[NOT A] STANDARDS PROJECT

Information Technology—
POSIX Ada Language Interfaces—
LaTeX Macro Usage

Author

Ted Baker
Department of Computer Science
Florida State University
Tallahassee, FL 32306-4019

Abstract: This report is a supplement to the standard LaTeX documentation, that explains how to use the special macros defined by the `psxcbln` style. The `psxcbln` style is for use in producing documents in the style required by the IEEE for POSIX standards, and in particular for POSIX Ada-language bindings.

Keywords: LaTeX, TeX, style, POSIX, Ada, IEEE, standards.

psxcbln.sty / D1.2
November 1993

Copyright ©1994 T. P. Baker
All rights reserved.

Contact Information

This is the first draft of a revised document, to reflect changes made in the style-file for the "thickening" of P1003.20, and the first phase of convertin POSIX.5 from `troff`. This was a "quickie" revision, and it has not yet been proof-read, so expect some defects.

Questions, problem reports, and comments on how this document might be improved are very welcome.

The 1003.5a technical editor:

Ted Baker
Florida State University
Department of Computer Science
207A Love Building
Tallahassee, FL 32306-4019

(904) 644-5452
(904) 644-0058 [FAX]

baker@cs.fsu.edu [e-mail]

(Electronic mail is preferred.)

This document was typeset using LaTeX and TeX.

Contents

PAGE

Contents 1

Section 1: General 2
1.1 Scope 2

Section 2: General LaTeX Information 4
2.1 How to Get LaTeX 4
2.2 Reserved Characters 4
2.3 Macros 5
2.4 Sectioning 5
2.4.1 Third-Level Heading 6
2.5 Spaces 6
2.6 Labels and Cross-References 7

Section 3: Programming Language Text 8
3.1 Names and Short Text 8
3.2 Large Ada Code Components 10
3.2.1 Part 2 15
3.3 Floating Code "Figures" 17

Section 4: Indexing 18

Section 5: Change Bars, Fuzz Bars, Line Numbers 20
5.1 Fuzz Bars 20
5.2 Change Bars 20
5.3 Frequent Errors 21
5.4 Line Numbers and Page Styles 22

Section 6: Lists 24

Section 7: Special "Boilerplate" Text 27
7.1 Error Handling 27
7.2 Options 27

Section 8: Tables 29

Section 9: Miscellaneous Macros 32

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

9.1 Standard Phrases 32
9.2 Lists of Types, Constants, and Operations 32
9.3 Annotations 33
9.4 Obsolescent Macros 33

Section A: Bibliography 34

Alphabetic Topical Index 34

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Information Technology — POSIX Ada Language Interfaces — LaTeX Macro Usage

Section 1: General

1.1 Scope

This document explains how to to use the macros defined in the LaTeX style file `psxcbln`[1], which is intended for producing Ada language bindings for IEEE POSIX standards.

The `psxcbln` style is based on the standard LaTeX `report` style (11pt size). For general information on LaTeX and the `report` style, see the LaTeX manual[1].

The `psxcbln` style differs from the `report` style documented in the LaTeX manual in that it has been modified to approximate the style requirements of the IEEE POSIX standards. Some of these changes include:

— new page styles, with headers and footers

— use of New Century Schoolbook and Courier fonts instead of Computer Modern fonts.

— virtual PostScript fonts

— line numbering

— change bars

— 'fuzz' bars (for non-normative text)

— multi-level indexing

[1] The names stands for **P**osi**X** with **C**hange-**B**ars and **L**ine-**N**umbering

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

24 — Ada code inclusions, with output of compilable code to separate files, and re-
25 peatable synopses
26 — Ada, C, and LIS identifiers, with optional indexing
27 — special tables, for showing LIS/C-to-Ada mappings

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Section 2: General LaTeX Information

2.1 How to Get LaTeX

LaTeX is a standard set of fonts and macro-definitions developed by Leslie Lamport for use with Donald Knuth's TeX document preparation system, developed by Leslie Lamport. LaTeX is normally distributed in a bundle with TeX, for free.

To use this style-file you will need a more or less complete TeX installation, including PostScript virtual fonts and the `dvips` dvi-to-PostScript translator. TeX is available from many public `ftp` sites. In particular, it can be found in the `/pub/tex` directory on `byron.u.washingon.edu`. Using the `archie` utility to scan the internet, one can find many public ftp sites where versions of TeX are available. For example, there is a directory `/.2/micro/modes/simtel20/tex` on the host `gatekeeper.dec.com`.

2.2 Reserved Characters

Generally, the following characters are reserved by TeX/LaTeX, and therefore should not appear in plain text.

```
#$%&_^\{}
```

Of these, the biggest problem for typesetting Ada code using LaTeX is the underscore (_), though # and & sometimes also crop up in Ada code. Ada text should always be coded using special macros provided with this style-file. These macros allow underscores in their arguments. The `\adacomp` macro, described in Section 3.2, also allows the other Ada characters. For other situatations, where you need these special characters outside of the special Ada macro calls, you can use the forms `\$`, `\&`, `\%`, `\#`, `\{`, and `\}`, or you can use the LaTeX `\verb` macro. The form of the latter macro call is `\verbXtextX`, where `text` is the string to be printed verbatim, and `X` stands for any character that does not appear inside `text`.

The character `%` is used for TeX comments. This character and the remainder of the line following it are ignored. To include this character in text, type `\%`.

Note that LaTeX *(unlike Scribe) cares about the difference between* `()`, `[]`, *and* `{}`. *If an example shown here uses* `{}`, *do not try to subsitute* `[]`, *or to do any other such subsitutions of bracketing characters.*

LaTeXmacro names are also case-sensitive.

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

2.3 Macros

Symbols beginning with \ are macro-names. Macros have arguments, and there are a variety of syntaxes for the arguments. Usually, the argument is just the next item. In this case, if the argument is more than one token long, it must be enclosed in {}'s. For consistency, it is best to always enclose the argument in {}s, even if it is only one token long.

For some macros there is a special syntax, often requiring []s to enclose a parameter.

In general, avoid nesting macro-calls inside the []s or {}s of the parameter to another macro call.

2.4 Sectioning

This documentstyle supports six (6) levels of sectioning. These are represented by the section-header macros:

- `\adasect{Top-Level (Section) Title}`
- `\adasectt{Second-Level (Clause) Title}`
- `\adasecttt{Third-Level (Subclause) Title}`
- `\adasectttt{Fourth-level Title}`
- `\adasecttttt{Fifth-level Title}`
- `\adasectttttt{Sixth-level Title}`
- `\pkgsect{Some_Ada_Package_Name}`
- `\adarat{Rationale Section Title}`
- `\adaratt{Rationale Subsection Title}`
- `\adarattt{Third-level Rationale Title}`
- `\adaratttt{Fourth-level Rationale Title}`
- `\adaerr` (expands to `\adasectttt{Error Handling}`)
- `\adarefs` (expands to `\adasectttt{References}`)

LaTeX takes care of incrementing section numbers automatically, starting at zero for each level, and incrementing the appropriate count when it processes a section

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

header macro. Occasionally, you may need to override this numbering, typically because you are specifying updates to an earlier standard (POSIX.5) and certain sections are left unchanged. You can explicitly set the section numbering counters using the following macros.

