Internal Document

Open Group Technical Publications Documentation Tools

The Open Group

Internal Document

Open Group Technical Publications Documentation Tools

ISBN: N/A
Document Number: I801B (Documentation Tools)

Published in the U.K. by The Open Group, December 1998.

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

List of Tables

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 describes The Open Group documentation tools for technical publications.

It should be followed to ensure a unified approach to producing document source.

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 do not.

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.

Trademarks

Referenced Documents

The following documents are referenced in this guide:

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.

Writing Reference Pages: A Style Guide for the Single UNIX Documentation, The Open Group Common Product Documentation Group, The Open Group, 1997, I701.

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/.

Chapter 1

Introduction

This chapter describes the various tools used by The Open Group to manage document source.

1.1 SGML Coding

How to code documents using SGML is documented in Chapter 2 on page 5.

1.2 troff Coding

How to code documents using *troff* is documented in Chapter 3 on page 33.

1.3 Source Control

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.

1.4 Bug-Tracking

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:
------------------------------------------------------------------
```

1.5 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 34.

Change bars are not used in published documents.

1.6 Conversion Programs

1.6.1 troff-to-HTML

The Open Group has a *troff*-to-HTML conversion program which is managed in Reading. For Reading editors, the instructions are located in xopubs:/usr/local/lib/tr2htmproc.txt.

1.6.2 SGML-to-troff

The Open Group has an SGML-to-*troff* conversion program which is managed in Reading. For Reading editors, the program source is held on xolit.

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**.

1.6.3 PostScript-to-GIF

The Open Group has a PostScript-to-GIF conversion program which is managed in Reading. For Reading editors, the program source is held in xopuk:/xopen/PCSHARE2/ps2gif.

To run ps2gif, place the postscript file(s) in the /in directory. The program will handle normal and encapsulated files and does not care what the files are called—.ps, .eps, .PS, and so on). Also add a .dat file which contains your email address plus a list of the postscript files on separate lines.

The .gif files will be placed in the /out directory and you will receive an email on completion.

Chapter 2

Using SGML

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

Note: Since this chapter was derived from the Style Guide developed for the Single UNIX Documentation Common Product Documentation project (see **Referenced Documents** on page xii), it is mainly concerned with reference pages. There are several sections marked "TBD" for this reason.

2.1 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.

2.2 SGML Coding

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

2.2.1 Front Matter

Title and Copyright Pages

TBD.

Preface

TBD.

Trademarks

TBD.

Referenced Documents

TBD.

2.2.2 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>
```

2.2.3 Changebars

TBD.

2.2.4 Comments

TBD.

2.2.5 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.

2.2.6 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.

2.2.7 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.

2.2.8 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.

2.2.9 Footnotes

TBD.

2.2.10 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.

2.2.11 Graphics

TBD.

Figure Titles

TBD.

2.2.12 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.

2.2.13 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.

2.2.14 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.

2.2.15 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.

2.2.16 Part Pages

TBD.

2.2.17 External References

The <citetitle> tag identifies a document title.

2.2.18 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**.

2.2.19 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”.

2.2.20 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.

2.2.21 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.

2.2.22 Warnings

TBD.

2.3 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"

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.

2.3.1 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”.

2.3.2 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.

2.3.3 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 2-1 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

2.3.4 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>
```

2.3.5 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.

2.3.6 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.
```

2.3.7 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.

2.3.8 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 A on page 55.)

— <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:

```
<refsect1 id="refentryid-divid-also">
<title>See Also</title>
<para>
<link linkend="refentryid-divid">
<citerefentry><refentrytitle>primaryname</refentrytitle>
<manvolnum>&divid;</manvolnum></citerefentry></link>, ...
</para>
<para>
<citetitle>full document title</citetitle>, ...
</para>
</refsect1>
```

Chapter 3

Using troff

This chapter contains instructions for building a document using the Open Group build procedure, and how to code document elements in *troff*.

3.1 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.

3.2 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 3.4.1 on page 37.

The order of the front matter files is defined by the contents of the file _vfiles (see **Build Files** on page 34).

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 3.4.23 on page 51.

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 3.3 on page 35). You must edit your copy of the _vars file to give the required SCCS version number of these files for the *CBSI* 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).

3.3 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.

3.4 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.

3.4.1 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.

3.4.2 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
```

3.4.3 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.

3.4.4 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
```

3.4.5 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
`.xR 12`	Unnumbered section (.Hi); for example, Glossary

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`.

3.4.6 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.

3.4.7 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.

3.4.8 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.

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 `\*?`.

3.4.9 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
```

The `.Ur` macro can also be used for email addresses. The first argument is the email address in double quotes. The second argument is the text “mailto:“ followed by the same email address.

3.4.10 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.

3.4.11 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.

3.4.12 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.

3.4.13 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.

3.4.14 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.

3.4.15 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.

3.4.16 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`.

3.4.17 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.

3.4.18 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.

3.4.19 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 Internal Document
.ds XP Part 2:
.ds Xp Title of Part
.ds XV Part\t2
.Tl
.eF e
```

3.4.20 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.

3.4.21 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.

3.4.22 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.

3.4.23 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:

IM 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.

3.4.24 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.

3.4.25 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, ˆ, 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.

3.4.26 Tabs

Tab stops are set by using the `.ta` command:

Note that *tbl* in a display resets tab stops for the following display.

3.4.27 Text

Normal text is broken into paragraphs with the `.P` macro.

3.4.28 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
```

3.5 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

3.6 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 A

SGML Examples

A.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>
```

A.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>
```

A.3 SYNOPSIS

A.3.1 Utilities

The following example shows a utility synopsis with an arbitrary 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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

A.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>
```

Appendix B

SUD-Specific SGML

Note: This appendix documents the SGML coding used specifically for the Single UNIX Specification Common Documentation project. Further detail can be found in the Style Guide for this project, which is listed in **Referenced Documents** on page xii.

B.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.

B.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]

B.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*.

B.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).

B.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.

B.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.

B.7 Equations

Vendor placeholders should be placed within marked sections using the %VendorEquation; parameter entity.

B.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.

B.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").

B.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:"

EXAMPLES

Use the &xshexamsection; text entity for the title.

Add a vendor placeholder for command and utility examples, so that each vendor can show the output produced by the command. Use the &VendorComputerOutput; marked section.

