-divid*	**Section 1 ID** *-sect1id*	**Section 2 ID** *-sect2id*
Reference page	*refentryid*	*-divid*	N/A	N/A
NAME	*refentryid*	*-divid*	-name	N/A
SYNOPSIS	*refentryid*	*-divid*	-synp	N/A
DESCRIPTION	*refentryid*	*-divid*	-desc	N/A
Subsections	*refentryid*	*-divid*	-desc	-sect2id
OPTIONS	*refentryid*	*-divid*	-opts	N/A
OPERANDS	*refentryid*	*-divid*	-oper	N/A
PARAMETERS	*refentryid*	*-divid*	-parm	N/A
EXTENDED DESCRIPTION	*refentryid*	*-divid*	-exde	N/A
Subsections	*refentryid*	*-divid*	-exde	-sect2id
Standard Input	*refentryid*	*-divid*	-exde	-stdi
Standard Output	*refentryid*	*-divid*	-exde	-stdo
Standard Error	*refentryid*	*-divid*	-exde	-stde
Input Files	*refentryid*	*-divid*	-exde	-inpu
Output Files	*refentryid*	*-divid*	-exde	-outp
Consequences of Errors	*refentryid*	*-divid*	-exde	-erro
EXIT STATUS	*refentryid*	*-divid*	-exit	N/A
RETURN VALUES	*refentryid*	*-divid*	-rtrn	N/A
ERRORS	*refentryid*	*-divid*	-erro	N/A
EXAMPLES	*refentryid*	*-divid*	-exam	N/A
nth Example	*refentryid*	*-divid*	-exam	-n
ENVIRONMENT VARIABLES	*refentryid*	*-divid*	-envr	N/A
FILES	*refentryid*	*-divid*	-file	N/A
SEE ALSO	*refentryid*	*-divid*	-also	N/A

6.4.4 Metainformation

Metainformation is included for all reference pages. It provides information about the reference page itself, but the information does not appear in the body of the reference page. Some of the information is used in running heads when the reference page is formatted for printing. This information is placed inside the <refmeta> tag.

The following information is included:

- <refentrytitle>

  The title of the reference page (used in links to the page). This name is derived from the first name used on the NAME line of the reference page (either the <refname>, or the <refdescriptor> if that tag is present).

- <manvolnum>

  This tag contains an identifier for the reference page division.

- <refmiscinfo class="copyright">

  The copyright statement, typically using a text entity.

- <refmiscinfo class="date">

  The date of the reference page. The format of the date is day month year, without abbreviation; for example, 1 January 1996.

- <refmiscinfo class="sectdesc">

  The title of the reference manual division. The format of this entity is &divid-div;.

- <refmiscinfo class="conformance">

  Conformance information for the item documented.

- <indexterm>

  Index entries for the reference page. Include at least 1 entry for each <refname> or <refdescriptor>.

The following example shows a sample metainformation section:

```
<refmeta>
<refentrytitle>primaryname</refentrytitle>
<manvolnum>reference page division identifier</manvolnum>
<refmiscinfo class="copyright">copyright statement</refmiscinfo>
<refmiscinfo class="date">date</refmiscinfo>
<refmiscinfo class="sectdesc">reference page division title</refmiscinfo>
<refmiscinfo class="conformance">conformance-level</refmiscinfo>
<indexterm><primary>indexentry</primary></indexterm>
</refmeta>
```

6.4.5 Shadow Pages

Using the example of *writev*, the shadow page would consist of a file named **writev.sgm**, with the following contents (not including any fragment coding):

```
<!DOCTYPE DOCBOOK PUBLIC "-//The Open Group//DTD DocBook
V2.4.1-Based Extension SUD V1.0//EN" [
<!ENTITY write SYSTEM "./write.sgm">
]>
&write;
```

As shown in this example, the shadow file declares a file entity for the file to which it is pointing. The SYSTEM identifier of the form "./filename" indicates that the file being pointed to resides in the same directory as the shadow file. The file entity is then embedded in the file in place of the content that would normally be there.

6.4.6 Cross-references

You can only refer to another reference page as a whole, not to a specific section or example. Add the words ‘‘the ... reference page’’ around a reference to another reference page.

In general, when the reference page is discussing the action of a utility or function, use the <command> or <function> tag.

Use <link> when the reference page is discussing the information or documentation for a utility or function. Choose one or the other; do not combine a <link> with a <command> or <function> tag.

Do not put links to other reference pages within parentheses, because links include the division number of the reference page within parentheses (automatically generated).

The <link> tag contains the <citerefentry> tag, which contains a <refentrytitle> tag (the title of the reference page being referred to), and a <manvolnum> tag (an entity identifying the division for the reference page):

```
see the
<link linkend="grep-user">
<citerefentry>
<refentrytitle>grep</refentrytitle>
<manvolnum>&user;</manvolnum>
</citerefentry>
</link>
reference page.
```

6.4.7 Tables

In references pages, use the <informaltable> tag, not the <table> tag.

When it is necessary to refer to a table from elsewhere in a reference page, enclose the table and any related text in a separate section.

6.4.8 Reference Page Sections

NAME

This section is enclosed within the <refnamediv> tag. This tag does not specify a title style; the style sheet determines the convention. This tag contains an identifier.

The <refname> tag occurs with the <refnamediv> tag, and contains the name of the item described by the reference page. There must be at least 1 <refname> tag, but there may be more than one. If multiple <refname> values are included, use the first <refname> value for the identifier, and do not type separating characters between them. The contents of this tag must be a single word.

Group names for a set of functions are coded using the <refdescriptor> tag. The <refdescriptor> tag is used when a reference page name is not one of the names documented on the reference page. This tag must be a single word, and it can only be used in the NAME section (that is, not in general text). Within this tag, each item is tagged as a separate <refname>. Do not include separating characters between <refname>s. There can only be 1 <refdescriptor> tag, and it must occur before any <refname> tags. The first name is the reference page name.

Note: If multiple items are documented on a reference page, create a shadow page for each one.

The name used for an item documented on a reference page cannot include a slash (/). For this reason, this name may differ from the actual system name of the item. For example, the reference page for the **<sys/time.h>** header is **systime.h**.

The brief description of the item(s) is tagged using the <refpurpose> tag. Do not include separating characters between the <refname> and <refpurpose> tags; an em-dash will be generated automatically by the style sheet.

The NAME section is tagged as follows:

```
<refnamediv id="refentryid-divid-name">
[<refdescriptor>primaryname</refdescriptor>]
<refname>refentryname</refname>
<refpurpose>purposetext</refpurpose>
</refnamediv>
```

SYNOPSIS

The SYNOPSIS is coded using the <refsynopsisdiv> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Synopsis’’, the style of which will be determined by the style sheet.

Depending on the type of item being documented, the <refsynopsisdiv> tag contains <cmdsynopsis>, <funcsynopsis>, or <synopsis> tags.

- Utility Synopses

  Code utility synopses using the <cmdsynopsis> tag.

  Do not type the following special characters in a utility synopsis; they will be generated automatically by the style sheet:

  ```
  [ ] | { } ... - +
  ```

  The <cmdsynopsis> tag can be used more than once to show mutually-exclusive (or different) uses of a utility. Obsolescent forms of a utility should be placed in a <refsect2> tag with the title ‘‘Obsolescent Form.’’

  Code the utility name with the <command> tag. Following tags should be placed inside an <arg> or a <group> tag.

  The <arg> tag contains an argument to a utility. It may be used individually, or as a group of mutually-exclusive arguments using the <group> tag. A <group> tag must contain at least 2 <arg> tags. Within the <arg> or <group> tags, code options with the <option> tag, and code operands with the <replaceable> tag. For option-arguments which are literal values, use the <literal> tag; for option-arguments which are identified by a symbolic name (such as *target-file*), use the <replaceable> tag.

  The following attributes can be used with the <arg> and <group> tags:

  choice — Use to specify whether an argument is required. The following values can be used for this attribute:

    - opt — Indicates the argument is optional. This is the default.
    - plain — Indicates the argument is not optional.
    - req — Indicates the argument is required.

  rep — Use to indicate whether an argument is repeatable. The following values can be used for this attribute:

    - norepeat — Indicates the argument cannot be repeated. This is the default.
    - repeat — Indicates the argument can be repeated.

  Use a nested <arg> construction to achieve an ellipsis following an item inside a set of brackets. For example:

  ```
  <arg>
  <arg choice="plain" rep="repeat">
  <replaceable>file</replaceable>
  </arg>
  </arg>
  ```

  is used to produce [file ...].

  The <option> tag contains one or more option characters. If an option character is followed by an option-argument, you should type the space following options within the <option> tag.

Do not type a hyphen or other character as part of the <option> tag value.

The following values can be used for the <role> attribute on the <option> tag—it must be set explicitly because there is no default setting:

dash Produces a hyphen in front of the option.

nodash Produces nothing in front of the option. (In text, the <literal> tag can be used.)

plus Produces a plus.

A utility synopsis should be coded as follows:

```
<refsynopsisdiv id-"refentryid-divid-synp">
<title>Synopsis</title>
<cmdsynopsis>
<command>utility name</command>
<arg>
<arg>literal or nested</arg>
<option>option character</option>
<replaceable>operand</replaceable>
</arg>
</cmdsynopsis>
</refsynopsisdiv>
```

Variables that represent the user-supplied names of option-arguments and operands, when they consist of more than one word, should be hyphenated. (Do not use underscores.)

- Interface Synopses

Code programming interface (system calls and library routines) and macro synopses using the <funcsynopsis> tag. (The tagging style is the same for ANSI C and K&R C.) Note that the <funcsynopsis> tag can also be used in the Function Prototypes section of header reference pages.

Do not type the following special characters in a function or macro synopsis; they will be generated automatically by the style sheet:

```
( ) , ;
```

This tag can be used more than once to show multiple function synopses.

The following elements should appear in a <funcsynopsis> tag:

— <funcsynopsisinfo>

Contains one or more include statements for the headers required by the function. Type line breaks between each statement.

— <funcdef>

Contains the function definition, consisting of the return type of the function and the function name. Enclose the function name in a <function> tag. Type a space between the return type and the <function> tag. If the function name is a pointer, type an asterisk (*) just before the <function> tag (with no space).

Here are some examples:

```
<funcdef>int <function>fprintf</function></funcdef>
<funcdef>char *<function>foo</function></funcdef>
```

— <paramdef>

Contains the parameter definitions, including the parameter type and the parameter name.. Each definition is enclosed with a separate <paramdef> tag. The parameter names are enclosed in <parameter> tags. Give the type of the parameter first, followed by a space, and then the <parameter> tag. If the parameter is a pointer, type an asterisk (*) just before the <parameter> tag (with no space).

If a user-supplied function is provided as a parameter to a function, the user-supplied name is tagged as <parameter> within the <paramdef> tag.

Here are some examples:

```
<paramdef>int <parameter>flags</parameter></paramdef>
```

```
<paramdef>char *<parameter>string</parameter></paramdef>
```

— <funcparams>

This tag may be used within the <paramdef> tag. Contains the parameters of a user-supplied function that is supplied as a parameter to the function. Tag each parameter using the <parameter> tag. You must type commas between each set of parameters. (Example included in Appendix D on page 143.)

— <varargs>

If the parameters of a function can be supplied as a variable argument list, the <varargs> tag can be used in place of the <paramdef> tag. For example:

```
<funcdef>int <function>hmmm</function></funcdef>
<varargs>
```

— <void>

If a function does not have any parameters, the <void> tag can be used in place of the <paramdef> tag. For example:

```
<funcdef>int <function>big</function></funcdef>
<void>
```

If the function definition is followed by an external variable definition, enclose the definition in a <synopsis> tag outside the <funcsynopsis> tag, using a <literal> tag inside the <synopsis> tag.

An alternative method for coding multiple functions is to use a single <funcsynopsis> tag with multiple <funcprototype> tags within it. The tagging inside a <funcprototype> tag is the same as for the <funcsynopsis> tag.

A function or macro synopsis should be coded as follows:

```
<refsynopsisdiv id-"refentryid-divid-synp">
<title>Synopsis</title>
<funcsynopsis>
<funcsynopsisinfo>include statement(s)</funcsynopsisinfo>
<funcdef>return-type <function>function name</function></funcdef>
<paramdef>parameter-type <parameter>parameter name</parameter></paramdef>
</funcsynopsis>
</refsynopsisdiv>
```

- Other Synopses

Code header, sample program, and other synopses (such as external variables) using the <synopsis> tag.

The <synopsis> tag must include a <literal> tag which is used to format the content of the synopsis as literal text, and can include other tags such as <command>, <function>, and <parameter> as needed.

A <synopsis> should be coded as follows:

```
<refsynopsisdiv id-"refentryid-divid-synp">
<title>Synopsis</title>
<synopsis>
<literal>synopsis coding</literal>
</synopsis>
</refsynopsisdiv>
```

DESCRIPTION

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Description’’, the style of which will be determined by the style sheet. It has at least 1 <para> tag, and may include other appropriate in-line tags.

The description is coded as follows:

```
<refsect1 id="refentryid-divid-desc">
<title>Description</title>
<para>text</para>
</refsect1>
```

In descriptive reference pages, this section may include 2 or more <refsect2> tags, which have their own identifiers.

Vendor placeholders, if required, should be placed after the end of a paragraph.

If the synopsis includes symbolic names to represent user-supplied values which contain underscores, substitute the underscores for hyphens. (Do not change underscores to hyphens in literal names that must be types exactly as shown.)

OPTIONS

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Options’’, the style of which will be determined by the style sheet.

Begin this section with an introductory paragraph that should be entered using a text entity.

A <variablelist> tag is used to list the options. Each entry consists of a <term> tag containing the option character and its option-argument, if applicable. Use the <option> tag for the option character. Use the <replaceable> tag for the option-argument if it is a user-supplied value, or the <literal> tag if it is a literal value. (To show a space between the option and its option-argument, include a space following the option character within the <option> tag.) Use the role attribute to determine whether an option has a preceding hyphen—do not type one.

The <listitem> tag contains a <para> tag for the description.

If no options are included with a utility, use a text entity to indicate this fact and do not include any other text for the section.

The Options section is coded as follows:

```
<refsect1 id="refentryid-divid-opts">
<title>Options</title>
<para>introductory text</para>
<variablelist>
<varlistentry>
<term><option role="dash">option character</option>
<replaceable>option-argument</replaceable></term>
<listitem>
<para>option description</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

If the letter ‘‘ell’’ is used as an option, show the actual option as the list entry (–l), and begin the description with the text ‘‘(the letter ‘‘ell’’)’’.

If the number ‘‘one’’ is used as an option, show the actual option as the list entry (–1), and begin the description with the text ‘‘(the number ‘‘one’’)’’.

OPERANDS

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Operands’’, the style of which will be determined by the style sheet.

Begin this section with an introductory paragraph that should be entered using a text entity.

A <variablelist> tag is used to list the operands. Each entry consists of a <term> tag containing the operand. Use the <replaceable> tag for the operand if it is a user-supplied value, and the <literal> tag if it is a literal value.

The <listitem> tag contains a <para> tag for the description.

The Operands section is coded as follows:

```
<refsect1 id="refentryid-divid-oper">
<title>Operands</title>
<para>introductory text</para>
<variablelist>
<varlistentry>
<term><replaceable>operand</replaceable></term>
<listitem>
<para>operand description</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

PARAMETERS

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Parameters’’, the style of which will be determined by the style sheet.

Begin this section with an introductory paragraph that should be entered using a text entity.

A <variablelist> tag is used to list the parameters. Each entry consists of a <term> tag containing the parameter. Use the <parameter> tag for the parameter. Do not include the parameter’s type.

The <listitem> tag contains a <para> tag for the description.

