ProofPower

DOCUMENT PREPARATION

PPTex-2.9.1w2.rda.110727

Copyright © : Lemma 1 Ltd. 2006

Information on the current status of ProofPower is available on the World-Wide Web, at URL:

http://www.lemma-one.demon.co.uk/ProofPower/index.html

This document is published by:

Lemma 1 Ltd.
2nd Floor
31A Chain Street
Reading
Berkshire
UK
RG1 2HX
e-mail: pp@lemma-one.com

PPTex-2.9.1w2.rda.110727 13:54:16 27/07/2011

CONTENTS

0 ABOUT THIS PUBLICATION **5**
- 0.1 Related Publications . . . 5
- 0.2 Assumptions . . . 5

1 INTRODUCTION **7**

2 FORMATTING DOCUMENTS **9**
- 2.1 Getting Started . . . 9
- 2.2 Including Standard ML . . . 10
- 2.3 Formal and Narrative Text . . . 10
- 2.4 Making an Index of Defined Terms . . . 11
  - 2.4.1 Indexing Extended Characters . . . 12
  - 2.4.2 Indexing Strings . . . 12
  - 2.4.3 Restrictions on Indexing . . . 12
  - 2.4.4 Indexing and `.sml` Files . . . 13

3 CATEGORIES OF TEXT IN A DOCUMENT **15**
- 3.1 Basic Categories . . . 15
  - 3.1.1 Narrative Text . . . 15
  - 3.1.2 Standard ML . . . 15
  - 3.1.3 Formal Text Within Narrative Text . . . 16
  - 3.1.4 Ignored Text . . . 16
  - 3.1.5 Example . . . 17
- 3.2 Z Language Material . . . 19
- 3.3 HOL Language Material . . . 21
  - 3.3.1 Constants . . . 21
  - 3.3.2 Labelled Products . . . 22
- 3.4 Other Categories . . . 22
  - 3.4.1 Standard ML Item Descriptions . . . 22
  - 3.4.2 Manipulating Files . . . 23
  - 3.4.3 Other Standard ML Categories . . . 24
  - 3.4.4 Theory Documentation . . . 25
  - 3.4.5 General Formal Text . . . 25
  - 3.4.6 Miscellaneous . . . 26
  - 3.4.7 Formal Rules . . . 27

4 LaTeX ENVIRONMENTS **29**
- 4.1 Input Format . . . 29
- 4.2 Formatting Formal Rules . . . 29
  - 4.2.1 Examples . . . 31
  - 4.2.2 LaTeX Environment For Rules . . . 32
  - 4.2.3 Format of Rule Environments . . . 32
  - 4.2.4 Types of Rule . . . 33

4.3 Miscellaneous . . . 33
4.4 Quirks and Foibles . . . 34
4.5 Controlling The Layout . . . 34
4.5.1 Left and Right Margins . . . 34
4.5.2 Page Breaks in Formal Text . . . 36
4.5.3 Space Width . . . 36
4.5.4 Tab Intervals . . . 36
4.5.5 Centering In Rule Environments . . . 37
4.5.6 Width of Rule Environments . . . 37
4.5.7 Tabs In Rule Environments . . . 38
4.5.8 Showing Extended Character Images . . . 39
4.5.9 Underscores . . . 39
4.5.10 Controlling Indexing . . . 39
4.6 Vertical Bars on Formal Texts . . . 40
4.7 Labels on Formal Texts . . . 40
4.8 Line Numbers in LaTeX Error Messages . . . 41
4.9 Error Handling . . . 41

5 COMMONLY USED PROGRAMS 43
5.1 Steering Files for Sieving Programs . . . 43
5.2 `doctex` . . . 43
5.3 `docsml` . . . 43
5.4 `doctch` . . . 44
5.5 `doctds` . . . 44
5.6 `texdvi` . . . 44
5.7 `docpr` . . . 44
5.8 `docdvi` . . . 45

6 SUPPORTING PROGRAMS AND FILES 47
6.1 `sieve` . . . 47
6.1.1 Synopsis . . . 47
6.1.2 Description . . . 47
6.1.3 Options . . . 47
6.1.4 Standard Input Format . . . 48
6.1.5 Conversion of Extended Characters . . . 48
6.1.6 Conversion of Percent Keywords . . . 49
6.1.7 Format of Steering Files . . . 49
6.1.8 Keyword File . . . 49
6.1.9 View File Format . . . 51
6.1.10 View-File Macros . . . 54
6.1.11 Output Redirection . . . 54
6.1.12 Directive Lines . . . 54
6.1.13 Examples . . . 55
6.1.14 Errors . . . 57
6.1.15 Limits . . . 58
6.2 `sieveview` — Steering File for `sieve` . . . 58
6.3 `sievekeyword` — Steering File for `sieve` . . . 58
6.4 `findfile` . . . 59
6.5 Font Files . . . 59
6.6 Style File `hol1.sty` . . . 59

7 INSTALLING THE PACKAGE 61

8 FILE FORMATS **63**

9 EXTENDED CHARACTER SET **65**
9.1 Greek Letters . . . 65
9.2 Logic, Equivalence and Related Symbols . . . 65
9.3 Set Symbols . . . 65
9.4 Arrows . . . 65
9.5 Formal Text Brackets . . . 65
9.6 Padding Symbols . . . 66
9.7 Index Brackets . . . 66
9.8 Bracketing Symbols . . . 66
9.9 Subscription and Superscription . . . 66
9.10 Underlining . . . 66
9.11 Relation, Sequence and Bag Symbols . . . 67
9.12 Miscellaneous . . . 67
9.13 Extended Character Images . . . 67
9.14 ASCII Keywords for Special Symbols . . . 69
9.15 Using `Xpp` to work with the Extended Character Set . . . 70

REFERENCES **71**

INDEX **73**

ABOUT THIS PUBLICATION

This document, one of several making up the user documentation for the ProofPower system, describes the facilities for preparing documents containing HOL, Z orwww similar formal material.

0.1 Related Publications

A bibliography is given at the end of this document. Publications relating specifically to ProofPower are:

1. ProofPower Tutorial *[4]*, tutorial covering the basic ProofPower system.
2. ProofPower Z Tutorial *[7]*, tutorial covering ProofPower Z support option.
3. ProofPower-HOL Reference Manual *[8]*, the reference manual for the ProofPower-HOL system.
4. ProofPower-Z Reference Manual *[10]*, the reference manual for the ProofPower-HOL system.
5. ProofPower Xpp User Guide *[9]*, the user guide for the X Windows interface to ProofPower.

0.2 Assumptions

Some familiarity with LaTeX is assumed.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

INTRODUCTION

Text files, often termed 'document files' or just 'documents', form the basis of a user's interactions with the ProofPower system. They contain the formal material comprising specifications and proofs as well as narrative material for documentation purposes.

The formal material is used as input to various programs, most notably to the ProofPower specification and proof tool. This material is held in the document file in the *same* form that is presented to these programs, this also the *same* form as it will appear on paper when printed. A strong point of the documentation system is this *sameness* and that any differences are simple to explain. Hence an auditor of a system produced using the ProofPower system can easily see that the text in the source file *is* the text compiled or submitted to the proof system and *is* the same as the text printed.

Some utility programs and supporting files are provided to assist in the handling of literate scripts, *i.e.*, files containing a mixture of formal material (*e.g.*, HOL, Standard ML code, or Z) and narrative text. The utilities support the extraction of formal material for processing (*e.g.*, by the Standard ML compiler) or for typesetting. The typesetting aspects are oriented towards LaTeX, but some are suitable for use with other typesetting systems, such as `ptroff.`

For typesetting the package consists of a LaTeX style file (*i.e.*, a macro suite) named `hol1.sty`; some font files for screen use, namely, `holnormal` for normal use and `holdouble` for demonstrations; two shell scripts `doctex` and `texdvi` which are the recommended means of preparing documents for typesetting and for running LaTeX in normal use. There are additional supporting programs which are normally invoked via the shell scripts but which may be called directly for special effects.

The shell script `doctex` (with the supporting programs) allows the user to prepare a LaTeX file using an extended character set described in section 9. With this character set the formal material appears on the screen much as it does in the printed document, and the document can be processed directly with ProofPower. Tools are available for editing files in this format. Under the X Windows System, `xpp` is the recommended tool; Consult ProofPower `Xpp` *User Guide* [9] for further information.

This document describes the typesetting facilities, largely by example. We assume some familiarity with LaTeX, which is described in [2] and Z which is described in [3].

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

FORMATTING DOCUMENTS

This section describes the basic facilities of the document processing system, these provide for formatting narrative text, inclusion of Standard ML source code and indexing of defined names. Many other types of material may be contained in a document, including Z paragraphs (see [3]). Section 3 describes the full set of text types and section 3.2 describes the support for Z paragraphs.

2.1 Getting Started

To install the document processing package see section 7.

To use the package you prepare a file, say `myfile.doc`, using an editor which supports an appropriate extended character font. The font `holnormal.vfont` is recommended. The characters available in this font are described in section 9 together with the character codes and the keystrokes used to access them. Another way of entering the extended characters is by copying from the palette which may be displayed with the command `palette`.

The form of the file is essentially that of a LaTeX file except that you can use extended (*i.e.*, non-ASCII) characters to type Z or HOL constructs, or constructs of other languages. These characters are shown in section 9 below. The remaining extended characters are used for bracketing formal text for printing and other processing, these characters are discussed elsewhere in this document.

The first LaTeX command in the document (normally this is the `\documentstyle` command) should be preceded with a line containing just the characters '`=TEX`'. Any lines before this are simply ignored. The significance of this line will be made clear in section 6.1 where the `sieve` program and the view-file are discussed. For the moment, be aware that any line whose first character is an equals sign '`=`' is a directive line which indicates the type of processing for the lines that follow.

What to do next depends on whether you want to use LaTeX2e mode or LaTeX2.0.9 compatibility.

For LaTeX2e, there is a package called `ProofPower.sty`. To use it, use the normal LaTeX2e `\documentclass` command for the type of document you are preparing (book, article, etc.) and then put the following command in your document preamble:

```
\usepackage{ProofPower}
```

The LaTeX2.0.9 style file is called `hol1.sty`. To use it, the `\documentstyle` command line that you use should include the option `hol1`. *e.g.*, this document starts with:

```
=TEX
\documentstyle[TQ,hol1,11pt,ifthen]{article}
```

to get the options `TQ` (which gives our house style which is used at an eleven point size) and `hol1`.

For LaTeX2e, there is a package called `ProofPower.sty`. To use it, use the normal LaTeX2e `\documentclass` command for the type of document you are preparing (book, article, etc.) and then put the following command in your document preamble:

| *\usepackage{ProofPower}*

To process file `myfile.doc` and produce a `.dvi` file you run `doctex` to form the `.tex` file then run LaTeX (via `texdvi`) as follows:

| *doctex myfile*
| *texdvi myfile*

The above recipe assumes that you have put the directory containing the programs on your search path and the LaTeX style files where LaTeX can find them — all subsequent mentions of programs, etc., will assume that the directories are set up correctly, section 7 describes how to do this. If your document does not include a request to make an index as described in 2.4 below, `texdvi` will report on the fact that the file `myfile.idx` does not exist or cannot be read. This message may be ignored if you do not wish your document to have an index.

The basic job in preparing the `.tex` file is to translate each non-ASCII character into LaTeX macro calls which typeset that character and to add LaTeX macro calls to properly format the formal material. The lines starting with an equals '`=`' sign and some of the extended characters are directive lines which control the way in which it does this, as we shall see.

Because of the way LaTeX works when producing the self-referential aspects of a document (*e.g.*, table of contents, page and section references, *etc.*) the program `texdvi` will need to be run up to four times. If bibliographies use BibTeX then the program `bibtex` should be run between the first and second execution of `texdvi`.

Program `doctex` needs to be run once before the first execution of `texdvi`, thereafter it only needs to be executed after the source file (e.g., `myfile.doc`) is amended.

2.2 Including Standard ML

Program text written in Standard ML may be included, it is introduced by a directive line containing the characters '`=SML`'. After this comes the Standard ML code, then a directive line containing the characters '`=TEX`'. The document may contain many segments of Standard ML code, each surrounded[1] by '`=SML`' and '`=TEX`', the command '`docsml myfile`' extracts the code segments and writes them into the file `myfile.sml`. There are no restrictions on the size or contents of the code segments, they may contain complete or partial declarations etc., a segment may contain several declarations. Section 4.5.2 gives help about printing Standard ML segments which are over a page in length.

2.3 Formal and Narrative Text

The Standard ML code segment is an example of the range of types of 'formal text' supported. The segment starting with the '`=TEX`' directive line is one of the types of 'narrative text'. Other types of formal and narrative text are discussed later on.

Within narrative text segments extended characters may be used to obtain many symbols not found in the normal keyboard. These include many mathematical symbols, particularly those of Z, plus the Greek letters.

[1] More precisely, the segments of Standard ML code are introduced with an '`=SML`' and are normally concluded by an '`=TEX`' directive, but any other directive line may be used.

Within formal text segments the extended characters are available plus a number of percent keywords. Each extended character has a corresponding percent keyword, but not vice versa. The percent keywords give access to further symbols.

To use percent keywords to get symbols in text that is to be typeset in a paragraph, also to get the same formatting style as is used in formal text, a segment starting with the directive '`=INLINEFT`' is used, an '`=TEX`' directive returns to narrative text. Blank lines before and after these two directive lines should be avoided or LaTeX will interpret them as paragraph separations. Section 3.1.3 gives more details of '`=INLINEFT`'.

Formal text is typeset in LaTeX math mode, but with some changes. Extended characters and percent keywords are recognised and the corresponding symbols are displayed. Space characters are significant. The LaTeX special characters, such as '`\`', '`$`', '`&`', '`#`', '`$`' and '`_`' lose their special meaning, they are just printed.

2.4 Making an Index of Defined Terms

An additional pair of extended characters, '⦏' and '⦐' may be used to help make an index of defined terms for the document. If you type, say, '`⦏InterestingThing⦐`', somewhere in `myfile.doc` then InterestingThing will be printed in bold and the LaTeX macro `\index` will be called with 'InterestingThing' as parameter. If, following the instructions in [2], you have used `\makeindex` earlier in your document this will cause an entry of the following form (note the "*11*" is because this text is on page number *11*):

```
\indexentry{InterestingThing }{11}
```

to be written in the file `myfile.idx`. The program `texdvi` then sorts this file and puts the output in `myfile.sid`.