ENVIRONMENT VARIABLES

Use the &xshenvrsection; text entity for the title.

A description of the standard environment variables is given in the *environ*( ) reference page; they are:

COLUMNS	LC_COLLATE	LC_TIME	PATH
DATEMSK	LC_CTYPE	LINES	SHELL
HOME	LC_MESSAGES	LOGNAME	TMPDIR
LANG	LC_MONETARY	MSGVERB	TERM
LC_ALL	LC_NUMERIC	NLSPATH	TZ

The following text entities can be used in this section:

- &envrleadstd;

  Use for reference pages that include references to one of the standard environment variables listed above, but do not include any information that is specific to the <refname>.

  This translates to: ‘‘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.’’

- &envrleadother;

  Use to provide supplementary information for a particular variable if the generic description is not sufficient. (Do not repeat information that is included in the environ(5) reference page.)

  This translates to: ‘‘Some environment variables interact with specific features of this utility, as follows:’’

Vendor placeholders may be added to allow additional vendor-specific information.

FILES

Use the &xshfilesection; text entity for the title.

The following text should be used to introduce a list of files: ‘‘The following files are used by this utility:’’

SEE ALSO

Use the &xshalsosection; text entity for the title.

For references to other reference pages, code them as follows:

```
<link linkend="environ-misc"><citerefentry>
<refentrytitle>environ</refentrytitle><manvolnum>&misc;</manvolnum>
</citerefentry></link>
```

For references to external documents, use text entities.

B.11 Reference Page Templates

B.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>
```

```
</citerefentry></link>, <link linkend="myheader.h-file"><citerefentry>
<refentrytitle>myheader.h</refentrytitle><manvolnum>&file;</manvolnum>
</citerefentry></link>, <link linkend="myprogram.c-exmp"><citerefentry>
<refentrytitle>myprogram.c</refentrytitle><manvolnum>&exmp;</manvolnum>
</citerefentry></link></para>
<para><citetitle>Document Title</citetitle></para>
</refsect1>
</refentry>
```

B.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>
```

```
<para>&envrleadstd;
<systemitem class="environvar">EnvironmentVariables</systemitem>
</para>
<para>&envrleadother;</para>
<variablelist>
<varlistentry>
<term><systemitem class="environvar">EnvironmentVariable
</systemitem></term>
<listitem>
<para></para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
<refsect1 id="myutil-user-file">
<title>&xshfilesection;</title>
<para></para>
</refsect1>
<refsect1 id="myutil-user-also">
<title>&xshalsosection;</title>
<para><link linkend="myother-util"><citerefentry>
<refentrytitle>myother</refentrytitle><manvolnum>&user;</manvolnum>
</citerefentry></link>, <link linkend="myfunc-sysc"><citerefentry>
<refentrytitle>myfunc</refentrytitle><manvolnum>&sysc;</manvolnum>
</citerefentry></link>, <link linkend="myheader.h-file"><citerefentry>
<refentrytitle>myheader.h</refentrytitle><manvolnum>&file;</manvolnum>
</citerefentry></link>, <link linkend="environ-misc"><citerefentry>
<refentrytitle>environ</refentrytitle><manvolnum>&misc;</manvolnum>
</citerefentry></link>, <link linkend="xbdusyntax-misc"><citerefentry>
<refentrytitle>xbdusyntax</refentrytitle><manvolnum>&misc;</manvolnum>
</citerefentry></link>, <link linkend="myprogram.c-exmp"><citerefentry>
<refentrytitle>myprogram.c</refentrytitle><manvolnum>&exmp;</manvolnum>
</citerefentry></link></para>
<para><citetitle>Document Title</citetitle></para>
</refsect1>
</refentry>
```

B.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>
```

```
</refsynopsisdiv>
<refsect1 id="myfunc-sysc-desc">
<title>&xshdescsection;</title>
<para></para>
</refsect1>
<refsect1 id="myfunc-sysc-parm">
<title>&xshparmsection;</title>
<para>&newparmlead;</para>
<variablelist>
<varlistentry>
<term><parameter>ParameterName</parameter></term>
<listitem>
<para>Indicates</para>
</listitem>
</varlistentry>
<varlistentry>
<term><parameter>PointerName</parameter></term>
<listitem>
<para>Points to</para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
<refsect1 id="myfunc-sysc-exde">
<title>&xshexdesection;</title>
<para></para>
</refsect1>
<refsect1 id="myfunc-sysc-rtrn">
<title>&xshrtrnsection;</title>
<para>The <function>myfunc</function> function &rtrnlead;</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>
<varlistentry>
<term>Nonzero value</term>
<listitem>
<para></para>
</listitem>
</varlistentry>
</variablelist>
</refsect1>
```