The Parameters section is coded as follows:

```
<refsect1 id="refentryid-divid-parm">
<title>Parameters</title>
<para>introductory text</para>
<variablelist>
<varlistentry>
<term><parameter>parameter</parameter></term>
<listitem>
<para>parameter description</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

EXTENDED DESCRIPTION

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Extended Description’’, the style of which will be determined by the style sheet. It has at least 1 <para> tag, and may include other appropriate in-line tags. It may also use the <refsect2> tag to code subsections. This tag also contains an identifier, and the text of the title should be typed within this tag.

The extended description is coded as follows:

```
<refsect1 id="refentryid-divid-exde">
<title>Extended Description</title>
<para>text</para>
<refsect2 id="refentryid-divid-exde-sect2id">
<title>sect2title</title>
<para>text</para>
</refsect2>
</refsect1>
```

Vendor placeholders, if required, should be placed after the end of a paragraph.

Illustrative examples should be coded using the <informalexample> tag.

If the synopsis includes symbolic names to represent user-supplied values which contain underscores, substitute the underscores for hyphens. (Do not change underscores to hyphens in literal names that must be types exactly as shown.)

EXIT STATUS

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Exit Status’’, the style of which will be determined by the style sheet.

Begin this section with an introductory paragraph that should be entered using a text entity, .

A <variablelist> tag is used to list the exit status values. Each entry consists of a <term> tag containing the exit status value. Use the <returnvalue> tag for the exit status value, unless the value is a descriptive phrase, such as ‘‘Nonzero value.’’

The <listitem> tag contains a <para> tag for the description, which is the condition that produced the exit status value.

The Exit Status section is coded as follows:

```
<refsect1 id="refentryid-divid-exit">
<title>Exit Status</title>
<para>introductory text</para>
<variablelist>
<varlistentry>
<term><returnvalue>exit status value</returnvalue></term>
<listitem>
<para>exit status value description</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

RETURN VALUES

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Return Values’’, the style of which will be determined by the style sheet.

Begin this section with an introductory paragraph that should be entered using a text entity. This sentence indicates the name of the functions to which each set of return values applies.

A <variablelist> tag is used to list the return values (although this may not be the best approach when the return values are not literals). Each entry consists of a <term> tag containing the return value. Use the <returnvalue> tag for the return value, unless the value is a descriptive phrase, such as ‘‘Nonzero value.’’

The <listitem> tag contains a <para> tag for the description, which contains the condition that produced the return value. When the term *errno* is used, it is contained in the <symbol> tag.

The Return Values section is coded as follows:

```
<refsect1 id="refentryid-divid-rtrn">
<title>Return Values</title>
<para>introductory text</para>
<variablelist>
<varlistentry>
<term><returnvalue>return value</returnvalue></term>
<listitem>
<para>return value description</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

ERRORS

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Errors’’, the style of which will be determined by the style sheet.

Begin this section with an introductory paragraph that should be entered using a text entity. Errors may be organized into logical groupings, using separate lists; each list should have its own introductory paragraph.

A <variablelist> tag is used to list the errors. Each entry consists of a <term> tag containing the error code. Use the <systemitem role="errno"> tag for the error code. (Do not type brackets around error codes.)

The <listitem> tag contains a <para> tag for the condition description.

If no errors are defined, use a text entity to indicate this fact and do not include any other text for the section.

The Errors section is coded as follows:

```
<refsect1 id="refentryid-divid-erro">
<title>Errors</title>
<para>introductory text</para>
<variablelist>
<varlistentry>
<term><systemitem role="errno">error code</systemitem></term>
<listitem>
<para>error code description</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

EXAMPLES

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Examples’’, the style of which will be determined by the style sheet.

Each example is coded using the <example> tag. Each <example> tag has a title and an identifier. To create the <title> tag for <example>, type in the text.

Each example may be preceded by introductory text in a <para> tag.

Within the <example> tag, use the <programlisting> tag for C source file or shell script examples, or the <screen> and <userinput> and <computeroutput> tags for examples of using a utility.

Do not manually number the examples.

The examples section is coded as follows:

```
<refsect1 id="refentryid-divid-exam">
<title>Examples</title>
<example id="refentryid-divid-exam-1">
<title>example title</title>
<para>introductory text</para>
<programlisting>
example text
</programlisting
<screen>
<userinput>example text typed by the user</userinput>
<computeroutput>example text output by the system</computeroutput>
</screen>
</example>
</refsect1>
```

Vendor placeholders may be added for commands and utilities.

ENVIRONMENT VARIABLES

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Environment Variables’’, the style of which will be determined by the style sheet.

Begin this section with an introductory paragraph that should be entered using a text entity.

A <variablelist> tag is used to list the environment variables. Each entry consists of a <term> tag containing the environment variable. Use the <systemitem class="environvar"> tag for the environment variable.

The <listitem> tag contains a <para> tag for the description.

The Environment Variables section is coded as follows:

```
<refsect1 id="refentryid-divid-envr">
<title>Environment Variables</title>
<para>introductory text</para>
<variablelist>
<varlistentry>
<term><systemitem role="environvar">environment variable</systemitem></term>
<listitem>
<para>environment variable description</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

FILES

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘Files’’, the style of which will be determined by the style sheet.

Begin this section with an introductory paragraph that should be entered using a text entity.

A <variablelist> tag is used to list the parameters. Each entry consists of a <term> tag containing the full path name of the file. Use the <filename> tag for the file and path name. Do not include the parameter’s type.

The <listitem> tag contains a <para> tag for the description.

The Files section is coded as follows:

```
<refsect1 id="refentryid-divid-file">
<title>Files</title>
<para>introductory text</para>
<variablelist>
<varlistentry>
<term><filename>path name</filename></term>
<listitem>
<para>description</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

SEE ALSO

This section is contained within a <refsect1> tag. This tag contains an identifier. It also contains a <title> tag containing the text ‘‘See Also’’, the style of which will be determined by the style sheet.

Each group of references is grouped in a <para> tag.

For the list of references to other reference pages, the <link> tag contains the reference information. The linkend identifier is the reference page ID of another reference page. Use a comma to separate each link, except for the last link. Do not type a period after the last entry.

Within the <link>, the <manvolnum> and <refentrytitle> tags are contained within a <citerefentry> tag. The <manvolnum> tag contains the divid, as an entity reference. The <refentrytitle> tag contains the name of the reference page to which the link is referring.

For the list of references to other documents, the <citetitle> tag contains the reference information.

The See Also section is coded as follows:

```
2949          <refsect1 id="refentryid-divid-also">
2950          <title>See Also</title>
2951          <para>
2952          <link linkend="refentryid-divid">
2953          <citerefentry><refentrytitle>primaryname</refentrytitle>
2954          <manvolnum>&divid;</manvolnum></citerefentry></link>, ...
2955          </para>
2956          <para>
2957          <citetitle>full document title</citetitle>, ...
2958          </para>
2959          </refsect1>
```

Chapter 7

Using Troff

This chapter contains instructions for building a document using the Open Group build procedure, and how to code document elements in *troff*.

7.1 Introduction

The Open Group has a shell script called build that performs all the activities necessary for building a document.

The following items (available from The Open Group) should be installed on your system:

- The READ.ME file — this gives instructions for how to set up your troff environment.
- The DocTemplate directory — this contains templates for chapters, appendices, and front matter.
- The tools directory — this contains files required by the build procedure, including the macros definitions in a file called macros.xo. There are executable files in the Make directory which should be compiled (as described in the READ.ME file). This directory needs to be in your path.

Note that amendments are made as necessary to these files. The email group OGEdit is notified of such amendments.

7.2 Directory Structure

To create a new document, copy the DocTemplate into a new directory. This will give you the correct structure as follows:

```
<Document> directory /
    front matter files, build files, and Text directory /
        body text files and build files
```

Front Matter Files

Each front matter file should be a separate file called *something.r*. Use the following filenames:

```
title.r
preface.r
acknowl.r
trademar.r
refs.r
```

The contents of these files are described in Section 7.4.1 on page 99.

The order of the front matter files is defined by the contents of the file _vfiles (see **Build Files** on page 96).

Body Text Files

Each body text file should be a separate file called *something.r*. Use the following filenames:

chap*n*.r (for chapters)
apdx*x*.r (for appendixes)
item.mm (for individual reference pages)

All graphics files should use a filename extension that indicates the format of the file (for example, .eps, .pic).

The order of the body text is defined by the contents of the file _files (see. **Build Files**).

Build Files

The _strings.def file should be the same in the Document and Text directories. The contents of _strings.def are described in Section 7.4.23 on page 113.

The _vars file should be the same in the Document and Text directories. This file sets environment variables correctly for the build. You will need to make sure this file refers to the correct pathnames for your system. The following templates are provided in the tools directory — check each template for commented instructions:

vars_xo.bsh	Final camera-ready copy without shading.
vars_xs.bsh	Final camera-ready copy with shading.
vars_mcb.bsh	Draft with no changebars or manual changebars inserted where indicated by the `.mc |` and `.mc` macros.
vars_dft.bsh	Draft with changebars inserted automatically by comparison with a specific SCCS version. For this to work, you must have an SCCS delta of the files prelims.r and text.r (see Section 7.3 on page 97). You must edit your copy of the _vars file to give the required SCCS version number of these files for the *CBSID* variable.
vars_nro.bsh	ASCII output.

The _xref.inc file should be present in the Document and Text directories. The build procedure produces an updated version of this file.

The _vfiles file should be placed in the Document directory. It contains a list of the front matter files (without filename extensions). The complete list is:

title
contents
preface
acknowl
trademar
refs

The _files file should be placed in the Text directory. It contains a list of the body text files (without filename extensions).

7.3 Building Documents

From the Document directory, issue the command:

```
build
```

or:

```
build|& tee log
```

where *log* is the name of a file which stores output messages from the script.

The build procedure creates the following files automatically in the Document directory:

contents.r
prelims.r
front.ps
index.r
index.ps

and the following files in the Text directory:

text.r
text.ps

The following intermediate files are also created during the build:

,troff_err
,pg_first
,ridx
,cont
,ex
,fig
,tab
,xref

During the build procedure, messages are displayed indicating progress, any errors, and the number of pages written to the various postscript output files.

The build procedure should be repeated until all cross-references are resolved; that is, the _xref.inc file is the same in the Document and Text directories. Any unresolved cross-references will be labeled <REFERENCE_UNDEFINED>.

You can build single elements of a document as follows:

```
build chap1
```

This creates a postscript file of just Chapter 1 called chap1.ps. Note that this process uses the _xref.inc file that was created during the last complete build, and therefore may not be correct.

You can build the front matter only (front.ps) by issuing the following command in the Document directory:

```
build Prel
```

You can build the body text only (text.ps) by issuing the following command in the Text directory:

```
build SUB
```

Manual edits to contents.r and index.r are sometimes necessary.

For contents.r, edit the file, and then rebuild the front matter using `build Prel`.

For index.r, edit the file, add the correct page number as a second argument to `.Hi`, and then rebuild the index by using `build index`.

ASCII

To produce an ascii version of your document, copy the file vars_nro.bsh into the _vars file in the Document and Text directories, then issue the command:

```
buildnroff
```

The output files are prelims.asc and index.asc in the Document directory, and text.asc in the Text directory.

Note that .eps files will not be recognized in ascii, and that complex tables and graphics coded using pic may not format correctly.

No Index

To produce a document without an index, issue the command:

```
buildnoindex
```

A5

To produce an A5 size document, issue the command:

```
builda5
```

Simplified Build

Provided the contents.r and index.r files have been created, you can build a document from the command line.

The following files must be included (using `.so`) at the start of each file: macros.xo, _strings.def, and _xref.inc.

For troff, issue the command:

```
pic file.r|tbl|eqn|psfig| \
troff -rL27c -rOnc -rS10 -rW16.5c -rX1 -rZ1 -mm>outfile
```

For eroff, issue the command:

```
pic file.r|tbl|eqn|psfig| \
eroff -p -rL27c -rOnc -rS10 -rW16.5c -rX1 -rZ2 -mm>outfile
```

where *n* is the page offset required by your printer, probably in the region of 1.5 centimetres.

For nroff, issue the command:

```
pic file.r|tbl|eqn nroff -Tascii -e -rX1 -rZ3 -mm>outfile
```

To create a draft with line numbers use `-rX0` instead of `-rX1`, or omit the `-rX` argument.

The printable files produced differ slightly from those created by the *build* environment.

For a document that does not require all the preprocessors, each command can be modified.

psdraft

The *psdraft* utility can be used to add a greyscale watermark to a postscript file. This can be useful for identifying drafts (in addition to line numbers) or specific versions. The command:

```
psdraft -s "<string>" <file> > newfile.ps
```

adds the string in a diagonal line across every page of the file. The output can also be directed straight to a printer. The string is limited to one diagonal line of text on the printed page.

7.4 Troff Coding

This section describes document components, starting with the front matter, followed by other components arranged in alphabetical order.

The text formatter *troff* uses commands that start with a period or apostrophe, for example:

```
.P
```

Each command starts at the beginning of a line. If you start a text line with either a period or an apostrophe, the formatter treats it as a command. Frequently this results in the line being ignored, because the text is not a valid command. If you need to start a text line with a period or an apostrophe, precede it with:

```
\&
```

which represents a zero-width non-printing character.

In general, avoid starting a normal text line with a period or single quote.

troff source files should not include any blank lines—space is defined in the macros.

All raw *troff* commands consist of two lower-case characters.

To simplify coding, macros are available, which combine several *troff* commands into a meaningful formatting operation. As far as possible the macros are from the *mm* macro package. A few are written specifically for The Open Group's requirements.

Note: Do not write new macros for an Open Group document.

Do not use nested `.so` commands in document source.

Each file must end with the `.eF [e]` macro.

7.4.1 Front Matter

Title and Copyright Pages

The coding that starts the title page is as follows:

```
.\" Copyright 1997, The Open Group
.ds SI %Z% %I% %E%
.tL "June 1997" Innn
.eF e
```

The coding has the following meanings and uses:

- The first line is the copyright notice. Whenever you change a file, ensure that the copyright notice contains the current year.

- The `.ds SI` line contains SCCS keywords, which should not be changed.
- The `.tL` line contains the copyright date and document number. The `.tL` macro also takes optional third and fourth arguments. The third argument is the ISBN. The fourth argument—a `.P`—allows you to add additional copyright text. Place the additional text between the `.tL` and `.eF` macros.
- The `.eF` macro specifies the end of the file. The `e` argument specifies that the file should end on an even page. The end of every file must have the `.eF` macro; it can be used with no arguments if the last page need not be even.

The copyright page text is generated as part of the `.tL` macro.

Preface

The coding that starts the Preface is as follows:

```
.\" Copyright 1997, The Open Group
.ds SI %Z% %I% %E%
.Ho Preface
.so <pathname>/prefintro.r
.HU "This Document"
```

The `.Ho` macro creates a front matter heading that must start on an odd page. It takes an optional second argument which can be used to specify the page number.

The `.so <pathname>/prefintro.r` instruction includes the standard preface text. Amend this line or comment it out as required to suit the environment of your own system.

Further sections of the Preface should be placed under `.HU`s.

Trademarks

The coding that starts the Trademarks section is as follows:

```
.\" Copyright 1997, The Open Group
.ds SI %Z% %I% %E%
.Hp Trademarks
```

The `.Hp` macro creates a front matter heading that starts on an odd or even page. It takes an optional second argument which can be used to specify the page number.

Referenced Documents

The coding that starts the Referenced Documents section is as follows:

```
.\" Copyright 1997, The Open Group
.ds SI %Z% %I% %E%
.Hp "Referenced Documents"
The following documents are referenced in this <document type>:
.VL 4
.LI "Short Name"
.br
Full bibliographic details, including ISBN and Publisher.
.LE
.eF e
```

Each document is a list item.

