Authors' Guide
to writing
HOL Documentation

Introduction

This note describes the organization of the HOL documentation and the conventions used in it. It also explains how to write entries for the reference manual and how to write documentation for libraries. All authors of reference manual entries and library documentation should read this document carefully before starting to write. General users of the system may find the beginning sections of this note useful when they wish to typeset the HOL documentation at their site.

1 Structure of the HOL documentation

The HOL system documentation consists of three volumes:

- *TUTORIAL*, which contains a brief tutorial introduction to the HOL system. This is intended to be the first item read by new users of HOL and provides a self-study introduction to the structure and use of the system.

- *DESCRIPTION*, which is intended to serve both as a systematic definition and description of the HOL system, and as an advanced guide for users with some prior experience of the system.

- *REFERENCE*, which provides full reference documentation in the form of individual descriptions for all the predefined ML variable bindings in the HOL system.

In addition, there is a separate volume which is not, strictly speaking, part of the HOL manual set, but which is also described in this note:

- *LIBRARIES*. This consists of stand-alone documentation for each of the libraries distributed in the HOL system library.

The documentation in this *LIBRARIES* volume is written by the contributors of the coresponding HOL system libraries.

2 Structure of the documentation directories

Most of the LaTeX sources for HOL system documentation reside in the subdirectory `Manual` of the main HOL distribution directory. Certain source files for the *REFERENCE* volume reside in the directory `help`. The reason for this and for the organization of these files is described later in this document. For similar reasons, also explained later, much of the *LIBRARIES* volume resides in the `Library` directory.

The `Manual` directory contains the following subdirectories:

- `Description`: LaTeX sources for the *DESCRIPTION* volume of the documentation.
- `Tutorial`: LaTeX sources for the *TUTORIAL* volume of the documentation.
- `Reference`: LaTeX sources for the *REFERENCE* volume of the documentation.
- `Libraries`: `Makefile` for the *LIBRARIES* volume of the documentation.
- `Covers`: LaTeX and PostScript sources for cover pages for all volumes of the manual.
- `LaTeX`: LaTeX style and macro files for the HOL documentation.
- `Reference/bin`: various sed and shell scripts needed to build the reference manual.
- `Guide`: the LaTeX source for the present document.

Each of these subdirectories contains a `READ-ME` file explaining its contents. Authors working on parts of the HOL documentation in these directories are advised to first read the appropriate `READ-ME` files.

3 Typesetting the HOL documentation

Each volume of the HOL documentation is typeset using LaTeX. More specifically, the documentation is designed to be typeset with LaTeX version 2.09 of 7 Dec 1989, based on TeX, the C version 3.0. Building the documentation also requires the '`makeindex`' program (portable version 2.4). The script `Manual/LaTeX/makeindex` will run the appropriate binary from `Manual/LaTeX/makeindex.bin` if it is present. If not, then an appropriate warning will result: in this case the user should build a binary from the sources in `Manual/LaTeX/makeindex.src` (instructions are included) and place it in the directory `Manual/LaTeX/makeindex.bin`. The TeX psfig macros are required for the fancy cover pages. See Section 3.7 below for further information on this.

The following sections describe how to obtain typeset versions of each of the four volumes of the HOL system documentation.

3.1 Choice of paper size

The HOL documentation is designed to be printed either on ISO A4 sized paper (297mm high by 210mm wide) or on American 8.5 by 11 inch paper. The pagination and text size is designed to be identical in both formats.

Before typesetting any part of the documentation, the choice of paper size must be set by editing the style file `Manual/LaTeX/layout.sty`. Near the top of this file there is the declaration of a LaTeX boolean flag '`Afour`' followed by a command to set the flag. One may select the paper size by editing this command. For A4 size paper, the appropriate command is '`\Afourtrue`' (the distribution default). For American paper, the command should be '`\Afourfalse`'.

3.2 Makefile for typesetting the HOL documentation

The `Makefile` in the `Manual` directory allows one to typeset all four volumes of the HOL documentation with a single command. When working in the `Manual` directory, one can type '`make all`' to create typeset versions of the documentation from scratch. Note that the desired paper size should be set before doing this (see above). After LaTeX has typeset the documentation, the following `dvi` files will have been created:

- `Tutorial/tutorial.dvi`: the typeset *TUTORIAL* volume.
- `Description/description.dvi`: the typeset *DESCRIPTION* volume.
- `Reference/reference.dvi`: the typeset *REFERENCE* volume.
- `Libraries/libraries.dvi`: a typeset titlepage and preface for the *LIBRARIES* volume.

Additionally, the files `Covers/endpages.ps` and `Covers/titlepages.ps` should also be present. These are the PostScript versions of the cover pages for the manual volumes. All these documents are then ready to be printed on a laser-printer. Note that some of them are rather long, and therefore not lightly to be printed.