```
<refsect1 id="myfunc-sysc-erro">
<title>&xsherrosection;</title>
<para>&willfail1; <function>myfunc</function> function &willfail2;</para>
<variablelist>
<varlistentry>
<term><systemitem role="errno">ERROR_CODE</systemitem></term>
<listitem>
<para></para>
</listitem>
</varlistentry>
</variablelist>
<para>&mayfail1; <function>myfunc</function> function &mayfail2;</para>
<variablelist>
<varlistentry>
<term><systemitem role="errno">ERROR_CODE</systemitem></term>
<listitem>
<para></para>
</listitem>
</varlistentry>
</variablelist>
<![ %VendorExtension; [&myfunc-sysc-entity1;]]>
</refsect1>
<refsect1 id="myfunc-sysc-exam">
<title>&xshexamsection;</title>
<example id="myfunc-sysc-exam-1">
<title>Title Text</title>
<para>The following example...</para>
<para>&compexp1; <link linkend="examplename-exmp"><citerefentry>
<refentrytitle>examplename.c</refentrytitle><manvolnum>&exmp;</manvolnum>
</citerefentry></link> &compexp2;</para>
<programlisting>#include &lt;header-name>
&vellip;
variable declarations for the function calls
function calls</programlisting>
</example>
</refsect1>
<refsect1 id="myfunc-sysc-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>
</citerefentry></link>, <link linkend="myheader.h-file"><citerefentry>
<refentrytitle>myheader.h</refentrytitle><manvolnum>&file;</manvolnum>
</citerefentry></link>, <link linkend="myprogram.c-exmp"><citerefentry>
<refentrytitle>myprogram.c</refentrytitle><manvolnum>&exmp;</manvolnum>
</citerefentry></link></para>
<para><citetitle>Document Title</citetitle></para>
</refsect1>
</refentry>
```

B.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>
```

```
<listitem>
<para><xref linkend="myheader.h-file-desc-daty"></para>
</listitem>
<listitem>
<para><xref linkend="myheader.h-file-desc-extv"></para>
</listitem>
<listitem>
<para><xref linkend="myheader.h-file-desc-funp"></para>
</listitem>
<listitem>
<para><xref linkend="myheader.h-file-desc-macr"></para>
</listitem>
<listitem>
<para><xref linkend="myheader.h-file-desc-stru"></para>
</listitem>
</itemizedlist>
<refsect2 id="myheader.h-file-desc-cons">
<title>Constants</title>
<para>The <filename class="headerfile">my/header.h</filename>
header defines the following constants:</para>
<variablelist>
<varlistentry>
<term><systemitem class="constant">CONSTANT_NAME</systemitem></term>
<listitem>
<para></para>
</listitem>
</varlistentry>
</variablelist>
</refsect2>
<refsect2 id="myheader.h-file-desc-daty">
<title>Data Types</title>
<para>The <filename class="headerfile">my/header.h</filename>
header defines the following data types:</para>
<variablelist>
<varlistentry>
<term><literal>DataType</literal></term>
<listitem>
<para></para>
</listitem>
</varlistentry>
</variablelist>
</refsect2>
<refsect2 id="myheader.h-file-desc-extv">
<title>External Variables</title>
<para>The following are declared as external variables:</para>
<variablelist>
<varlistentry>
<term><literal>DataType VariableName</literal></term>
<listitem>
<para></para>
</listitem>
</varlistentry>
```

```
</variablelist>
</refsect2>
<refsect2 id="myheader.h-file-desc-funp">
<title>Function Prototypes</title>
<para>&funcmacro;</para>
<![ %VendorExtension; [&myheader.h-file-entity1;]]>
<funcsynopsis>
<funcdef>DataType <function>funcname1</function></funcdef>
<paramdef>ParameterDataType <parameter>Parameter</parameter></paramdef>
</funcsynopsis>
<funcsynopsis>
<funcdef>void <function>funcname2</function></funcdef>
<void>
</funcsynopsis>
<funcsynopsis>
<funcdef>struct StructureName *<function>funcname3</function></funcdef>
<paramdef>ParameterDataType <parameter>Parameter</parameter></paramdef>
</funcsynopsis>
<![ %VendorExtension; [&myheader.h-file-entity2;]]>
</refsect2>
<refsect2 id="myheader.h-file-desc-macr">
<title>Macros</title>
<para>The <filename class="headerfile">my/header.h</filename>
header defines the following macros:</para>
<programlisting>
DataTypeWithoutDesc
<systemitem class="macro">MacroName</systemitem>
(ParameterDataType <parameter>Parameter</parameter>);
</programlisting>
<variablelist>
<varlistentry>
<term><literal>DataTypeWithDesc
<systemitem class="macro">MacroName</systemitem>
(ParameterDataType <parameter>Parameter</parameter>);
</literal></term>
<listitem>
<para>Description of Data Type</para>
</listitem>
</varlistentry>
</variablelist>
</refsect2>
<refsect2 id="myheader.h-file-desc-stru">
<title>Structures</title>
<para>The <filename class="headerfile">my/header.h</filename>
header defines the following structures:</para>
<variablelist>
<varlistentry>
<term><structname>StructureName</structname></term>
<listitem><para>The <structname>StructureName</structname>
structure includes at least the following members:</para>
<variablelist>
<varlistentry>
```

```
<term><literal>MemberDataType</literal>
<structname>DifferentStructureNameIfApplicable</structname>
<structfield>StructureMemberOrField</structfield></term>
<listitem>
<para></para>
</listitem>
</varlistentry>
</variablelist>
</listitem>
</varlistentry>
</variablelist>
</refsect2>
</refsect1>
<refsect1 id="myheader.h-file-file">
<title>&xshfilesection;</title>
<para></para>
</refsect1>
<refsect1 id="myheader.h-file-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>
</citerefentry></link>, <link linkend="myprogram.c-exmp"><citerefentry>
<refentrytitle>myprogram.c</refentrytitle><manvolnum>&exmp;</manvolnum>
</citerefentry></link>, <link linkend="mydesc-misc"><citerefentry>
<refentrytitle>mydesc</refentrytitle><manvolnum>&misc;</manvolnum>
</citerefentry></link></para>
<para></para>
<para><citetitle>Document Title</citetitle></para>
</refsect1>
</refentry>
```