7.4.2 Cautions

A caution should be coded as follows:

```
.As
Text of caution
.Ae
```

Several cautions should be coded as follows:

```
.As s
.AL
.LI
Text of first caution
.LI
Text of second caution
     .
     .
     .
.LE
.Ae
```

7.4.3 Changebars

Use the macro `.mc |` to start a changebar; use `.mc` to stop a changebar. The identification of changed text must be done carefully to avoid incorrect output. The `.mc` macro does not break the current line, so it is possible to turn the changebar on and off again within one line, resulting in no mark on the printout. Make sure you include enough text between the macros to ensure that you reach the end of a line before switching the changebar off. At the end of a paragraph, place the `.mc` macro after the next `.P` macro.

To mark changes in displays, tables and footnotes, make sure that the changebars are turned on and off again within the boundaries of the display, table or footnote. In a table you cannot mark part of a text block formed by the `T{` and `T}` delimiters; you must mark the whole block.

Changebars can be inserted automatically using a special preprocessor in the build procedure.

7.4.4 Comments

The comment command is `.\"`. Any text following a comment command is not included in the output file. For example, each file should start with a line similar to the following but with the correct copyright date:

```
.\" Copyright 1997, The Open Group
```

7.4.5 Cross-references

When referring to other parts of the same document use the cross-referencing macros.

To identify an item to which you refer elsewhere, use the `.xR` macro. To call a cross-reference, use the `.cX` macro.

The `.xR` takes two arguments. The first identifies the type of item as follows:

`.xR 1`	Chapter or Appendix (first-level heading)
`.xR 2`	Section (second-level heading)
`.xR 3`	Subsection (third-level heading)
`.xR 4`	Subsection (fourth-level heading)
`.xR 5`	Unnumbered heading
`.xR 6`	Figure
`.xR 7`	Table
`.xR 8`	Example
`.xR 9`	Front matter section
`.xR 10`	Reference page
`.xR 11`	Glossary entry

The second argument is a short string that identifies the item, for example:

```
.H 2 Changebars
.xR 2 chbar
```

The cross-reference macro must be placed after the item heading you need to identify, most usefully immediately below. To refer to an item identified as `chbar`, use the `.cX` macro as follows:

```
.cX chbar
```

The `.cX` macro takes an optional second argument which is printed immediately following the reference text. It is normally used for punctuation, for example:

```
For details of how to use changebars see
.cX chbar .
```

To suppress the page number, specify `1` as the third argument to `.cX`.

7.4.6 Displays

A display is used to keep a block of information together, unless it is too long to fit on one page.

The content is kept together on a page; if it does not fit below existing material, it is placed on the next page. This is useful for items like tables.

The `.DS` and `.DE` macros define the start and end of a display.

The content of a display has a small space above and below. The content is normally not adjusted or filled. It is possible to add an `F` second argument which causes text to be filled in the normal way. If text blocks are present in a table within a display, you must use the `F` second argument.

Displays can be indented using `I` as the first argument to `.DS` as follows:

```
.DS I
```

If the display is very long, try to place `.DS` and `.DE` pairs around logical blocks of material.

Note that displays operate in a different *troff* environment from the body of the text. Problems can usually be traced back to a previous display.

7.4.7 Examples

If numbered examples are required (for example, because a reference is made to the example), use the `.EX` macro as follows:

```
.EX "Example Title"
```

Place the `.EX` instruction immediately *before* the example. Do not place the `.EX` macro in a display, because the page number in the contents is incorrect if you do.

Code examples should be placed between the `.Cs` and `.Ce` macros. Examples can be indented using `I` as a first argument, `.Cs I`.

Code examples in IDL should be placed between the `.Is [I]` and `.Ie` macros.

Code examples in PIDL should be placed between the `.Ps [I]` and `.Pe` macros.

7.4.8 Extensions

Extensions are coded using the shading macros.

To start shading, use the `.sS` macro with one of the arguments described in the list below. (The full definition of each term is contained in Section 2.13 on page 12.)

eX Extension.

eF FIPS Requirements.

eJ Job Control Extension.

oB Obsolescent.

oF Output format incompletely specified.

oH Optional header.

oP Dependent on optional service in XSI.

pI The behavior cannot be guaranteed to be consistent.

rT Realtime.

tT Realtime Threads.

uN Possibly unsupportable feature.

Index entries are created automatically by the `.sS` macro.

You must not use the `.sS` macro in a display.

To end the shading use the macro `.sE`.

It is sometimes necessary to switch shading on or off part-way through a line, in which case you should use in-line coding. To switch shading on use `\*!`. To switch shading off use `\*?`.

7.4.9 External References

When referring to another document use a string. For example, the following string definitions:

```
.ds ZA ANSI COBOL standard
.ds Z3 \f3XPG3\fP guide
```

should be used in text as follows:

```
see the \*(ZA
see the \*(Z3
```

You can use a string to reference a section in another document, although this should be avoided unless absolutely necessary, for example:

```
.ds ZY \f3SQL CLI\fP specification, Chapter 5, Diagnostics
```

URLs can be added using the `.Ur` macro, so that on conversion to HTML each reference to an external document can be shown as a link.

This macro takes three arguments. The first argument is the name of an external document taken from _strings.def, placed inside double quotes. The second argument is the actual URL, also taken from _strings.def. This argument is ignored in *troff* output. The third argument is used for punctuation, and can be omitted.

For example, the following string definitions from _strings.def:

```
.ds ZO Document Name
.ds Zo http://www.sitename.docname
```

would be used with the `.Ur` macro as follows:

```
.Ur "\*(ZO" \*(Zo
```

7.4.10 Fonts

In *troff*, you can refer to a font either explicitly by name, or numerically by its typesetter position. In Open Group documents, you must specify the font by its number, for these reasons:

- Some sites may not have the preferred fonts.
- The Open Group may change its selection of fonts.
- The Open Group may distribute documentation to member companies, in which case, each member may elect to publish it using fonts consistent with its own house style for documentation.

Open Group documents use the following fonts:

Font Number	Weight	In-Line Coding	Start-of-Line Coding
1	Palatino Roman	`\f1`	`.R`
2	Palatino italic	`\f2`	`.I`
3	Palatino boldface	`\f3`	`.B`
4	Palatino bold italic	not used	
5	Courier Roman	`\f5`	
6	Courier italic	`\f6`	
7	Courier bold	`\f7`	
8	Helvetica	`\f8`	
9	Helvetica bold	`\f9`	

Fonts can be specified with in-line commands of the form:

```
\f3bold font\fP
```

Never use `\fB`, `\fI`, or `\fR`. These are absolute references to Times Bold, Times Italic, and Times Roman, respectively. Use only the generic forms shown in the above table. You can use the *mm* macros, `.R`, `.I`, and `.B` because they call fonts by position, not by name.

In general, documents should select the previous font (font P) after shifting to one of the other fonts, rather than specifying the previous font by number. This ensures that if the text is cut and pasted to another location where the surrounding font is different, it still selects the correct font. The previous font is selected by using the `.ft P` request, or in-line by specifying `\fP`. The formatter's memory of the previous font is only 1 deep. This means that forgetting to insert a `\fP` directive affects the font of all the text immediately following up to the next heading.

Note: A missing `\fP` directive in a display or footer will continue to affect displays and footers for the rest of the document.

The rules for font usage and typography are the same in chapters and in reference pages; in regular text and in tables.

7.4.11 Footnotes

To include a footnote use the automatically numbered footnotes provided by the *mm* macros. To do this type the three characters `\*F` immediately after the text requiring a footnote. Then use the macros `.FS` and `.FE` to enclose the footnote text.

It is sometimes useful to mark several items with the same footnote indicator. In this case, you should use a symbol such as an asterisk or dagger to mark the items and then define the symbol as follows:

```
.FS *
Text which defines use of the asterisk.
.FE
```

Automatic footnotes do not work within displays or tables; the footnote number is incremented by 2. Either use a symbol or place the `.FS` and `.FE` outside the display or table.

7.4.12 Glossary

The Glossary is coded as follows:

```
.\" Copyright 1997, The Open Group
.ds SI %Z% %I% %E%
.Hi Glossary
.gS
.gT "term"
Definition of term.
    .
    .
    .
.gE
.eF e
```

`.gT` automatically inserts a main entry into the index.

7.4.13 Graphics

The *pic* preprocessor is used to produce line diagrams. You will need to refer to the *pic* documentation in your *troff* manual for details.

For more complex artwork, encapsulated PostScript illustrations can be built into the book.

Do not use crude, typewriter-style (constant-width) illustrations. If the document author cannot supply source material, ask them to supply a clear drawing which will enable an editor to recreate the figure.

Graphics Coded in pic

pic code is any line between the `.PS` and `.PE` macros.

Keep your *pic* code simple and observe the following rules:

- Use the labeling facility of *pic* to assign meaningful labels to the boxes or other objects in your picture. Refer to positions in the picture in terms of these labels, rather than with absolute coordinates.
- Comment the *pic* source.
- Use tools that produce *pic* code with caution as their output is excessively precise. The absolute coordinates make it hard to edit.

Note that the `.PS` and `.PE` pair define their troff environment differently from that of the preceding text. Unexpected results can usually be traced back to the environment setting inside the preceding `.PS`/`.PE` pair.

PostScript

The standard *psfig* preprocessor is used to include encapsulated PostScript (.EPS) files. These files can be generated by hand or by a drawing package.

If you generate EPS files by hand, keep them clear and simple and include plenty of comments. You should also use the PostScript structuring conventions.

If you use a drawing tool to generate EPS, you should supply the EPS files with the rest of the *troff* source. In addition, you should supply:

- Copies of the files containing your drawing tool's internal format for the diagrams
- Information stating the name and version of the application that created the diagrams

In addition to supplying the EPS files, you must indicate where in the text they should print. This is done by directives to the *psfig* preprocessor between the commands `.F+` and `.F-`. For example:

```
.F+
figure ./Figures/filename width 10c
.F-
```

This includes the EPS file located in the directory **Figures** into the file at this point, and adjusts the size of the figure so that its width is 10cm and its height is scaled proportionately.

Notes:

1. Some applications have a PostScript prologue which is downloaded to the printer at the start of the day, rather than including it with every print job. Ensure that you supply such a prologue with your EPS files.
2. When you create the file, specify EPS and not PostScript.
3. Specify the application's version of the fonts rather than the printer's version, if you have the choice.
4. Do not include a bit-mapped image header, if you have the option.
5. Specify that labels in diagrams be set in Palatino; if this is not available, choose the font most similar to it.

Figure Titles

Illustrations should normally have titles. Use the `.FG` macro to create numbered figure titles, for example:

```
.FG "Figure Title"
```

Place the `.FG` instruction immediately *after* the figure. Do not place the `.FG` macro in a display as this will result in an incorrect page reference in the contents and automatic cross-references.

7.4.14 Headings

Headings longer than one word should be enclosed in quotes.

Chapters

The coding that starts a chapter is as follows:

```
.\" Copyright 1997, The Open Group
.ds SI %Z% %I% %E%
.H 1 "Chapter Title"
```

The `.H 1` line contains the chapter title. Since this string will be used in other places, do not include an apostrophe in the title since it may cause formatting errors (for example, in a running header). Instead, consider using \(mt or similar.

The `.H 1` macro takes an optional third argument which can be used to specify the page number.

The first chapter of a book must have the `.fC` macro before the `.H 1` macro as follows:

```
.\" Copyright 1997, The Open Group
.ds SI %Z% %I% %E%
.fC
.H 1 "Chapter Title"
```

The `.fC` macro sets up the number registers for chapter headings and specifies page 1. It can be used with an optional argument at the start of any chapter to allow processing of the individual element. For example, if this chapter were to be built on its own, the addition of `.fC 4` before the `.H 1` line would set the number registers correctly for Chapter 4.

Appendixes

The start of an appendix is coded in nearly the same way as a chapter. Since appendices have letters instead of chapter numbers, the first appendix must include `.fA` before the `.H 1` line. As with `.fC`, an optional argument can be used with `.fA` in appendices other than the first. This is how an Appendix C would start if it had to be built on its own.

```
.\" Copyright 1997, The Open Group
.ds SI %Z% %I% %E%
.fA 3
.H 1 "Appendix Title" 69
```

This generates the third appendix (C) starting on page 69.

Sections

Code section headings (second-level headings) using the `.H 2` macro, as follows:

```
.H 2 "Section Title"
```

Since this string will be used in other places (the `XC` string), do not include an apostrophe in the title since it may cause formatting errors (for example, in a running header). Instead, consider using \(mt or similar.

By default, all second-level headings start a new page. If the subject matter is such that only first-level headings (`.H 1`) should start a new page, insert the line `.nr Ej 1` after the `.H 1` line.

An optional third argument `S` keeps a short section on the same page.

Subsections

Code subsection headings (third-level headings) using the `.H 3` macro, as follows:

```
.H 3 "Subsection Title"
```

An optional third argument `S` keeps a short section on the same page.

Lower Levels

Use of `.H 4` is discouraged, and use of lower levels is disallowed.

Unnumbered

If you need headings within a subsection, use the `.HU` macro to give unnumbered headings as follows:

```
.HU "Unnumbered Section Title"
```

An optional third argument `S` keeps a short section on the same page.

7.4.15 Index

Index entries must be located in the text immediately below the first line of a subject. An index entry is provided automatically for:

- each reference page
- each glossary entry
- each shaded extension

Do not put index entries in the front matter, because they do not appear in the index.

To code index entries use the `.iX` macro as shown in the following examples:

```
.iX "main entry"            (single entry only)
.iX "main entry" "" 1       (single main entry)
.iX "main" "secondary"      (single and secondary entry)
.iX "main" "secondary" 1    (single main and secondary entry)
```

Do not use any font changes in index entries.

Do not use quotes in index entries as they cause formatting errors.

Some macros automatically generate main index entries — any other main index entry to the same term will be treated as a single entry.

7.4.16 Lists

Unordered lists can be coded either with bullets (`.BL`) or with dashes (`.DL`). Unordered lists use `.LI` to introduce a list element; the text follows on the next line. Use bullets for all top-level unordered lists, and dashes for all sublists. Avoid any further nesting of lists.

Ordered (or numbered) lists are coded with `.AL`.

Variable-item lists are coded with `.VL` *n*. The parameter *n* is the text indentation, as a number of ens. For short keywords, choose a multiple of 4 that is larger than the longest item. For long keywords, use `.VL 4` and add a `.br` to separate the keyword from the following text. The term is an argument to the `.LI`. For example:

```
.VL 8
.LI "<list-item>"
```

Note that if the short keyword consists of more than one item, the automatic justification may produce an uneven result. You should therefore join them with a fixed space; for example, `object\ 1`.

Do not indent variable lists by using the optional 3rd argument to the `.VL` command.

All lists must have `.LE` at the end followed by a `.P`.

7.4.17 Notes

A note should be coded as follows:

```
.Ns
Text of note
.Ne
```

Several notes should be coded as follows:

```
.Ns s
.AL
.LI
Text of first note
.LI
Text of second note
     .
     .
     .
.LE
.Ne
```

Invisible Notes

You can insert notes to reviewers which only appear when the document is processed as a draft. To do this, use the `.iN` macro before the text and the `.sA` macro afterwards.

7.4.18 Pagination

In general, avoid hard-coding page breaks in source since the output media and use of fonts will vary. Conditional instructions are preferred.

Second-level headings typically start on a new page.

If the chapter is short (say, 10 pages or less in run-on form) and all second-level sections comprise less than two pages (so that every time you turn a page, you see a new section head), all sections may be run on, since there is no need to aid the reader's navigation. However, if one or more sections comprise more than two pages, page breaks at second-level headings become the default (unless the consideration below applies).

In a chapter of any length, if the whole of a second-level section can be contained on the current page, it should be. Otherwise start the section on a new page.

By inserting `.nr Ej 1` and `.nr Ej 2` at appropriate places in the file, you can arrange for short sections to be run on and for long sections to start a new page.