This documentstyle supports six (6) levels of sectioning. These are represented by the section-header macros:

- `\adaset{n}`
- `\adasett{n}`
- `\adasettt{n}`
- `\adasetttt{n}`
- `\adasettttt{n}`
- `\adasetttttt{n}`

2.4.1 Third-Level Heading

This is an example of a second-level document division, i.e. a "subclause" in ISO terminology.

2.4.1.1 `Fourth-Level_Heading_With_Underscores`

This is an example of a third-level division. Note that the section name has underscores. Ordinarily, underscores can't be relied upon to work in table-of-contents entries, but the table of contents does not go down to this level. Therefore, you can get away with this only at the subsubsection and deeper levels in the hierarchy. For other section heading with Ada names, special tricks are needed.

The main case is the `\pkgsect` macro, which is used for the headings that have the form "Package `Some_Package_Name`".

```
\adasectttt{\ada[*]{Subsubsection_Header_With_Underscores}}
```

2.4.1.1.1 Fifth-Level Heading

This is an example of a fourth-level document division.

2.4.1.1.2 Sixth-Level Heading

This is an example of a sixth-level document division.

2.5 Spaces

LaTeX generally assumes it is free to adjust white-space. Line-breaks are ignored, unless there are two in a row, in which case it indicates the start of a new paragraph. Thus

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

```
  this                                text
        here
```

and

```
  this text here
```

produce the same typeset output.

Macro calls generally gobble up the following spaces. One way to preserve spaces is to make sure the spaces you want are between some ordinary text and the following macro call. A more general way to avoid the space-gobblin effect of a macro call is to enclose the macro call in braces. For example, `\thistandard` is a macro with no parameters. `\thistandard text` produces "this standardtext", but `{\thistandard} text` produces "this standard text".

2.6 Labels and Cross-References

The standard LaTeX macros for defining symbolic labels and making references to them are supported with this style. To define a label `labelname`, put a macro-call `\label{labelname}` inside the section, subsection, etc. to which the label should be attached. Then to generate a reference to the section number where the label was defined, you use `\secref{labelname}`. This generates both the number and the prefix "Section". Of course, LaTeX needs to be run over the document twice, so that forward references can be filled in correctly.

In addition, certain other macros automatically generate labels, notably the macros for generating C/LIS-to-Ada mapping tables.

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Section 3: Programming Language Text

This section describes the support for typesetting programming language text, including Ada, C, and the LIS language.

3.1 Names and Short Text

Most Ada identifiers should be coded using the macro `\ada`. The general form of this macro call is `\ada[x]{text}`. The parameter `x` must be a single character, and specifies the category of the name or text. The text would ordinarily be an Ada-language identifier, though this macro can also be used for short fragments of in-line Ada code. For example `\ada[t]{Some_Type_Identifier}` will print `Some_Type_Identifier`. As a side-effect, it will also generate an index entry for this as an Ada type identifier.

Always use `\adadot` for the dot in qualified names. This will cause the two components of the name to be indexed separately separately. For example, you can write `\ada[t]{POSIX_IO.File_Descriptor}`, and you will get `POSIX_IO.File_Descriptor`, but the index will show an entry for the long form of the name, listed under P. In contrast, if you write `\ada[k]{POSIX_IO}\adadot\ada[t]{File_Descriptor}` you will get `POSIX_IO.File_Descriptor`, but there will be two index entries, one under P for the package and the other under F for the type.

You can use * as the category name to suppress automatic indexing on a call to `\ada`. The latter is good for references to unimportant Ada identifiers, and chunks of code that may be used in the discussion of an example, like:

"`X+Y=Z` is an example of an Ada expression."

Similarly, C-language identifiers should be coded as `\C[X]{text}`, and language-independent specification identifiers should be coded as `\lis[X]{text}`, where `X` identifies the category of name.

Presently, the `\lis` macro should not be used, since the Ada bindings refer to the C-language bindings.

The categories for Ada and the other languages are shown in Table 3.1:

The category * should generally be used for in-line fragments of Ada text which are intended not to be indexed. However, for some categories of C and LIS text, this may result in incorrect font choice.

For environments where no indexing is desired, and * does not give the correct font,

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Table 3.1 – Letter Codes for Categories of Names

Category	Meaning	Indexed
a	attribute (Ada)	
b	variable	
c	constant	
d	record/structure datatype component/member	not indexed
e	error status value (C) or exception (Ada)	
f	function	
g	generic unit	
h	check (Ada)	
i	in-parameter	not indexed
j	not used yet	
k	package	
l	limit	
m	pragma (Ada) or macro (C)	
n	enumeration literal	
o	out-parameter	not indexed
p	procedure	
q	not used yet	
r	in-out (or reference) parameter	not indexed
s	subtype	
t	type	
u	not used yet	
v	environment variable	
w	not used yet	
x	POSIX option	
y	not used yet	
z	not used yet	
*	miscellaneous	not indexed

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

one can temporarily override the indexing of a given category. For example, to print a C-language function name *Function_Not_To_Index*, without generating an index entry, one can use:

```
\begingroup\def\dontindexf{*}\C[f]{Function_Not_To_Index}
\endgroup
```

Note that the letter `X` in `\dontindexX` can be any one-letter identifier. This macro specifies that the category `X` is not to be indexed. This technique should be avoided whenever possible, since there is risk of forgetting to use the `\begingroup` and `\endgroup` will result in globally changing the treatment of one or more name categories.

Ada reserved words should be coded using the `\key` macro. For example, `\key{begin}` results in the output **begin**. This can be used in plain text, and inside small Ada text fragments, like `\ada[*]{\key{begin} do_something; \key{end}}`, which produces **begin** `do_something;` **end**. It can also be used inside large Ada code components (see Section 3.2).

Take care not to attempt to use the `\ada`, `\C`, or `\lis` macros inside of section or section headers, since it interacts badly with the macros that produce the table of contents.

The `\ada`, `\C`, and `\lis` macros do allow underscores in names, but (unlike the `\adacomp` macros, used for large blocks of Ada code) they do not allow the other characters reserved by TEX. If you need any of these other characters (e.g. `&`) you can generate them as described in Section 2.2.

The special macros shown int Table 3.2are defined for use in C-to-Ada mapping tables.