3.3 Typesetting the tutorial volume

The `Makefile` in the `Tutorial` directory is used to typeset the *TUTORIAL* volume of the HOL documentation. Working in the `Tutorial` directory, the `make` commands that are implemented are the following:

1. Type '`make clean`' to remove all traces of previous executions of LaTeX in this directory. This removes all files with suffixes `.dvi`, `.aux`, `.toc` and `.log`.
2. Type '`make tutorial`' to typeset the *TUTORIAL* volume using LaTeX.

3. Type '`make all`' to typeset the *TUTORIAL* volume completely from scratch, running it through LaTeX twice to ensure consistency of the references.

Note that, as usual, `latex` must be run at least twice to guarantee that various forms of reference (i.e. citataions, page references, entries in the table of contents and so on) are consistent. The command '`make tutorial`' (number (2) above) runs the tutorial volume through LaTeX only once. One may therefore have to type `make tutorial` more than once to guarantee a correct result (twice should be sufficient). Typing '`make all`' is guaranteed to typeset the *TUTORIAL* volume correctly and completely from scratch.

3.4 Typesetting the description volume

The `Makefile` in the `Description` directory is used to typeset the *DESCRIPTION* volume of the HOL documentation. Working in the `Description` directory, the `make` commands that are implemented are the following:

1. Type '`make clean`' to remove all traces of previous executions of LaTeX in the `Description` directory. This removes all files with suffixes `.dvi`, `.aux`, `.toc`, `.log`, `.idx` and `.ilg`.

2. Type '`make description`' to typeset the *DESCRIPTION* volume using LaTeX.

3. Type '`make index`' to create the *DESCRIPTION* index. This creates the LaTeX source file `index.tex` using the program `makeindex`.

4. Type '`make all`' to typeset the *DESCRIPTION* volume completely from scratch, running it through LaTeX twice to ensure consistency of the references and the index.

As usual, `latex` must be run at least twice to guarantee that various forms of reference are consistent. The command '`make description`' runs the *DESCRIPTION* through LaTeX only once and does not create the index file. One may therefore have to type `make description` then `make index` and then `make description` again in order to guarantee a correct result (running *DESCRIPTION* through LaTeX twice should be sufficient). Typing '`make all`' is guaranteed to typeset the *DESCRIPTION* volume correctly (including index) and completely from scratch.

3.5 Typesetting the reference manual

The `Makefile` in the `Reference` directory contains entries to typeset the entire *REFERENCE* manual from scratch, as well as entries for the individual steps required in this process. For a full description of the `make` facilities for *REFERENCE*, see section 6.1 below; only the `make` facility for building the entire manual is described here. Working in the `Reference` directory, the `make` command for typesetting *REFERENCE* is:

- Type '`make all`' to create the LaTeX sources for the *REFERENCE* volume and typeset it completely from scratch, running it through LaTeX twice to ensure consistency of the references and the index.

Typing '`make all`' is guaranteed to typeset *REFERENCE* correctly, with all index and page entries resolved, and to rebuild it completely from scratch.

3.6 Typesetting the library documentation

Library documentation for a particular library may be typeset by entering the `Manual` subdirectory *of the relevant library directory* and typing `make all`. The procedure is essentially the same as that for typesetting one of the main HOL manuals.

To make all the library manuals, go into the directory `Manual/Libraries` and type `make all`. This will build manuals for all libraries which have them. Currently, many libraries do not have proper manuals. It is intended that all new libraries should eventually have a proper manual. One or two older libraries (such as `unwind`) have had a manual written for them.

More details on `writing` library documentation are given in a later section. Again, the intention is that the format should correspond closely to that for the main documentation.

3.7 Typesetting the cover pages

The `Makefile` in the `Covers` directory is used to typeset the fancy cover pages for all the volumes in the HOL documentation. Working in the `Covers` directory, the `make` variables used are:

1. LATEX: the LaTeX command.

2. DVI2PS: name of the DVI to PostScript converter.

3. DVI2PSOPTS: options given to DVI2PS.

and the commands that are implemented are the following:

1. Type '`make clean`' to remove all traces of previous executions of LaTeX in the `Covers` directory. This removes all files with suffixes `.dvi`, `.aux`, `.toc`, `.log`, `.idx` and `.ilg`.

2. Type '`make all`' to typeset the fancy cover pages using LaTeXand DVI2PS.

`latex` needs to be run only once, and DVI2PS must be able to understand the TeX psfig macros. We supply a version of the macros, and information on obtaining conversion programs in the directory `Covers/psfig`.

4 Structure of the reference manual

The manual is organized into three chapters. Chapter 1 provides manual entries for all the pre-defined ML identifiers in the HOL system, except the identifiers that are bound to pre-proved theorems. These include: general-purpose functions, such as ML functions list processing, arithmetic, input/output, and interface configuration; functions for processing the types and terms of the HOL logic, for setting up theories, and for using the subgoal package; primitive and derived forward inference rules; and tactics and tacticals. The arrangement of this chapter is strictly alphabetical.