B.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>
```

```
</variablelist>
<![ %VendorExtension; [
<!--VENDOR ADDITION: Use the entity following each item in this list to
identify the location of the sample file on your UNIX system.-->
&myprog.c-exmp-entity1;]]>
</refsect1>
<refsect1 id="myprog.c-exmp-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>
</citerefentry></link>, <link linkend="myheader.h-file"><citerefentry>
<refentrytitle>myheader.h</refentrytitle><manvolnum>&file;</manvolnum>
</citerefentry></link>, <link linkend="mydesc-misc"><citerefentry>
<refentrytitle>mydesc</refentrytitle><manvolnum>&misc;</manvolnum>
</citerefentry></link></para>
<para><citetitle>Document Title</citetitle></para>
</refsect1>
</refentry>
```

Appendix C

DCE-Specific SGML

This appendix is provided for information only. It provides an alphabetical list of the DocBook elements that were used in the DCE Product Documentation developed by OSF. It also includes some formatting information which is specific to the FOSI developed for DCE Product Documentation.

Where an element is not to be processed, but is included for future reference, ‘‘Not used in DCE 1.2’’ appears at the end of the description.

Where a description is borrowed from the *Guide to the DocBook DTD*, the words ‘‘DocBook Guide’’ are appended in parentheses.

<Abstract>

This is the abstract of the book intended for use by browsing systems. Not used in DCE 1.2.

<Ackno>

Not used in DCE 1.2.

<Acronym>

There is no particular processing for this element. (That is, it is unchanged from its parent element.) A local application could check acronyms used against a database of known acronyms for correctness.

<Address>

When keydata is in running text, an address has no special presentation. The project FOSI for DCE 1.2 will generate the address of OSF automatically. However, the OSF address will be contained within <BookBiblio> information for use with other formatting tools.

<Anchor>

An Anchor marks a target for a Link. It requires no special processing in its own right. See the <XRef> element. Not used in DCE 1.2.

<Appendix>

An appendix begins on a new, recto page. In a book with parts, appendices are supplemental to the *entire* book, not a single part.

<Arg>

This is an argument in a command synopsis. An argument can have several parts, such as an option name and variable, or a single part. The content of this is in the child elements, not the element itself. Attributes of this element control various aspects of its presentation. See the <CmdSynopsis> element description for processing instructions of <Arg>.

<Author>

<CorpAuthor> will be used instead of <Author>, since OSF is the author of all DCE docset books.

<AuthorInitials>

Not used in DCE 1.2. (Typically used with <RevHistory>.)

<BiblioEntry>

An entry in a bibliography. Not used in DCE 1.2.

<Bibliography>

A bibliography. Not used in DCE 1.2.

<BlockQuote>

A <BlockQuote> is running text within a paragraph-like element; that is, it is set off vertically from previous and following paragraphs. It is indented 6mm, and right and left-justified. It is surrounded by the proper style quote marks. OSF style is double quotes.

In DCE 1.2, <BlockQuote> is always starts with <Para>.

<Book>

No processing in its own right; it semantically encapsulates a single book. The <Book role="publish"> attribute setting will cause the publish look to be formatted. The <Book role="draft"> setting will cause the book draft look to be formatted.

<BookBiblio>

Information about the author (OSF), including <CorpAuthor>, <Publisher>, its <Address>, and <Copyright> should be kept in <BookBiblio> (though the FOSI will generate the name and address information automatically on the title page). It is still kept in <BookBiblio> for other formatting tools to use.

<BookInfo>

The Contents attribute will not be needed. The order of elements within the book can define the order of the sections in the output, making the Contents attribute unnecessary.

<BridgeHead>

The Renderas attribute typically defines which section format (that is, <Sect1>, <Sect2>, and so on) to use. Not used in DCE 1.2.

<Caution>

A <Caution> is set off from the running text by a blank line of vertical space and is automatically labeled with the word Caution, just as <Note> is (by the word Note).

<Chapter>

A chapter begins on a new, recto page. There is a horizontal rule spanning the live area 60mm from the top margin. The word Chapter followed by the chapter number is above the line, right-justified. Chapter titles are left-justified in the live area just below the line. The point size is 24.

<CiteRefEntry>

A reference to an external <RefEntry> should be rendered in bold.

<CiteTitle>

A non-link, non-cross-reference citation of a work should be rendered in italics.

<CmdSynopsis>

OSF uses traditional UNIX-style syntax notation, which is described in the DocBook Guide. A command synopsis is a container for other synopsis elements. It is set off by one blank line before and after it. It is left-justified in the text area. The DocBook Guide gives a detailed explanation of the elements and attributes used within <CmdSynopsis>. OSF's processing instructions follow those in the DocBook Guide.

<CO>

ArborText did not have processing capability for this at the time of DCE publication.

<Command>

A command name appears in bold.

<Comment>

Commentary about some content of a document. These generally do not appear in final published output, but are used during development of the document and associated software. During development drafts, the appearance stands out on the page so that it is not confused with the running text (for example, boxed text or small print at the bottom of a page).

<ComputerOutput>

Computer generated text, as would be seen on a terminal, is presented in (constant-width) Courier.

<Copyright>

This is a container for individual copyright notices. The copyrights appear after the title page, with other meta information. The font size should be smaller than the 10 points default for running text in the main book.

The primary use of <Copyright> will be for OSF's copyright to the books. The list of other copyright holders (that is, those who have contributed to the books) will use <Trademark class="Copyright"> to produce the copyright glyph, but need not tag each with <Copyright>.

<CorpAuthor>

Appearing within <AuthorGroup>, the author of OSF books is OSF's full name, Open Software Foundation, to be right-justified on the title page, 4cm below the Release number line. None of the other author information items are used. Since OSF is always the author, its address will be generated on the title page.

<Date>

For the draft look, the date the book is formatted will be generated, placed in the running footers. Not used in DCE 1.2.

<DocInfo>

Container for book meta information.

<Edition>

Used for Release number on title page. Contents appear on title page, preceded by word Release. (Formerly, the title page has called this the Revision number, but it actually refers to the Release the book documents.)

<Email>

Rendered as (constant-width) Courier. A local application could check e-mail addresses for validity.

<Emphasis>

Text marked for emphasis is presented in italics.

<EndChange>

An extension to the DocBook 2.4.1 DTD. See <StartChange> and <Book>.

<Entry>

A cell in a table. See the DocBook Guide for an extensive description.

<EntryTbl>

A form of a subtable. See the DocBook Guide for an extensive description.

<Equation>

Equations should be set as other paragraph-like elements, centered within the text area, with paragraph spacing above and below.

<Example>

Used for blocks of program source code. Typically contains a <Title> and a <ProgramListing>. Character spacing and line feeds are preserved. The title is, like captions in figures and tables, Helvetica.

<Figure>

Figures are centered in the text area. If the figure does not fit, it is left-justified in the live area. (See discussion of role="column-wide" below.) Captions, in Helvetica, appear above the figure, and are left-justified in the live area, as section titles are. Figure titles are preceded by the word Figure and the chapter and figure number in that chapter separated by an en-dash. If a figure is just simple text, such as a part of a file listing, it may break across pages. Graphics do not break at page boundaries.

The default attribute setting for how the Figure will be centered is <Figure role="column-wide">. Figures that are too large for this setting will be adjusted in the source to <Figure role="page-wide">. If <Figure role="page-wide">, then the output will set in the text area plus the width of the scanning column. If the default (<Figure role="column-wide">) is used, the output will set in the text area.

<Filename>

A filename appears in bold. The <Filename MoreInfo="RefEntry"> attribute indicates that there is a RefEntry for this filename.