Some required space is built into each section or subsection heading, so that they cannot start near the bottom of the page. If you have a very short section or subsection and you want to force it to print near the bottom of a page, use `S` as a third argument to the `.H 2`, `.H 3`, or `.HU` macro, for example:

```
.H 2 "Short Section" S
```

There are two ways of creating other page breaks.

A forced page break breaks regardless of the space left on the page. Use the `.SK` macro. A first argument skips `$1` pages.

If you force a page break immediately before a second-level heading, you must specify the running header (the `XC` string) before the page break. The running header text must be the same as the text of the following second-level heading.

Conditional page breaks depend on the space left. Use the *troff* `.ne` (need) request which takes the number of lines as an argument:

```
.br
.ne 5
```

This instructs *troff* to break the page if fewer than 5 lines remain. This request is the most useful form for headings or list items, where a number of lines must be kept on the same page, but the space available on the current page cannot be predicted.

For widows and orphans, if a multi-line paragraph or list element is split across two pages, there should be at least two lines on each page.

7.4.19 Part Pages

Part pages for large books can be produced using the `.Tl` macro. When used with no arguments it produces a part page containing a page number derived from the build. To force a page number, specify it as the first argument. A second argument is also available; when this is set to 1 the macro produces arabic page numbers.

A part page is likely to require a revised footer. This is achieved by including new string definitions. Note that the XQ string must be explicitly defined and that the XV is needed to produce the correct result in the contents.

Code a part page as follows:

```
.\" Copyright 1997, The Open Group
.ds SI %Z% %I% %E%
.ds XO Series Title
.ds XQ Style Guide for Technical Publications
.ds XP Part 2:
.ds Xp Title of Part
.ds XV Part\t2
.Tl
.eF e
```

7.4.20 Point Size

The character size can be specified with in-line commands of the form:

```
\s+1larger type\s0
```

Try to avoid this markup except where absolutely necessary; for example, code fragments that are too wide for the page.

7.4.21 Reference Pages

The *troff* source is often in separate files, one for each reference page. This is not essential; short reference pages embedded within a chapter are best coded as part of the chapter.

Each reference page starts with the macro:

```
.mS name
```

This macro does the following:

- It redefines the section header string so that the *name* appears in the page header.
- It forces the reference page to a new page.
- It places an entry in the contents.
- It places an entry in the index. (This is treated as a main entry and so the page number will appear in bold.)
- It specifies a page number if the optional third argument is used.
- It specifies a string to be placed in the center of the header if the optional fourth argument is used.

For C-language functions with parameters, specify a second argument as follows:

```
.mS function (\|)
```

For COBOL functions, specify a second argument as follows:

```
.mS function c
```

For C-language headers specify a second argument as follows:

```
.mS <header.h> h
```

For macros without parameters or utilities, do not use a second argument:

```
.mS macro
```

For data types specify a second argument as follows:

```
.mS data_type d
```

For X Windows widgets specify a second argument as follows:

```
.mS widget w
```

Each subsection of the reference page begins with the `.mH` macro; for example:

```
.mH NAME
text
.mH SYNOPSIS
      .
      .
      .
```

If the section head is more than one word, you must put the argument to `.mH` in double quotes. Start the text of the section on the following line. Do not use a `.P` macro after `.mH`.

Use `.yS` before the text of the SYNOPSIS, and `.yE` after. Between these two macros, adjusting and filling is disabled. The `.yS` macro displays font 5 (courier).

The `.mE` macro must be invoked at the end of each reference page.

Use `.HU` headings for any other headings required in reference pages. Do not use numbered headings.

7.4.22 Special Characters

To make a backslash appear in the document, code it as `\e`.

Do not use `\\` as a command to produce a single backslash character, because the two-to-one translation is repeated for each stage of processing. For example, if the text in question appears within a macro or is operated on by a preprocessor such as *tbl*, four backslashes would be required.

7.4.23 Strings

All strings used in a document are defined in the _strings.def file. (The `.ds` command specifies a string.) This file is coded as follows:

```
.\" Copyright 1997, The Open Group
.ds SI %Z% %I% %E%
.ds XO <series title>
.ds XP <document title>
.ds Xp <document subtitle, if required>
.ds XQ \*(XO (1997)
.ds XZ \*(DT
.\" the DT string puts a date in the footer for drafts.
.\" If you are using SCCS you can define the XZ string
.\" to be the SI string.
.\"
.\" Add all your string definitions for the book,
.\" using Z as the first character.
.\"
.ds Za <local string>
```

You may define your own strings for a document, using any two-character string starting with a Z or z. It is useful to define strings for frequently used items such as references to other documents, because it ensures consistency. It is important to restrict local strings to those starting with Z or z because formatting errors occur if a string is defined with the same name as an existing macro.

The following strings are used in the build procedure:

lM Used internally to define the left margin.

xC Used internally to define Chapter or Appendix string.

XC Used internally to define section part of running header.

Xm Used internally to define the chapter number or appendix letter.

XR Used internally in footer.

XT Used internally to define title part of running header.

7.4.24 System Items

Use the following macros for system items:

`.Ar` Argument

`.Er` Error

`.Fn` Function

`.Hd` Header

`.In` Function (IDL)

`.LM` Limit

`.Pn` Function (PIDL)

`.tK` Keyboard legend

These macros take 2 arguments. The first argument is the name of the system item; the second argument is used for punctuation.

7.4.25 Tables

The *tbl* preprocessor is used to enable the use of coded tables within a *troff* source file. You will need to refer to the *tbl* documentation in your *troff* manual for details.

Coding for a typical table is as follows:

```
.TS
box center tab(@);
cB | cB | cB | cB      (This line controls the heading of the table)
l  | l  | l  | l.      (This line controls the body of the table)
Column 1@Column 2@Column 3@Column 4
_                      (This line contains an underscore in column 1)
Text@Text@Text@Text
.TE
```

Follow these rules for tables:

- You can use absolute units of measurement in the table format (inches, centimetres, points, picas, basic units, or ens), for example:

  ```
  lw(12c) cw(2c).
  ```

- If the width of a table is 80% or more of the width of the column, justify the table using the `expand` keyword. Otherwise, center tables between the left and right margin by using the `center` keyword.

  If you use the `expand` option, you must first set the line length to the width of the text using `.ll -\*(lM`. Reset the line length after the table with `.ll +\*(lM`. Alternatively, place the table within a left-aligned display, because a display uses the correct line length.

- The column headings should be centered in bold font (font 3). Do not specify any font for the body of the table, unless some special rule applies.

- Specify a tab character other than tab, ^, or &. Alternatives include @ and !.

- Box tables with the `box` option of *tbl*. Avoid using the `allbox` option.

- Draw a line between the header and the body of the table using the underscore character in the body of the table. Note that this does not count as a line in the structure of the table.

- Vertical rules between columns may be used at the author's discretion. This is done by inserting the | (bar) character in the format line.
- Horizontal lines in the body of the table are discouraged, except to separate logical sections within a table.

Setting Tables within Displays

By placing a table within a display (`.DS` and `.DE`), it will ensure that it is kept as one unit.

Multi-page Tables

If a table is longer than a single page, use `.TS H` and add a `.TH` macro after any heading or at the start of the table text if there is no heading. This ensures that the table set-up information is repeated at the top of each page of the table.

Note: You cannot put a multi-page table in a display.

Table Titles

If a numbered table is required (for example, if you need to refer to it), use the `.TB` macro as follows:

```
.TB "Table Title"
```

Place this instruction immediately *before* the `.TS` macro so that the title appears above the table. Subsequent `.TB` instructions in the same file increment the table number. Do not place the `.TB` macro in a display as this will result in an incorrect page reference in the contents and automatic cross-references.

7.4.26 Tabs

Tab stops are set by using the `.ta` command:

Note that *tbl* in a display resets tab stops for the following display.

7.4.27 Text

Normal text is broken into paragraphs with the `.P` macro.

7.4.28 Warnings

A warning should be coded as follows:

```
.Ws
Text of warning
.We
```

Several warnings should be coded as follows:

```
.Ws s
.AL
.LI
Text of first warning
.LI
Text of second warning
     .
     .
     .
.LE
.We
```

7.5 Troff Coding for Popular Titles

The following changes should be applied to the troff source files to produce postscript files which use the popular title style.

1. In the _strings.def file, include the macros file tools/consort.mac using `.so`.
2. In title.r, arrange the arguments to the `.tL` macro as follows:

   $1 The Open Group

   $2 Copyright date (month, year)

   $3 Document number

   $4 ISBN

7.6 Checking Source Files

Spelling

To verify the spelling in a troff source file, use the command:

```
spell source_file
```

This command sends a list of unrecognized words to standard output in alphabetical order. Alternatively, you can redirect this output into a file.

Although this command is very useful, it is not a substitute for careful proofreading.

Coding

For limited checking of code, use the *checkmm* program, which runs a basic check of the use of eroff macros.

This command sends a list of unrecognized words to standard output. Alternatively, you can redirect this output into a file.

Note that *checkmm* does not recognize Open Group-specific macros, such as `.Ns` and `.mS`.

You can also run the *pairs* program if you need to locate a missing macro. The syntax is as follows:

```
pairs "<macro 1>" "<macro 2>" <filename>
```

This will run a check on the named file to make sure all occurrences of macro 1 are matched by a macro 2. Any errors will be directed to standard output, together with the relevant line number in the named file.. Because the program understands nesting of macros, you can used regular expressions to group several macros together; for example, multiple types of lists.

Appendix C

SUD-specific SGML

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

The following SUD Style Guide material has not yet been added: Chapter 5 and Appendix C.

C.1 Introduction

The Single UNIX Documentation reference pages are based on the Single UNIX Specification, which is an industry standard to which UNIX operating systems conform.

The five documents below constitute the Single UNIX Specification:

XBD, Issue 4, Version 2
: CAE Specification, August 1994, System Interface Definitions, Issue 4, Version 2 (ISBN: 1-85912-036-9, C434).

XSH, Issue 4, Version 2
: CAE Specification, August 1994, System Interfaces and Headers, Issue 4, Version 2 (ISBN: 1-85912-037-7, C435).

XCU, Issue 4, Version 2
: CAE Specification, August 1994, Commands and Utilities, Issue 4, Version 2 (ISBN: 1-85912-034-2, C436).

XNS, Issue 4
: CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438).

XCURSES, Issue 4, Version 2
: CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), plus Corrigendum U018.

The Single UNIX Documentation reference pages are written for a range of users that includes novice and experienced UNIX system users, as well as application developers. These reference pages provide information on the utilities, functions, and files defined in the Single UNIX Specification.

The Single UNIX Specification is used by operating system developers who create utilities, functions, and files that conform to the standard.

The Single UNIX Documentation is a set of documentation for UNIX operating systems that conform to the Single UNIX Specification. In its simplest form, this documentation covers only those features that are common to all conforming systems. However, the documentation is designed to be extended by UNIX system vendors, so that they can simply add information about the extensions provided by their systems. In this way, the Single UNIX Documentation serves as a base set of information that can support the documentation efforts of individual

system vendors.

The Single UNIX Specification uses the term *utility* to refer to a program that can be called by name from the shell, and *command* to refer to the complete string that is submitted to the shell for execution. Reference pages for the Single UNIX Documentation follow this convention.

In the Single UNIX Specification the classification of items is very broad. One reference volume provides information about programming interfaces (including functions, macros, and headers); another covers commands and utilities. The other volumes contain information about system definitions, files, and devices, as well as specialized utilities and programming interfaces.

The Single UNIX Documentation classifies items more narrowly, to support the needs of individual UNIX system vendors. For example, vendors may make some utilities available to all users (*user utilities*), but restrict the use of other utilities to system administrators (*administration utilities*). Similarly, programming interfaces may be classified as *system calls* or *library routines*, depending on how the interfaces are made available on the system.

This guide uses the term *division* to refer to the category or grouping to which a reference page belongs. For example, there is a *printf* utility and a *printf*( ) function. The reference page for the *printf* utility is in the user utilities division and the reference page for the *printf*( ) function is in the library routines division.

The term *section* refers only to sections inside each reference page. For example, almost every reference page has Name, Synopsis, and Description sections. Most have additional sections.

Reference pages usually reside in different directories on a UNIX system. Each directory contains reference pages for a particular kind of component. For example, reference pages that describe utilities for which users do not need special privileges often reside in a directory named *man1*, and reference pages that describe system calls often reside in a directory named *man2*.

These directories are often called *section directories*, because they correspond to the sections of a printed manual. Furthermore, UNIX users often describe reference pages in terms of belonging to Section 1, Section 2, and so forth because reference pages on most UNIX systems reside in numbered directories. For example, reference pages for system administration utilities may belong to Section 1m on one UNIX system and Section 8 on another system. This guide does not use the word *section* in this sense.

C.2 Doctype Declaration

Reference pages in the Single UNIX Documentation are tagged using elements defined for the DocBook document type definition (DTD), Version 2.4.1 (1995), with extensions.

Reference pages for the Single UNIX Documentation use the following identifier for the DTD:

"-//The Open Group//DTD DocBook V2.4.1-Based Extension SUD V1.0//EN"

The following DOCTYPE statement must be at the beginning of each file:

<!DOCTYPE DOCBOOK PUBLIC "-//The Open Group//DTD DocBook V2.4.1-Based Extension SUD V1.0//EN" [entity-definitions-specific-to-this-reference-page]

The reference page files for the Single UNIX Documentation are created as SGML fragments using ADEPT*Editor. The DOCTYPE statement for each individual reference page is commented out, so that the file can be treated as a fragment, rather than a complete SGML document instance. The following example shows the ADEPT*Editor coding for a fragment:

<!-- Fragment document type declaration subset: ArborText, Inc., 1988-1995, v.4001 DOCTYPE statement -->

This coding enables you to open each reference page file in ADEPT*Editor and edit it as a unit. At the same time, this approach enables you to use a wrapper file to process a collection of reference pages.

To enable each reference page to be handled as a valid SGML document instance, without fragment coding, a vendor can replace the word DOCBOOK in the DOCTYPE declaration with REFENTRY, and remove the fragment coding. The DOCTYPE declaration would then be as follows:

<!DOCTYPE REFENTRY PUBLIC "-//The Open Group//DTD DocBook V2.4.1-Based Extension SUD V1.0//EN" [entity-definitions-specific-to-this-reference-page]

C.3 Filenaming

For descriptive pages that do not document a specific system construct that already has a name (such as a function or utility name), you must create a name for the page. The convention used for descriptive reference pages in the Single UNIX Documentation is to begin the name with the abbreviation that is used to refer to the volume from which the information comes:

xsh The XSH and XSI volumes (programming interfaces and headers)

xcu The XCU volume (utilities)

xbd The XBD volume (system interface definitions)

xns The XNS volume (networking services)

xcur The XCURSES volume (curses interface)

The remainder of the name should be a mnemonic string that suggests the content for the page (for example, the reference page for the information on conformance from the XSH volume is named *xshconform*.

C.4 Metainformation

The following information should be included inside the <refmeta> tag:

- <manvolnum>

  Use the text entity that identifies the reference manual division (*divid*) for the page.

- <refmiscinfo class="sectdesc">

  The title of the reference manual division. The format of this entity is &divid-div;.

- <refmiscinfo class="copyright">

  Use the text entity &xosudcopyright;.

- <refmiscinfo class="date">

  Use the text entity &suddate; for Version 1 of the Single UNIX Specification.

- <refmiscinfo class="conformance">

  Use the information that is shown in the Single UNIX Specification as top-of-page headers (such as BASE or X/OPEN UNIX), or as supplementary information in the NAME line (such

as DEVELOPMENT).