A macro `\printindex` is supplied in the `hol1` style option for including the index in the printed document. To use it type `\printindex` in your file where you want the index to appear. It reads the file `myfile.sid` in an environment in which each `\indexentry` produces entries whose form may be seen in the index to this document. Long indexes are best printed in two columns by preceding the `\printindex` call by a call of `\twocolumn`, and smaller characters than usual might be used. The index for a document might be included by using the following LaTeX commands (this document actually uses this two column style rather than just calling `\printindex`):

```
\twocolumn[\section{INDEX}]
{       \footnotesize
        \printindex
}
\onecolumn
```

Note that the calls of `\twocolumn` and `\onecolumn` will force page breaks. If the index is the last part of the whole document then the `\onecolumn` may be omitted.

In formal text, where percent keywords are understood, the keywords '`%SX%`' and '`%EX%`' may be used instead of '⦏' and '⦐' respectively.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

2.4.1 Indexing Extended Characters

Most of the extended characters (which are shown in section 9) may be used in names to be indexed, these characters may be the whole name. Characters that cannot be indexed are described in section 9.

2.4.2 Indexing Strings

Some identifiers in ProofPower are defined via Standard ML strings, these names may be indexed by enclosing the whole string including its quotation character between the indexing characters or percent keywords. If the quotation characters are outside the indexing characters or percent keywords then the name includes those characters or keywords. The actual text of the index entry omits the quotation characters.

Consider the following example of a section of a document. (The '=SMLLITERAL' is explained in section 3.4.3.)

```
=SML
⌊"Index−1"⌋
"⌊Index−2⌋"
=SMLLITERAL
"⌊Index−3⌋"
```

Which prints as:

SML

```
"Index−1"
"Index−2"
```

SML

```
"⌊Index−3⌋"
```

These lines produce index entries (as shown in the index at the end of this document) for `Index-1` and `Index-2`, i.e., without the quotation characters. No index entry is made for `Index-3`. Processing by `docsml` discards the first two pairs of indexing characters but retains those around `Index-3`. When the text is compiled in the extended Standard ML of ProofPower the indexing characters in the `"⌊Index-3⌋"` are retained as part of the string.

2.4.3 Restrictions on Indexing

The characters '⌊' and '⌋' and the processing that is applied to the LaTeX indexing system are intended for the names of defined terms and symbols in the formal text. Using it for the general purpose indexing of other text, or for text containing tabs or spaces is not supported.

This document includes two types of indexed entries. First, many of the examples of formal text define some name, and these are indexed using the '⌊' and '⌋' characters. Second, the names of various programs described as part of the text formatting package are indexed, for example, the program `doctex` is described in a section starting:

```
\section{{\tt doctex}\index{doctex }}
```

If an index-like feature for general text is required then the use of the LaTeX glossary system, see [2], is suggested.

2.4.4 Indexing and `.sml` Files

When `docsml` creates the `.sml` file any indexing characters and keywords in '`=SML`' sections are deleted. If indexing characters or keywords are required in the `.sml` file then the '`=SMLLITERAL`' category, see section 3.4.3, should be used.

Note that `docsml` attempts to convert text in an '`=SMLLITERAL`' section to Standard ML by a simple algorithm: all extended characters are converted to their string literal form, i.e., to a backslash followed by their character code as a three-digit decimal number. This means that extended characters outside of Standard ML string literals will provoke error messages from the Standard ML compiler.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

CATEGORIES OF TEXT IN A DOCUMENT

A source document is divided into a number of segments of text of various categories. Each segment starts with a directive line whose first letter is an equals '=' sign or is one of a small number of the extended characters. These extended characters are used for categories such as Z schemas and they are discused later, see section 3.2.

In the rest of this document we will refer to categories and directive lines by merely quoting the first few letters of the directive line. For example, the idiom "... an '=TEX' category ..." should be understood to mean a segment of text whose first line is the directive line starting with the characters '=TEX'.

3.1 Basic Categories

The getting started section, section 2.1 above, introduced some of the categories. Here they and some other basic categories are described. Several other categories are provided, they are described in later sections, particularly in sections 3.4.

3.1.1 Narrative Text

Text to be processed by the normal rules of LaTeX is introduced by a directive line containing the four characters '=TEX', this text is copied into the `.tex` file by `doctex` but discarded by `docsml`. Program `doctex` additionally converts all extended characters in this category into calls of LaTeX macros that, in most cases, will print the image of the extended character.

3.1.2 Standard ML

Standard ML source code, which is an example of formal text, is headed by a directive line containing the four characters '=SML', this text is copied into the `.sml` file by `docsml` to form the text to be read by the Standard ML compiler. Program `docsml` actually handles an extended Standard ML which is described in [5]. Text in the '=SML' category is written in a WYSIWYG[1] manner where the text is written exactly as it would be presented to the Standard ML compiler. Such text is processed by `doctex` to obtain a verbatim-like layout on the printed page.

[1] Acronym: WYSIWYG = what you see is what you get.

3.1.3 Formal Text Within Narrative Text

Short sections of formal text may be included in paragraphs by heading them with a '=INLINEFT' directive line. These section may be split over line breaks. This category is intended mostly to provide the space processing performed in '=SML' categories.

For example, the text fragments " $f\ a \wedge g\ b$ " and "h "i" $\wedge$ j k_1 $\wedge$ $l__m$" were produced by the following source file text.

```
... text deleted
        the text fragments "
        =INLINEFT
        f a ∧ g b
        =TEX
        {} " and "%
        =INLINEFT
        h "i" ∧ j k_1
         ∧ l__m
        =TEX
        " were produced
... text deleted
```

Note how the spacing before and after the formal texts is controlled. A space is normally produced before the inlined text, it may be suppressed with a LaTeX comment (i.e., the % character). Spaces after the inlined text are normally suppressed, an explicit space may be included by placing an empty group (as shown above) or by using the ~ character. Within the text newline characters are ignored, space characters may need to be added at the start of lines.

3.1.4 Ignored Text

Text headed by the directive line '=IGN' or '=IGNORE' is discarded by both doctex and docsml, such text segments may be used for adding commentary and notes into a document but which are not to be extracted. They might also be used to remove (or comment out) sections of a document without actually deleting the text.

The part of a document file before the first directive line is always discarded.

3.1.5 Example

A simple document in `myfile.doc` containing two Standard ML functions might be written as follows. Recall that the text before the first directive line is ignored, thus this area may be used to include version information about the document including keywords understood by the UNIX `sccs` program.

Document example

```
File: myfile.doc
SCCS version: %Z% %E% %I% %M%

=TEX
\documentstyle[hol1,11pt]{article}
\makeindex
\begin{document}
This example document contains a Standard ML function
to calculate factorials and Fibonacci numbers.

=SML
fun %SX%factorial%EX% (n:int) : int = (
        if n > 0
        then    n * factorial(n-1)
        else    1
);
=TEX

=SML
fun ⌊fibonacci⌋ (n:int) : int = (
        if n <= 0
        then    0
        else    if n = 1
                then    1
                else    fibonacci(n-1) + fibonacci(n-2)
);
=TEX

Index of defined names.

\printindex
\end{document}
```

Notice the keywords and extended characters used for creating an index of defined names. In particular notice how they are interchangeable which is illustrated by indexing *factorial* using percent keywords and *fibonacci* using extended characters.

The body of this example document would print as follows. Notice the vertical bar on the left hand side which is headed with 'SML'.

This example document contains a Standard ML function to calculate factorials and Fibonacci numbers.

SML
```
fun factorial (n:int) : int = (
        if n > 0
        then    n * factorial(n-1)
        else    1
);
```

SML
```
fun fibonacci (n:int) : int = (
        if n <= 0
        then    0
        else    if n = 1
                then    1
                else    fibonacci(n-1) + fibonacci(n-2)
);
```

Index of defined names.

factorial 18
fibonacci 18

Processing the same example with `docsml` would produce the file `myfile.sml` containing the following text.

```
fun factorial (n:int) : int = (
        if n > 0
        then    n * factorial(n-1)
        else    1
);
fun fibonacci (n:int) : int = (
        if n <= 0
        then    0
        else    if n = 1
                then    1
                else    fibonacci(n-1) + fibonacci(n-2)
);
```

3.2 Z Language Material

Two categories of Z material are supported, both will be printed (via `doctex`), but only one is included in the output from `docsml`. Here they are termed formal and informal. The formal Z material is labelled with 'Z' above it top left hand corner whereas the informal material has 'Informal Z', see section 4.7.

We give an example of each sort of Z paragraph (using the terminology of Spivey's Z book [3]).

A given type definition, from page 3 of [3]. Notice that the printed form differs from [3] by the addition of the vertical line labelled with a "Z", this is in keeping with all of the formal text categories provided by the document processing suite.

z

| [**NAME**, **DATE**]

This was typed in as follows.

```
ⓈZ
|[⌊NAME⌋, ⌊DATE⌋]
■
```

A Z schema box, from page 13 of [3].

z

BirthdayBook1

$names: \mathbb{N}_1 \rightarrow NAME$
$dates: \mathbb{N}_1 \rightarrow DATE$
$hwm: \mathbb{N}$

$\forall i,j: 1 \,..\, hwm \bullet$
$\quad i \neq j \Rightarrow names(i) \neq names(j)$

This was typed in as follows.

```
┌─⌊BirthdayBook1⌋────────────────────
│    names:ℕ↑1→NAME
│    dates:ℕ↑1→DATE
│    hwm:ℕ
├──────────────────────────
│    ∀i,j: 1 .. hwm •
│        i≠j ⇒ names(i) ≠ names(j)
└──────────────────────────────
```

An axiomatic description box, from page 39 of [3].

z

limit:$\mathbb{N}$

$limit \leq 65535$

This was typed in as follows.

```
│        ⓈZAX
│        │      ⌊limit⌋:ℕ
│        ├──────────────
│        │      limit ≤ 65535
│        ■
```

A Generic constant, from page 41 of [3].

z

$[X, Y]$

first $: X \times Y \to X$

$\forall x{:}X;\ y{:}Y \bullet first(x, y) = x$

This was typed in as follows.

```
│        ╒══[X, Y]═════════════
│        │⌊first⌋ : X × Y → X
│        ├────────────────────
│        │∀x:X; y:Y • first(x, y) = x
│        └─────────────────────
```

The remaining forms of paragraph do not have a boxed form in [3]. They are entered in a similar fashion to given types. The examples are, in order: schema definition, from page 9 of [3]; abbreviation definition, page 19; free type definition, from page 82; and, constraint, page 51.

z

RAddBirthday $\hat{=}$ $(AddBirthday \land Success) \lor AlreadyKnown$

Informal Z

DATABASE $== ADDR \to PAGE$

z

TREE $::= tip \mid fork \langle\langle \mathbb{N} \times TREE \times TREE \rangle\rangle$

z

$n_disks < 5$

The first of these was typed in as follows, the other have the same start and end symbols.

```
│        ⓈZ
│        ⌊RAddBirthday⌋ ≙ (AddBirthday ∧ Success) ∨ AlreadyKnown
│        ■
```

Points to note.

- The '|', '—' and '═' characters are optional. In many cases they allow the source text to look like very much like the final printed Z text.

- The '├' character separates declarations from predicate in the Z boxes. This character should be the first on a line, the only other characters allowed on that line are '│', '—' and '═'. Adding other characters will normally provoke LaTeX error messages.
- For document printing purposes the characters '■' and '└' are interchangeable, however the language specific processors may impose further restrictions.
- Examples of Z material may be required for enhancing the narrative in a document, these are Z paragraphs that are *not* to be sieved out for formal processing by the Z support system. Such text is known as informal Z material. To include such informal Z material the category names should be replaced as follows.

Formal	Informal
┌	ⓈSimple┌
╒	Ⓢ╒
ⓈZAX	ⓈIZAX
ⓈZ	ⓈIZAX

  In informal schema and generic constant boxes the category name must be followed by a padding character (i.e., '│', '—' or '═') or by white space. In the formal equivalents the category name is a single character which does not require separation from the rest of the line. It is recommended that these category names are always followed by a padding character, as in the examples above, then the only distinction between the formal and informal boxes is the leading 'Ⓢ'.

- In ProofPower-Z abbreviation definitions and horizontal schema definitions both use the ' $\widehat{=}$ ' character whereas in [3] the abbreviation definition uses ' == '.
- The vertical bars and the "Z" labels at the left and top of the Z paragraphs may be suppressed, see sections 4.6 and 4.7, thus making Z paragraphs display closer to the style of [3], but this may obscure the distinction between formal and narrative texts and between formal texts in different languages.

3.3 HOL Language Material

3.3.1 Constants

HOL constants may be defined together with a constraining predicate. Each declaration may define one or many HOL constants.

HOL Constant

```
│c1 : type1;
│c2 : type2
├──────────────────────────
│p c1 c2
```

The above constant was typed in as follows.

```
ⓈHOLCONST
c1 : type1;
c2 : type2
├
p c1 c2
■
```

For details of the use of HOL constants refer to function '*const_spec*' in [8].

3.3.2 Labelled Products

HOL labelled product types may be defined. The labelled product type has a name plus a series of HOL declarations separated by semicolons.

Example: the type *LAB_PROD* which has four components.

HOL Labelled Product

LAB_PROD_EXAMPLE

```
l1 : type1;
l2 : type2;
l3 : type3;
l4 : type4
```

The above labelled product type was typed in as follows.

```
ⓈHOLLABPROD ⌜LAB_PROD_EXAMPLE⌝
l1 : type1;
l2 : type2;
l3 : type3;
l4 : type4
■
```

For details of the use of labelled products refer to function '*labelled_product_spec*' in [8].

3.4 Other Categories

Programs `doctex` and `docsml` support the categories described in the preceding sections via a 'view-file' and a 'keyword-file' (i.e., command files for the program `sieve`, see section 6.1) which describes the categories available and the processing that they require. This view-file has many other entries which are described here in various groups.