Table 3.2 – Constraint Operators

source	output
\cand	&
\Cequal	==
\Cneq	!=
\Cgeq	>=

3.2 Large Ada Code Components

A set of interraleted macros is provided for Ada package specifications, which provide the following functionality:

- — Output of the compilable part of the Ada code to a separate file, so that the syntax can be checked by a compiler.
- — Repetition of fragments of the package text at the point where they are described in the document.
- — Enforced consistency between the descriptive headings and section numbers that are included as comments in a package specification and the corresponding sections of the document.

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Ada code components of one or more full lines are coded using the \adacomp macro. These macros have the side-effect of outputting the Ada text, stripped of formatting and ready to compiler, to a separate file. If no other file is specified, the Ada code goes to the LaTeX .log-file. To override this default and send the output to a file someprefix.a, use the macro call \adaopen{someprefix}.

Note that the argument of \adaopen should not contain any of the LaTeX control characters. In particular, it should not contain underscores.

Note: *This is a weakness of the style file, which should some day be corrected.*

The \adacomp macro has four forms:

\adacomp{title}{X}{text} This is the general form.

The second argument must be a single letter. In the example above, it is /verb/X/. This letter is used to generate the names of various macro that will be associated with this Ada code fragment.

The first argument is a descriptive title. A side-effect of this macro expansion is to define a macro \adacmphX, which will expand to the given title if it is called later.

The third argument is some Ada text. A side-effect of this macro expansion is to define a macro \adacompX, which will expand to the given text if it is called later.

What Ada text is also inserted in-line at the point of the current call to \adacomp depends on whether the symbol \abbreviate is set to t. (The default setting is currently t.) These effects will be illustrated by an example, further below.

\adacomp*{X}{text} omits the title. It still defines \adacompX to stand for the text, so that it can be expanded later.

\adacomp{title}*{text} omits the definition of a macro to stand for the Ada text.

\adacomp**{text} omits the definition of an title or body macros, and just inserts the Ada text in place.

Here, text can be any Ada code. **Contrary to the normal rule in LaTeX, in this environment spaces and line-breaks are generally preserved verbatim. This allows you to lay out the code in the normal way you would when programming.**

The effect of (empty) blank lines depends on whether \abbreviate is set to t. If \abbreviate is t, the full text of each Ada component is printed out at the point where the \adacomp macro is first called. Blank lines are preserved in this text. If \abbreviate is t, only "..." is printed out at the first point.

In all cases, empty lines are suppressed on expansion of the \adacompX macros, which are later used to reproduce the code components for the synopses.

a

Just a few macro-calls are safe inside Ada code segments. These are:

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

`\key`
: This macro should be used for reserved words, so they will appear in bold. This macro can also be used outside of Ada code sections.

`\doc`
: This macro should be used for text that is not legal Ada code, which is only to appear in the typeset document. Calls to `\doc` are expanded in the typeset copy of the text, and omitted from the copy that is written to the compilable code file. Do *not* atempt to use this macro outside of Ada code sections.

`\cod`
: The effect is complementary to `\doc`. This macro should be used for text that is needed to make the Ada code compilable, but which should *not* appear in the typeset document. Calls to `\cod` are omitted from the typeset copy, but are expanded in the compilable code file. The general idea is is to use these macros in pairs where the typeset form includes text, like "*implementation defined*", that cannot be compiled. Do *not* atempt to use this macro outside of Ada code sections.

`\fromdotfive`
: This macro prints out a standard elipsis for sections of code that are already defined by POSIX.5.

`\indx`
: Use `\indx[t]{name}` to create an index entry for `name` of category `t`.

`\impldef`
: Use `\impldef` inside a call to `\doc` to generate the standard text for something that is implementation defined.

`\comment`
: Use `\comment{text}` to create a comment that should appear in both the typset and compilable versions of the Ada code.

`\cbon` and `\cboff`
: If the change only affects one line, just put `\cbon[a]\cboff[a]` at the end of the line (with `a` replaced by the appropriate change-set number).

The use of these macros is illustrated in the example below.

```
\adaopen{code}
\adacomp**{
\key{with} System;
\comment{first part}
\key{package} Package_1 \key{is}\fromdotfive}\indx[k]{Package_1}
\adacomp{Part 2}{a}{

   \comment{second part}

   \key{type} Type_2 \key{is private};

   \key{procedure} Procedure_2
      (L: \key{out} Type_2;
       R: \key{in} Type_2);
```

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

```
}
\adacomp{Part 3}*{

    \comment{third part}

    \key{function} Function_3
       (L: \key{in out} Type_2)
       \return{Type_3};

}
\adacomp*{c}{
    \comment{fourth part}

    \key{procedure} Procedure_4
       (L: \key{in out} Type_2);

}
\adacomp**{
\comment{fifth part}

\key{private}

    \key{type} Type_2 \key{is} \doc{\impldef;}\cod{new Integer;}

\key{end} Package_1;}
\adaclose
```

The input above produces the following output, if `\abbreviate` is `t`.

```
with System;
-- first part
package Package_1 is
    -- Declarations carried forward from POSIX.5 are not shown here.
      ...k]Package_1
    -- 3.2.1 Part 2   ...
    --  Part 3

    -- third part

    function Function_3
       (L: in out Type_2)
       <return-value>Type_3;


-- fifth part

private

    type Type_2 is implementation-defined;

end Package_1;
```

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Please take note of the indentation and spacing of this example, which is the current style to be used for P1003.20.

It also defines the macros `\adacompa`, `adacmpha`, and `\adacompc`, and it writes the following text to the file `code.a`.

```
--This code extracted from file macros.tex by Latex.

with System;
-- first part
package Package_1 is


   -- second part

   type Type_2 is private;

   procedure Procedure_2
      (L: out Type_2;
       R: in Type_2);




   -- third part

   procedure Procedure_3
      (L: in out Type_2);



   -- fourth part

   procedure Procedure_4
      (L: in out Type_2);



-- fifth part

private

   type Type_2 is new Integer;

end Package_1;
```

> **Note:** *This code does look a bit ugly, due to the extra blank lines. I have not yet had time to figure out how to drop these blank lines without also losing blank lines in the formatted document.*

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

3.2.1 Part 2

Calling \adacmph{a} produces the subsection heading above. This is equivalent to

```
\adasecttt{\adacmpha}\label{c:\thesection:a}
```

To produce a heading for a section describing an Ada code component at a different sectioning level, you need to use the long form, with the appropriate other-level sectioning macro instead of \subsection.

Calling \adacmpha produces

Part 2

This is just the text of the section heading, without the sectioning effect. This is useful in creating captions for tables or figures to match the section heading.