C.5 Conformance

Because vendor-specific information may discuss conformance to other standards, it is important that you clearly identify conformance issues as applying to systems that conform to the Single UNIX Specification in reference pages for the Single UNIX Documentation. In a paragraph that discusses conformance issues, try to use the phrase ‘‘systems that conform to the Single UNIX Specification’’ early in the paragraph. Later references can use the phrase ‘‘conforming systems’’ as long as it is clear what the systems are conforming to.

C.6 Cross-references

The format used for links in the Single UNIX Documentation includes an identifier for the reference manual division in parentheses, as in ‘‘see the grep(1) reference page’’. Because the use of nested parentheses should be avoided, do not include such links within parentheses.

The preferred method for referring to any document is to use one of the standard text entities defined for the Single UNIX Documentation.

Replace specification language saying ‘‘this document’’ with a more specific reference. If you can refer to a specific reference page, that is the preferred treatment. If you must refer to the specifications, use a general reference to ‘‘the Single UNIX Specification.’’ Avoid general references to a specific volume of the Single UNIX Specification (such as ‘‘see the XSH specification’’).

Do not refer to a Single UNIX Documentation glossary entry for content. Copy the information from the glossary to the reference page.

If the Single UNIX Specification source for a reference page indicates that the reader should see another function for information about errors, return values, or other information, copy the information from the reference page that is cited and adapt it as needed for the reference page. If the operation of a function is described entirely in terms of another reference page—for example, indicating that the function you are documenting acts like another function called with specific parameters—determine what effect would be produced by the call that is cited, and explain to the reader what happens without the reference to the second function. (You may still wish to include the comparison between the two functions in the EXTENDED DESCRIPTION section.)

The definition of a utility in the Single UNIX Specification will sometimes refer to a particular use of a function to define the implementation of a utility precisely. If this definition provides information that clarifies the operation of the utility, that information should be restated directly, in a way that the reader can understand. If the definition is only a restatement of information that is already provided in descriptive form, you can retain the information, but you should move it to the EXTENDEd DESCRIPTION section.

C.7 Equations

Vendor placeholders should be placed within marked sections using the %VendorEquation; parameter entity.

C.8 Examples

Examples showing commands (<screen><userinput>) or the output format for a utility (<programlisting>), should be followed by a vendor placeholder for sample output. The placeholder should be within a marked section using the %VendorComputerOutput; parameter entity.

C.9 Notes

Include a title with the text entity &xosudnote; (which produces the word ‘‘Note’’) when you use the DocBook <note> tag.

For situations that involve potential damage to software or data, include a note tagged with the DocBook <caution> tag. Enter the title for the note using the text entity &xosudcaution; (which produces the word ‘‘Caution’’).

C.10 Reference Page Sections

SYNOPSIS

Use the &xosudsynptitle; text entity for the title.

If operand or option-argument names are shown with underscores in the specifications, substitute hyphens for the underscores in the Single UNIX Documentation. Do not change underscores in literal names that must be typed as shown.

If parameter names are shown with underscores in the specifications, substitute hyphens for the underscores in the Single UNIX Specification. Do not change underscores in literal names that must be typed as shown.

If the specification source does not show a space between an option and its argument, this indicates that no space is permitted on any conforming system.

If the specification shows a group of options together, you can put all of them within a single <option> tag.

DESCRIPTION

Use the &xshdescsection; text entity for the title.

OPTIONS

Use the &xosudoptstitle; text entity for the title.

A vendor placeholder should be included before the list to indicate whether the vendor has additional options, and after the list to allow a list of vendor options to be added.

The following text entities can be used in this section:

- &optleadsyn;

  Use if the specification states that the utility conforms to the standard utility syntax guidelines for the Single UNIX Specification, with no exceptions.

  This translates to: ‘‘This utility supports the utility syntax guidelines described in the xbdutsyntax(5) reference page.’’

- &optleadexcpt;

  Use if the specification states that the utility conforms to the standard utility syntax guidelines for the Single UNIX Specification, but lists exceptions. Follow this with an unnumbered list with a list item for each exception (even if there is only 1).

  This translates to: ‘‘This utility supports the utility syntax guidelines described in the xbdutsyntax(5) reference page, except that:’’

- &optlead;

  Use if any options are defined for the utility by the Single UNIX Specification. This should follow the 2 previous entities if they are used.

  This translates to: ‘‘Systems that conform to the Single UNIX Specification support the following options:’’

- &optnone;

  Use if no options are defined for the utility by the Single UNIX Specification. Do not add any other text for the section.

  This translates to: ‘‘No options are defined for this utility by the Single UNIX Specification.’’

  Add a vendor placeholder following this paragraph, in case vendors have options defined. This placeholder should include an introductory sentence, such as: ‘‘The *name* system also includes the following options:’’

OPERANDS

Use the &xosudopertitle; text entity for the title.

A vendor placeholder should be included before the list to indicate whether the vendor has additional operands, and after the list to allow a list of vendor operands to be added.

The following text entities can be used in this section:

- &operandlead;

  Use as the lead-in sentence for the list of operands.

  This translates to: ‘‘Systems that conform to the Single UNIX Specification support the following operands:’’

  The vendor placeholder after this list should include an introductory sentence, such as: ‘‘The *name* system also includes the following operands:’’

PARAMETERS

Use the &xshparmsection; text entity for the title.

The following text entities can be used in this section:

- &newparmlead;

  Use as the lead-in sentence for the list of parameters.

  This translates to: ‘‘The parameter descriptions follow in alphabetical order:’’

EXTENDED DESCRIPTION

Use the &xshexdesection; text entity for the title.

The Single UNIX Specification uses standard headings; the information under those headings can be mapped to reference pages in the Single UNIX Documentation either as paragraphs without subheads or, if the information is extensive, as subsections with standard titles. Typical examples are:

- STDERR

  Use the &nostderr; text entity for the text ‘‘This utility writes only diagnostic messages to standard error.’’

- ASYNCHRONOUS EVENTS

  Use the &asyncdefault; text entity for the text ‘‘If a signal is received during execution of this utility, the action of the utility follows the guidelines described in the xcuutildef() reference page.’’

- CONSEQUENCES OF ERRORS

  Use the &errdedefault; text entity for the text ‘‘If an error condition occurs, the effects of using this utility may vary among systems that conform to the Single UNIX Specification. For more information, see the xcuutildef() reference page.’’

EXIT STATUS

Use the &xosudexittitle; text entity for the title.

The following text should be used as the lead-in sentence: ‘‘This utility returns the following exit values:’’

RETURN VALUES

Use the &xshrtrnsection; text entity for the title.

The following text entities can be used in this section:

- &rtrnlead;

  Use as the lead-in sentence for the list of return values, following the phrase ‘‘The *function-name* function ...’’, if 1 function is named.

  This translates to: ‘‘... returns the following:’’

- &rtrnlead2;

  Use as the lead-in sentence for the list of return values, following the phrase ‘‘The *function-name* function ...’’, if more than 1 function is named.

This translates to: ‘‘... return the following:’’

ERRORS

Use the &xsherrosection; text entity for the title.

You must use separate lists to distinguish between errors that must be defined on all conforming systems (the Single UNIX Specification uses the term *will*), and errors that are optional for conforming systems (*may*).

Errors for multiple interfaces documented on the same page should be divided by interface, unless grouping can eliminate duplication without sacrificing clarity.

The following text entities can be used in this section:

- &willfail1;, &willfail2;, and &willfail3;

  Use to introduce a list of required errors with the name of the function, as follows:

  ```
  <para>willfail1;
  <function>function</function> function
  &willfail2;</para>
  ```

  This translates to: ‘‘On all systems that conform to the Single UNIX Specification, the *function* function sets *errno* as listed for the following conditions:’’

  When lists of required errors can be combined, use the text entities &willfail1; and &willfail3; with the function names, as follows:

  ```
  <para>&willfail1;
  <function>function</function> and <function>function</function>
  functions &willfail3;</para>
  ```

  This translates to: ‘‘On all systems that conform to the Single UNIX Specification, the *function* and *function* functions set *errno* as listed for the following conditions:’’

- &mayfail2;

  Use to introduce a list of optional errors with the name of the function, as follows:

  ```
  <para>The <function>function</function> function &mayfail2;</para>
  ```

  This translates to: ‘‘The *function* function can set *errno* to one of the following if the corresponding error condition occurs, but systems that conform to the Single UNIX Specification are not required to detect these conditions:’’

- &noerrors;

  Use when no errors are defined by the Single UNIX Specification.

  This translates to: ‘‘Systems that conform to the Single UNIX Specification are not required to detect error conditions for this function.’’

- &nootherrors;

  Use when the Single UNIX Specification states that no other errors will occur. This indicates that a conforming system cannot define additional errors for the specified interface.

  This translates to: ‘‘No other errors will occur.’’

A vendor placeholder may be added to list additional errors defined by a vendor. This list should include an introductory sentence, such as: ‘‘The *name* system specifies errors for the *function* function for the following condition:’’

4063 **EXAMPLES**

4064 Use the &xshexamsection; text entity for the title.

4065 Add a vendor placeholder for command and utility examples, so that each vendor can show the
4066 output produced by the command. Use the &VendorComputerOutput; marked section.

4067 **ENVIRONMENT VARIABLES**

4068 Use the &xshenvrsection; text entity for the title.

4069 A description of the standard environment variables is given in the *environ*( ) reference page;
4070 they are:

4071 COLUMNS	LC_COLLATE	LC_TIME	PATH
4072 DATEMSK	LC_CTYPE	LINES	SHELL
4073 HOME	LC_MESSAGES	LOGNAME	TMPDIR
4074 LANG	LC_MONETARY	MSGVERB	TERM
4075 LC_ALL	LC_NUMERIC	NLSPATH	TZ

4076 The following text entities can be used in this section:

4077 • &envrleadstd;

4078 Use for reference pages that include references to one of the standard environment
4079 variables listed above, but do not include any information that is specific to the <refname>.

4080 This translates to: ‘‘The environ(5) reference page provides general information about the
4081 following standard environment variables, which can affect the operation of this utility: LANG,
4082 LC_ALL, LC_CTYPE, LC_MESSAGES, LC_TIME, LP_DEST, and NLSPATH.’’

4083 • &envrleadother;

4084 Use to provide supplementary information for a particular variable if the generic description is
4085 not sufficient. (Do not repeat information that is included in the environ(5) reference page.)

4086 This translates to: ‘‘Some environment variables interact with specific features of this utility,
4087 as follows:’’

4088 Vendor placeholders may be added to allow additional vendor-specific information.

4089 **FILES**

4090 Use the &xshfilesection; text entity for the title.

4091 The following text should be used to introduce a list of files: ‘‘The following files are used by this
4092 utility:’’

4093 **SEE ALSO**

4094 Use the &xshalsosection; text entity for the title.

4095 For references to other reference pages, code them as follows:

```
4096 <link linkend="environ-misc"><citerefentry>
4097 <refentrytitle>environ</refentrytitle><manvolnum>&misc;</manvolnum>
4098 </citerefentry></link>
```

4099 For references to external documents, use text entities.

C.11 Reference Page Templates

C.11.1 Descriptive Reference Pages

This template can be used to create pages that contain general information that does not apply to a specific function, utility, or file.

Substitute the correct information for the following placeholders:

- Change all occurrences of `mydesc` to the refentry name for the reference page.
- In the Name section, change `purpose` to the purpose of this refentry.
- In the Files section, change `filename` to an appropriate file name.
- In the See Also section, substitute appropriate entries for `myutil`, `myfunc`, `myheader.h`, `myprogram.c` and `Document Title`. Delete unused entries and their tags.

```
<refentry id="mydesc-misc">
<refmeta>
<refentrytitle>mydesc</refentrytitle>
<manvolnum>&misc;</manvolnum>
<refmiscinfo class="copyright">&xosudcopyright;</refmiscinfo>
<refmiscinfo class="date">&suddate;</refmiscinfo>
<refmiscinfo class="sectdesc">&misc-div;</refmiscinfo>
<refmiscinfo class="conformance"></refmiscinfo>
<indexterm><primary></primary><secondary></secondary>
</indexterm>
</refmeta>
<refnamediv id="mydesc-misc-name">
<refname>mydesc</refname>
<refpurpose>purpose</refpurpose>
</refnamediv>
<refsect1 id="mydesc-misc-desc">
<title>&xshdescsection;</title>
<para></para>
</refsect1>
<refsect1 id="mydesc-misc-file">
<title>&xshfilesection;</title>
<para>The <citerefentry><refentrytitle>mydesc</refentrytitle>
<manvolnum>&misc;</manvolnum></citerefentry> reference
page uses the following file:</para>
<variablelist>
<varlistentry>
<term><filename>filename</filename></term>
<listitem><para>Contains</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
<refsect1 id="mydesc-misc-also">
<title>&xshalsosection;</title>
<para><link linkend="myutil-user"><citerefentry><refentrytitle>
myutil</refentrytitle><manvolnum>&user;</manvolnum>
</citerefentry></link>, <link linkend="myfunc-sysc"><citerefentry>
<refentrytitle>myfunc</refentrytitle><manvolnum>&sysc;</manvolnum>
```

```
4148        </citerefentry></link>, <link linkend="myheader.h-file"><citerefentry>
4149        <refentrytitle>myheader.h</refentrytitle><manvolnum>&file;</manvolnum>
4150        </citerefentry></link>, <link linkend="myprogram.c-exmp"><citerefentry>
4151        <refentrytitle>myprogram.c</refentrytitle><manvolnum>&exmp;</manvolnum>
4152        </citerefentry></link></para>
4153        <para><citetitle>Document Title</citetitle></para>
4154        </refsect1>
4155        </refentry>
```

C.11.2 Utility Reference Pages

This template can be used to create reference pages that document system utilities.

Substitute the correct information for the following placeholders:

- Change all occurrences of `myutil` to the refentry name for the reference page.
- If the reference page you are creating is for an administration utility, change all occurrences of `user` to `admn`. Also, declare the file entity.
- In the Name section, change `purpose` to the purpose of this refentry.
- In the Synopsis section, substitute appropriate entries for `myutil`, `Option-Argument`, and `Operand`. Delete unused entries and their tags.
- In the Options section, substitute appropriate entries for `Option` and `Option-Argument`. Delete unused entries and their tags.
- In the Operands section, change `Operand` to an appropriate operand name.
- In the Examples section, change `Doing Something Useful` to an appropriate Example title. Change `myutil -o filename` to an appropriate Example command.
- In the Environment Variables section, if the &envrleadother; entity is used, change `EnvironmentVariable` to an appropriate environment variable name.
- In the See Also section, substitute appropriate entries for `myother`, `myfunc`, `myheader.h`, `myprogram.c`, and `Document Title`. Delete unused entries and their tags.