The text in this section and in sections 2.1 and 3.1 should be reguarded as a description of the facilities provided by the `sieve` program, see section 6.1, and the standard view-file and keyword-file

3.4.1 Standard ML Item Descriptions

A standard format is used by the developers of ProofPower for documenting Standard ML functions etc. This is the format used, for example, in the ProofPower reference manuals, [8, 10]. The categories shown in table 3.1 are provided to support this format.

Category name	Copied by	Type
=DOC	doctex	Formal
	docsml	
=SYNOPSIS	doctex	Narrative
=DESCRIBE	doctex	Narrative
=FAILURE	docsml	Formal
	doctex	
=FAILUREC	doctex	Narrative
=EXAMPLE	doctex	Narrative
=USES	doctex	Narrative
=COMMENTS	doctex	Narrative
=SEEALSO	doctex	Narrative
=KEYWORDS	doctex	Narrative
=ENDDOC	doctex	Narrative

Table 3.1: Help Box Contents

3.4.2 Manipulating Files

Categories are provided for manipulating other files.

Text may be written or appended to files. This is useful for documents which contain shell scripts, make files and data files, it allows them to be written and documented using the same mechanisms as, for example, Standard ML code. Category ‘=DUMP’ allows formal text to be printed by `doctex` plus `texdvi` and extracted into a named file by `docsml`. The directive line ‘=DUMP auxfile1’ indicates to `docsml` that the following lines are to be written to the file `auxfile1` overwriting if it already exists. When printed, the vertical line at the left of the formal text is headed with the text ‘`Text dumped to file auxfile1`’. Category ‘=DUMPMORE’ allows formal text to be appended to a file, the directive line ‘=DUMPMORE auxfile2’ indicates to `docsml` that the following lines are to be appended to the file `auxfile2`, the header of the printed form replaces `dumped` with `appended`.

Within the formal text of ‘=DUMP’ and ‘=DUMPMORE’ categories extended characters are allowed. The text written out by `docsml` will have any indexing characters (i.e., ‘⌊’ and ‘⌋’) deleted. This allows, e.g., makefile target names to be indexed but not have the indexing characters included in the makefile.

Categories ‘=VDUMP’ and ‘=VDUMPMORE’ are similar to ‘=DUMP’ and ‘=DUMPMORE’ respectively, the difference being that all characters in the category are written to the file. This allows, e.g., the indexing characters to be written to a file. The `tex` view of these characters is the same as for category ‘=GFTSHOW’.

No checks are made to ensure that the filename is valid. Failing to open the file is considered a serious error: sieving will stop.

Categories ‘=SH’ and ‘=CSH’ may be used to execute arbitrary shell commands. The text following the directive is written to the standard input of a Bourne shell or a C-shell, respectively, by `docsml` (see `sh(1)` and `csh(1)` in the Sun UNIX manual set). The text in these categories is printed with the shell name at the top of the left hand vertical bar. These categories are useful for, e.g., setting the execute flag on shell scripts created with the ‘=DUMP’ category.

Text from other files may be included by using the ‘=INCLUDE’ category. The text of this category is a list of file names. When printed these names are just listed. With `docsml` the contents of the

files are included by supplying the full list of file names as arguments to the UNIX command `cat`, see `cat(1)` in the Sun UNIX manual set.

An example of these file manipulation commands occurs in the ProofPower Parser implementatiojn where the grammar of ProofPower is written to an auxiliary file by an '`=DUMP`', the parser generator is invoked with an '`=SH`' which writes the generated parser to another auxiliary file which is included in the Standard ML to be compiled by an '`=INCLUDE`' directive. The following lines are extracted from the parser implementation to show how the file manipulation commands are used.

Extract from ProofPower Parser Implementation

```
... text deleted

        =DUMP imp019.grm.txt
        (* Terms *)
                Tm      =

... text deleted

        =TEX

... text deleted

        =SH
        poly `arch`slrp.db >imp019.grm.run <<!
        Slrp.slrp{infile="imp019.grm.txt", outfile="imp019.grm.sml",
                        logfile="imp019.grm.log",  eos="HLEos", loglevel=2};
        PolyML.quit();
        !
        =TEX

... text deleted

        =INCLUDE
        imp019.grm.sml
        =TEX

... text deleted
```

3.4.3 Other Standard ML Categories

Text in the category '`=SMLLABELLED`' is treated exactly the same as that in category '`=SML`' except that a user supplied label is used rather than `SML` for the text placed on the left hand vertical bar. The user's label is supplied as arguments on the directive line, there may be none in which case no label is printed.

Text in the category '`=SMLPLAIN`' is treated similarly to that of text in category '`=SMLLABELLED`', the distinction is that keywords are not recognised when processed by `docsml`. This category is useful for

text that includes percent characters that are not to be otherwise processed. Examples include: TEX and LATEX macro files, where percent characters normally denote comments; and text that includes SCCS keywords.

Some sections of Standard ML include indexing characters that are part of the program text rather than controlling how that text is to be printed. A small number of other characters, those used for Z box drawing, need are treated in the same manner. Text in the category '=SMLLITERAL' will be printed so that all the extended characters are visible, in particular the '⌜' and '⌝' are displayed and do not cause indexing. When processed by `docsml` all of the extended characters are converted to the Standard ML string literal form (namely backslash plus three decimal digits) and thus will be read by Standard ML but keywords will not be recognised. Thus '=SMLLITERAL' is useful when extended characters occur only in Standard ML strings. Note that in the extended Standard ML of ProofPower indexing characters in strings are retained but elsewhere they are discarded, see also section 2.4.2.

Category '=SMLLITERAL' is intended for use when extended characters are wanted in Standard ML strings where the normal purpose of the characters is to achieve some printing effect. For example, when initialising a data structure containing such strings. Note that in this case the Standard ML identifiers within the segment may not contain extended characters — Standard ML compilation errors will be reported if this rule is violated. Standard ML code which contains a mixture of identifiers with extended characters and strings with the extended characters for printing effects needs a combination of the '=SML...' categories and the '\Show...' macros (see 4.5.8) to obtain the correct output.

3.4.4 Theory Documentation

Describe[2] the categories '=THDOC' and '=THSML'.

3.4.5 General Formal Text

Sections of formal text which are to be printed in the style of the '=SML' but discarded by `docsml` are included in the '=GFT' category. This allows an optional label on the directive line which is printed at the head of the left hand vertical bar. This category is used by many of the examples in this document. The extract from the parser shown in section 3.4.2 above starts with the following lines.

```
=GFT Extract from ProofPower Parser Implementation
... text deleted

        =DUMP imp019.grm.txt
```

A similar environment is provided by the '=GFTXQ' category, but here the quotation characters ' " ' and ' ` ' are displayed as themselves rather than being toggled between ' “ ' and ' ” ' and between ' ‘ ' and ' ’ ' respectively.

Category '=GFTSHOW' is similar to '=GFTXQ' but here all of the extended characters are shown, none of the special interpretations of (e.g.) indexing and padding characters are made.

[2] To be done.

3.4.6 Miscellaneous

For the convenience of the authors of documents two further ignored categories are provided, namely '=TEMP' and '=TEST'.

3.4.7 Formal Rules

Formal rule structures having three main parts, conventionally the name, the assumptions, and the conclusions may be formed using the '`=FRULE`' category. The '`=FRULE`' directive has two arguments. First the value '`1`' or '`2`' showing how many horizontal lines are wanted. Second the label to use at the head of the left hand vertical bar. They are displayed in the following form, similar forms are used for conversions, tactics and theorems.

Rule

$$\frac{\textit{assumptions}}{\textit{conclusions}} \qquad \textit{name}$$

Formal rules are discussed in detail in section 4.2.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

LaTeX ENVIRONMENTS

Several LaTeX environments are provided by the `hol1` style, they are not normally invoked by typing the `\begin{...}` and `\end{...}` commands of the environments defined in the LaTeX book [2], rather they are used via the capabilities of the sieving process and the various categories of text they support.

Table 4.1 lists the environments available in the `hol1` style. Each environment corresponds to a piece of formal text. To use the environments you use the directive or characters shown under 'opening' and 'closing' in the table.

The environments marked with an asterisk (*) take a parameter giving the name of the schema, the parameters of the generic constant or the title for the formal text. For the last case the parameter may be empty in which case the label field is suppressed.

4.1 Input Format

Within any of the above environments, text is typeset with each 'line of input' corresponding to a line in the printed document. Here if one of the bracketing pairs listed above encloses the text then the phrase 'line of input' actually means a line of input from the source file; if you are using `\begin{...}` and `\end{...}` to delimit the environments, then a 'line of input' must have its beginning and end marked with `\+` and `\\` respectively (apart from the lines containing a '├' plus optional '—' characters).

Tab and space characters may be used to control the layout within each line of input. All space characters give a small amount of space in the printed output. Tab characters are used to give type-writer style tabbing, see section 4.5.4 for more details.

Some of the environments support Z schema boxes where the visual layout of the source text is important. For these a range of extended characters are provided that allow the schema boxes to be drawn. Some other environments, including formal rules and ..., use the same characters to allow a good layout.

The shape of these box drawing characters has been chosen so that you can use them, together with the extended characters '|', '—', '═' and '├', to draw boxes which look on the screen like the printed form. The characters '|', '—' and '═' are merely padding and have no effect on the form of the printed document, although other programs may use them.

4.2 Formatting Formal Rules

Rule structures have three main parts, conventionally the name, the assumptions, and the conclusions. They are displayed thus:

Environment	Opening	Closing	Purpose
	character or directive		
FRULE	=FRULE	=TEX	Rules, tactics, conversions, etc.
GFT*	=GFT	=TEX	General formal text using a single font and with a user supplied label, quote character are toggled
GFTSHOW*	=GFTSHOW	=TEX	General formal text where all of the extended characters are shown rather than specially treated, uses a single font and with a user supplied label, quote character are not toggled,
GFTXQ*	=GFTXQ	=TEX	General formal text using a single font and with a user supplied label, quote character are not toggled
HELPDOC	=DOC	=ENDDOC	Help documentation
HOLConst	ⓈHOLCONST	■	HOL constants
HOLLabProd	ⓈHOLLABPROD	■	HOL labelled product types
MLCode	=ML	=TEX	Environment for ML code, toggles between two fonts, font changes at term bracketing characters
ZAxDes	ⓈZAX	■	Z axiomatic description
ZAxDes	ⓈIZAX	■	Informal Z axiomatic description
ZGenConst*	╒	└	Draw a Z generic constant, the argument is the generic parameters
ZGenConstInformal*	Ⓢ╒	└	Draw an informal Z generic constant, argument is generic parameters
ZOther	ⓈZ	■	Other Z paragraphs (i.e., those which are not schemas, generic constants or axiomatic descriptions)
ZOtherInformal	ⓈIZ	■	Other informal Z paragraphs (i.e., those which are not schemas, generic constants or axiomatic descriptions)
ZSchema*	┌	└	Draw a Z schema box, argument is schema name
ZSchemaInformal*	Ⓢ┌	└	Draw an informal Z schema box, argument is schema name

Table 4.1: LaTeX Environments

Rule

$$\frac{\textit{assumptions}}{\textit{conclusions}} \qquad \textit{name}$$

Each of the three parts may have multiple lines. The name part is always aligned centrally with the horizontal line. There may be one or two horizontal lines. Additionally, each structure may be labelled, the word 'Rule' labels the above display. Similar layouts are used for conversions, tactics and theorems.

4.2.1 Examples

The *Modus Ponens* rule and the *induction tactic* may be displayed as follows.

Rule

$$\frac{\Gamma 1 \vdash t1 \Rightarrow t2;\ \Gamma 2 \vdash t1'}{\Gamma 1 \cup \Gamma 2 \vdash t2} \qquad \Rightarrow_\mathbf{elim}$$

Tactic

$$\frac{\{\ \Gamma\ \}\ t1 \wedge t2}{\{\ \Gamma\ \}\ t1;\ \{\ \Gamma\ \}\ t2} \qquad \wedge_\mathbf{tac}$$

These are typed in the source text as follows.

```
=FRULE 1 Rule
⌊⇒_elim⌋
├
Γ1 ⊢ t1 ⇒ t2; Γ2 ⊢ t1′
├
Γ1 ∪ Γ2 ⊢ t2
=TEX

=FRULE 2 Tactic
│⌊∧_tac⌋
├────────────────
│{ Γ } t1 ∧ t2
├────────────────
│{ Γ } t1; { Γ } t2
=TEX
```

Significant points shown above are as follows.

- The name, assumptions and conclusions are given in that order and are separated by the large turnstile '├' character, which should occur on a line on its own.
- The use of the vertical '│' and horizontal '—' bars to make the second example more readable.
- The first argument to the '=FRULE' directive gives the number of horizontal lines, either 1 or 2.
- The type of rule box is indicated by the second argument to the '=FRULE' directive, the type is used as the label.
- The line '=TEX' completes each rule.

4.2.2 LaTeX Environment For Rules

The rule environments are implemented with the LaTeX environment '`=FRULE`'. It takes two parameters. First, a single digit, either 1 or 2, indicating the number of horizontal lines wanted. Second the label for the rule environment, or an empty parameter if no label is wanted.

Within the name, assumptions and conclusions the lines of input must be formatted according to the rules given in section 4.1.

4.2.3 Format of Rule Environments

A rule environment contains the following 7 parts, in the stated order.

1. A line containing the rule type, types are given in section 4.2.4 below.
2. The name part, which should include any non-theorem arguments.
3. A large turnstile '├' character with optional horizontal bars '—' on a line on their own.
4. The assumptions part.
5. Another large turnstile '├' character with optional horizontal bars '—' on a line on their own.
6. The conclusions part.
7. Start another sieving section, normally a directive line containing just the text '`=TEX`'.

Each of the name, assumptions and conclusions may comprise of several lines, each of which may start with the optional vertical '|' bar, if desired. Lines in the assumptions and conclusions are individually centered above or below the horizontal lines. The name part is left justified. An example:

Tactic

Assumptions line 1
2
Third assumptions line
and the fourth
═══════════════════
First conclusions line
second
3
4, 4, 4, 4
fifth

Name line 1
2
Third name line

The source text for this example is as follows.