<FirstTerm>

The first occurrence of an important term in the document is italicized.

<Footnote>

The body of a footnote appears at the bottom of the page from which it is referenced. A left-justified, 3-cm rule appears between the running text and the footnote, and the footnote text is in 7 point text. Footnotes are numbered. The number appears as a superscript where the <FootnoteRef> containing the footnote ends. Long footnotes may spill over to the next page. When this happens, the rule on the following page is the width of the text area.

<FootnoteRef>

Identifies location (indicated by the Linkend attribute) for a footnote mark (a superscripted number). See <Footnote>.

<ForeignPhrase>

Text marked as being a phrase in foreign language is presented in italic.

<FuncDef>

Bold. An open parenthesis should follow the content of this element.

<FuncParams>

Bold. All but the last in a sequence of <FuncParams> must be followed by a comma, and the output line broken (that is, a new line begun).

<FuncSynopsis>

The layout of its contents mimics that of a UNIX reference page, with spacing, parentheses, and semicolons generated (in bold) by the application, except for:

- Spaces surrounding function and parameter names
- Parentheses, commas, or spacing inside lists of data types of parameters that are pointers to functions
- Parentheses around those parameter names themselves

These bulleted items must be supplied by the writer.

<FuncSynopsisInfo>

Bold, with line breaks preserved, usually starting in the same line as the section title, Synopsis, or following it on its own line. A vertical space of 1 pica separates these lines from each other, and from the <FuncDef> that typically follows. (Typically, <FuncSynopsisInfo> contains #includes.)

<Function>

A function name appears in bold.

<Glossary>

A glossary division containing introductory prose, followed by a glossary list or lower-level glossary divisions. The glossary is presented at the end of the book, after the appendixes. Glossaries are formatted similarly to a <VariableList> (or labeled list), but use the entire live area. Term names are left-justified in the live area. Definitions are left-justified in the text area. They appear as running text, indented the width of the scanning column, below the definition line, with intervening space.

<GlossDef>

The glossary definition is treated like a <ListItem> in a <VariableList>.

<GlossEntry>

A container for a single entry in a glossary.

<GlossList>

Wrapper for a list of <GlossEntry>s.

<GlossSee>

Processed as italicized "See" followed by bold term.

<GlossSeeAlso>

Processed as italicized "See" followed by bold term.

<GlossTerm>

The term being defined. <GlossTerm> is treated like a <Term> in a <VariableList>.

<Graphic>

Displays containing graphics are centered in the text area of a page, if possible. Graphics do not break at page boundaries.

If <Graphic Role="column-wide"> (the default) is used, then the graphic will fit into and center in the text area. If <Graphic Role="page-wide">, then the graphic will fit into and center in the combined space of both the scanning *and* text areas. It will be necessary for large graphics to have this attribute set to prevent the graphic from oversetting into the right margin.

The <Graphic> scale-to-fit width attribute is used to control scaling of large graphics.

<Group>

See <CmdSynopsis> for processing of <Group>.

<GUILabel>

Used in DCE 1.2 to put up labels in the Menu Title Bar of a screen. See <Screen>.

<Holder>

For holder of copyright. Unchanged from parent element.

<Index>

The Index wrapper is included, so that books that have no index will not have an empty index generated automatically. See <IndexTerm>.

<IndexEntry>

Not used in DCE 1.2.

<IndexTerm>

Contents are collected for creation of a keyword index. See the DocBook Guide for explanation of use of SpanEnd, PageNum, Significance, Scope, and Zone attributes.

<InformalExample>

An untitled example. (See <Example>.)

<InformalTable>

An untitled table. (See <Table>.)

<InlineEquation>

The <Graphic> containing the equation is placed in the same line (that is, wrapping region) as the surrounding text.

<InlineGraphic>

The <InlineGraphic> is placed on the same line as the surrounding text, causing no line breaks other than those that would occur during normal left justification (that is, it appears in the wrapping flow).

<Interface>

Not used in DCE 1.2.

<ItemizedList>

In a first-level unordered (or itemized) list, each item is preceded by a bullet for the mark. When nested, second-level list items use an em-dash as the mark. The text indent should be the same as that for other types of lists. A blank line separates the list from any the running text it is embedded in. The text of the item is indented 6 mm. A mark (a bullet for a first-level list, an em-dash for second-level nested list) appears two en-dashes to the left of the item text. Items are offset by one blank line of vertical white space, just as a paragraph is. A blank line also follows the list end.

<KeyCap>

This is a key on a keyboard, such as the return key. If the formatter permits, the key name is surrounded by a graphic (a box with rounded corners). If not, it is presented in bold and is surround by angle brackets.

<KeyCombo>

This is a container, typically, for combinations of <KeySym> elements (for example, <Ctrl-c>.

<KeySym>

<KeySym> contains a keyboard symbol. Like <KeyCap>, it is boxed and shaded.

<LegalNotice>

Follows copyrights. The font size should be smaller than the 10 points default for running text in the main book.

<Link>

Not used in DCE 1.2.

<ListItem>

Processing of an item depends on the context. In an itemized list, the text begins after two intervening spaces from the mark, and this beginning point serves as the left justification point for any subsequent wrapped lines of the item.

In an ordered list, the text begins after two intervening spaces from the mark, and this beginning point serves as the left justification point for any subsequent wrapped lines of the item. The mark (a number, letter, or Roman numeral followed by a period) is vertically aligned by the periods, not by the left sides of the numbers, letters, or Roman numerals.

In a variable list, the text will begin with at least two spaces between it and the <Term>, and this beginning point serves as the left justification point for any subsequent wrapped lines of the item.

<Literal>

Literal text appears in bold Times Roman.

<LiteralLayout>

Line breaks and leading white space are preserved. Text indented 6mm on left and right.

<ManVolNum>

Saved and placed in bold within bold parentheses directly after <RefEntryTitle> under the page headers of reference pages (left justified on verso and right-justified on recto pages). May also appear (with the same presentation) in running text within a <CiteRefEntry>.

<Markup>

Markup tokens appear in bold.

<Member>

See <SimpleList>.

<MenuChoice>

No special processing.

<MouseButton>

No special processing.

<Msg>

For DCE 1.2, the <MsgSet> subtree is used to describe the error messages extracted from the source code and compiled into the *OSF DCE Problem Determination Guide*. The main body of the book comprises thousands of <MsgEntry> units, all encoded in exactly the same way. This is the only use of the <MsgSet> subtree in the documentation set.

The <Msg> wrapper is used to contain the message number, its symbolic name, and the message text.

The FOSI will place the Severity and Component fields (which follow this in the SGML encoding because of the DocBook context rules) *before* the <MsgText> content. (This is in keeping with previous releases formatting.)

The message number, followed by a space and then the symbolic name, are in bold, placed on the same line, starting at the left of the scanning column. Following 6 points of vertical spacing, with a hanging indent of 20mm for the publish look and 13mm for the draft look, the FOSI generates the bolded string "Severity:" (no quotes), followed by a space, then the contents of the <MsgInfo><MsgLevel> tag. The contents of the <MsgLevel> tag should align horizontally with the contents of the <MsgOrig> tag. The width of the <MsgLevel> content shall not exceed 30mm. The FOSI should generate the bolded string "Component:" (no quotes), followed by a space before the content of <MsgOrig>.

The FOSI then, after 6 points of vertical spacing, and, aligned with the same hanging indent, generates the bolded string "Text:" (no quotes), followed by a space, then the contents of the <MsgText> tag. This is followed by the contents of the <MsgExplan> tags (see <MsgExplan>), which completes the <MsgEntry>.

<MsgAud>

Not used in DCE 1.2.

<MsgEntry>

The main body of the *OSF DCE Problem Determination Guide* comprises thousands of identically coded <MsgEntry> units.