Calling \adacompa produces the following Ada text.

```
   -- second part
   type Type_2 is private;
   procedure Procedure_2
      (L: out Type_2;
       R: in Type_2);
```

Calling \adacmphc is an error, since it is not defined.

Calling \adacompc produces the following text.

```
   -- fourth part
   procedure Procedure_4
      (L: in out Type_2);
```

Do not try calling \adacmph*. If you do, it will produce a spurious subsection heading, and override the section number associated with the code segment where it originally appeared.

> **Note:** *This is a defect, which ought to be fixed.*

Now, if \abbreviate is f, the effects at the point where \adacomp is expanded are different. The typeset output is as follows.

```
with System;
-- first part
package Package_1 is
   -- Declarations carried forward from POSIX.5 are not shown here.
     ...k]Package_1
   -- 3.2.1 Part 2

   -- second part

   type Type_2 is private;

   procedure Procedure_2
      (L: out Type_2;
```

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

```
282          R: in Type_2);

283      -- Part 3

284      -- third part

285      procedure Procedure_3
286         (L: in out Type_2);


287      -- fourth part

288      procedure Procedure_4
289         (L: in out Type_2);


290  -- fifth part

291  private

292      type Type_2 is implementation-defined;

293  end Package_1;
```

294 The definitions of \adacompa, \adacmpha, and Calling \adacompc should be un-
295 changed.

296 Note that the naming scheme for the macros \adacompX, is limited to 52 names
297 (using the letters 'a'..'z','A'..'Z' in the position of for 'X'). In practice, the more severe
298 limit is likely to be TEX/LATEX's macro-definition storage space. It is probably best to
299 recycle code-section names, once they are no longer needed. By convention, this is
300 no later than the end of the section where they are defined.

301 Note that TEX has a limited capacity for buffering macro arguments. Due to the
302 nature of the \adacomp macro, it is easy to exceed this limit if you try to to cram
303 lots of declarations into a single component. Try to keep each component down to
304 a moderate number of declarations. If TEX complains about running out of storage,
305 try splitting up the components into smaller chunks.

306 There is also a limit on the total space for macro definitions. This is a good reason to
307 reuse names of code segments, after the last reference to the corresponding macros
308 \adacmphX and \adacompX.

309 Beware that line-breaks in Ada code segments are preserved in the formatted text.
310 Since the font used in the final document is smaller than that used in most editors,
311 and because the macro calls for such things as change-bars, comments, etc. collapse
312 down to nothing, it will generally be necessary to have some lines in the code sections
313 that are longer than will fit on one text editor line. *Do not break these lines, or the*
314 *line break will appear in the output.*

315 As mentioned elsewhere, empty blank lines will be deleted automatically from the
316 code synopses when they are printed, but the *breaks* will still be there.

a

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

3.3 Floating Code "Figures"

If we follow the style of POSIX.5, Ada package specification will need to be treated as floating figures. For this purpose, the `\begin{adafig}{Package_Name}` and `\end{adafig}` macros are provided. They are to be used in pairs, around the clusters of `\adacomp` calls that define the package.

At present, these bodies of these macros are nulled-out, so that the Ada package spec's always appear in-line (i.e. they do not float). However, the policy is to put these around each package spec., so that we can quickly convert them between in-line text and floating bodies.

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Section 4: Indexing

This style file is set up to allow automatic generation of a multi-level index. The other components of this system are shown in Table 4.1.

Table 4.1 – Tools

file name	content
`psxcbln.sty`	the main style file
`index.sty`	the index style file
`headers.tex`	index section headers
`sort`	the UNIX file-sorting utility

Producing an index has several steps:

(1) Run LaTeX on the document, with `\makeindex` set. For a document file `doc.tex` This will generate a raw index file `doc.idx`.

(2) Concatenate a list of alphabetic section headers with the index file. For example,

```
cat doc.idx headers.tex > tmp.idx
```

(3) Sort the raw index file into the desired order. For example,

```
sort -t\{ +2 -3df +3 -4df +4 -5df +5n < tmp.idx > sorted.idx
```

(4) Use the sorted index file, with the `index.sty` file, to produce the typeset index. It generally looks better if this is done in a smaller font. For example, the index of this document was produced by:

```
\begin{theindex}
\input{index.sty}
\downtwo % reduces font size by two steps
\input{sorted.idx}
\end{theindex}
```

Some of the index entries are produced automatically, as explained in Section 3.1. Other indexing must be done by manually-inserted macro calls.

Because the POSIX documents use a two-level indexing scheme, and distinguish between identically spelled names in different fonts, the standard LaTeX `\index` macro must be used in a special way. The form of a call is shown below, where `k` must be a single letter, indicating the language categorization of the index entry.

```
\index{{primary_phrase}{k}{category_letter_or secondary_phrase}}
```

This macro call does not produce any immediate output. The effect is to create an entry on the `.idx` file, for later sorting and processing to produce the document index.

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

The first parameter is the word or phrase to be indexed. The second parameter must be a single character. The values presently supported, in alphabetical order, are shown in Table 4.1.

Table 4.1 – Language Classes

language classification	significance
*	plain English text
a	Ada name
c	C-language name
l	LIS name

Look at the index of this document to see the effects of the following:

```
\index{{single-level English phrase}{*}{*}}
\index{{two-level English phrase}{*}{second-level phrase}}
\index{{single-level English phrase, with comma}{*}{*}}
\index{{Ada_Procedure_Name}{a}{p}}
\index{{C_Function_Name}{c}{f}}
\index{{LIS_Procedure_Name}{l}{p}}
```

There is no automatic indexing inside Ada code segments. To generate an index entry for an identifier within a code segment, use the special macro-call `\indx[t]{name}`, as shown in the example in Section 3.2. Look at the index to see the effect. The bracketed parameter identifies the category of Ada name. Please use this only once for each defined name — for the declaration — to avoid cluttering the index with redundant entries.

Short form macros are provided for indexing English phrases, which need to be inserted by hand. One form is `\indx{English text}`. The effect is equivalent to `\index{{English text}{*}{}}`. It only creates an index entry. Another macro is provided to both create an index entry and insert the name in-line. This is `\indxself{English text}`. For example, this is used here to index the prhase "indexing in-line text".

With the self-inserting form, take care to write in a style that does require the word to be capitalized, since that will result in confusing index entries (some with capitalization and some without).

For the defining occurrences of an English term or phrase, use the macro-call `\definition{text}`. This will enter the phase in the index, and also print out the term. For example, the call `\definition{defining occurrence}` is inserted at the point of the following text: defining occurrence. It both produces the immediate text and creates an index entry. Thereafter, use the macro `\use{text}` for occurrences of the defined term, so that they also appear in the index.

The warning about self-inserting macros above also applies to these macros.