=FRULE 2 Tactic
Name line 1
2
Third name line
⊢
Assumptions line 1
2
Third assumptions line
and the fourth
⊢
First conclusions line
second
3
4, 4, 4, 4
fifth
=TEX

Rule environments that contain only one large turnstile '⊢' character are assumed to contain just the name and conclusion parts, those with no large turnstiles just the name part. Any third or subsequent turnstile is ignored, any text that follows them is treated as further lines of the conclusions.

Any rule environment with fewer, or greater, than two large turnstiles '⊢' characters will cause a warning message to be written to the terminal and to the log file for the LaTeX run.

4.2.4 Types of Rule

FST project conventions use a small number of types of rule, namely: conversions, rules and theorems with one horizontal line; and, tactics with two horizontal lines. All of these types are illustrated within this document.

The directive lines for these four types of rule are as follows.

=FRULE 1 Conversion
=FRULE 1 Rule
=FRULE 1 Theorem
=FRULE 2 Tactic

4.3 Miscellaneous

The final environments may be used for any text that is to be displayed in the verbatim-like fashion of the other formal text environments.

Categories '`=GFT`', '`=GFTXQ`' and '`=GFTSHOW`' correspond to environments of the same name, i.e., `GFT`, `GFTXQ` and `GFTSHOW`.

The `GFTSHOW` environment is used, via the '`=GFTSHOW`' directive, for the examples of 'how to type ...' throughout this document.

4.4 Quirks and Foibles

The package is designed on the assumption that text in the various categories uses the conventional category codes and macro definitions of LaTeX plus the style file `hol1.sty`.

For the formal text categories (of which only the '`=SML`' and '`=INLINEFT`' categories have been introduced so far), characters such as '\', '$', '&', '#', '$' and '_' are not to be treated specially, they are just to be printed. The intention is that the formal text categories give a verbatim representation of their contents apart from extended characters and percent keywords which are replaced by the symbols.

The package has also been designed on the assumption that if you use the LaTeX `\begin{...}` command to invoke one of the environments described in section 4 you are doing it to use LaTeX to get some special effect[1] (but, see 3.4.4 below for an exception to this). The package therefore only serves to translate extended characters in text which is not enclosed in a pair of bracketing characters. It does not put in controls for extra spaces or for printing LaTeX special characters, on the assumption that you will want to set the spacing yourself and to use the special characters as special characters.

Extended characters may be used in the LaTeX sectioning commands, but in such cases they should not be used in maths mode, *i.e.*, do not enclose them in dollar '$' signs. The sectioning commands are the commands that create entries for the table of contents, such as `\section`, `\subsection` and`\subsubsection`. Having the extended characters in maths mode will often cause LaTeX error messages to be issued when it reads the '`.toc`' file to generate the table of contents.

4.5 Controlling The Layout

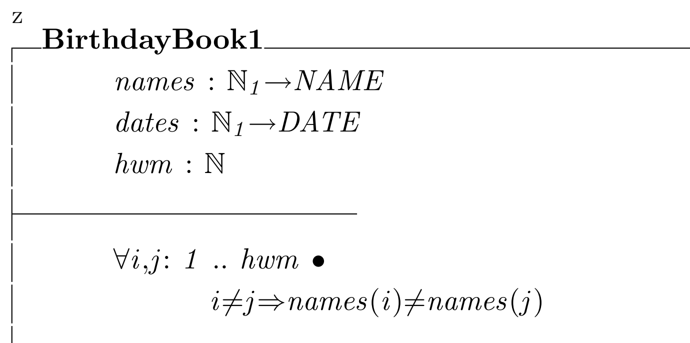
4.5.1 Left and Right Margins

The dimension registers `\ftlmargin` and `\ftrmargin` may be used to adjust the additional left and right margin space used for the boxes. These are set to give no additional space by default. For example, the schema below starts and finishes with the following lines

```
|         {\ftlmargin=1.5in \ftrmargin=1.5in
|         ┌─[ BirthdayBook1 ]──────────────────
|... text deleted
|         └──────────────────────────────────
|
|         }
```

to the source text yields the following:

[1] If you want to do this you will need to know that lines of text in these environments are typeset in math mode, adjusted so that all spacing (even around relation symbols *etc.*) must be requested explicitly. See the style file for details of how this is done.



When formal text is included in LaTeX list environments (namely: `description`, `enumerate`, `itemize` and `list`) it is, by default, placed against the left hand margin rather than against the indented margin implied by the list environment. To move the formal text to the indented margin `\ftlmargin` should be set to the sum of the LaTeX list indentations for the enclosing lists. These values are held as the dimensions `\leftmargini`, `\leftmarginii`, `\leftmarginiii`, `\leftmarginiv`, `\leftmarginv` and `\leftmarginvi`. The usage of these dimensions can be seen in the following examples.

1. The next formal text is at the default left hand margin.

| *Formal text*

   Paragraphs of narrative text are indented.

2. This formal text is once indented, the commands required to achieve this effect are shown.

   | *{\ftlmargin=\leftmargini*
   | *=GFT*
   | *Formal text*
   | *=TEX*
   | *}*

   The above formal text has the same indentation as paragraphs of narrative text.

   - Double nested list environments.
   - This formal text is twice indented, the commands required to achieve this effect are shown.

     | *{\ftlmargin=\leftmargini*
     | *\advance\ftlmargin by \leftmarginii*
     | *=GFT*
     | *Formal text*
     | *=TEX*
     | *}*

     (a) Triple nested list environments.
     (b) This formal text is three times indented, the commands required to achieve this effect are shown.

```
{\ftlmargin=\leftmargini
\advance\ftlmargin by \leftmarginii
\advance\ftlmargin by \leftmarginiii
=GFT
Formal text
=TEX
}
```

The above formal text has the same indentation as paragraphs of narrative text.

(c) For more deeply nested lists add in further indentation dimensions in a similar manner.

4.5.2 Page Breaks in Formal Text

The count register `\ftlinepenalty` sets the penalty which applies at the end of each line of formal text in the various environments. It is set to *10000* by default. This value means that no page breaks will be allowed within such an environment. If you want to allow page breaks use *e.g.*, `\ftlinepenalty=0`.

No page breaks are possible in rule environments, so changing the value has no effect for rules.

It is recommended that large pieces of formal text, such as HOL theory listings and some HOL proofs, that may spread across more than one page are preceded with a `\ftlinepenalty=0` so that LaTeX may choose convenient page breaks.

4.5.3 Space Width

Spaces in formal texts have a width given by the TeX maths skip `\ftspaceskip` which has a default value of `7 mu`. Maths skip distances can be seen from the following example.

Skips

```
4 4, 5 5, 6 6,7 7, 8 8, 9 9,10 10, 11 11
some words that show off
the above maths skip sizes
```

Which was produced as follows.

```
\begin{GFT}{Skips}
\+4\mskip4mu4, 5\mskip5mu5, 6\mskip6mu6,
7\mskip7mu7, 8\mskip8mu8, 9\mskip9mu9,
10\mskip10mu10, 11\mskip11mu11\\
\+some\mskip4muwords\mskip5muthat\mskip6mushow\mskip7muoff\\
\+the\mskip8muabove\mskip9mumaths\mskip10muskip\mskip11musizes\\
\end{GFT}
```

4.5.4 Tab Intervals

In all of the environments tab characters are interpreted specially and give typewriter-style tab stops. The size of the tabbing interval is given by the dimension `\tabstop` whose default value is $\frac{1}{2}$ inch.

For instance, the example in section 4.5.1 above uses one tab at the beginning of each line but the last, which has two. By putting tabs in before the colons in the signature part of the schema, one can get the types to line up in columns:

z
BirthdayBook1

$names : \mathbb{N}_1 \rightarrow NAME$
$dates : \mathbb{N}_1 \rightarrow DATE$
$hwm : \mathbb{N}$

$\forall i,j\colon 1 \mathinner{..} hwm \bullet$
$\quad i \neq j \Rightarrow names(i) \neq names(j)$

It may require some experiment to get the right number of tabs. The tab stop interval can be changed by setting the dimension register `\tabstop` to the required interval. *E.g.*, with

$\backslash tabstop{=}1.0in$

our example looks like:

z
BirthdayBook1

$names \quad : \mathbb{N}_1 \rightarrow NAME$
$dates \quad : \mathbb{N}_1 \rightarrow DATE$
$hwm \quad : \mathbb{N}$

$\forall i,j\colon 1 \mathinner{..} hwm \bullet$
$\qquad i \neq j \Rightarrow names(i) \neq names(j)$

4.5.5 Centering In Rule Environments

Text in the assumptions and conclusions is normally centered above and below the horizontal lines. It may be left justified above by calling the macro `\FruleLeftJustify`. An example of the use of this command is in the next section, 4.5.7.

4.5.6 Width of Rule Environments

By default the left and right parts of a rule are set in formal text areas which are each 45% of the whole width available, i.e., the line width less the space for margins as held in `\ftlmargin` and `\ftrmargin`, the remaining 10% of the width is used for spacing between the items. The widths of the two parts may be adjusted, the 10% for spacing is fixed.

To alter the sizes of the left and right parts of a rule macros `\FruleLeftWidth` and `\FruleRightWidth` may be redefined with the required widths. Their sum should not exceed 90% of the whole width available.

For example, with the default widths the rule below would cause an 'overfull hbox' error message, but by adjusting the widths in the manner shown the error message may be avoided.

```
{\def\FruleLeftWidth{0.7\textwidth}
\def\FruleRightWidth{0.2\textwidth}
=FRULE 2 Tactic
z_∀_elim
⌜_Z t⌝
├
Γ ⊢ ∀ D[x↾1,...] | P↾1[x↾1,...]• P↾2[x↾1,...]
├
Γ ⊢ "D[t.x↾1,...] as predicate" ∧ P↾1′[t.x↾1,...] ⇒ P↾2′[t.x↾1,...]
=TEX
}
```

Tactic

$$\frac{\Gamma \vdash \forall\ D[x_1,...]\ |\ P_1[x_1,...]\bullet\ P_2[x_1,...]}{\Gamma \vdash \text{"}D[t.x_1,...]\ as\ predicate\text{"} \land P'_1[t.x_1,...] \Rightarrow P'_2[t.x_1,...]} \qquad \begin{array}{l} z_\forall_elim \\ ⌜_Z t⌝ \end{array}$$

4.5.7 Tabs In Rule Environments

The lines of input in rules may contain TAB characters for alignment purposes in the same way that they may be used in other formal text environments, but when the text is centered the alignment may not be as expected. It is recommended that TAB characters be used in rules *only* when `\FruleLeftJustify` has been selected. An example of the use of this command is as follows.

Theorem

```
Tab−free assumption line
A        ssum   ptio    ns
Some more ass           unpt    ions              N      a      m      e
-----------------------------------------         Name line without tabs
Concl   usions
Furt    her c   onc     lusi    ons
Tab−free line
```

Which was produced with the following input text.

```
{\FruleLeftJustify
=FRULE 1 Theorem
N[tab]a[tab]m[tab]e
Name line without tabs
├
Tab−free assumption line
A[tab]ssum[tab]ptio[tab]ns
Some more ass[tab]unpt[tab]ions
├
Conclus[tab]ions
Furt[tab]her c[tab]onc[tab]lusi[tab]ons
Tab−free line
=TEX
}
```

Omitting the `\FruleLeftJustify` has the following effect.

Theorem

Tab−free assumption line
A ssum ptio ns
Some more ass unpt ions
Concl usions
Furt her c onc lusi ons
Tab−free line

N a m e
Name line without tabs

Note how the tabs are not lined up, they appear to give random gaps.

4.5.8 Showing Extended Character Images

Normally some of the extended characters cause formatting effects such as indexing but it may be necessary to cause these characters to be printed. A range of macros are provided to allow various sections of the extended character set to be shown. These controls are normally invoked automatically via one of the sieving categories but they may be called by the user to get particular printing effects.

Macro `\ShowScripts` causes the subscripting and superscripting characters to be shown.

Macro `\ShowBars` causes the padding characters to be shown.

Macro `\ShowIndexing` causes the indexing characters to be shown.

Macro `\ShowBoxes` causes the big turnstile plus the characters that start and end Z schemas and generic constants to be shown.

Macro `\ShowAllImages` calls all of the above showing macros.

These macros may be called before the formal text environments to make their characters be shown on the printed result. They may have their scope limited by enclosing them in TEX groups.

In many simple cases the '`=SML`...' categories (see section 3.4.3) provide an alternative method for displaying extended characters.

4.5.9 Underscores

Some document authors want underscores to have their normal TEX meaning, i.e., as subscription, others want underscores to print as themselves. Macro `\underscoreoff` makes underscore a normal printing character, and `\underscoreon` restores its usual TEX meaning. The default is that underscore have their normal TEX meaning. These macros may have their scope limited by enclosing them in TEX groups.

This document is written with an `\underscoreoff` near its start, the effects of this may be seen in several of the examples where underscore characters are freely used without needing to be protected by backslash characters.

4.5.10 Controlling Indexing

Indexed terms are normally shown in a bold font. Macro `\HOLindexPlain` causes indexed terms to be shown in the same font as their enclosing text and macro `\HOLindexBold` restores the bold font.

It may useful to be able to turn of indexing over some blocks of text even though the indexing macros (or extended characters) are still used. Macros `\HOLindexOff` and `\HOLindexOn` turn on and off, respectively, the indexing actions of the indexing extended characters and keywords. (The `\index` macro of LATEX is not affected.)

All of these macros may have their scope limited by enclosing them in TEX groups.

For example, this paragraph uses the extended characters for indexing. This term **ci_one** uses the default style of indexing. This term *ci_two* is displayed with the enclosing font. These terms **ci_three** and *ci_four* are not indexed. This term **ci_five** uses the default style of indexing.

The above paragraph was typed as follows.

For example, this paragraph uses the extended characters
for indexing. This term ⌊ci_one⌋ uses the default style
of indexing. This term {\HOLindexPlain ⌊ci_two⌋} is
displayed with the enclosing font. These terms \HOLindexOff
⌊ci_three⌋ and \HOLindexPlain ⌊ci_four⌋ \HOLindexOn
\HOLindexBold are not indexed. This term ⌊ci_five⌋ uses
the default style of indexing.

Additional index entries may be added with the `\HOLindexEntry` macro, it takes one argument which is the item to be indexed. The text is not included at the point of call, indexing is controlled by the `\HOLindexOff` and `\HOLindexOn` commands. Thus, the following might be a useful idiom for some documents.