The processing instructions for a <MsgEntry> are found under <Msg> and <MsgExplan>. <MsgEntry> units are separated by 12 points of vertical space.

<MsgExplan>

The explanation and action to be taken for error messages are contained in <MsgExplan> tags, differentiated by the <Title> used.

Following the <MsgInfo> processing, the FOSI generates 6 points of vertical spacing, and, aligned with the same hanging indent, the FOSI places the contents of <MsgExplan<Title> in bold (the first one will be the word Explanation) followed by a colon and a space. The text of the explanation follows.

The FOSI then generates 6 points of vertical spacing, and, aligned with the same hanging indent, places the contents of the second <MsgExplan<Title> in bold—this will be the word Action—followed by a colon and a space. The text of the action follows.

<MsgInfo>

<MsgInfo> is used to contain the severity level and component for the error message.

<MsgLevel>

<MsgLevel> contains the severity of the error. The word Severity is generated by the FOSI.

<MsgMain>

<MsgMain> contains the symbolic name and text for an error message. (See <Msg> for processing instructions.)

<MsgOrig>

<MsgOrig> contains the component in which the error occurred. This is perhaps not the use intended by the DocBook DTD, but was the most suitable of the tags available. (See <Msg> for processing instructions.)

<MsgRel>

Not used in DCE 1.2.

<MsgSet>

<MsgSet> is the wrapper for the entire group of <MsgEntry>

<MsgText>

<MsgText> contains the text of the error message. (See <Msg> for processing instructions.)

<Note>

A note is set off from the running text by a blank line of vertical space and is automatically labeled Note.

<OLink>

Not used in DCE 1.2. Reserved for future hyperlinking.

<Option>

A command option name appears in bold. A dash is prepended to the contents, and, in a <CmdSynopsis> section, the contents are separated with a space from the following element.

<OrderedList>

An ordered list has items that are numbered. The mark is an Arabic numeral followed by a period. In a second-level ordered list, items use lowercase letters for the mark. Third-level items use lowercase Roman numerals for the mark. The mark is vertically aligned by the periods, not by the left sides of the numbers, letters, or Roman numerals.

In an ordered list, the <ListItem> text begins after two intervening spaces from the mark, and this beginning point serves as the left justification point for any subsequent wrapped lines of the item.

<Para>

This produces what most people expect as a paragraph appearance. There is vertical space of one blank line immediately preceding and following a paragraph. The right margin is justified, and the first line is not indented. It wraps its content.

<ParamDef>

Data type information appears in bold. The output line should be broken following each <ParamDef>.

<Parameter>

Parameters appear in italics.

<Part>

A part begins on a new, recto page, containing only the word Part, the part number as an uppercase Arabic numeral, and part title. That is, the initial page looks like a chapter title page, except that there is no running text. The first contained chapter begins on the following recto page.

A part may contain text instead of hierarchical elements. This is intended for documents that are under development, where the part is briefly described with this text. In this case, the text should appear on the same page as the Part title.

<PartIntro>

Treated as a <Sect1>.

<Preface>

A preface is a division of the preliminaries element and appears after the Table of Contents (and Lists of Figures and Tables). Its title, Preface, is processed in the same way as a chapter's except that it has no chapter number next to it.

<Primary>

Nested under <IndexTerm>, the primary entry for the index term appears at the left of its column. See <IndexTerm>.

<PrimaryIE>

Not used in DCE 1.2.

<Procedure>

This is presented the same way as an ordered list, except that it can have a <Title>. <Step> is treated like <ListItem> in an ordered list, and <SubSteps> serves as a wrapper for nesting steps. The nested steps are marked like second and third-level list items.

<ProductName>

Bold.

<ProductNumber>

Bold.

<ProgramListing>

Line feeds and character spacing are preserved in a <ProgramListing>. The presentation is (constant-width) Courier of point size 9 with leading of 11.

The role="page-wide" attribute is supported for <ProgramListing>. It can be used for listings that extend into (and beyond) the right margin, to bring them to the left. If <ProgramListing role="page-wide"> is set, the output will start in the scanning column.

<Property>

Not used in DCE 1.2.

<PubDate>

Not used in DCE 1.2.

<Publisher>

Used to indicate publisher. However, the FOSI generates the publisher (OSF) on the title page automatically.

<PubsNumber>

Not used in DCE 1.2.

<Quote>

Wrap the contents of <Quote> in double quote marks.

<RefDescriptor>

Not used in DCE 1.2, since it is a substitute for <RefName> when more than one topic is covered and none of the topic names is to be used as the sort name. (*All* names should be used to sort on in the DCE docset.) However, if <RefDescriptor> *is* used, it should appear in the <RefNameDiv> section, with the reference names separated from it by an em-dash.

<RefEntry>

Each reference page starts on a new page. The scanning column is 20mm. Reference page entry titles are left-justified in the live area and are mixed-case. Titles are 13 point bold. Subsection titles are bold and indented by half the width of the scanning column, and are in the same typeface as the running text (Times Roman, 9 points).

The leading for running text, and for Synopsis lines is 12 (in the publish look).

<RefEntryTitle>

Bold when used within a <CiteRefEntry>; otherwise, as part of meta information, it is ignored.

<Reference>

A section which contains a collection of <RefEntry>s. For a book in which the reference pages form a part, both the part and the chapter titles require a full separate page. The part page is as described for <Part>. The chapter page is as described for <Chapter>, except that there is no running text on the page after the chapter title (that is, it is just like a part page, except that it uses the word Chapter instead of Part).

For a book which consists entirely of reference pages, each chapter (or group of reference pages) is preceded by a separate chapter title page, like that just described.

<RefMeta>

Contains <RefEntryTitle> and <ManVolNum>. Ignored, other than providing those values to other processing.

<RefNameDiv>

The <RefNameDiv> division comes after the <RefMeta> division. It consists of the title of the reference page (which coincides with the name), followed by the purpose. Reference page titles are left-justified in the live area and are mixed-case. They are 13 point bold, and are followed, below them, with a horizontal line.

The <RefPurpose> line follows, separated by a vertical space of 2 picas.

<RefName>

This is the name of a reference page. The name appears (in bold mixed case) in the page header under the horizontal rule, on the outside margins. The <ManVolNum> contents, enclosed in parentheses, are appended.

The FOSI generates Table of Contents entries *only* for those <RefName> instances that contain <Command>, <Function>, <Filename>, or <Literal>. These names do not carry their formatting (that is, bold) into the Table of Contents, but appear as bold in the reference page itself.

The <Refname> element itself (and thus, any #PCDATA inside <RefName> without the <Filename>, <Function>, <Command>, or <Literal> markup) will *not* generate a Table of Contents entry. Further, any element inside of <RefName> other than the four elements listed above will not generate a Table of Contents entry.

<RefPurpose>

Appears after <RefName> in <RefNameDiv>. Its contents appear (6 points) beginning 5 picas in from the scanning column, following the word Purpose, which is 8 points bold and begins at the scanning column. If the purpose text wraps, it indents to below where it started.

<RefSect1>

The <Title> is 16 points bold and starts at the leftmost edge of the scanning column.

<RefSect2>

A <RefSect2> <Title> is 13 points bold and starts 2.5 picas in from the left margin.

<RefSect3>

A <RefSect3> <Title> is 13 points bold and starts 5 picas in from the left margin.

<RefSynopsisDiv>

The <RefSynopsisDiv> division follows the <RefNameDiv> division, separated by a vertical space of 2 picas. It begins with the word Synopsis in 16 point bold, starting at the leftmost edge of the scanning column.

<ReleaseInfo>

Not used in DCE 1.2.

<Replaceable>

Appear in italics Times Roman.

<ReturnValue>

Used within a <Sect1> of <Title> Return Values. This will typically be encoded as a <VariableList>, with the <ReturnValue> in italics (encoded within <Term>).

<RevHistory>

Not used in DCE 1.2, since ODE manages the revision history.

<Revision>

Not used in DCE 1.2.

<RevNumber>