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Section 5: Change Bars, Fuzz Bars, Line Numbers

This style file provides macros to generate PostScript commands that will cause the printer to insert line numbers, change bars, and fuzz bars along the edges of the printed text. The macros just toggle these effects on and off. The usage rules are similar.

In general, avoid inserting these (or any other) macro calls inside the actual parameters of other macros. In particular, avoid this within the section-header argument of the sectioning commands, the argument of an indexing command, or the arguments of the `\ada`, `\C`, and `\lis` macros, or between or within the parameters of the table-building macros.

You may use these and a few other macros within the code sections of `\adacomp`, but be careful to keep the on/off calls properly paired within the same parameter. If the change is confined to one line, the best thing is to simply put `\cbon[a]\cboff[a]` at the end of the line.

Note that putting change-bars around a complete call to `\adacomp` with not have the desired effect, since the change-bar will not appear later when the component is expanded for the synopsis-part.

5.1 Fuzz Bars

Non-normative text should be enclosed in paired calls to `\fbon` and `\fboff`. This generates 'fuzz' bars in the right and left margins.

For example, the paragraph above was generated by:

```
\fbon Non-normative text should be enclosed in paired calls to
\mcro/\fbon/ and \mcro/\fboff/.  This generates 'fuzz' bars
in the right and left margins. \fboff
```

Because the `\fbon` occurs before the first word of the paragraph, the fuzz bars start at the preceding line. To avoid this, put the `\fbon` after the first word of the line where you want the fuzz-bar to start.

5.2 Change Bars

The change-bar macros are similar to those for fuzz bars, but they also have a version identifier argument. Read the section on fuzz bars before reading this section. a

The paragraph above was generated by the text:

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

```
The \cbon[a] change-bar macros are similar to those for fuzz
bars, but they also have a version identifier argument.  Read
the section on fuzz bars before reading this section.
\cboff[a]
```

Notice that the `\cbon[a]` is after the first word of the first line, so the change-bar does not include the section header.

The version identifier is a single letter, and identifies the revision in which the change was introduced. The convention is to start with the letter `a`, and proceed through the alphabet for subsequent revisions.

To indicate a deleted section of text, you can use

```
 \cbon[a]\cboff[a]
```

a

5.3 Frequent Errors

Do not attempt to nest or overlap change/fuzz-bar regions. For example the following will not be accepted:

```
\cbon[a] Pairs of \cbon[b]change-bar on/off \cboff[b]macros
{\em are not allowed} to be nested \cboff[a], \cbon[a]
neither are \cbon[b] they \cboff[a] allowed
to \cboff[b]overlap.
```

Note that incorrect closure of regions can happen accidentally if you use fuzz bars or change bars in the argument of a macro that causes the text to be replicated later. The rule to be safe is:

If an on-operation appears within a set of braces, make sure the matching off-operation appears within the same set of braces.

The place where this error most often occurs is within the arguments of `\adacomp`, when changes are made to Ada package specifications.

The change-bar and fuzz-bar on/off macros (like all normal TEX macros) gobble up trailing spaces. When inserting calls in the space between words, make sure the space is *before* the interword space.

Be very careful about using bars in Ada code sections (see `\adacomp`).

- If you have a bar on coming up to a call to `\adacomp`, turn it off first.
- If you turn on the bar inside the braces of the `\adacomp` parameter, you must turn it off before the closing brace.
- If only one line is affected, just put `\cbon[X] \cboff[X]` at the end of the line.

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Also be careful about using bars in tables, because they (like Ada code components) are "floating" bodies of text. As LaTeX moves these blocks of text around (to avoid splitting over page boudaries), any unmatched bar on/off macros will result in errors. In particular:

— If you have a bar on coming up to a call to the beginning of a table, turn it off.

— If you want to mark a particular line, turn on the bar at the end of the line. If that is the only line to be marked turn the bar off again immediately.

5.4 Line Numbers and Page Styles

Whether lines are numbered or not depends on the page style. Pages styles also determine the headers and footers. Normally, the page style is set up automatically by the the major section macro, i.e. `\adasect`. The places where it may need special attention are the front-matter and appendices.

Besides the standard pages styles of the LaTeX `report` document style, we have the following special page styles:

`\pagestyle{c}`
: Display the copyright footer only. There are no headers, no other footers, no page numbers, and no line numbers.

`\pagestyle{cp}`
: Display the copyright and page numbers only. There are no headers, no other footers, and no line numbers.

`\pagestyle{cl}`
: Display the copyright and line numbers only. There are no headers, no other footers, and no page numbers.

`\pagestyle{cflp}`
: Display the copyright, line numbers, page numbers, and full footers. There are no headers.

`\pagestyle{cfhlp}`
: Display the copyright, partial headers, full footers, line numbers, and page numbers.

`\pagestyle{cfhp}`
: Display the copyright, partial headers, page numbers, and full footers. There are no line numbers.

`\pagestyle{cfhhlp}`
: Display everything, i.e. copyright, full headers, full footers, line numbers, and page numbers.

To help in remembering these names, and in defining new page styles, it helps to understand the naming system followed. This is shown in Table 5.1.

The default text that appears in the full page headers is for P1003.20. This can be overriden. For example, to override the default page header text for this macro usage document, the following macro calls were used:

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Table 5.1 – Page Style Mnemonics

feature	abbreviation
copyright notice	c
page numbering	p
line numbering	l
full headers	hh
abbreviated headers	h
footers	f
none of the above	empty
all of the above	full

```
\def\draftnumber{1.1}
\def\evenheadtext{POSIX Ada INTERFACES}
\def\oddheadtext{{\LaTeX} MACRO USAGE}
\def\docversion{\parbox[b]{1in}{\scriptsize
   FSU\\psxcbln.sty/D\draftnumber}}
```

The actual line numbering, as well as the implementation of change bars and fuzz bars is done by PostScript functions defined in the file `tex.pro`. This file is supposed to be inserted automatically into the PostScript output produced by the program `dvips`, which is used to translate the `.dvi` output of LaTeX into PostScript.

Make sure you have a copy of this file in your local directory when you run `dvips`.

Though line numbering is turned on and off automatically by the page-style files, if more direct control of the line numbering is required for some special purpose it can be obtained by direct output of PostScript text. The raw PostScript commands are described below. However, direct use of these commands is dangerous, as it is likely to interfere with the other usages of the same commands by the `psxcbln` macros.

Table 5.1 – Macros for Line Numbers and Side Bars

macro call	effect
`\special{ps: lineszero}`	resets line number to zero
`\special{ps: lineson}`	turns on line numbering
`\special{ps: linesoff}`	turns off line numbering
`\special{ps: ()fbon}`	turns on fuzz-bar
`\special{ps: ()fboff}`	turns off fuzz-bar
`\special{ps: (a)cbon}`	turns on change-bar
`\special{ps: (a)cboff}`	truns off change-bar

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Section 6: Lists

The standard LaTeX `itemize` list format has been modified to use dashes.