\def\myindex#1{\HOLindexOn\HOLindexEntry{#1}\HOLindexOff}

4.6 Vertical Bars on Formal Texts

Formal text is normally printed with a vertical bar on its left hand edge. This bar may be suppressed by using `\vertbarfalse`, the corresponding `\vertbartrue` may be used to bring back the vertical bars as may the normal LATEX grouping mechanism. For example, the following is printed

SML
some text

by entering the following text.

{\vertbarfalse
=SML
some text
=TEX
}

4.7 Labels on Formal Texts

Several of the types of formal text provide a label at the top left hand corner or above the left hand vertical line. The text of these labels may be altered by redefining their macros. For long labels on the Z boxes it may be necessary to include `\strut` in the macro body. Table 4.2 gives gives details of the labelling macros and their default label.

Sieving category	LaTeX environment	Label macro	Default label
ⓈHOLCONST	HOLConst	\HOLConstLabel	HOL Constant
ⓈHOLLABPROD	HOLLabProd	\HOLLabProdLabel	HOL Labelled Product
=ML	MLCode	\MlLabel	ML
ⓈIZAX	ZAxDesInformal	\ZAxDesInformalLabel	Informal Z
ⓈZAX	ZAxDes	\ZAxDesLabel	Z
�together	ZGenConstInformal	\ZGenericInformalLabel	Informal Z
Ⓢ�together	ZGenConst	\ZGenericLabel	Z
ⓈZ	ZOther	\ZOtherLabel	Z
ⓈIZ	ZOtherInformal	\ZOtherInformalLabel	Informal Z
┌	ZSchema	\ZSchemaInformalLabel	Informal Z
Ⓢ┌	ZSchemaInformal	\ZSchemaLabel	Z

Table 4.2: Formal Text Labelling

4.8 Line Numbers in LaTeX Error Messages

Error and warning messages produced by LaTeX include line numbers, these refer to the `.tex` file that LaTeX actually reads, not to the `.doc` file. Whenever LaTeX opens a file for reading it prints an open bracket '`(`' to the terminal and to the `.log` file, followed by the file name. A matching closing bracket '`)`' is printed when LaTeX closes the file.

4.9 Error Handling

A few error messages are reported by the style file, most faults are left to be reported by LaTeX or other parts of the documentation system. Some of these errors may include translations of extended characters into their '\Pr$\mathcal{NN}${}' form, see section 6.1.5, which is used internally by the document processing system. To assist the user in understanding these error messages the translations are shown in section 9.13.

Some extended character values are reserved for future expansion. If one of these is found in a `.doc` file then the default action is for the style file to issue an error message and display the character as its hexadecimal value enclosed in a box. For example, the character value 128_{10} would be shown as "0x80" if it were one of the reserved characters. The error message may be suppressed by calling the macro `\NoMoaning` in a group surrounding such characters.

A number of error messages may be produced when formatting rules, section 4.2.3 gives details.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

COMMONLY USED PROGRAMS

Most of the processing tasks required to produce a printed form of a document are encapsulated in shell scripts documented here. These scripts use other programs described later in this document. The idiom for typesetting the literate script, `myfile.doc` is to run the following scripts.

doctex myfile
texdvi myfile

This generates the file `myfile.dvi` containing the LaTeX output for printing with `pstex` or displaying with `dvipage`. The names of these programs are formed by concatenating the extension of the source file with the extension of the main output file.

The reason for having separate commands, `doctex` and `texdvi`, is that one usually needs to run LaTeX several times (*i.e.*, using `texdvi` up to four times) to ensure that contents lists, references, indexes and similar are all in step.

Similar shell scripts are provided to extract Standard ML text from a document to create the `.sml` file.

5.1 Steering Files for Sieving Programs

Some shell scripts invoke the `sieve` program with a view-file and keyword-file found by program `findfile` to do the sieving. The default view-file and keyword-file give the categories described in the previous sections of this document. For these shell scripts the `-f` option may be used to name an alternate view-file, similarly `-k` may be used to name additional keyword-files. Option `-K` may be used to suppress the default keyword-file.

5.2 `doctex`

doctex [−*v*] [−*f view_file_name*] [−*K*] [−*k keyword_file_name*] *<files...>*

Each source file is read and sieved to produce a corresponding `.tex` file.

File name arguments may be the whole name of the `.doc` file or just the stem part, i.e., for file `myfile.doc` either `myfile.doc` or `myfile` may be used.

If the `-v` option is given the programs prints out the names of the source and main output files. The `-f`, `-K` and `-k` options are described in section 5.1.

5.3 `docsml`

docsml [−*v*] [−*f view_file_name*] [−*K*] [−*k keyword_file_name*] *<files...>*

Each source file is read and sieved to produce a corresponding `.sml` file.

File name arguments and options are the same as with program `doctex` in section 5.2.

5.4 doctch

doctch [−*v*] [−*f view_file_name*] [−*K*] [−*k keyword_file_name*] <*files...*>

Each source file is read and sieved to produce a corresponding `.tch` file.

File name arguments and options are the same as with program `doctex` in section 5.2.

5.5 doctds

doctds [−*v*] [−*f view_file_name*] [−*K*] [−*k keyword_file_name*] <*files...*>

Each source file is read and sieved to produce a corresponding `.tds` file.

File name arguments and options are the same as with program `doctex` in section 5.2.

5.6 texdvi

texdvi [−*v*] [−*b*] [−*p TeX_program*] <*files...*>

This program runs LaTeX on each source file in turn to produce a `.dvi` file ready for printing or for viewing on screen. Normally it will be necessary to run `texdvi` up to four times to ensure that the page numbers, tables of contents and inter-page references are correct. Also, `bibtex` should be run after the first run of `texdvi` to form the `.bib` file which contains the body of the cross references section of the document.

File name arguments may be the whole name of the `.tex` file or just the stem part, i.e., for file `myfile.tex` either `myfile.tex` or `myfile` may be used.

After running LaTeX any `.idx` file is sorted to create a `.sid` file which may be read in to form the body of an index to the document, see also section 2.4. Before running LaTeX program `texdvi` ensures that a `.sid` file (possibly and empty one) exists so that the first run of LaTeX will not complain because of a non-existent `.sid` file, this allows `texdvi` to be used from make files.

The `-p` option may be used to select a program other than LaTeX (the default program is `latex`).

If the `-v` option is given the programs prints out the names of the source and main output files.

If the `-b` option is given, BibTeX is run after running LaTeX.

5.7 docpr

docpr [−*n*] [−*p*] [−*s*] [−*v*] [−*w width*] <*files...*>

This program produces a verbatim listing of one or more files which may contain characters in the extended character set. By default the listings are sent to a printer using `pstex` The `-n` option is used to add line numbers to the listings. The `-p` option is preserve the `.dvi` file, i.e., it is not deleted when `docpr` completes. The `-s` option causes the output to be sent to the screen previewer `dvipage`, allowing selected pages to be printed using the appropriate `dvipage` commands. The `-v` option causes details of some of the files read and written (but not all of the LaTeX auxiliary files) to be listed on the standard output. Long lines are folded to a width of 80 characters to allow the whole line to be listed, the `-w` option may be used to alter the folding point.

5.8 docdvi

docdvi [−*v*] [−*f view_file_name*] [−*K*] [−*k keyword_file_name*]
  [−*p TeX_program_name*] [− $\mathcal{N}$] *<filename> ...*

This program that combines the actions of `doctex`, `bibtex` (which is part of the basic TeX distribution) and `texdvi` with the intention of fully processing a simple document from its `.doc` form to a printable `.dvi` file. The option $-\mathcal{N}$ controls how many times LaTeX should be invoked, the default is three (i.e., '*−3*'), the values of $\mathcal{N}$ may be in the range 1 to 4 inclusive. LaTeX and `bibtex` are run so that if they detect errors and thus would normally prompt for input they will read an end of file and thus stop immediately.

In some cases an extra run of LaTeX may be required. In these cases LaTeX will output the message: '*LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.*

The remaining potions are the same as for the File name arguments and options are the same as with programs `doctex` in section 5.2 and `texdvi` in section 5.6.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

SUPPORTING PROGRAMS AND FILES

Several programs and data files are used to support the programs in section 5 above.

6.1 `sieve`

The program `sieve` implements a general literate script mechanism. It is used to implement the specific mechanisms for scripts containing LaTeX and Standard ML. Its description is more complex than that of the other utilities. We follow the style of the UNIX manual pages.

6.1.1 Synopsis

sieve [−*l*] [−*v*] [−*d number*] [−*f view_file_name*] [−*K*] [−*k keyword_file_name*] *view*

Multiple '`-k` ...' options may be given.

6.1.2 Description

This program is a filter which enables a single text file to be used to hold data for processing by a range of different programs. The view argument is used to indicate which program sieve is preparing the output for. (The idea is similar to Knuth's Web system, but much simpler).

Directive lines are used to indicate that portions of the file belong to particular 'categories', where a category is just a tag. A view file relates categories to 'views' and shows the processing required of each category in each view. In the filtering process for a particular view the text of some categories is copied to the standard output, perhaps with modifications. The text of other categories may be discarded.

6.1.3 Options

`-f` Names a view file, default is `sieveview`. If the name does not begin with a '`/`' and does not identify a file in the current directory the program looks for the view file in the directories named in the environment variable `PATH`.

`-K` Suppress reading of the default keyword file `sievekeyword` which the program looks for in the directories named in the environment variable `PATH`.

`-k` Names a keyword file to be read. If the name does not begin with a '`/`' and does not identify a file in the current directory the program looks for the keyword file in the directories named in the environment variable `PATH`. Multiple keyword files may be read by repeating this option.

`-l` Shows the values of program limits, see section 6.1.15, in addition to the other actions of the program.

`-v` Shows the program version in addition to the other actions of the program.

`-d <number>` Causes copious diagnostic tracing on the standard output. The information produced depends on the number given which is interpreted as a series of binary flags. Multiple `-d` options are cumulative, their values being or-ed together. An aggregate value of zero (the default) means no diagnostic output.

Two of the bits have defined values. The effect of setting other bits may vary from issue to issue of the program and users must not rely upon the outputs produced by these other bits.

If the least significant bit is set (e.g., by '`-d 1`') then details of the view-file as interpreted for the requested view are printed, the exact format of this is not defined and may vary from issue to issue of the program.

If the second-least significant bit is set (e.g., by '`-d 2`') then details of the keyword-file are printed, the exact format of this is not defined and may vary from issue to issue of the program.

Options may be given in any order. White space is optional between an option and its argument, thus '`-d 1`' and '`-d1`' have the same effect.

Options may be merged provided all but the last have no arguments. Thus the string '`-v -l -d 1`' may be written as '`-vld 1`'. However, the following strings will not have the desired effect: '`-vld 1`', '`-ldv 1`' and '`-ld1v`' — options are recognised by the UNIX library function `getopt(3)`, see the Sun UNIX manual for further details.

6.1.4 Standard Input Format

The sieving process reads the program's standard input which comprises of a mixture of directive and other lines. Directive lines indicate how the next lines, up to the next directive or the end of the file, are to be processed. Directive lines take either of the following forms.

Directive lines

=*<equals category>* *<arguments>*

<extended category> *<arguments>*

The '=' of an equals category or the initial character of an extended category must be the first characters on the line.

Any text on lines preceding the first directive line is ignored.

The `<arguments>` are a white space separated set of words, their number should correspond to the numbers given to the `<arguments>` specification for the category in the view-file. The ninth argument, if used, is formed from all of the words on the line after the eighth argument.

Section 6.1.12 explains in full detail how directive lines are processed.

6.1.5 Conversion of Extended Characters

In some of the copying actions extended characters are converted to LaTeX macro calls of the form '`\Pr`$\mathcal{NN}$`{}`' where the '$\mathcal{N}$' are derived from the hexadecimal value of the character: *0* is converted to '`A`', *1* to '`B`' and so on, up to 15_{10} which is converted to '`P`'. The call is completed by a pair of curly brackets so that space characters in the source text are preserved. The non-standard encoding for the hexadecimal numbers is chosen so that conventional LaTeX macro names may be used. These

names should never need to be typed by a user of the document processing system and they should only be seen when examining the contents of a `.tex` file to determine the cause and location of LaTeX error messages.

The full set of extended characters provided by the default steering and font files are shown in section 9.13. Note that the sieve program is independent of the actual values and images of the characters, the steering files provide all the information required.

6.1.6 Conversion of Percent Keywords

In some of the copying actions percent keywords, that is strings of the form '%*ldots*%', are converted into an appropriate LaTeX macro.

Keywords corresponding to extended characters which are specified as `directive` type in the keyword file are always recognised when they occur at the start of a line of the source file. When recognised at the start of the line these keywords introduce a new category of text.

Keywords are always recognised on directive lines. Unknown or mal-formed keywords will be reported.

6.1.7 Format of Steering Files

Two steering files are used by `sieve`, the view-file and the keyword-file. Throughout these steering files the two character sequence backslash-newline (i.e., in C language terms the string ' `"\\\n"` ') is ignored, thus long lines may be split into two or more shorter lines. After this line merging, blank lines and comment lines are ignored. Comment lines are those whose first non white space character is a hash '`#`' sign.

To get a line that finishes with a backslash either add an extra backslash and then a blank line, or add a space after the backslash. In the second case the trailing space will be retained (it will be significant in the echoing actions described below).

6.1.8 Keyword File

The keyword file gives the association of extended characters with keywords and the LaTeX macros needed to print them.

The standard keyword file, `sievekeyword`, may be used as an example. It may be found, assuming the document processing system is installed, by using the `findfile` program as follows.

> *findfile sievekeyword $PATH*

Multiple keyword file may be used, the effect is as if they were concatenated and then read as a single file. The first keyword file read is the default `sievekeyword`, unless the '`-K`' option is given. After this any other keyword files named on the command line are read. Having no keyword files at all (i.e., by using the '`-K`' option and no '`-k` ...' options) is valid.

Each keyword specification line has three or four elements separated by white space.

The first element is the keyword being defined. This includes the percent '`%`' characters.