Then make the following changes:

- In the Synopsis section, delete the Obsolescent Forms tagging if it is not used.
- In the Options section, choose the entities (and their associated Vendor Extension tags) that apply to this reference page. Delete unused entities.
- In the Operands section, choose the entity (and its associated Vendor Extension tags) that applies to this reference page. Delete the unused entity.
- In the Exit Status section, delete the &otherexitlead; entity and associated variable list if it is not used.
- In the Environment Variables section, delete the &envrleadother; entity and associated variable list if it is not used.

```
<refentry id="myutil-user">
<refmeta>
<refentrytitle>myutil</refentrytitle>
<manvolnum>&user;</manvolnum>
<refmiscinfo class="copyright">&xosudcopyright;</refmiscinfo>
<refmiscinfo class="date">&suddate;</refmiscinfo>
<refmiscinfo class="sectdesc">&user-div;</refmiscinfo>
<refmiscinfo class="conformance"></refmiscinfo>
<indexterm><primary>myutil utility</primary></indexterm>
</refmeta>
<refnamediv id="myutil-user-name">
<refname>myutil</refname>
<refpurpose>purpose</refpurpose>
</refnamediv>
```

```
<refsynopsisdiv id="myutil-user-synp">
<title>&xosudsynptitle;</title>
<cmdsynopsis>
<command>myutil</command>
<group>
<arg choice="plain"><option role="dash">a</option></arg>
<arg choice="plain"><option role="nodash">b</option></arg>
</group>
<arg choice="plain" rep="repeat">
<arg><option role="plus">c </option>
<replaceable>Option-Argument</replaceable></arg></arg>
<arg choice="plain" rep="repeat">
<replaceable> Operand</replaceable></arg>
</cmdsynopsis>
<refsect2 id="myutil-user-synp-obso">
<title>Obsolescent Forms</title>
</refsect2>
</refsynopsisdiv>
<refsect1 id="myutil-user-desc">
<title>&xshdescsection;</title>
<para></para>
</refsect1>
<refsect1 id="myutil-user-opts">
<title>&xosudoptstitle;</title>
<para>&optleadsyn;</para>
<para>&optleadexcpt;</para>
<![ %VendorExtension; [&myutil-user-entity1;]]>
<para>&optlead;</para>
<![ %VendorExtension; [&myutil-user-entity2;]]>
<variablelist>
<varlistentry>
<term><option role="dash">Option </option>
<replaceable>Option-Argument</replaceable></term>
<listitem>
<para></para>
</listitem>
</varlistentry>
</variablelist>
<![ %VendorExtension; [&myutil-user-entity3;]]>
<para>&optnone;</para>
<![ %VendorExtension; [&myutil-user-entity7;]]>
</refsect1>
<refsect1 id="myutil-user-oper">
<title>&xosudopertitle;</title>
<para>&operandlead;</para>
<![ %VendorExtension; [&myutil-user-entity4;]]>
<variablelist>
<varlistentry>
<term><replaceable>operand</replaceable></term>
<listitem>
<para></para>
</listitem>
```

```
</varlistentry>
</variablelist>
<![ %VendorExtension; [&myutil-user-entity5;]]>
</refsect1>
<refsect1 id="myutil-user-exde">
<title>&xshexdesection;</title>
<para></para>
</refsect1>
<refsect1 id="myutil-user-exit">
<title>&xosudexittitle;</title>
<para>&exitlead;</para>
<variablelist>
<varlistentry>
<term><returnvalue>0</returnvalue></term>
<listitem>
<para>Successful completion.</para>
</listitem>
</varlistentry>
<varlistentry>
<term>><returnvalue>0</returnvalue></term>
<listitem>
<para>An error occurred.</para>
</listitem>
</varlistentry>
</variablelist>
<para>&otherexitlead;</para>
<variablelist>
<varlistentry>
<term><returnvalue>0</returnvalue></term>
<listitem>
<para>Successful completion.</para>
</listitem>
</varlistentry>
<varlistentry>
<term>&lt;<returnvalue>0</returnvalue></term>
<listitem>
<para>An error occurred.</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
<refsect1 id="myutil-user-exam">
<title>&xshexamsection;</title>
<example id="myutil-user-exam-1">
<title>Doing Something Useful</title>
<para>The following example ...</para>
<screen><userinput>myutil -o filename</userinput></screen>
<![ %VendorComputerOutput; [&myutil-user-output1;]]>
</example>
</refsect1>
<refsect1 id="myutil-user-envr">
<title>&xshenvrsection;</title>
```

```
4303        <para>&envrleadstd;
4304        <systemitem class="environvar">EnvironmentVariables</systemitem>
4305        </para>
4306        <para>&envrleadother;</para>
4307        <variablelist>
4308        <varlistentry>
4309        <term><systemitem class="environvar">EnvironmentVariable
4310        </systemitem></term>
4311        <listitem>
4312        <para></para>
4313        </listitem>
4314        </varlistentry>
4315        </variablelist>
4316        </refsect1>
4317        <refsect1 id="myutil-user-file">
4318        <title>&xshfilesection;</title>
4319        <para></para>
4320        </refsect1>
4321        <refsect1 id="myutil-user-also">
4322        <title>&xshalsosection;</title>
4323        <para><link linkend="myother-util"><citerefentry>
4324        <refentrytitle>myother</refentrytitle><manvolnum>&user;</manvolnum>
4325        </citerefentry></link>, <link linkend="myfunc-sysc"><citerefentry>
4326        <refentrytitle>myfunc</refentrytitle><manvolnum>&sysc;</manvolnum>
4327        </citerefentry></link>, <link linkend="myheader.h-file"><citerefentry>
4328        <refentrytitle>myheader.h</refentrytitle><manvolnum>&file;</manvolnum>
4329        </citerefentry></link>, <link linkend="environ-misc"><citerefentry>
4330        <refentrytitle>environ</refentrytitle><manvolnum>&misc;</manvolnum>
4331        </citerefentry></link>, <link linkend="xbdusyntax-misc"><citerefentry>
4332        <refentrytitle>xbdusyntax</refentrytitle><manvolnum>&misc;</manvolnum>
4333        </citerefentry></link>, <link linkend="myprogram.c-exmp"><citerefentry>
4334        <refentrytitle>myprogram.c</refentrytitle><manvolnum>&exmp;</manvolnum>
4335        </citerefentry></link></para>
4336        <para><citetitle>Document Title</citetitle></para>
4337        </refsect1>
4338        </refentry>
```

C.11.3 Program Interface Reference Pages

This template can be used to create reference pages that document a function or macro.

Substitute the correct information for the following placeholders:

- Change all occurrences of `myfunc` to the refentry name for the reference page.
- If the reference page you are creating is for a library routine, change all occurrences of `sysc` to `libr`. Also, declare the file entity.
- If the reference page is for a macro, change the word `function` to `macro` in the index entry.
- In the Synopsis section, change `refdescriptor` to an appropriate refdescriptor, if there is one, or delete the tags. Change `purpose` to the purpose of this refentry.
- In the Synopsis section, substitute appropriate entries for `header-name`, `ReturnValue`, `myfunc`, `ParameterType`, and `Parameter`. Delete unused entries and their tags.
- In the Parameters section, substitute appropriate entries for `ParameterName` and `PointerName`.
- In the Return Values section, change `[ERROR_CODE]` to an appropriate error name.
- In the Examples section, change `Doing Something Useful` to an appropriate Example title. Substitute appropriate entries for `header-name`, `variable declarations for the function calls`, and `function calls`. Delete unused entries and their tags.
- In the See Also section, substitute appropriate entries for `myutil`, `myfunc`, `myheader.h`, `myprogram.c`, and `Document Title`. Delete unused entries and their tags.
- In the Errors section, delete the &mayfail1; and &mayfail2; entities and associated variable list if this text does not apply.

```
<refentry id="myfunc-sysc">
<refmeta>
<refentrytitle>myfunc</refentrytitle>
<manvolnum>&sysc;</manvolnum>
<refmiscinfo class="copyright">&xosudcopyright;</refmiscinfo>
<refmiscinfo class="date">&suddate;</refmiscinfo>
<refmiscinfo class="sectdesc">&sysc-div;</refmiscinfo>
<refmiscinfo class="conformance"></refmiscinfo>
<indexterm><primary>myfunc function</primary></indexterm>
</refmeta>
<refnamediv id="myfunc-sysc-name">
<refdescriptor>
<replaceable>refdescriptor</replaceable>
</refdescriptor>
<refname>myfunc</refname>
<refpurpose>purpose</refpurpose>
</refnamediv>
<refsynopsisdiv id="myfunc-sysc-synp">
<title>&xosudsynptitle;</title>
<funcsynopsis>
<funcsynopsisinfo>#include &lt;header-name></funcsynopsisinfo>
<funcdef>ReturnValue <function>myfunc</function></funcdef>
<paramdef>ParameterType <parameter>Parameter</parameter></paramdef>
</funcsynopsis>
```

```
4385        </refsynopsisdiv>
4386        <refsect1 id="myfunc-sysc-desc">
4387        <title>&xshdescsection;</title>
4388        <para></para>
4389        </refsect1>
4390        <refsect1 id="myfunc-sysc-parm">
4391        <title>&xshparmsection;</title>
4392        <para>&newparmlead;</para>
4393        <variablelist>
4394        <varlistentry>
4395        <term><parameter>ParameterName</parameter></term>
4396        <listitem>
4397        <para>Indicates</para>
4398        </listitem>
4399        </varlistentry>
4400        <varlistentry>
4401        <term><parameter>PointerName</parameter></term>
4402        <listitem>
4403        <para>Points to</para>
4404        </listitem>
4405        </varlistentry>
4406        </variablelist>
4407        </refsect1>
4408        <refsect1 id="myfunc-sysc-exde">
4409        <title>&xshexdesection;</title>
4410        <para></para>
4411        </refsect1>
4412        <refsect1 id="myfunc-sysc-rtrn">
4413        <title>&xshrtrnsection;</title>
4414        <para>The <function>myfunc</function> function &rtrnlead;</para>
4415        <variablelist>
4416        <varlistentry>
4417        <term><returnvalue>0</returnvalue></term>
4418        <listitem>
4419        <para>Success.</para>
4420        </listitem>
4421        </varlistentry>
4422        <varlistentry>
4423        <term><returnvalue>&minus;1</returnvalue></term>
4424        <listitem>
4425        <para>Failure: <symbol>errno</symbol> is set to indicate
4426        the error.</para>
4427        </listitem>
4428        </varlistentry>
4429        <varlistentry>
4430        <term>Nonzero value</term>
4431        <listitem>
4432        <para></para>
4433        </listitem>
4434        </varlistentry>
4435        </variablelist>
4436        </refsect1>
```

```
4437        <refsect1 id="myfunc-sysc-erro">
4438        <title>&xsherrosection;</title>
4439        <para>&willfail1; <function>myfunc</function> function &willfail2;</para>
4440        <variablelist>
4441        <varlistentry>
4442        <term><systemitem role="errno">ERROR_CODE</systemitem></term>
4443        <listitem>
4444        <para></para>
4445        </listitem>
4446        </varlistentry>
4447        </variablelist>
4448        <para>&mayfail1; <function>myfunc</function> function &mayfail2;</para>
4449        <variablelist>
4450        <varlistentry>
4451        <term><systemitem role="errno">ERROR_CODE</systemitem></term>
4452        <listitem>
4453        <para></para>
4454        </listitem>
4455        </varlistentry>
4456        </variablelist>
4457        <![ %VendorExtension; [&myfunc-sysc-entity1;]]>
4458        </refsect1>
4459        <refsect1 id="myfunc-sysc-exam">
4460        <title>&xshexamsection;</title>
4461        <example id="myfunc-sysc-exam-1">
4462        <title>Title Text</title>
4463        <para>The following example...</para>
4464        <para>&compexp1; <link linkend="examplename-exmp"><citerefentry>
4465        <refentrytitle>examplename.c</refentrytitle><manvolnum>&exmp;</manvolnum>
4466        </citerefentry></link> &compexp2;</para>
4467        <programlisting>#include &lt;header-name>
4468        &vellip;
4469        variable declarations for the function calls
4470        function calls</programlisting>
4471        </example>
4472        </refsect1>
4473        <refsect1 id="myfunc-sysc-also">
4474        <title>&xshalsosection;</title>
4475        <para><link linkend="myutil-user"><citerefentry><refentrytitle>
4476        myutil</refentrytitle><manvolnum>&user;</manvolnum>
4477        </citerefentry></link>, <link linkend="myfunc-sysc"><citerefentry>
4478        <refentrytitle>myfunc</refentrytitle><manvolnum>&sysc;</manvolnum>
4479        </citerefentry></link>, <link linkend="myheader.h-file"><citerefentry>
4480        <refentrytitle>myheader.h</refentrytitle><manvolnum>&file;</manvolnum>
4481        </citerefentry></link>, <link linkend="myprogram.c-exmp"><citerefentry>
4482        <refentrytitle>myprogram.c</refentrytitle><manvolnum>&exmp;</manvolnum>
4483        </citerefentry></link></para>
4484        <para><citetitle>Document Title</citetitle></para>
4485        </refsect1>
4486        </refentry>
```

C.11.4 Header Reference Pages

This template can be used to create reference pages that document a header.

Substitute the correct information for the following placeholders:

- Change `myheader.h` to the refentry name for the reference page.
- Change all occurrences of `my/header.h` to the name of the header page.
- In the Constants section, change `CONSTANT_NAME` to an appropriate constant name.
- In the Data Types section, change `DataType` to an appropriate data type.
- In the External Variables section, change `VariableName` to an appropriate external variable name.
- In the Function Prototypes section, substitute appropriate entries for `DataType`, `funcname1`, `ParameterDataType`, `Parameter`, `funcname2`, `StructureName`, and `funcname3`. Delete unused entries and their tags.
- In the Macros section, substitute appropriate entries for `DataTypeWithoutDesc`, `MacroName`, `ParameterDataType`, `Parameter`, `DataTypeWithoutDesc`, and `Description of Data Type`. Delete unused entries and their tags.
- In the Structures section, substitute appropriate entries for `StructureName`, `MemberDataType`, `DifferentStructureNameIfApplicable`, and `StructureMemberOrField`. Delete unused entries and their tags.
- In the See Also section, substitute appropriate entries for `myutil`, `myfunc`, `myprogram.c`, and `Document Title`. Delete unused entries and their tags.

```
<refentry id="myheader.h-file">
<refmeta>
<refentrytitle>myheader.h</refentrytitle>
<manvolnum>&file;</manvolnum>
<refmiscinfo class="copyright">&xosudcopyright;</refmiscinfo>
<refmiscinfo class="date">&suddate;</refmiscinfo>
<refmiscinfo class="sectdesc">&file-div;</refmiscinfo>
<refmiscinfo class="conformance"></refmiscinfo>
<indexterm><primary>my/header.h header</primary></indexterm>
</refmeta>
<refnamediv id="myheader.h-file-name">
<refname>myheader.h</refname>
<refpurpose>include definitions for </refpurpose>
</refnamediv>
<refsynopsisdiv id="myheader.h-file-synp">
<title>&xosudsynptitle;</title>
<synopsis><literal>#include &lt;my/header.h></literal></synopsis>
</refsynopsisdiv>
<refsect1 id="myheader.h-file-desc">
<title>&xshdescsection;</title>
<para>The <filename class="headerfile">my/header.h
</filename> &headerdefs;</para>
<itemizedlist>
<listitem>
<para><xref linkend="myheader.h-file-desc-cons"></para>
</listitem>
```

```
4533        <listitem>
4534        <para><xref linkend="myheader.h-file-desc-daty"></para>
4535        </listitem>
4536        <listitem>
4537        <para><xref linkend="myheader.h-file-desc-extv"></para>
4538        </listitem>
4539        <listitem>
4540        <para><xref linkend="myheader.h-file-desc-funp"></para>
4541        </listitem>
4542        <listitem>
4543        <para><xref linkend="myheader.h-file-desc-macr"></para>
4544        </listitem>
4545        <listitem>
4546        <para><xref linkend="myheader.h-file-desc-stru"></para>
4547        </listitem>
4548        </itemizedlist>
4549        <refsect2 id="myheader.h-file-desc-cons">
4550        <title>Constants</title>
4551        <para>The <filename class="headerfile">my/header.h</filename>
4552        header defines the following constants:</para>
4553        <variablelist>
4554        <varlistentry>
4555        <term><systemitem class="constant">CONSTANT_NAME</systemitem></term>
4556        <listitem>
4557        <para></para>
4558        </listitem>
4559        </varlistentry>
4560        </variablelist>
4561        </refsect2>
4562        <refsect2 id="myheader.h-file-desc-daty">
4563        <title>Data Types</title>
4564        <para>The <filename class="headerfile">my/header.h</filename>
4565        header defines the following data types:</para>
4566        <variablelist>
4567        <varlistentry>
4568        <term><literal>DataType</literal></term>
4569        <listitem>
4570        <para></para>
4571        </listitem>
4572        </varlistentry>
4573        </variablelist>
4574        </refsect2>
4575        <refsect2 id="myheader.h-file-desc-extv">
4576        <title>External Variables</title>
4577        <para>The following are declared as external variables:</para>
4578        <variablelist>
4579        <varlistentry>
4580        <term><literal>DataType VariableName</literal></term>
4581        <listitem>
4582        <para></para>
4583        </listitem>
4584        </varlistentry>
```

```
4585        </variablelist>
4586        </refsect2>
4587        <refsect2 id="myheader.h-file-desc-funp">
4588        <title>Function Prototypes</title>
4589        <para>&funcmacro;</para>
4590        <![ %VendorExtension; [&myheader.h-file-entity1;]]>
4591        <funcsynopsis>
4592        <funcdef>DataType <function>funcname1</function></funcdef>
4593        <paramdef>ParameterDataType <parameter>Parameter</parameter></paramdef>
4594        </funcsynopsis>
4595        <funcsynopsis>
4596        <funcdef>void <function>funcname2</function></funcdef>
4597        <void>
4598        </funcsynopsis>
4599        <funcsynopsis>
4600        <funcdef>struct StructureName *<function>funcname3</function></funcdef>
4601        <paramdef>ParameterDataType <parameter>Parameter</parameter></paramdef>
4602        </funcsynopsis>
4603        <![ %VendorExtension; [&myheader.h-file-entity2;]]>
4604        </refsect2>
4605        <refsect2 id="myheader.h-file-desc-macr">
4606        <title>Macros</title>
4607        <para>The <filename class="headerfile">my/header.h</filename>
4608        header defines the following macros:</para>
4609        <programlisting>
4610        DataTypeWithoutDesc
4611        <systemitem class="macro">MacroName</systemitem>
4612        (ParameterDataType <parameter>Parameter</parameter>);
4613        </programlisting>
4614        <variablelist>
4615        <varlistentry>
4616        <term><literal>DataTypeWithDesc
4617        <systemitem class="macro">MacroName</systemitem>
4618        (ParameterDataType <parameter>Parameter</parameter>);
4619        </literal></term>
4620        <listitem>
4621        <para>Description of Data Type</para>
4622        </listitem>
4623        </varlistentry>
4624        </variablelist>
4625        </refsect2>
4626        <refsect2 id="myheader.h-file-desc-stru">
4627        <title>Structures</title>
4628        <para>The <filename class="headerfile">my/header.h</filename>
4629        header defines the following structures:</para>
4630        <variablelist>
4631        <varlistentry>
4632        <term><structname>StructureName</structname></term>
4633        <listitem><para>The <structname>StructureName</structname>
4634        structure includes at least the following members:</para>
4635        <variablelist>
4636        <varlistentry>
```

```
4637        <term><literal>MemberDataType</literal>
4638        <structname>DifferentStructureNameIfApplicable</structname>
4639        <structfield>StructureMemberOrField</structfield></term>
4640        <listitem>
4641        <para></para>
4642        </listitem>
4643        </varlistentry>
4644        </variablelist>
4645        </listitem>
4646        </varlistentry>
4647        </variablelist>
4648        </refsect2>
4649        </refsect1>
4650        <refsect1 id="myheader.h-file-file">
4651        <title>&xshfilesection;</title>
4652        <para></para>
4653        </refsect1>
4654        <refsect1 id="myheader.h-file-also">
4655        <title>&xshalsosection;</title>
4656        <para><link linkend="myutil-user"><citerefentry><refentrytitle>
4657        myutil</refentrytitle><manvolnum>&user;</manvolnum>
4658        </citerefentry></link>, <link linkend="myfunc-sysc"><citerefentry>
4659        <refentrytitle>myfunc</refentrytitle><manvolnum>&sysc;</manvolnum>
4660        </citerefentry></link>, <link linkend="myprogram.c-exmp"><citerefentry>
4661        <refentrytitle>myprogram.c</refentrytitle><manvolnum>&exmp;</manvolnum>
4662        </citerefentry></link>, <link linkend="mydesc-misc"><citerefentry>
4663        <refentrytitle>mydesc</refentrytitle><manvolnum>&misc;</manvolnum>
4664        </citerefentry></link></para>
4665        <para></para>
4666        <para><citetitle>Document Title</citetitle></para>
4667        </refsect1>
4668        </refentry>
```

C.11.5 Sample Program Reference Pages

This template can be used to create reference pages that document a sample program.

Substitute the correct information for the following placeholders:

- Change all occurrences of `myprog.c` to the refentry name for the reference page.
- In the Description section, substitute appropriate entries for `_XOPEN_SOURCE 1` and `_XOPEN_SOURCE_EXTENDED 1`.
- In the See Also section, substitute appropriate entries for `myutil`, `myfunc`, `myheader.h`, `mydesc`, and `Document Title`. Delete unused entries and their tags.

```
<refentry id="myprog.c-exmp">
<refmeta>
<refentrytitle>myprog.c</refentrytitle>
<manvolnum>&exmp;</manvolnum>
<refmiscinfo class="copyright">&xosudcopyright;</refmiscinfo>
<refmiscinfo class="date">&suddate;</refmiscinfo>
<refmiscinfo class="sectdesc">&exmp-div;</refmiscinfo>
<refmiscinfo class="conformance"></refmiscinfo>
<indexterm><primary>myprog.c sample program</primary></indexterm>
</refmeta>
<refnamediv id="myprog.c-exmp-name">
<refname>myprog.c</refname>
<refpurpose>sample application for </refpurpose>
</refnamediv>
<refsynopsisdiv id="myprog.c-exmp-synp">
<title>&xosudsynptitle;</title>
<synopsis><literal>c89 -o myprog myprog.c</literal></synopsis>
</refsynopsisdiv>
<refsect1 id="myprog.c-exmp-desc">
<title>&xshdescsection;</title>
<para></para>
<para>This application is designed to be compilable
and linkable C code that can be used on any system
conforming to the &suspecs;.</para>
<informalexample>
<programlisting>
#define _XOPEN_SOURCE 1
#define _XOPEN_SOURCE_EXTENDED 1
</programlisting>
</informalexample>
</refsect1>
<refsect1 id="myprog.c-exmp-file">
<title>&xshfilesection;</title>
<para>The following files are used by this example application:</para>
<variablelist>
<varlistentry>
<term><filename>myprog.c</filename></term>
<listitem>
<para></para>
</listitem>
</varlistentry>
```

```
4718        </variablelist>
4719        <![ %VendorExtension; [
4720        <!--VENDOR ADDITION: Use the entity following each item in this list to
4721        identify the location of the sample file on your UNIX system.-->
4722        &myprog.c-exmp-entity1;]]>
4723        </refsect1>
4724        <refsect1 id="myprog.c-exmp-also">
4725        <title>&xshalsosection;</title>
4726        <para><link linkend="myutil-user"><citerefentry><refentrytitle>
4727        myutil</refentrytitle><manvolnum>&user;</manvolnum>
4728        </citerefentry></link>, <link linkend="myfunc-sysc"><citerefentry>
4729        <refentrytitle>myfunc</refentrytitle><manvolnum>&sysc;</manvolnum>
4730        </citerefentry></link>, <link linkend="myheader.h-file"><citerefentry>
4731        <refentrytitle>myheader.h</refentrytitle><manvolnum>&file;</manvolnum>
4732        </citerefentry></link>, <link linkend="mydesc-misc"><citerefentry>
4733        <refentrytitle>mydesc</refentrytitle><manvolnum>&misc;</manvolnum>
4734        </citerefentry></link></para>
4735        <para><citetitle>Document Title</citetitle></para>
4736        </refsect1>
4737        </refentry>
```

Appendix D

SGML Examples

D.1 Metainformation

The following example shows the metainformation section for the *grep* utility:

```
<refmeta>
<refentrytitle>grep</refentrytitle>
<manvolnum>&user;</manvolnum>
<refmiscinfo class="copyright">Copyright 1997, The Open Group
</refmiscinfo>
<refmiscinfo class="date">August 1997</refmiscinfo>
<refmiscinfo class="sectdesc">&user-div;</refmiscinfo>
<refmiscinfo class="conformance">Base</refmiscinfo>
<indexterm><primary>grep utility</primary></indexterm>
</refmeta>
```

The following example shows the metainformation section for the **<sys/time.h>** header. Note that the slash character (/) in the name of the header is omitted from the reference page name in the <refentrytitle, but not from the index entry.

```
<refmeta>
<refentrytitle>systime.h</refentrytitle>
<manvolnum>&file;</manvolnum>
<refmiscinfo class="copyright">Copyright 1997, The Open Group
</refmiscinfo>
<refmiscinfo class="date">August 1997</refmiscinfo>
<refmiscinfo class="sectdesc">&file-div;</refmiscinfo>
<refmiscinfo class="conformance">X/Open UNIX</refmiscinfo>
<indexterm><primary>sys/time.h header</primary></indexterm>
</refmeta>
```

D.2 NAME

The following example shows the Name section for the *grep* utility:

```
<refnamediv id="grep-user-name">
<refname>grep</refname>
<refpurpose>search a file for a pattern</refpurpose>
</refnamediv>
```

The following example shows the Name section for the *exec* family of functions:

```
<refnamediv id="exec-sysc-name">
<refdescriptor>exec</refdescriptor>
<refname>execl</refname>
<refname>execle</refname>
<refname>execlp</refname>
<refname>execv</refname>
<refname>execve</refname>
<refname>execvp</refname>
<refpurpose>execute a file</refpurpose>
</refnamediv>
```

D.3 SYNOPSIS

D.3.1 Utilities

The following example shows a utility synopsis with an arbritrary collection of options, option-arguments, and operands. The utility in the example is named *utility-name*.

The syntax:

```
utility-name [-a|-c] [-bd] [-e opt-arg1]... [-f opt-arg2...]
    [oper1]... oper2 ...
```

is coded as follows:

```
<cmdsynopsis>
<command>utility-name</command>
<group>
<arg choice="plain"><option role="dash">a</option></arg>
<arg choice="plain"><option role="dash">c</option></arg>
</group>
<arg><option role="dash">bd</option></arg>
<arg rep="repeat"><option role="dash">e </option>
<replaceable>opt-arg1</replaceable></arg>
<arg><option role="dash">f </option>
<arg choice="plain" rep="repeat"><replaceable>opt-arg2</replaceable>
</arg>
</arg>
<arg rep="repeat"><replaceable>oper1</replaceable></arg>
<arg choice="plain" rep="repeat"><replaceable>oper2</replaceable></arg>
</cmdsynopsis>
```

The following example shows the Synopsis for the *lpstat* utility.

The syntax:

```
lpstat [-drst] [-a list] [-c list] [-o list] [-p list]
    [-u list] [-v list] ID ...
```

is coded as follows:

```
<cmdsynopsis conformance="unix-extension">
<command>lpstat</command>
<arg><option role="dash">drst</option></arg>
<arg><option role="dash">a</option><replaceable>list</replaceable></arg>
<arg><option role="dash">c</option><replaceable>list</replaceable></arg>
<arg><option role="dash">o</option><replaceable>list</replaceable></arg>
<arg><option role="dash">p</option><replaceable>list</replaceable></arg>
<arg><option role="dash">u</option><replaceable>list</replaceable></arg>
<arg><option role="dash">v</option><replaceable>list</replaceable></arg>
<arg choice="plain" rep="repeat"><replaceable>ID</replaceable></arg>
</cmdsynopsis>
```

The following example shows the coding used for the *mailx* utility, which has different synopses to reflect different uses of the utility.

The syntax:

Send Mode

```
mailx [-s subject] address...
```

Receive Mode

```
mailx -e
mailx [-HiNn] [-F] [-u user]
mailx -f[-HiNn] [-F] [file]
```

is coded as follows:

```
<refsect2>
<title>Send Mode</title>
<cmdsynopsis>
<command>mailx</command>
<arg><option role="dash">s </option><replaceable>subject</replaceable>
</arg>
<arg choice="plain" rep="repeat"><replaceable>address</replaceable>
</arg>
</cmdsynopsis>
</refsect2>
<refsect2>
<title>Receive Mode</title>
<cmdsynopsis>
<command>mailx</command>
<arg choice="plain"><option role="dash">e</option></arg>
</cmdsynopsis>
<cmdsynopsis>
<command>mailx</command>
<arg><option role="dash">HiNn</option></arg>
<arg conformance="extension"><option role="dash">F</option></arg>
<arg><option role="dash">u </option><replaceable>user</replaceable></arg>
</cmdsynopsis>
<cmdsynopsis>
```

```
<command>mailx</command>
<arg choice="plain"><option role="dash">f</option>
<arg><option role="dash">HiNn</option></arg>
</arg>
<arg conformance="extension"><option role="dash">F</option></arg>
<arg><replaceable>file</replaceable></arg>
</cmdsynopsis>
</refsect2>
```

D.3.2 Functions and Macros

The following example shows the synopsis for the *ftw*() function, including use of the <funcparams> tag.

The syntax:

```
#include <ftw.h>
int ftw(
    const char *path,
    int (*fn)(const char *, const struct stat *ptr,
    int flag),
    int ndirs);
```

is coded as follows:

```
<funcsynopsis conformance="extension">
<funcsynopsisinfo>#include &lt;ftw.h></funcsynopsisinfo>
<funcdef>int <function>ftw</function></funcdef>
<paramdef>const char *<parameter>path</parameter></paramdef>
<paramdef>int (*<parameter>fn</parameter>)
<funcparams>const char *, const struct stat *<parameter>ptr</parameter>,
int <parameter>flag</parameter></funcparams>
</paramdef>
<paramdef>int <parameter>ndirs</parameter></paramdef>
</funcsynopsis>
```

The following example shows the Synopsis of the *exec*() function, which includes multiple function synopses and an external variable definition.