Not used in DCE 1.2.

<RevRemarks>

Not used in DCE 1.2.

<Screen>

Causes a rectangle to be drawn around the content. The depth of of the rectangle depends on the amount of vertical content within it. To get screens of equal depth, the author has to provide the same number of lines inside the screen.

The FOSI has been customized to reproduce a particular menu screen for the *OSF DCE GDS Administration Guide and Reference*. A Remap attribute has been provided that puts a Menu Title Bar with the words "DIRECTORY SYSTEM" in the middle of the bar. <GUILabel> can be used to place text in the corners of this Menu Title Bar. The menu content itself follows the </GUILabel>.

<ScreenCO>

Not used in DCE 1.2.

<ScreenInfo>

Not used in DCE 1.2.

<ScreenShot>

Not used in DCE 1.2.

<Secondary>

Nested under <IndexTerm>, following <Primary>, the secondary entry for the index term appears indented 10mm in its column, and follows the same rules as primary entries. See <IndexTerm>.

<SecondaryIE>

Not used in DCE 1.2.

<Sect1>

<Sect1> titles appear in 16 point bold Times, with 3 blank lines above and 2 below. The generated section number appears before the title, separated from it by one space. The title is left-justified within the scanning column. Both margins are justified. The right-hand margin is 5 picas.

There should be text between the <Chapter> and the <Sect1>, and that text has the same margins as <Sect1>.

<Sect2>

<Sect2> titles appear in 13 point bold Times, with 3 blank lines above and 2 below. The title is left-justified within the scanning column. The generated section number appears before the title, separated from it by one space. The margins are as for <Sect1>.

<Sect3>

<Sect3> titles appear in 13 point Times (*not* bold), with 3 blank lines above and 2 below. The title begins 2.5 picas from the left margin. The generated section number appears before the title, separated from it by one space. The margins are as for <Sect1>. <Sect3> titles are *not* collected for the Table of Contents.

<Sect4>

<Sect4> titles appear in 11 point Times (*not* bold), with 3 blank lines above and 2 below. The title is left-justified within the text area. The generated section number appears before the title, separated from it by one space. The margins are as for <Sect1>. <Sect4> titles are *not* collected for the Table of Contents.

<Sect5>

It is not recommended that <Sect5>-level elements be used. If one appears, it should be handled like the <Sect4> level.

<SGMLTag>

This should appear in bold, and have the angle brackets generated by the application. The case convention should be preserved; that is, a mixture of upper and lowercase.

<Seg>

Reserved for future use.

<SegListItem>

Reserved for future use.

<SegmentedList>

Reserved for future use.

<SegTitle>

Reserved for future use.

<SideBar>

Reserved for future use.

<SimpleList>

Intended for long lists of single words or short phrases. It consists of one or more <Member>, and has Column and Type attributes that are used to specify how the list will be laid out (for instance, as an array reading left to right then top to bottom, or top to bottom then left to right).

<StartChange>

This element is an extension to the DocBook DTD to preserve the change information put into OSF source files by writers to indicate what Change Request (CR) the modification addresses. See <Book>.

<Step>

In a <Procedure>, which is presented the same way as an ordered list, except that it can have a <Title>. <Step> is treated like <ListItem> in an ordered list, and <SubSteps> serve as a wrapper for nesting steps. The nested steps are marked like second and third-level list items.

<StructField>

Bold.

<StructName>

Bold. <StructName Role="TypeDef"> indicates a typedef.

<SubScript>

Subscripted text is two points smaller than surrounding text in its context and is lowered by half of a vertical space.

<SubSteps>

See <Step>.

<SubTitle>

Not used in DCE 1.2.

<SuperScript>

Superscripted text is two points smaller than surrounding text in its context and is raised by half a vertical space.

<Symbol>

Rendered in italics.

<Synopsis>

<Synopsis> is an alternative to <CmdSynopsis> or <FuncSynopsis>, which are the preferred elements for DCE documentation. <Synopsis>, however, unlike those two, preserves line feeds and white space and may contain <Graphic>. If it is embedded in a <Para>, the flow of the paragraph will be broken, and the synopsis will be displayed as a block-oriented element.

<SynopFragment>

Not used in DCE 1.2.

<SynopFragmentRef>

Not used in DCE 1.2.

<SystemItem>

Bold. <SystemItem Class="Constant"> is used for constants, <SystemItem Class="EnvironVar"> for environment variables, and <SystemItem Class="Resource"> for resources.

<Table>

This is a container for a table. Presentation depends on the content, but in all cases it is offset by vertical white space, as for a paragraph. The presentation of the table is controlled via the Table Editor, rather than via the FOSI.

The default attribute setting for how the Table will be centered is <Table role="column-wide">. Tables that are too large for this setting will be adjusted in the source to <Table role="page-wide">. If <Table role="page-wide">, then the output will set in the text area plus the width of the scanning column. If the default (<Table role="column-wide">) is used, the output will set in the text area.

<Term>

This is a label portion of a <VariableList> (or labeled list). It is left-justified in the text area. The semantic content of the label itself is marked up, which produces the correct appearance.

<Tertiary>

Nested under <IndexTerm>, following <Secondary>, the tertiary entry for the index term appears indented 10mm past the indentation of the secondary entry, and follows the same rules as primary entries. See <Index>.

<TertiaryIE>

Not used in DCE 1.2.

<Tip>

Not used in DCE 1.2.

<Title>

The appearance of titles varies with their context. Chapter titles are bold, in a large font and occur near the top of the page. Section titles are bold, in a smaller font (though larger than the surrounding text) and appear on a line by themselves.

<ToC>

Not used in DCE 1.2, since the Table of Contents is generated automatically.

<TradeMark>

The trademark symbol depends on the Class attribute. If Class=Trade (or no Class is specified), a stylized TM symbol is appended. If Class=Registered, an R in a circle is appended. If Class=Copyright, a C in a circle is appended.

<ULink>

Reserved for future use.

<UserInput>

Appears in bold Times Roman. <Replaceable> and <Symbol>, when used within <UserInput>, inherit the Times Roman, but not the bold. That is, they will appear as unbolded Times Roman italics.

<VarArgs>

An ellipsis is output.

<VariableList>

The label is left-justified in its text area, while the item text is indented by some amount. Nominally, the indent is 20mm, the same as the scanning column. Second-level nested lists are indented an additional 20mm. The indent may vary for lists with very long or very short labels.

<VarListEntry>

Each <VarListEntry> is separated from the next by a blank line. See <VariableList>.

Because the FOSI cannot detect and accommodate long terms, writers will have to use the role="Linebreak" attribute to force linebreaks and avoid terms crashing into definitions. The default will be role="Aligned", which places the definition start on the same line as the term.

<Void>

The string void is output.

<VolumeNum>

Not used in DCE 1.2.

<XRef>

A cross-reference points (via the ID attribute) to another element in the document. If an Endterm attribute is specified, the content of the element whose ID matches the Endterm becomes the replacement text for the <XRef>. If no Endterm is specified, the ID specified in the Linkend attribute is used. If the target of the Linkend has an XRefLabel attribute, the content of that attribute becomes the replacement text for the <XRef>.

<Year>

Used for OSF copyright, but no change from parent element.

Appendix D

Known troff Problems and Solutions

Blank Pages

Occasionally the bottom of page trap is sprung at the end of a document element, although all the text appears on the page. If this occurs, a blank page is produced, possibly creating a pagination error. The next document element then starts on an even page instead of an odd page.

To avoid this, force some text onto the additional blank page.

Minus Sign Prints as Hyphen

If you are using *eroff*, this is its default behavior. To obtain a minus sign, code `\(mi` instead.

eqn Effects

- The % character should not be used as an *eqn* delimiter because it affects the header macros.
- An H included in an *eqn* using the $ character as the delimiter can have unexpected effects if it immediately follows the $ to form $H.

Tab Stops

In general, tab stops should not be used in Open Group documentation.

However, if tab stops are required, set them by using the `.ta` command:

```
.ta 4m +4m +4m +4m +4m +4m +4m +4m +4m
```

This sets the tab stops to 4 ems. An em is horizontal distance which is equal to the current point size.

Note that *tbl* resets tab stops. Also, a *tbl* in a display resets the tabs stops for the following display.

Table Sizing

This is a known bug resulting from the use of the ampersand character (&) as a field delimiter (tab) in *tbl*. If you are using this character, you should change it.

Formatting in Displays

Displays operate in a different *troff* environment from the body of the text, and all displays operate in the same environment. This means that if you change font or size in one display, that font is used for all the others as well. There are several formatting features that make up an environment: line length, page size, and so on. If you see any unusual formatting occur in a display, the coding error may well be in the previous display.

EPS Files Do Not Print

Some postscript printers cannot print files which contain encapsulated postscript created using CricketDraw. To solve this, try commenting out the line near the start of the .eps file which contains:

```
[] settransfer
```

Positioning of Included EPS Files

Encapsulated postscript files included using `.F+` and `.F-` are left-justified in a complete build, but centered in a partial build.

To make sure a figure is centered, use:

```
.in +\*(lM
.DS C
.F+
figure <file>.eps
.F-
.DE
.in -\*(lM
```

If the figure can be the full page width (15c), you should code:

```
.DS
.F+
figure <file>.eps width 15c
.F-
.DE
```

Cross-Reference Error Messages

During a complete build, a list of unresolved cross-references is displayed on the screen. If your cross-reference label contains an uppercase letter, the screen message for that item will not make sense. This is because postscript places uppercase letters on separate lines in its output file. In general, it is better to use lowercase letters only for cross-references.