The second element gives both the nature of this keyword and the interpretation of the third and fourth elements. It is one of the following words.

simple The third element gives the character code of the equivalent extended character or if there is no character then the value `-1` is used. The fourth element may be omitted if the third is not `-1`, if present it gives the LaTeX macros used to print this keyword, they may contain white space.

The LaTeX macro is used when producing text from the `cat` action with the `latex` option set. If it is not provided then the extended character or keyword is output in the '`\Pr`$\mathcal{NN}$`{}`' form (see section 6.1.5). If both a character code and a LaTeX macro are given then the macro element is output rather than the '`\Pr`$\mathcal{NN}$`{}`' form of the extended character code.

The fourth element may also include a hash sign "`#`" followed by a specification of an argument to the LaTeX macro to be extracted from the text on an input line following the keyword or extended character. This specification comprises an optional minus sign "`-`" followed by a regular expression (in the POSIX extended regular expression syntax). If the minus sign is omitted, the argument is taken to be the longest sequence of input characters that match the regular expression (or empty if no match is found). If the minus sign is present, the argument is taken to be shortest sequence of input characters delimited by and not including a sequence of characters matching the regular expression (or the rest of the input line if no match is found).

index Same format as `simple` keywords, but denotes an extended character and keyword used for surrounding text to indexed.

directive Same format as `simple` keywords, but denotes that the extended character is a directive character that is a complete category name. These keywords must not[1] have a character code of `-1`.

startdirective Same as `directive` keywords, but the extended character is used as the first character of category names.

verbalone Same format as `simple` keywords, but indicates the extended characters and percent keywords that are recognised by the `verbalone` option to the `cat` action of the view-file.

sameas The third element is another keyword which must have been given elsewhere in the file. There is no fourth element. The third element must not be a keyword which itself was defined by a `sameas` entry. The first-element keyword is defined to be the same as the third-element keyword.

white Same format as `simple` keywords, but indicates that the keyword and extended character are of a type that is generally ignored or treated as a space character.

The character codes may be numeric hexadecimal, octal or decimal. They must be in the range 128_{10} to 255_{10} inclusive or the value -1. Hexadecimal numbers start with `0x` and may use upper or lower case letters. Octal numbers start with `0`.

To illustrate the distinction between `directive` and `startdirective` keywords consider the following definitions where the start of Z schema character '┌' is of type `directive` and the start of formal text character 'Ⓢ' is of type `startdirective`. Then, the source file lines '┌`abcd`' and '┌ `abcd`' both denote the Z schema `abcd`, i.e., the category name in both cases is '┌' and the argument is `abcd`. By contrast, in the following three cases: (1) the line 'Ⓢ `abcd`' starts the 'Ⓢ' category with argument `abcd`; (2) the line 'Ⓢ`abcd`' starts a category of that name and has no arguments; (3) the line 'Ⓢ`ab cd`' starts the 'Ⓢ`ab`' category with argument `cd`.

[1] The reason is that the extended category name has the extended character as its first character. Source file lines starting with keywords are, effectively, modified so that the extended character replaces the keyword.

To avoid keywords being interpreted by SCCS their closing '%' character may be replaced with a double quote '"' character — but only within the keyword file. This will only be necessary for keywords comprising a single uppercase letter.

Example of the keyword file. These lines are taken from the default keyword-file.

```
# keywordfile  @(#) 92/01/15 1.1 sievekeyword
%=>%           simple 183
%>"            simple 174
%BH%           white  0xfc
%BT%           verbalone       247
%EFT%                 directive        176
%EX%           index  221
%Pi%           simple 144
%Q"            simple 81
%SFT%                 startdirective   185
%SX%           index  219
%calA%                simple -1        \MMM{\cal A}
%fn%           simple 204
%lambda%       sameas %fn%
```

6.1.9 View File Format

The view file contains the specifications of the categories and the processing they require.

The standard view file, `sieveview`, may be used as an example. It may be found, assuming the document processing system is installed, by using the `findfile` program as follows.

```
findfile sieveview $PATH
```

View file entries for the categories and their processing are lines of the following forms.

```
Form 1:    <category> <view> [?<var>[=<val>]] <filter>
Form 2:    <category> <view> [?<var>[=<val>]]
Form 3:    <category> <view> [?<var>[=<val>]]
                   <redirection> <action>
                   <redirection> <action>
                   ...
```

These all indicate that lines of the category `<category>` are to be processed in the indicated manner when sieving to get view `<view>`. The category names always start as the first character on a line. The sieving process is to read lines of text following a directive line upto, but not including the next directive line and to process them as indicated. The output texts from each of these processes are concatenated to form the output of the sieving process.

In any of the three forms, the optional field `?<var>[=<val>]` comprises a question mark followed by an environment variable name, optionally followed by an equals sign and a possible value for the environment variable. If `?<var>` is supplied (without a value), lines in the given category will be ignored in this view unless the indicated environment variable is set. If `?<var>=<val>` is supplied,

lines in the given category will be ignored in this view unless the indicated environment variable is set and has the specified value.

In form 1 source file lines are to be piped through the command `<filter>`. Form 2 is equivalent to form 1 with a `<filter>` of `cat`. Form 3 allows a number of fixed actions selected from a range of options, each action line starts with one or more spaces or tabs. It is intended primarily for the cases where a form 1 filter would be '`echo ... ; filter_program ; echo ...`' or similar. In general form 3 is preferred over form 1 for performance reasons. The `<redirection>` is optional, it is discussed in section 6.1.11 below. Redirection allows the output of an action to be written to a file rather than to the standard output of the sieving process.

Two form 1 filters are detected specially and processed internally rather than by Unix pipes. The filter `cat` is used to copy text unchanged and the filter `ignore` may be used to discard text. These cases are provided to allow a shorter and neater view-file.

Category names are of two forms depending on their first character. Those whose first character has been specified as a '`directive`' character in the keyword file, see section 6.1.8 below, are known as 'extended categories' because they are invoked without any special prefix. All others are known as 'equals categories' because they are invoked by source file lines starting with an '`=`' character. (Note that the source file may introduce an extended category using either the extended character or the corresponding keyword on the directive line.)

The actions allowed with the form 3 categories are divided into three classes, as follows.

Informative These specify additional information for the category entry. They must be specified before any others for the category.

'`arguments <number> <number>`' The two decimal numbers give the minimum and maximum number of arguments that may be given on a directive line in the source file. These arguments are made available as replacement texts via view-file macros as described in section 6.1.10 below.

One of the numbers may be omitted in which case the minimum and maximum are the same. Both numbers must be in the range zero to nine inclusive. If not specified then both numbers are assumed to be zero.

A source file directive line with arguments outside the range allowed will cause a warning message, that directive line will then be considered to be of an unwanted category so the following text will be discarded. The action entry '`arguments 2 4`' means that between two and four inclusive arguments are allowed.

'`argoptions <options...>`' Indicates any additional processing required for the directive line arguments when they are used as replacement texts for view-file macros, see section 6.1.10 below. The options are indicated by (space separated) arguments to this action line.

'`delindex`' Extended characters and percent keywords for indexing are deleted.

'`latex`' See discussion under the `cat` copying action, below.

Copying These cause text to be read from the source file, processed in some manner, then written to the standard output. The processing may be null (i.e., just copy the text), it may delete all the text so nothing remains to be written, it may be an internal or an external filtering operation. There must be exactly one of these actions per category.

'`cat <options...>`' Copy subsequent lines. With no options this action is equivalent to a `<filter>` action of `cat`, but faster.

Additional processing to form text compatible with the document style `hol1.sty` (see section 6.6) and for other purposes is indicated by the options which are indicated by (space separated) arguments.

'char' Extended characters are converted to their LaTeX equivalent. This option is implied by option `verbatim`.

'convkw' Percent keywords are recognised are converted to their corresponding extended character. Only meaningful when option `verbatim` is not set.

'convext' Extended characters are converted to their keyword form. This option is not allowed with options `mlchar`, `latex` or `verbatim`.

'delindex' Extended characters for indexing are deleted. Additionally, with option `kw` percent keywords for indexing are deleted.

'kw' Percent keywords are to be recognised. With option `verbatim` all keywords will be converted to their LaTeX equivalent. With option `char` the keyword `%%` which stands for a single `%` character is just copied, but all other keywords are converted to their LaTeX equivalent.

'kwflag' Convert unknown keywords to a call on the LaTeX macro `\UnknownKeyword`. Convert malformed keywords to a call on the LaTeX macro `\MallFormedKeyword`. Only meaningful when options `verbatim` and `kw` are set.

'kwwarn' Issue a warning message when unknown keywords are found. Only meaningful when option `kw` is set.

'latex' Text is converted for the LaTeX verbatim-like fashion provided by the LaTeX style file `hol1.sty`. In this style characters with special meaning to LaTeX are converted so that they are printed in a verbatim-like fashion. Extended characters are converted to their LaTeX equivalent. This option is not allowed with options `convext` or `mlchar`.

'mlchar' Extended characters are converted to their Standard ML string form, a backslash followed by three decimal digits. This option is not allowed with options `convext`, `latex` or `verbatim`.

'verbalone' Modifies option `verbatim` so that lines containing at least one character of type `verbalone` and whose other characters are all of type `white` do not have their line-ends converted. This corresponds to the idea that some keywords have meaning outside of the line of text. When option `kw` is set this option accepts percent keywords of types `verbalone` and `white` on the line. Note that white space characters (e.g., space and tab) are not of type `white`.

'verbatim' As option `latex` but additionally line-ends are indicated for `hol1.sty`. This option is not allowed with options `convext` or `mlchar`.

'filter <command>' Pipe subsequent lines through the filter given by `<command>`.

The pipes are set up using `popen(3S)`. Thus the filter is interpreted by the Bourne shell `sh(1)`.

For performance reasons the other copying actions should be used in preference to this action whenever possible

'mlstring' Filter subsequent lines converting text to a Standard ML string, but without the enclosing string quotes. Convert non-printing characters to their decimal form, e.g., the byte value decimal `234` is written as the four characters `\234`. Characters ' `"` ' and ' \ ' are prefixed with a ' \ '. Across the whole input (not just each use of this action) convert even numbered ' ` ' characters (as counted when read) into ' ' '. All other chars are left unchanged.

'ignore' The text is discarded.

Echoing These perform some action that does not require text to be read from the source file, typically these are similar to the Unix `echo` program. There may be any number of these.

'echonl <text>' The characters of `<text>` are copied to the output, followed by a newline.

'echo <text>' The characters of `<text>` are copied to the output.

'nl' A newline is output.

6.1.10 View-File Macros

View file copying and echoing actions allow a limited form of macro expansion. Macros calls are indicated by a dollar sign '$' followed by a single character. Dollar signs in macro bodies are not treated as macro calls. The macros available are as follows.

'$$' A single dollar sign is obtained by using two dollar signs.

'$digit' The ten decimal digits indicate the arguments from the most recently read directive line. The macro calls '$1', '$2' and '$9' denote, respectively, the first, second and ninth arguments, '$0' denotes the directive. Invoking a macro which has not been set yields an empty string. Requesting a macro outside the range allowed by the `<arguments>` specification of the category is an error.

'$*' All the arguments of the directive line, separated by single space characters.

'$&' The whole directive line.

A dollar sign followed by anything else is erroneous, all such cases are reserved for future expansion.

6.1.11 Output Redirection

The output of most of the echoing and copying actions may be redirected to a named file, the default is the standard output of the program. Output from `filter` actions may be redirected by using an explicit redirection in the filter command itself, see the examples of categories '=SLOWDUMP' and '=SLOWDUMPMORE' in section 6.1.13 below. Redirection of `ignore` is meaningless as it gives no output.

Two forms of redirection are provided, for writing (or creating) a new file and for appending to a file (creating it if necessary). To write to a file use the redirection '`write` filename' and to append use '`append` filename'. The filename may be indicated by view-file macros. It may not contain white space and it may not be empty, otherwise there is very little validation of the name. A faulty filename or failing to open the file is considered a serious error: sieving will stop.

For examples see '=DUMP' and '=DUMPMORE' in section 6.1.13 below.

6.1.12 Directive Lines

A line from standard input is taken to be a directive line in the following cases.

- When its first character is an equals character '=' and its second character is not a white space character. This is an equals category.
- When its first character is a directive character (of type either `directive` or `startdirective`, see section 6.1.8). This is an extended category.
- When its first characters are the percent keyword for a directive character (of type either `directive` or `startdirective`, see section 6.1.8). This is an extended category.

A directive line is split into its category and arguments as follows.

1. The line is processed to convert all extended characters and keywords of type `white` to spaces. Other keywords are converted to their extended character where possible. Note that directive keywords must have extended characters — see 6.1.8. Any unknown or mal-formed keywords will provoke warning messages.
2. If the first character is of type `directive` then a space character is inserted after it.
3. The resulting line is split into words which are separated by one or more space characters.
4. The first word is then the category.
5. The next eight words are the first eight arguments, the remaining words form the ninth argument.

When a directive line is found is category is determined, as above. If this category is not specified in the view file then a warning message is issued and the whole segment of following the directive line is ignored.

Options specified with `argoptions` for the category in the current view are applied to the arguments after they are extracted by the rules above.

6.1.13 Examples

One might hold a program containing Fortran and C together with its documentation using the following view file. (Note, extra spacing has been added at the left hand side in all of these examples of the view-file and the standard input text. This is needed to prevent the source text of these examples being interpreted as directive lines.)

View file example

```
        =FORTRAN nroff echo .nf ; cat; echo .fi
        =C nroff
                echonl .nf
                cat
                echonl .fi
        =C cc
        =FORTRAN f77
        =TEXT nroff
        =IGNORE ignore ignore
```

Here the code segments are included surrounded by `.nf` and `.fi` when the file is viewed to make a document and the relevant languages only are presented when the file is viewed for compilation. Note that the '`=FORTRAN`' category in the `nroff` view uses a `filter` action which would be more efficiently written in the style of the `C` category in the `nroff` view.

In the above example the input for sieve might look as follows.

Standard input

```
An ignored line.
=TEXT
This program ...

This Fortran routine ...
=FORTRAN
        SUBROUTINE ...

=TEXT
This C routine ...

=C
        main() ...

=IGNORE
End of file.
```

Assuming the view-file is file `myviewfile`, that no keyword file is wanted and the source text is in file `mydoc.doc` then the following sieve commands would be used to obtain the various views of the file.