A special form of list environment is defined for the "Error Handling" sections of the standard. Such a list begins with `\begin{errlst}` and ends with `\end{errlst}`. Items within the list should use the macro `\erritm{error_code_name}`. This macro uses the right font for the error code name, allowing underscores in the text, and automatically makes an index entry for it. Therefore, this macro should *only* be used for Ada identifiers that correspond to POSIX error code constants.

```
\errtxt
\begin{errlst}
\erritm{Filename_Too_Long} The length of \ada[i]{Name} exceeds
 \lis[l]{PATH_MAX}.

 A pathname component is longer than
 \lis[l]{NAME_MAX} while \ada[l]{_POSIX_NO_TRUNC} is in effect.

\erritm{File_Exists} \ada[f]{Open_Or_Create_Shared_Memory} was
 called with \ada[c]{Exclusive} in \ada[i]{Options} and the named
 shared memory object already exists.
\erritm{Interrupted_Operation} The operation was interrupted
 by a signal.
\end{errlst}
```

The text above produces the following effect.

If any of the following conditions occur, the exception `POSIX_Error` shall be raised, and the error code returned by `Get_Error_Code` shall be set to the corresponding value:

`Filename_Too_Long`
: The length of `Name` exceeds {PATH_MAX}.

`A pathname component is longer than {NAME_MAX} while _POSIX_NO_TRUNC is in effect.`
: A pathname component is longer than {NAME_MAX} while `_POSIX_NO_TRUNC` is in effect.

`File_Exists`
: `Open_Or_Create_Shared_Memory` was called with `Exclusive` in `Options` and the named shared memory object already exists.

`Interrupted_Operation`

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

The operation was interrupted by a signal.

(See Section 7 for explanation of the \errtxt macro.)

For a list with a similar appearance, but with an adjustable indentation, use the \varlst environment. This has a parameter, which specifies the indentation.

```
\begin{varlst}{20}
 \varitm{Indentation.} The indentation argument specifies the
 number of widths of the letter 'M' to indent.
 \varitm{Labels in English text} The \mcro/\varitm/ macro is
 intended for use with this form of list, for labels
 with ordinary English text.  No underscores are allowed in
 the text of such a label. This macro does not create an index
 entry for the label.
 \adaitm[*]{Ada_Identifiers} You can use the \mcro/\adaitm/ macro
 with this form of list, if you want to use Ada identifiers as
 labels.  This has an extra parameter, to specify the semantic
 category of the Ada name. This macro does create an index
 entry for the label.
 \varitm{No Macro Calls Inside Labels} Don't try to put macro
 calls inside an item label.
 \varitm{No_Underscores} If you try including words with
 underscores inside the label, you will get strange results.
 Don't do it.
\end{varlst}
```

The text above produces the following output:

Indentation.
: The indentation argument specifies the number of widths of the letter 'x' to indent.

Labels in English text
: The \varitm macro is intended for use with this form of list, for labels with ordinary English text. No underscores are allowed in the text of such a label. This macro does not create an index entry for the label.

Ada_Identifiers
: You can use the \adaitm macro with this form of list, if you want to use Ada identifiers as labels. This has an extra parameter, to specify the semantic category of the Ada name. This macro does create an index entry for the label.

No Macro Calls Inside Labels
: Don't try to put macro calls inside an

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

item label.

No_Underscores
: If you try including words with underscores inside the label, you will get strange results. Don't do it.

These lists can also be used with the normal LaTeX `\item` macro, where different effects are desired. For example, the example, the following list is produced using a `varlst` environment with indentation 20.

With a label. This item is generated using
: `\item[With a label.]`

With a label. This item is generated the same way.

This item is generated using the `\item` macro without any label. The only difference noticeable is in the indentation of the first line.

Below is the effect of using `\item` with the `errlst` environment.

With a label. This item is generated using
: `\item[With a label.]`

With a label. This item is generated the same way.

This item is generated using the `\item` macro without the optional label parameter. The only difference noticeable is in the indentation.

> **Note:** *It would be better if these environments could be defined in a way that could cause a variable space after the label, long enough to align the rest of the text on the line with the label with the left margin of the following lines.*

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Section 7: Special "Boilerplate" Text

7.1 Error Handling

Special macros are provided for frequently-used sentences that we want to be uniform throughout the document.

The macro `\errtxt` produces the standard POSIX.5 introductory text for a list of errors that are required to be detected, i.e.

> If any of the following conditions occur, the exception `POSIX_Error` shall be raised, and the error code returned by `Get_Error_Code` shall be set to the corresponding value:

The macro `errtxto` produces similar text, but for errors that are not required to be detected, i.e.

> If any of the following conditions is detected, the exception `POSIX_Error` shall be raised, and the error code returned by `Get_Error_Code` shall be set to the corresponding value:

The macro `\errtexta` produces the standard warning that in situations where multiple error codes may be returned, which of the possible error codes will be returned is unspecified, i.e.

> In those situations where multiple error codes may be returned, which possible error code may be returned is unspecified.

7.2 Options

The macro `\option{Limit_Function}` produces the text `Ada_Function_Name` and creates an index entry with the category "option". It should be used for configurable system parameters in contexts where they are used as implementation options.

Similarly, the `\limit{Limit_Function}` produces the text `Limit_Function` and creates an index entry with the category "limit".

The macro `\optional` produces the text

> *The functionality described in this section is optional.*

The macro `\optxt{Option_Function}` produces the text

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

The functionality described in this section is optional.

If `Option_Function` returns `False`, the implementation may cause all calls to the operations defined in this section to raise `POSIX_Error`. Otherwise, the behavior shall be as specified below.

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Section 8: Tables

The normal LaTeX `tabular` environment can be used.

In addition, special macros are provided to make it easier to write the tables that are used to show the mapping from C or LIS names to Ada names. Since these are likely to be phased out, now that we are converting to the POSIX.5 document format, they are only explained briefly here.

A numbered (floating) table begins with a macro call of the form `\begin{table_kind}{title_text}` and ends with a macro call of the form `\begin{table_kind}`, for the same `table_kind`. These environments take care of the table header, numbering, and column headers. For each type of table, there are also predefined macros that define lines of the table. After any line, one can have calls to the `\note` macro, so indicate references to numbered notes. Finally, there is a macro that marks the beginning of the (optional) "notes" section of the table, and a macro that defines a note.

A forward reference to the next table can be made from within the text that preceeds it using the macro `\listabnext`. This can be seen in the following example.