The syntax:

```
#include <unistd.h>
int execl (
    const char * path,
    const char * arg0,
    ... /*,
    (char *)0 */);
int execle (
    const char * path,
    const char * arg0,
    ... /*,
    (char *)0
    char *const envp[] */);
int execlp (
    const char * file,
    const char * arg0,
```

```
         ... /*,
        (char *)0 */);
    int execv
        const char * path,
        char *const  argv[]);
    int execve (
        const char * path,
        char *const  argv[],
        char *const  envp[]);
    int execvp
        const char * file,
        char *const  argv[]);
    extern char **environ;
```

is coded as follows:

```
    <funcsynopsis>
    <funcsynopsisinfo>#include &lt;unistd.h></funcsynopsisinfo>
    <funcdef>int <function>execl</function></funcdef>
    <paramdef>const char *<parameter>path</parameter></paramdef>
    <paramdef>const char *<parameter>arg0</parameter></paramdef>
    <paramdef> ... /*</paramdef>
    <paramdef>(char *)0 */</paramdef>
    </funcsynopsis>
    <funcsynopsis>
    <funcdef>int <function>execle</function></funcdef>
    <paramdef>const char *<parameter>path</parameter></paramdef>
    <paramdef>const char *<parameter>arg0</parameter></paramdef>
    <paramdef> ... /*</paramdef>
    <paramdef>(char *)0</paramdef>
    <paramdef>char *const <parameter>envp</parameter>[]*/</paramdef>
    </funcsynopsis>
    <funcsynopsis>
    <funcdef>int <function>execlp</function></funcdef>
    <paramdef>const char *<parameter>file</parameter></paramdef>
    <paramdef>const char *<parameter>arg0</parameter></paramdef>
    <paramdef> ... /*</paramdef>
    <paramdef>(char *)0 */</paramdef>
    </funcsynopsis>
    <funcsynopsis>
    <funcdef>int <function>execv</function></funcdef>
    <paramdef>const char *<parameter>path</parameter></paramdef>
    <paramdef>char *const <parameter>argv</parameter>[]</paramdef>
    </funcsynopsis>
    <funcsynopsis>
    <funcdef>int <function>execve</function></funcdef>
    <paramdef>const char *<parameter>path</parameter></paramdef>
    <paramdef>char *const <parameter>argv</parameter>[]</paramdef>
    <paramdef>char *const <parameter>envp</parameter>[]</paramdef>
    </funcsynopsis>
    <funcsynopsis>
    <funcdef>int <function>execvp</function></funcdef>
    <paramdef>const char *<parameter>file</parameter></paramdef>
```

```
<paramdef>char *const <parameter>argv</parameter>[]</paramdef>
</funcsynopsis>
<synopsis>
<literal>extern char **<symbol>environ</symbol>;</literal>
</synopsis>
```

D.3.3 Headers

The following example shows a header synopsis. The syntax:

```
#include <regex.h>
```

is coded as follows:

```
<synopsis><literal>#include &lt;regex.h></literal></synopsis>
```

D.3.4 External Variables

The following example shows an external variable synopsis: This synopsis may follow a function synopsis, or may be provided by itself.

```
<synopsis>
<literal>extern char **<symbol>environ</symbol>;</literal>
</synopsis>
```

D.3.5 Sample Programs

The following example shows a sample program synopsis. The syntax:

```
c89 -o dbengine dbengine.c dbworks.c
```

is coded as follows:

```
<synopsis>
<literal>c89 -o dbengine dbengine.c dbworks.c</literal>
</synopsis>
```

D.4 DESCRIPTION

The following example shows the Description section for the *a64l*( ) function:

```
<refsect1 id="a64l-libr-desc">
<title>Description</title>
<para>The <function>a64l</function> and <function>l64a</function>
functions maintain numbers that are stored in radix-64
ASCII-character format. This format allows 32-bit integers to be
represented by a string of six or fewer characters.</para>
</refsect1>
```

D.5 OPTIONS

The following example shows the Options section for the *iconv* utility, which includes some vendor placeholders.

```
<refsect1 id=iconv-user-opts>
<title>Options</title>
<para>Systems that conform to the Single UNIX Specification
support the following options:</para>
<![ %VendorExtension; [&iconv-user-entity1;]]>
<variablelist>
<varlistentry>
<term><option role="dash">f </option>
<replaceable>fromcode</replaceable></term>
<listitem>
<para>Identifies the codeset of the input file.</para>
</listitem>
</varlistentry>
<varlistentry>
<term><option role="dash">t </option>
<replaceable>tocode</replaceable></term>
<listitem>
<para>Identifies the codeset of the output file.</para>
</listitem>
</varlistentry>
</variablelist>
<para>Codeset values may be defined differently among systems
that conform to the Single UNIX Specification.</para>
<![ %VendorExtension; [&iconv-user-entity2;]]>
</refsect1>
```

The following example shows the Options section for the *batch* utility, which has no options, but includes a vendor placeholder in case a vendor does define options for the utility.

```
<refsect1 id="batch-user>
<title>Options</title>
<para>No options are defined for this utility by the Single UNIX
Specification.</para>
<![ %VendorExtension; [&batch-user-entity1;]]>
</refsect1>
```

The following example shows part of an Options section for a utility that does not always follow the standard utility syntax guidelines.

```
<refsect1 id="uniq-user">
<title>Options</title>
<para>This utility supports the utility syntax guidelines
described in the xbdutsyntax(5) reference page, except that:</para>
<itemizedlist conformance="obsolescent">
<listitem>
<para>The obsolescent usage of <command>uniq</command> does
not conform to the utility syntax guidelines, because
of the following non-standard syntax elements:</para>
<itemizedlist>
<listitem>
```

```
<para>One of the options begins with a plus sign
(<literal>+</literal>).</para>
</listitem>
<listitem>
<para>The <option role="dash"></option><replaceable>m</replaceable>
and <option role="plus"></option><replaceable>n</replaceable>
options do not have option letters.</para>
</listitem>
</itemizedlist>
</listitem>
</itemizedlist>
<para>Systems that conform to the Single UNIX Specification
support the following options:</para>
<replaceable>list-of-options</replaceable>
</refsect1>
```

The following example shows how to code an option that has an option-argument—note the space typed inside the <option> tag.

```
<option role="dash">f </option><replaceable>filename</replaceable>
```

D.6 OPERANDS

The following example shows the Operands section for the *uniq* utility.

```
<refsect2 id="uniq-user-oper">
<title>Operands</title>
<para>Systems that conform to the Single UNIX Specification
support the following operands:</para>
<![ %VendorExtension; [&uniq-user-entity1;]]>
<variablelist>
<varlistentry>
<term><replaceable>input-file</replaceable></term>
<listitem>
<para>Specifies the path name of the text file used for
input. If no <replaceable>input-file</replaceable>
operand is specified, or if the input file is specified
using a hyphen (<literal>-</literal>), standard input is
used.</para>
</listitem>
</varlistentry>
<varlistentry>
<term><replaceable>output-file</replaceable></term>
<listitem>
<para>Specifies the path name of the output
text file. If no <replaceable>output-file</replaceable>
operand is specified, standard output is used. If
the file named by <replaceable>output-file</replaceable>
is the same file named by <replaceable>input-file</replaceable>,
results may vary among systems that conform to the Single UNIX
Specification.</para>
<![ %VendorExtension; [&uniq-user-entity2;]]>
</listitem>
```

```
</varlistentry>
</variablelist>
<![ %VendorExtension; [&uniq-user-entity3;]]>
</refsect2>
```

D.7 PARAMETERS

The following example shows the Parameters section for the *putc*( ) function.

```
<refsect1 id="putc-libr-parm">
<title>Parameters</title>
<para>The parameter descriptions follow in alphabetical order:</para>
<variablelist>
<varlistentry>
<term><parameter>stream</parameter></term>
<listitem>
<para>Points to the file structure of an open file.</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

D.8 EXTENDED DESCRIPTION

The following example shows the coding for second and third-level sections.

```
<refsect1 id="tr-user-exde">
<title>Extended Description</title>
<para>text</para>
<para>The following sections provide additional information
for the <command>tr</command> utility:</para>
<itemizedlist>
<listitem>
<para><xref linkend="tr-user-exde-opts"></para>
</listitem>
<listitem>
<para><xref linkend="tr-user-exde-stri"></para>
</listitem>
</itemizedlist>
<refsect2 id="tr-user-exde-opts">
<title>Interactions Among Options</title>
<para>text</para>
</refsect2>
<refsect2 id="tr-user-exde-stri">
<title>Translation Control Strings</title>
<para>text</para>
<refsect3>
<title>Terminal Constructs</title>
<para>text</para>
</refsect3>
<refsect3>
```

```
<title>Non-Terminal Constructs</title>
<para>text</para>
</refsect3>
</refsect2>
</refsect1>
```

D.9 EXIT STATUS

The following example shows an Exit Status section:

```
<para>This utility returns the following exit values:</para>
<variablelist>
<varlistentry>
<term><returnvalue>0</returnvalue></term>
<listitem><para>Successful completion: all input files were
processed.</para>
</listitem>
</varlistentry>
<varlistentry>
<term>&gt;<returnvalue>0</returnvalue></term>
<listitem><para>An error occurred.</para>
</listitem>
</varlistentry>
</variablelist>
```

D.10 RETURN VALUES

The following example shows a Return Values section.

```
<refsect1 id="fflush-libr-rtrn">
<para>The <function>fflush</function> function returns the
following:</para>
<variablelist>
<varlistentry>
<term><returnvalue>0</returnvalue></term>
<listitem>
<para>Success.</para>
</listitem>
</varlistentry>
<varlistentry>
<term><returnvalue>&minus;1</returnvalue></term>
<listitem>
<para>Failure: <symbol>errno</symbol> is set to indicate the error.</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

D.11 ERRORS

The following example shows an Errors section:

```
<para>On all systems that conform to the Single UNIX Specification,
the msgrcv() function sets errno as listed for the following conditions:
</para>
<variablelist>
<varlistentry>
<term><systemitem role="errno">EFAULT</systemitem></term>
<listitem><para>The <parameter>msgp</parameter> argument
points to an illegal address.</para>
</listitem>
</varlistentry>
<varlistentry>
<term><systemitem role="errno">EINTR</systemitem></term>
<listitem><para>The <parameter>msgp</parameter> argument
points to a user address.</para>
</listitem>
</varlistentry>
</variablelist>
```

D.12 EXAMPLES

The following example shows how to code the Examples section for a utility:

```
<refsect1 id="tr-user-exam">
<title>Examples</title>
<example id="tr-user-exam-1">
<title>Listing the Words on a Line</title>
<para>text</para>
<screen><userinput>
tr -cs "[:alpha:]" "[0]" &lt;file1 >file2
</userinput></screen>
<![ %VendorComputerOutput; [&tr-user-output1;]];>
</example>
<example id="tr-user-exam-2">
<title>Translating Lowercase to Uppercase Characters</title>
<para>text</para>
<screen><userinput>
tr "[:lower:]" "[:upper:]" &lt;file1
</userinput></screen>
<![ %VendorComputerOutput; [&tr-user-output2;]]>
</example>
</refsect1>
```

The following example shows how to code the Examples section for a programming interface:

```
<refsect1 id="close-sysc-exam">
<title>Examples</title>
<example id="close-sysc-exam-1">
<title>Reassigning a File Descriptor</title>
<para>text</para>
<programlisting>
```

```
#include &lt;unistd.h>
&vellip;
int pfd;
&vellip;
close (1);
dup (pfd);
close (pfd);
&vellip;
</programlisting>
</example>
<example id="close-sysc-exam-2">
<title>Closing a File Descriptor</title>
<para>text</para>
<programlisting>
#include &lt;stdio.h>
#include &lt;unistd.h>
#include &lt;stdlib.h>

#define LOCKFILE "/etc/ptmp"
&vellip;
int pfd;
FILE *fpfd;
&vellip;
if ((fpfd = fdopen (pfd, "w")) == NULL) {
   close (pfd);
   unlink(LOCKFILE);
   exit (1);
}
&vellip;
</programlisting>
</example>
</refsect1>
```

D.13 ENVIRONMENT VARIABLES

The following example shows an Environment Variables section:

```
<refsect1 id="lp-user-envr">
<title>Environment Variables</title>
<para>The environ(5) reference page provides general information
about the following standard environment variables, which can
affect the operation of this utility: LANG, LC_ALL, LC_CTYPE,
LC_MESSAGES, LC_TIME, LP_DEST, and NLSPATH.</para>
<para>Some environment variables interact with specific features
of this utility, as follows:</para>
<variablelist>
<varlistentry conformance="extension">
<term><systemitem class="environvar">LC_TIME</systemitem></term>
<listitem><para>[descriptive text]</para>
</listitem>
</varlistentry>
<varlistentry>
```

```
<term><systemitem class="environvar">LPDEST</systemitem></term>
<listitem><para>[descriptive text]</para>
<![ %VendorExtension; [&lp-user-entity12;]]>
</listitem>
</varlistentry>
<varlistentry>
<term><systemitem class="environvar">PRINTER</systemitem></term>
<listitem><para>[descriptive text]</para>
<![ %VendorExtension; [&lp-user-entity4;]]>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

D.14 FILES

The following example shows a Files section:

```
<refsect1 id="mycron-user">
<title>Files</title>
<para>The following files are used by this utility:</para>
<variablelist>
<varlistentry>
<term><filename>/etc/default/mycron</filename></term>
<listitem><para>Contains default settings.</para>
</listitem>
</varlistentry>
<varlistentry>
<term><filename>/etc/group</filename></term>
<listitem><para>Lists group IDs for the
<userinput>ls -l</userinput>  and <userinput>ls -g</userinput>
commands.</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

D.15 SEE ALSO

The following example shows how to code a reference to another reference page and an external document:

```
<refsect1 id="iconv-user-also">
<title>See Also</title>
<para>
<link linkend="gencat-user"><citerefentry>
<refentrytitle>gencat</refentrytitle><manvolnum>&user;</manvolnum>
</citerefentry></link>,
<link linkend="environ-misc"><citerefentry>
<refentrytitle>environ</refentrytitle><manvolnum>&misc;</manvolnum>
</citerefentry></link>
</para>
<para>
<citetitle>ISO 6937:1983, Latin Alphabet No. 1</citetitle>
</para>
</refsect1>
```

Style Guide for Technical Publications

Part 3:
Documentation Tools

The Open Group

Chapter 8

Documentation Tools

This chapter describes the various tools used by The Open Group to manage document source.

8.1 Source Control

8.1.1 SCCS

SCCS is used for archiving and document version control.

To use SCCS, each file should contain the following string definition as its second line:

```
.ds SI %Z% %I% %E%
```

It must not be changed since these are SCCS keywords. When the file is extracted from SCCS, SCCS substitutes values for `%Z% %I% %E%`, indicating the exact version of the file. This feature is used to identify drafts built from a specific SCCS version.

8.1.2 ODE

TBD.

8.1.3 Clearcase

TBD.

8.2 Bug-tracking

8.2.1 OT

TBD.

8.2.2 DTTS

TBD.

8.2.3 Corrigenda

The Corrigenda process is used to publish known errors or omissions from published Open Group documents.

All corrigenda against the same specification are collected together, so that there is one file per document, organized as follows:

- New and any previous corrigenda text is provided in reverse chronological order.
- Each section is clearly marked and dated.
- A new document number is allocated to each revision.
- A list of superseded corrigenda items is provided at the start of the corrigendum file.

The following template should be used as far as possible:

```
Corrigendum:    Unnn
Date:           <date>
Document:       <Doc. No.>
                <Document Title>
Code:           <no. of bytes> <date> <doc no.>/Unnn
Contents:       This corrigendum incorporates: Unnn (<date>)
-----------------------------------------------------------------
Change Number:  Unnn/1
Title:
Qualifier:
Rationale:
Change:
-----------------------------------------------------------------
Change Number:  Unnn/2
Title:
Qualifier:
Rationale:
Change:
-----------------------------------------------------------------
-----------------------------------------------------------------
Start of Corrigendum Unnn (<date>).
-----------------------------------------------------------------
Change Number:  Unnn/1
Title:
Qualifier:
Rationale:
```

```
Change:
-----------------------------------------------------------------
```

8.3 Difference Marking Between Versions

At any time, it should be possible to display differences between two different versions of a document. (For example, by using change bars.)

For how to show change bars in *troff*, refer to **Build Files** on page 96.

Change bars are not used in published documents.

8.4 Conversion Programs

8.4.1 troff-to-HTML

To follow.

8.4.2 SGML-to-troff

SGML source, coded in accordance with this Style Guide, can be converted to *troff*. You will need to obtain a series of files from The Open Group to use this conversion.

To invoke the SGML-to-troff converter, type:

```
sgm_r <list> <character entities> <general entities> <errors>
```

where:

<list>
: is a file listing the SGML files to be processed, one file per line, with the .sgm extension omitted from each filename.

<character entities>
: is a file containing character entity definitions.

<general entities>
: is a file containing other entity definitions.

<errors>
: is the name of a file to which the converter is to write error messages.

For example, to convert files **a.sgm**, **b.sgm**, and **c.sgm** to *troff*, using character entities defined in **r_lits.ent** and other entities defined in **xosudent.ent**, create a file called **listfile** with the following contents:

```
a
b
c
```

and type:

```
sgm_r listfile r_lits.ent xosudent.ent errfile
```

Files **a.r**, **b.r**, and **c.r** will be created containing the *troff* output, and any error messages will be written to file **errfile**.

Style Guide for Technical Publications

Part 4:
Presentation

The Open Group

Chapter 9

Output Formats

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

This part is intended to document the output formats for:

- Hard copy (regardless of tagging used):
    1. Specifications: document X/Open specification output style
    2. Product Documentation: document DCE FOSI
- Soft copy (HTML):
    1. All documents: document output from troff->HTML conversion