Shell commands

```
sieve -k /dev/null -f myviewfile nroff < mydoc.doc > mydoc.nroff
sieve -k /dev/null -f myviewfile cc < mydoc.doc > mydoc.c
sieve -k /dev/null -f myviewfile f77 < mydoc.doc > mydoc.f77
```

File inclusion facilities might be implemented as extensions to the above example by adding view-file lines of the following form. With '`=INCLUDE`' the line following the directive names a file (or files); with '`=POORINCLUDE`' exactly one file is named on the directive line and a sieving error is reported if the number of names is not one. Note that with the '`=POORINCLUDE`' any text on lines after the directive line will be discarded, it will be written to the standard input of a UNIX pipe which does not read its standard input. Using this side effect of input output as shown in '`=POORINCLUDE`' to discard unwanted text is not recommended (hence the name '`=POORINCLUDE`') and it only works for small amounts of text, the reasons are given in section 6.1.14 about sieving errors.

View file example

```
=INCLUDE nroff cat `cat` | sieve nroff
=POORINCLUDE nroff
        arguments 1
        filter cat $1 | sieve nroff
```

Additional output files may be created with view-file lines of the form.

View file example

```
=DUMP nroff
        arguments 1
        echonl Write text to file $1
        echonl .nf
        cat
        echonl *** End ***
        echonl .fi
=DUMPMORE nroff
        arguments 1
        echonl Append text to file $1
        echonl .nf
        cat
        echonl *** End ***
        echonl .fi
=DUMP f77
        arguments 1
        write $1 cat
=DUMPMORE f77
        arguments 1
        append $1 cat
```

Find[2] a better example of redirecting the output of a `filter` action.

Writing to other files can be produced by redirecting the output of a `filter` action. The '`=SLOWDUMP`' and '`=SLOWDUMPMORE`' examples have the same functionality as '`=DUMP`' and '`=DUMPMORE`' but because they invoke UNIX sub processes and pipe the input through them are much slower.

View file example

```
=SLOWDUMP f77
        arguments 1
        filter cat > $1
=SLOWDUMPMORE f77
        arguments 1
        filter cat >> $1
```

6.1.14 Errors

Numerous error messages may be produced by `sieve`. The program works in three phases, (1) command line argument processing, (2) reading the steering files and (3) the sieving process. Phases will be started only if no errors have been previously been found and reported.

If `sieve` completes successfully it gives an exit status of zero, if any errors are found during its execution the exit status is one.

Most faults found whilst reading the steering files are such that the reading can continue, possibly

[2] To be done

finding more faults. Recovery after finding a steering file faults generally means discarding that line which may introduce spurious errors.

Faults found in directive lines whilst sieving cause the text following that directive line to be discarded. Steering file faults found whilst sieving cause the program to stop, note that many possible faults are detected whilst reading the steering files.

Filter actions which do not read all of the text on their standard input may cause problems. Short sections of text will be ignored, the text written to the pipe is simply lost when the filter completes. In this case `sieve` does not notice that the text has not been read and so the sieving will continue without reporting any problems. Larger sections of text will fill the pipe causing `sieve` to block, when the filter completes the UNIX signal `SIGPIPE` is raised as `sieve` tries to write to a pipe with no reader. In this case `sieve` reports the fault and then stops.

6.1.15 Limits

The values of the program's limits may be checked by using the `-l` option of `sieve`. For many of the these limits the maximum value actually used during the execution of `sieve` is also shown. The normal values are as follows.

No more than 80 categories per view-file.

No more than 20 actions per category.

Input lines from the all input files must be less than 1024 bytes long.

Merged input lines from the view and keyword files (see section 6.1.7) must be less than 1024 bytes long.

Lines after macro expansion and conversion of extended characters and percent keywords to their LaTeX equivalents must be less than approximately 2036 bytes. In the case of a line containing only extended characters which are converted to the '`\Pr`$\mathcal{NN}$`{}`' form (see section 6.1.5) this limit means a worst case where the input line must have no more than about *2036/7 ≈ 290* characters.

Lines produced by echo actions and the text of a filter command must be less than 2048 bytes.

File names for output redirection must be less than 2048 bytes.

Up to 20 keyword files may be used, one of these is always reserved for file `sievekeyword`. These files may jointly specify up to 1000 keywords. Each keyword may have a maximum length of 50 bytes.

6.2 `sieveview` — Steering File for `sieve`

The standard view-file contains entries for all the sieving categories described in this document. If additional features are required for a particular file then a copy of the view-file can be made, and then extended, in directory holding the document needing these additional features.

6.3 `sievekeyword` — Steering File for `sieve`

The standard keyword-file contains entries for all the percent keywords supported by ProofPower. If additional keywords are required for a particular file then a copy of the keyword file can be made, and then extended, in directory holding the document needing these additional features.

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

6.4 findfile

This program allows shell scripts to be written without including full path-names for steering files *etc.* It takes as command line arguments a file name followed by a list of search paths in a similar format to the shell's PATH variable, *i.e.*, a colon-separated list of directory or file names, where a file name is taken to identify the directory in which the file is contained. The C-shell's tilde convention may be used in the names in the paths. An example in a shell script might be:

Example

```
sed -f `findfile sedscript` $PATH $0`
```

Here the file sedscript is sought for first in the directories in the users PATH shell variable and then in the directory identified by the name with which this script was invoked.

If a file is found then its name is written to the standard output preceded by the name of the directory holding the file. If no file is found then the argument file name is written to the standard output.

The exit status is zero if a file is found; one if no file was found; and, two if insufficient command line arguments were given. In this last case an error message is written to standard error.

6.5 Font Files

Three font files are provided for use with literate scripts containing extended characters. They all provide the same character images but at different sizes, all include the standard ASCII characters (with codes *0* to 127_{10}) together with the extended characters (with codes 128_{10} to 255_{10}).

The extended characters can be entered from the keyboard by using the META key, although keyboard accelerators interfere with a small number of the codes.

Font holnormal.vfont is recommended for normal use in text editors and command or shell tools.

Font holpalette.vfont is recommended for use in making a palette of characters from which characters can be picked when required. Having the palette avoids the need to remember keyboard sequences for the extended characters. A suitable palette can be obtained using the palette command.

Font holdouble.vfont is recommended for use when demonstrating the system to an audience grouped around the terminal. The characters in this font are double sized versions of those in font holnormal.vfont but without the large empty boundaries of those in holpalette.vfont.

File holpalette is used by the palette command, it contains all the extended characters, grouped according to function.

6.6 Style File hol1.sty

Much of the document processing relies upon the LaTeX style file hol1.sty whose functions are described throughout this document.

INSTALLING THE PACKAGE

For most users, the standard installation procedure described in ProofPower Installation and Operation *[6]* will be adequate. This chapter describes the configuration of the document preparation facilities in more detail.

A number of installation-dependent directories are used to hold the components of the package. It is possible to hold all of the components in a single directory if support for only one architecture (Sun 3 or Sun 4) is required.

The programs of the package are located in an installation-dependent directory which should be placed on the shell search path. This directory also contains some of the supporting files. The LaTeX macro files `hol1.sty` and `TQ.sty` are contained in a second installation-dependent directory. These style files should be made available to TeX and LaTeX by setting the `TEXINPUTS` environment variable. This may be done by including a line based on the following in the C-shell startup file (`~/.cshrc`).

```
setenv TEXINPUTS .:local−macros:main−macros:
```

Where, *local-macros* is replaced with the installation-dependent directory containing `hol1.sty` and `TQ.sty`, and *main-macros* with the installation-dependent directory holding the standard TeX and LaTeX macro files.

Here, the environment variable `TEXINPUTS` contains a colon separated list of three directories which TeX and LaTeX scan from the left for any source files they want to read. Thus, files are chosen from the current directory (*i.e.*, the '.' entry) in preference to the other directories.

The font files (in `vfont(5)` format, see the Sun UNIX manuals for details) for use with Sunview are placed in a third installation-dependent directory. The recommended method of using these font files is by explicitly naming them with the '`-Wt`' option when invoking the text editor or command windows (*i.e.*, `textedit(1), shelltool(1)` or `cmdtool(1)`, which are described the Sun UNIX manuals). Using the font setting capabilities of `defaultsedit(1)` or the Sunview defaults file (normally, `$HOME/.defaults`) is not recommended. On many versions of Sunview setting a default font file means that any '`-Wt`' option is ignored, leading to *all* of the text editors and command windows using the same font.

Section 6.5 describes all the fonts available, but for initial use invoking the following commands is recommended.

```
textedit −Wt font_dir/holnormal.vfont &
palette
```

FILE FORMATS

Various files are read or written by the programs in the package. Suffices, normally of three letters, are used in file names to distinguish the various file formats involved. These are described in table 8.1. Note that the master format for a literate script is the `.doc` format; the other formats are derived from this by running programs which, in general, lose some information.

There are several other file suffices that are used by TEX and LATEX for various purposes. They are derived from the `.tex` file and are documented in the TEX and LATEX books, [1] and [2].

Suffix	Description
`.doc`	This is the usual format in which literate scripts are held under SCCS control and in which they are edited.
`.dvi`	This is the `texdvi` output file (more precisely, it is the TEX or LATEX output file) which may be viewed with a screen previewer, or printed.
`.tex`	This is a LATEX input file. It is created from a `.doc` file using the utilities `doctex`.
`.sml`	This is a Standard ML input file. It is created from a `.doc` file using the utilities `docsml`.
`.tch`	This is a theory check file. It is created from a `.doc` file using the utilities `docsml`.
`.tds`	This is a theory design file. It is created from a `.doc` file using the utilities `docsml`.

Table 8.1: Main File Suffixes

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

EXTENDED CHARACTER SET

The extended character set provides characters grouped into various categories.

9.1 Greek Letters

These may be freely used between the indexing characters.

$\mathit{\Gamma}$ $\mathit{\Delta}$ $\mathit{\Theta}$ $\mathit{\Lambda}$ $\mathit{\Xi}$ $\mathit{\Pi}$ $\mathit{\Sigma}$ $\mathit{\Upsilon}$ $\mathit{\Phi}$ $\mathit{\Psi}$ $\mathit{\Omega}$
α β γ δ ϵ ζ η θ ι κ λ μ
ν ξ π ρ σ τ υ ϕ χ ψ ω

9.2 Logic, Equivalence and Related Symbols

These may be freely used between the indexing characters.

$\wedge$ $\vee$ $\neg$ $\forall$ $\exists$ $\bullet$ $\times$ $\leq$ $\neq$ $\geq$ $\in$ $\notin$ $\Leftrightarrow$ $\Rightarrow$

9.3 Set Symbols

These may be freely used between the indexing characters.

$\mathbb{B}$ $\mathbb{C}$ $\mathbb{F}$ $\mathbb{N}$ $\mathbb{P}$ $\mathbb{Q}$ $\mathbb{R}$ $\mathbb{S}$ $\mathbb{U}$ $\mathbb{Z}$ $\subseteq$ $\varnothing$ $\subset$ $\cap$ $\bigcap$ $\cup$ $\bigcup$ $\ominus$

9.4 Arrows

These may be freely used between the indexing characters.

⟶ ⤀ ⤔ ⇸ ↔ ⇻ ⇸ ⟼ ⟼ ↠ ⤔

9.5 Formal Text Brackets

Indexing of these extended characters is not supported. Hence, unfortunately, the index to this document cannot include the viewing categories whose names include these characters. For this document categories using the ‘Ⓢ’ character are included in the index but without the ‘Ⓢ’ character itself.

⌜ ⌜: ⌜ML ⌜Z ⌝ Ⓢ ■ ┌ └ ╒ ├

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

9.6 Padding Symbols

Indexing of these extended characters is not supported. Attempting to do so tends to get an index entry with just a page number!

| — ═

9.7 Index Brackets

Indexing of the indexing brackets themselves is not supported.

⌊ ⌋

These characters must be used in pairs, they must not be nested, they should be on the same line, they should be in the correct order. The intention is to provide indexing for identifiers of the sort used in programming and other formal languages, it is not intended that this be used as a general indexing facility.

9.8 Bracketing Symbols

These may be freely used between the indexing characters.

⟨ ⟩ ⟦ ⟧ ⦇ ⦈

9.9 Subscription and Superscription

These characters may be used as part of the identifiers in indexed entries, but they will cause the appropriate subscription or superscription, their images will not be shown in the index.

⋎ ⋏ ↗ ↘ ↕

These subscription and superscription characters must be used carefully or LaTeX may give error messages, probably relating to unmatched grouping symbols (i.e., the `{` and `}` characters). The subscripted or superscripted text must not contain linebreaks. The LaTeX style file defines[1] these characters as follows.

Character:	⋎	⋏	↗	↘	↕
LaTeX:	`_`	`^`	`^{`	`_{`	`}`

9.10 Underlining

These characters may be used as part of the identifiers in indexed entries, but they will cause underlining, their images will not be shown in the index.

[1] The actual definitions are slightly different, they may be found in the style file `hol1.sty`.

$\underline{(}$ $\underline{)}$

These underlining brackets enclose a block of text that will be underlined. The brackets must balance. Underlining brackets may be nested to achieve multiple underlining. The underlined text may extend over multiple lines.

(To use the underlining brackets in LaTeX2.0.9 compatibility mode, you will need to supply the style option `ifthen` to the `\documentstyle` command at the beginning of the document.)

9.11 Relation, Sequence and Bag Symbols

These may be freely used between the indexing characters.

⩥ ▷ ⩤ ◁ ⁀ ⁀/ ↾ ↿ ⊕ ⊎ ⨾ ∘

9.12 Miscellaneous

These may be freely used between the indexing characters.

⊢ $\stackrel{\oplus}{\oplus}$ ≘ ⊥ ⊖

9.13 Extended Character Images

The table below shows all of the available extended characters. Each character is shown with its hexadecimal value using conventional hexadecimal digits, its octal value and its hexadecimal value using the '$\mathcal{NN}$' letters used in its '`\Pr`$\mathcal{NN}$`{}`' form (see section 6.1.5). User documents should not normally need to give the numerical forms of the characters. The '`\Pr`$\mathcal{NN}$`{}`' form may sometimes be seen in LaTeX error messages. Octal value are used in `xpp` applications defaults files (see ProofPower `Xpp` *User Guide* [9]).