The verbatim text below produces Table 8.1, which is an example of an operator-correspondence table. The reference to the table in the preceding sentence was produced by the macro call `\listabnext`.

```
\begin{optable}{Operator Correspondence Table}
\optr[f]{C_function_name}[p]{Ada_Procedure_Or_Function_Name}
\param[i]{C_in_parameter_name}{Ada_In_Parameter_Name}
\param[o]{C_out_parameter_name}{Ada_Out_Parameter_Name}
\param[r]{C_in_out_parameter_name}{Ada_In_Out_Parameter_Name}
\param[*]{\return}{\return}\note{1}
\notes
\note{1} This indicates a function return value.
\end{optable}
```

The macro `\optr` generates a line containing a C-language function name and the corresponding Ada subprogram name. The macro `\param` is used to generate a line containing a C-language argument name and the corresponding Ada formal parameter name. The mode, which is assumed to be the same for both parameters, is given by the first macro-parameter.

Note that this example table produced index entries for the names of the subprograms, though formal parameter names are never indexed. If that effect is not desired, one needs to override it explicitly, by enclosing the entire table in a

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Table 5.1 – Operator Correspondence Table

C-Language	Ada-Language	Notes
C_function_name	Ada_Procedure_Or_Function_Name	
C_in_parameter_name (I)	Ada_In_Parameter_Name *(I)*	
C_out_parameter_name (O)	Ada_Out_Parameter_Name *(O)*	
C_in_out_parameter_name (I,O)	Ada_In_Out_Parameter_Name *(I,O)*	
<return-value>	*<return-value>*	(1)

NOTES:

(1) This indicates a function return value.

`\begingroup` ... `\endgroup` pair and using `dontindexX` to override the rules, as described further above.

The uses of the various table kinds and their names are as follows:

`optable`
: This is a 3-column table, for correspondences of operators (functions, procedures), with notes. Parameters are indented.

`optable*`
: Same as above, but without notes, and with only two columns.

`chtable`
: This is a "custom header" 3-column table, for pairwise correspondences, with notes. There are no special line-generating macros, so this must be handled specially for each use.

`chtable*`
: Same as above, but without notes.

`pairtable`
: This is the general two-column table for pairwise correspondences, with notes.

`pairtable*`
: Same as above, but without notes.

`strctable`
: This the the form of table used to show the correspondence between compnents of C/LIS structured types and components of the corresponding Ada private types, with notes.

`strctable*`
: Same as above, but without notes.

The general rule is that the forms ending in `*` not *not* allow notes.

The macros for generating table lines are as follows:

`\proc{C_Procedure_Name}{Ada_Procedure_Name}`
: This generates a pair of procedure names.

`\func{C_Procedure_Name}{Ada_Function_Name}`
: This generates a pair of function names.

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

`\optr[f]{C_Name}[f]{Ada_Name}`
: This generates a pair of operator names, possibly of different kind.

`\param[m]{C_Parameter_Name}{Ada_Parameter_Name}`
: This generates a pair of parameters, both of the same mode.

`\pair[r]{C_Name}[o]{Aaa_Name}`
: This generates a pair of parameters, of separately specified modes.

`\triple{C_Component_Name}{Ada_Func_Name}{Ada_Proc_Name}`
: This generates a three-item line, consisting of an LIS/C-language structured-type component-name, followed by the names of an Ada "get" function and an Ada "set" procedure.

`\triplen{C_Component_Name}{Ada_Function_Name}{non-indexed Ada name}`
: This is similar to the macro above, except that the third component is a "miscellaneous" Ada name, which is not indexed.

As mentioned previously, the left-side items, with names like `C_...`, are typeset and indexed as C-language names. The C-language POSIX interface specifications are being used as a basis for the Ada language bindings, since Language-Independent Specifications that do not yet exist.

Within table lines, you may find uses of certain other special macros. One of these is `\constrain{text}`, which is used for constraints on the values of C-language formal parameters.

If a table (or a figure) does not fit onto one page, you will need to split it manually. To split a table, try inserting the macro call `\continuetable` or `\concludetable` at the point where the split sould occur. The form `\continuetable` generates a caption with the notation "(continued)", and the form `\concludetable` generates "(concluded)".

Note: *These macros have not been well tested yet.*

Note: *Still missing explanation for the "gtable" environment, which is for home-cooked* LaTeX *tables.*

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Section 9: Miscellaneous Macros

9.1 Standard Phrases

Macros are define for certain standard phrases which are expected to need to be changed as the document progresses, to make it easier to change the exact form of the name or phrase without editing all the occurrences. These are listed in the table below.

Table 9.1 – Standard Phrases

macro	meaning
\posix{5}	POSIX.5
\posix{4}	POSIX.4
\posix{4a}	P1003.4a
\posix{20}	P1003.20
\posix{1}	POSIX.1
\posixwg{5}	P1003.5 working group
\thiswriting	date time-sensitive statements (e.g. January, 1994)
\thistandard	this standard
\theAdaStandard	the Ada Standard {1}
\theAdaNineXStandard	the draft Ada 9X standard[?]
\Clanguage	C-language interface
\posixref{4a}{3.3}	P1003.4a Section 3.3
\impldef	*implementation-defined*, for use in \doc calls
\draftnumber	the current document draft number
\dotivdraftnumber	the draft number of POSIX.4
\coverdate	date to be printed on the cover
\copydate	year(s) for the copyright notice
\down	reduces the font size by one step
\downtwo	reduces the font size by two steps
\usernote{text of note}	used for notes in normative text

9.2 Lists of Types, Constants, and Operations

The following macros are defined to format the lists of types, constants, and operations found at the start of many sections of POSIX.5.

```
\constantsConstant_1, ... Constant_9/

\types{\ada[t]{Type_1}, ... \ada[t]{Type_9}}

\operations{ada[f]{Function_1}, ... \ada[f]{Function_9}}
```

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Note: *At present, the bodies of thes macros have been nulled-out, to experiment with the idea of leaving this information out, since it appears to be redundant with the information provided in the "Synopsis" sections.*

Additional macros may be defined, to make it easier to convert POSIX.5 from `troff` to LaTeX, and to redo the example, further belowP1003.20 document in a format consistent with POSIX.5.

9.3 Annotations

Use the macro call `\annote{text}` to insert notes to yourself or other editors, that are not supposed to appear in the final draft. The definition of the `\annote` macro can be changed, to either print these out or ignore them. The default is to print them out to the log file. Take care to keep the text inside the annotation simple, since the output routine for messages to the log file is easily broken if the text has any embedded macro calls.

Note: *This is a sample annotation*.

9.4 Obsolescent Macros

Certain macros are supported that are going away.