Value	Char	Value	Char	Value	Char	Value	Char
80 200 IA	⊆	A0 240 KA	⊂	C0 300 MA	∪	E0 340 OA	↣
81 201 IB	⩥	A1 241 KB	∩	C1 301 MB	α	E1 341 OB	⩤
82 202 IC	⊎	A2 242 KC	⟩	C2 302 MC	β	E2 342 OC	⊥
83 203 ID	𝕌	A3 243 KD	⊖	C3 303 MD	⊑	E3 343 OD	⇐
84 204 IE	Δ	A4 244 KE	⇔	C4 304 ME	δ	E4 344 OE	⊃
85 205 IF	∘	A5 245 KF	⋂	C5 305 MF	ϵ	E5 345 OF	⊇
86 206 IG	Φ	A6 246 KG	≙	C6 306 MG	ϕ	E6 346 OG	𝔽
87 207 IH	Γ	A7 247 KH	⟨	C7 307 MH	γ	E7 347 OH	↗
88 210 II	└	A8 250 KI	⦇	C8 310 MI	η	E8 350 OI	↘
89 211 IJ	ϒ	A9 251 KJ	⦈	C9 311 MJ	ι	E9 351 OJ	≡
8A 212 IK	Θ	AA 252 KK	↔	CA 312 MK	θ	EA 352 OK	↕
8B 213 IL	⁀/	AB 253 KL	⊕	CB 313 ML	κ	EB 353 OL	⌒
8C 214 IM	Λ	AC 254 KM	⌜	CC 314 MM	λ	EC 354 OM	⎤
8D 215 IN	∈	AD 255 KN	→	CD 315 MN	μ	ED 355 ON	↦
8E 216 IO	∉	AE 256 KO	⌝	CE 316 MO	ν	EE 356 OO	ℕ
8F 217 IP	⤖	AF 257 KP	ℝ	CF 317 MP	⤀	EF 357 OP	↠
90 220 JA	Π	B0 260 LA	■	D0 320 NA	π	F0 360 PA	ℙ
91 221 JB	⌜ML	B1 261 LB	∧	D1 321 NB	χ	F1 361 PB	⌜z
92 222 JC	▷	B2 262 LC	∨	D2 322 NC	ρ	F2 362 PC	◁
93 223 JD	Σ	B3 263 LD	¬	D3 323 ND	σ	F3 363 PD	ℚ
94 224 JE	⌜:	B4 264 LE	⇒	D4 324 NE	τ	F4 364 PE	⊢
95 225 JF	Υ	B5 265 LF	∀	D5 325 NF	υ	F5 365 PF	(
96 226 JG	𝔹	B6 266 LG	∃	D6 326 NG	ℂ	F6 366 PG	)
97 227 JH	Ω	B7 267 LH	•	D7 327 NH	ω	F7 367 PH	├
98 230 JI	Ξ	B8 270 LI	×	D8 330 NI	ξ	F8 370 PI	–spare–
99 231 JJ	Ψ	B9 271 LJ	Ⓢ	D9 331 NJ	ψ	F9 371 PJ	↾
9A 232 JK	∅	BA 272 LK	⊕⊕	DA 332 NK	ζ	FA 372 PK	ℤ
9B 233 JL	⅄	BB 273 LL	⨾	DB 333 NL	⦗	FB 373 PL	⟦
9C 234 JM	═	BC 274 LM	≤	DC 334 NM	|	FC 374 PM	—
9D 235 JN	⊨	BD 275 LN	≠	DD 335 NN	⦘	FD 375 PN	⟧
9E 236 JO	⤔	BE 276 LO	≥	DE 336 NO	⋃	FE 376 PO	⤔
9F 237 JP	⇻	BF 277 LP	𝕊	DF 337 NP	⇸	FF 377 PP	⌜

9.14 ASCII Keywords for Special Symbols

ASCII keywords may be used instead of the characters in the extended character set. Use the programs `conv_ascii` and `conv_extended` described in ProofPower *Reference Manual* [8] to convert a document from extended to ASCII format and vice versa. The following table shows the ASCII keyword corresponding to each of the extended characters.

Keyword	Char	Keyword	Char	Keyword	Char	Keyword	Char
%and%	∧	%BT%	├	%int%	ℤ	%Theta%	Θ
%or%	∨	%EFT%	■	%emptyset%	∅	%Lambda%	Λ
%not%	¬	%SZS%	┌	%subset%	⊆	%Pi%	Π
%forall%	∀	%EZ%	└	%psubset%	⊂	%Sigma%	Σ
%exists%	∃	%SZG%	╒	%supset%	⊇	%Upsilon%	Υ
%=>%	•	%BV%	│	%psupset%	⊃	%Omega%	Ω
%x%	×	%BH%	─	%intersect%	∩	%Xi%	Ξ
%leq%	≤	%BHH%	═	%dintersect%	⋂	%Psi%	Ψ
%neq%	≠	%SX%	[×	%union%	∪	%alpha%	α
%geq%	≥	%EX%	×]	%symdiff%	⊖	%beta%	β
%mem%	∈	%lseq%	⟨	%dunion%	⋃	%delta%	δ
%notmem%	∉	%rseq%	⟩	%rsub%	⩥	%select%	ϵ
%equiv%	⇔	%lbag%	⟦	%rres%	▷	%phi%	φ
%implies%	⇒	%rbag%	⟧	%dsub%	⩤	%gamma%	γ
%Leftarrow%	⇐	%lrelimg%	⦇	%dres%	◁	%eta%	η
%fun%	→	%rrelimg%	⦈	%cat%	⁀	%iota%	ι
%bij%	⤖	%down%	⋎	%dcat%	⁀/	%theta%	θ
%finj%	⤕	%up%	⋏	%filter%	↾	%kappa%	κ
%ffun%	⇻	%uptext%	↗	%extract%	↿	%fn%	λ
%rel%	↔	%dntext%	↘	%overwrite%	⊕	%mu%	μ
%psurj%	⤀	%cantext%	↕	%bagunion%	⊎	%nu%	ν
%pfun%	⇸	%ulbegin%	(	%thm%	⊢	%pi%	π
%inj%	↣	%ulend%	)	%fcomp%	∘	%chi%	χ
%map%	↦	%boolean%	𝔹	%rcomp%	⨾	%rho%	ρ
%surj%	↠	%complex%	ℂ	%bigcolon%	⊕⊕	%sigma%	σ
%pinj%	⤔	%fset%	𝔽	%def%	≙	%tau%	τ
%<%	⌜	%nat%	ℕ	%bottom%	⊥	%upsilon%	υ
%<:%	⌜:	%pset%	ℙ	%identical%	≡	%omega%	ω
%SML%	ML⌜	%rat%	ℚ	%sqsubseteq%	⊑	%xi%	ξ
%SZT%	Z⌜	%real%	ℝ	%Delta%	Δ	%psi%	ψ
%>%	⌝	%symbol%	𝕊	%Phi%	Φ	%zeta%	ζ
%SFT%	Ⓢ	%u%	𝕌	%Gamma%	Γ		

In addition to the keywords for the extended characters, there are keywords for the mathematical symbols from tables 3.4, 3.5, 3.6 and 3.7 of the LaTeX User's Guide and Reference Manual, [2], and for the calligraphic and blackboard bold letters, $\mathcal{A} \ldots \mathcal{Z}$, $\mathbb{A} \ldots \mathbb{Z}$. The keywords for the =mathematical symbols are given by the LaTeX name without the leading backslash, e.g., the `\bowtie` symbol, "⋈", has keyword `%bowtie%`. The keywords for the calligraphic and blackboard bold letters are `%calA%` ... `%calZ%` and `%bbA%` ... `%bbZ%`.

9.15 Using Xpp to work with the Extended Character Set

The xpp program is designed to help you work with documents containing symbols in the extended character set. Its palette tool provides a graphical interface for selecting symbols. It can also be configured to let you enter symbols using the keyboard. See ProofPower Xpp *User Guide* [9] for more information about xpp and how to work with and customise the keyboard layout.

REFERENCES

[1] Donald E. Knuth. *The TEXbook.* Addison-Wesley, 1984.

[2] Leslie Lamport. *A Document Preparation System LATEX, user's guide & reference manual.* Addison-Wesley, 1986.

[3] J.M. Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall, 1989.

[4] DS/FMU/IED/USR004. *ProofPower Tutorial Manual.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[5] DS/FMU/IED/USR005. *ProofPower Description Manual.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[6] DS/FMU/IED/USR007. *ProofPower Installation and Operation.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[7] DS/FMU/IED/USR011. *ProofPower Z Tutorial.* Lemma 1 Ltd., `http://www.lemma-one.com`.

[8] LEMMA1/HOL/USR029. *ProofPower HOL Reference Manual.* Lemma 1 Ltd., `rda@lemma-one.com`.

[9] LEMMA1/XPP/USR031. *ProofPower Xpp User Guide.* Lemma 1 Ltd., `rda@lemma-one.com`.

[10] LEMMA1/ZED/USR030. *ProofPower Z Reference Manual.* Lemma 1 Ltd., `rda@lemma-one.com`.

INDEX

= COMMENTS ... 23
= CSH ... 23
= DESCRIBE ... 23
= DOC ... 23
= DUMP ... 23
= DUMPMORE ... 23
= ENDDOC ... 23
= EXAMPLE ... 23
= FAILURE ... 23
= FAILUREC ... 23
= FRULE ... 27
= GFT ... 25
= GFT ... 33
= GFTSHOW ... 25
= GFTSHOW ... 33
= GFTXQ ... 25
= GFTXQ ... 33
= IGN ... 16
= IGNORE ... 16
= INCLUDE ... 23
= INLINEFT ... 11
= INLINEFT ... 16
= KEYWORDS ... 23
= ML ... 41
= SEEALSO ... 23
= SH ... 23
= SML ... 10
= SML ... 15
= SMLLABELLED ... 24
= SMLLITERAL ... 25
= SMLPLAIN ... 24
= SYNOPSIS ... 23
= TEMP ... 26
= TEST ... 26
= TEX ... 9
= TEX ... 15
= THDOC ... 25
= THSML ... 25
= USES ... 23
= VDUMP ... 23
= VDUMPMORE ... 23
argoptions ... 52
arguments ... 52
BirthdayBook1 ... 19
BirthdayBook1 ... 35
BirthdayBook1 ... 37
cat ... 52
char ... 53
ci_five ... 40
ci_one ... 40
ci_two ... 40
convext ... 53
convkw ... 53
DATABASE ... 20
DATE ... 19
delindex ... 52
delindex ... 53
directive ... 50
docdvi ... 45
docpr ... 44
docsml ... 43
doctch ... 44
doctds ... 44
doctex ... 43
echo ... 54
echonl ... 54
factorial ... 18
fibonacci ... 18
filter ... 53
findfile ... 59
first ... 20
hol1.sty ... 9
hol1.sty ... 59
HOLCONST ... 21
HOLCONST ... 41
holdouble.vfont ... 59
HOLLABPROD ... 22
HOLLABPROD ... 41
holnormal.vfont ... 59
holpalette ... 59
holpalette.vfont ... 59
ignore ... 53
index ... 50
Index − 1 ... 12
Index − 2 ... 12
InterestingThing ... 11
IZ ... 41
IZAX ... 21
IZAX ... 41
kw ... 53
kwflag ... 53
kwwarn ... 53
LAB_PROD_EXAMPLE ... 22
latex ... 52
latex ... 53
limit ... 19
mlchar ... 53
mlstring ... 53

© Lemma 1 Ltd. 2006 PPTex-2.9.1w2.rda.110727 - DOCUMENT PREPARATIONUSR001

NAME ... 19
nl ... 54
RAddBirthday ... 20
sameas ... 50
sieve ... 47
sievekeyword ... 58
sieveview ... 58
simple ... 50
startdirective ... 50
texdvi ... 10
texdvi ... 44
TEXINPUTS ... 61
TQ ... 9
TREE ... 20
verbalone ... 50
verbalone ... 53
verbatim ... 53
white ... 50
Z ... 19
Z ... 20
Z ... 21
Z ... 41
Z − paragraphs ... 19
Z − paragraphs ... 20
Z − paragraphs ... 21
Z − paragraphs ... 41
ZAX ... 19
ZAX ... 21
ZAX ... 41
$∗ ... 54
$0 − $9 ... 54
$$ ... 54
$& ... 54
\+ ... 29
\description ... 35
\doctex ... 10
\documentstyle ... 9
\enumerate ... 35
\FruleLeftJustify ... 37
\FruleLeftWidth ... 37
\FruleRightWidth ... 37
\ftlinepenalty ... 36
\ftlmargin ... 34
\ftlmargin ... 35
\ftrmargin ... 34
\ftspaceskip ... 36
\HOLConstLabel ... 41
\HOLindexBold ... 39
\HOLindexEntry ... 40
\HOLindexOff ... 40
\HOLindexOn ... 40
\HOLindexPlain ... 39
\HOLLabProdLabel ... 41
\index ... 11
\indexentry ... 11
\itemize ... 35
\leftmargini − leftmarginvi ... 35
\list ... 35
\makeindex ... 11
\MallFormedKeyword ... 53
\MlLabel ... 41
\NoMoaning ... 41
\PrNN ... 41
\PrNN ... 48
\PrNN ... 50
\PrNN ... 58
\PrNN ... 67
\printindex ... 11
\ShowAllImages ... 39
\ShowBars ... 39
\ShowBoxes ... 39
\ShowIndexing ... 39
\ShowScripts ... 39
\strut ... 40
\tabstop ... 36
\underscoreoff ... 39
\underscoreon ... 39
\UnknownKeyword ... 53
\vertbarfalse ... 40
\vertbartrue ... 40
\ZAxDesInformalLabel ... 41
\ZAxDesLabel ... 41
\ZGenericInformalLabel ... 41
\ZGenericLabel ... 41
\ZOtherInformalLabel ... 41
\ZOtherLabel ... 41
\ZSchemaInformalLabel ... 41
\ZSchemaLabel ... 41
\\ ... 29
$\wedge$_*tac* ... 31
$\Rightarrow$_*elim* ... 31