Table 9.1 – Obsolescent Macros

obsolescent macro	replacement
`\adapkg{Ada_name}`	`\pkgsect`
`\posixlis`	`\posix`
`\thisection`	this section
`\adaref{label}`	`\secref`
`\LIS`	`\Clanguage`
`\adarm`	`\theAdaStandard`
`\posixLIS{1}`	`\Clanguage` or `\posix`
`\posixC{1}`	`\Clanguage` or `\posix`
`\LISa`	`\Clanguage` or `\posix`

Section A
(informative)
Bibliography

{1} Leslie Lamport, LaTeX *A Document Preparation System*, Addison-Wesley Publishing Company (1986).

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Alphabetic Topical Index

[...13, 15
Fourth-
Level_Heading_With_Underscores...6

A

Ada_Function_Name
function ... 27
Ada_Procedure_Or_Function_Name
procedure ... 30
Annotations...33

B

Bibliography...34

C

C_function_name
function ... 30
Change Bars...20
Change Bars, Fuzz Bars, Line Numbers...20
Constant_1
constant ... 32
Constant_9
constant ... 32

D

defining occurrence...19

E

Error Handling...27
Exclusive
constant ... 24

F

Fifth-Level Heading...6
File_Descriptor
type ... 8
File_Exists
constant ... 24
Filename_Too_Long
constant ... 24
Floating Code "Figures"...17
Frequent Errors...21
Fuzz Bars...20

G

General...2
General LaTeX Information...4

H

How to Get LaTeX...4

I

Indexing...18
indexing in-line text...19
Interrupted_Operation
constant ... 24

J

K

L

Labels and Cross-References...7
Large Ada Code Components...10
limit...27
Limit_Function
function ... 27
Line Numbers and Page Styles...22
Lists...24
Lists of Types, Constants, and Operations...32

M

Macros...5
Miscellaneous Macros...32

N

{NAME_MAX}
limit ... 24
Names and Short Text...8

O

Obsolescent Macros...33
Open_Or_Create_Shared_Memory
function ... 24
option...27
Option_Function ... 28
Option_Function

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

function . . . 28
Options. . .27

P

Part 2. . .15
{PATH_MAX}
limit . . . 24
POSIX_IO
package . . . 8
POSIX_IO.File_Descriptor
type . . . 8
_POSIX_NO_TRUNC
. . . 24
Programming Language Text. . .8

Q

R

Reserved Characters. . .4

S

Scope. . .2
Sectioning. . .5
Sixth-Level Heading. . .6
Some_Package_Name
procedure . . . 6
Some_Type_Identifier
type . . . 8
Spaces. . .6
Special "Boilerplate" Text. . .27
Standard Phrases. . .32
\ abbreviate
macro . . . 11
\ ada
macro . . . 8, 10, 20
\ adacmph*
macro . . . 15
\ adacmph
macro . . . 15
\ adacmpha
macro . . . 15–16
\ adacmphc
macro . . . 15
\ adacmphX
macro . . . 11, 16
\ adacomp
macro . . . 4, 10–11, 15–17, 20–21
\ adacompa
macro . . . 14–16
\ adacompc
macro . . . 14–16
\ adacompX
macro . . . 11, 16
\ adadot
macro . . . 8
\ adaerr
macro . . . 5
\ adaitm
macro . . . 25
\ adaopen
macro . . . 11
\ adapkg
macro . . . 33
\ adarat
macro . . . 5
\ adaratt
macro . . . 5
\ adarattt
macro . . . 5
\ adaratttt
macro . . . 5
\ adaref
macro . . . 33
\ adarefs
macro . . . 5
\ adarm
macro . . . 33
\ adasect
macro . . . 5, 22
\ adasectt
macro . . . 5
\ adasecttt
macro . . . 5
\ adasecttttt
macro . . . 5
\ adasectttttt
macro . . . 5
\ adasecttttttt
macro . . . 5
\ adaset
macro . . . 6
\ adasett
macro . . . 6
\ adasettt
macro . . . 6
\ adasetttt
macro . . . 6
\ adasettttt
macro . . . 6
\ adasetttttt
macro . . . 6
\ annote
macro . . . 33
\ beginadafig
macro . . . 17
\ C
macro . . . 10, 20
\ cand
macro . . . 10
\ cboff
macro . . . 12

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

\cbon
macro . . . 12, 21
\Cequal
macro . . . 10
\Cgeq
macro . . . 10
\Clanguage
macro . . . 32–33
\Cneq
macro . . . 10
\cod
macro . . . 12
\comment
macro . . . 12
\concludetable
macro . . . 31
\constants
macro . . . 32
\constraintext
macro . . . 31
\continuetable
macro . . . 31
\copydate
macro . . . 32
\coverdate
macro . . . 32
\definition
macro . . . 19
\doc
macro . . . 12, 32
\dontindexX
macro . . . 10
\dotivdraftnumber
macro . . . 32
\down
macro . . . 32
\downtwo
macro . . . 32
\draftnumber
macro . . . 32
\endadafig
macro . . . 17
\erritm
macro . . . 24
\errtexta
macro . . . 27
\errtxt
macro . . . 25, 27
\fboff
macro . . . 20
\fbon
macro . . . 20
\fromdotfive
macro . . . 12
\impldef
macro . . . 12, 32
\index
macro . . . 18
\indx
macro . . . 12, 19
\indxself
macro . . . 19
\item
macro . . . 26
\key
macro . . . 10, 12
\limit
macro . . . 27
\lis
macro . . . 8, 10, 20
\LIS
macro . . . 33
\LISa
macro . . . 33
\listabnext
macro . . . 29
\makeindex
macro . . . 18
\note
macro . . . 29
\operations
macro . . . 32
\option
macro . . . 27
\optional
macro . . . 27
\optr
macro . . . 29
\optxt
macro . . . 27
\pagestyle
macro . . . 22
\param
macro . . . 29
\pkgsect
macro . . . 5–6, 33
\posix
macro . . . 32–33
\posixC
macro . . . 33
\posixLIS
macro . . . 33
\posixlis
macro . . . 33
\posixref
macro . . . 32
\posixwg
macro . . . 32
\secref
macro . . . 33
\special
macro . . . 23
\subsection
macro . . . 15
\theAdaNineXStandard
macro . . . 32

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

\theAdaStandard
macro . . . 32–33
\thisection
macro . . . 33
\thistandard
macro . . . 7, 32
\thiswriting
macro . . . 32
\types
macro . . . 32
\use
macro . . . 19
\usernote
macro . . . 32
\varitm
macro . . . 25
\varlst
macro . . . 25
\verb
macro . . . 4

T

Tables. . .29
Third-Level Heading. . .6

U

V

W

X

Y

Z

Copyright ©1994 Ted Baker. All rights reserved.
this is an unapproved Draft, subject to change.

Note: *The index macros do not alphabetize macro-names correctly, since they currently are protected from expansion by the "string" macro. This causes them all to be indexed under the letter "S". This is a defect.*