Chapter 2 lists all pre-proved theorems built into the system, together with the ML identifiers to which they are bound. These are grouped into sections, based on the 'subject matter' of the theorems. The sections describing groups of theorems more or less mirror the HOL theories that contain these theorems.

Chapter 3 is the index, which is automatically generated.

5 Structure of the reference manual directory

The reference manual for the system is distributed along with the HOL sources in the directories `hol/Manual/Reference` and `hol/help`. Each predefined ML identifier $\langle id \rangle$ should have, somewhere within the directory `hol/help`, a corresponding file $\langle id \rangle$`.doc` that documents it. This file is used to generate the LaTeX source for the manual entry on the identifier $\langle id \rangle$. A specification of the format of $\langle id \rangle$`.doc` files and an explanation of how they are converted into LaTeX sources are given in the sections that follow.

Note that a few characters cannot conveniently be used in filenames, so the filename does not follow the $\langle id \rangle$`.doc` scheme. This will certainly not affect alphanumeric identifiers, and library documenters will not generally have much to worry about.

Note that entries do not exist for ML keywords such as `let` and `or`. These are not strictly identifiers. However there is an entry for `do`, because although it is an ML keyword, it is also an identifier.

The source files for the manual are in the following subdirectories of the directory `hol/help`:

`ENTRIES`	—	all non-theorem identifiers (chapter 1)
`THEOREMS`	—	pre-proved theorems (chapter 2)

The directory `ENTRIES`, simply contains a `.doc` file for each relevant ML identifier in the system. The directory `THEOREMS` contains further subdirectories. These correspond to a classification of built-in theorems by subject matter:

`arith`	—	theorems about arithmetic
`axioms`	—	axioms of the HOL logic
`basic-logic`	—	definitions basic logical constants
`combin`	—	theorems about combinators
`functions`	—	basic theorems about functions
`list`	—	theorems about lists
`logic`	—	logical tautologies
`one`	—	theorems about the type `one`
`pairs`	—	theorems about pairs
`sum`	—	theorems about disjoint sums
`tree`	—	theorems about trees
`tydefs`	—	theorems used for type definitions

The manual entries for built-in theorems are in pretty much their final form, and these subdirectories contain a `.doc` file for every built-in theorem.

The directory `Reference/bin` contains various shell and `sed` scripts that are used to generate the LaTeX source for *REFERENCE* from the `.doc` files. These programs are explained in more detail below.

6 Generation of the LaTeX source

Each ⟨*id*⟩`.doc` file is in a special format (described below), from which the LaTeX source for a manual entry on the ML identifier ⟨*id*⟩ can be generated. The file `bin/doc-to-tex.sed` is the `sed` script used generate LaTeX source text from `.doc` files. Executing:

- `sed -f bin/doc-to-tex.sed` ⟨*id*⟩`.doc > `⟨*file*⟩

will create a file called ⟨*file*⟩, containing a translation into LaTeX source text of the documentation on the identifier ⟨*id*⟩ in the file ⟨*id*⟩`.doc`. The shell script `bin/doc-to-tex` finds, sorts and translates all `.doc` files in a given directory into LaTeX sources. This shell script is used to generate chapter 1 of the *REFERENCE* manual.

6.1 Typesetting the entire manual

The `Makefile` in the `Reference` directory can be used to create the LaTeX source for the entire reference manual. The command sequence for doing this, and then typesetting the reference manual, is as follows:

1. Type '`make tex`' to create the LaTeX source for the finished manual entries. This translates all `.doc` files into LaTeX and builds a source text for the entire manual from the results.

2. Type '`make reference`' to run the file `reference.tex` through LaTeX to typeset the manual.

3. Type '`make index`' to create the LaTeX source for the index.

Note that steps (1) and (2) must be done in that order. Note further that `latex` must be run again after step (3) to guarantee that the index is consistent with the body of the text. Step (3) may, of course, be omitted until a final typeset version, complete with index, is required for printing.

The *REFERENCE* makefile also supports the command `make all`, which is described above in section 3.5.

6.2 Typesetting individual manual entries

To facilitate the debugging of individual manual entries (i.e. individual `.doc` files), the shell script `bin/mktex` is provided to generate a stand-alone LaTeX source file from individual `.doc` files. Executing:

- `mktex` ⟨*file* 1⟩ ⟨*file* 2⟩ ··· ⟨*file n*⟩ `>` ⟨*file*⟩`.tex`

Will translate the `.doc` files:

⟨*file* 1⟩`.doc` ⟨*file* 2⟩`.doc` ··· ⟨*file n*⟩`.doc`

into a stand-alone LaTeX source file called ⟨*file*⟩`.tex`, which can then be run through LaTeX in order to inspect the typeset manual entries generated from these files.

Note that the shell script `mktex` assumes it is being run in the `Reference` directory, and generates a LaTeX source which makes reference (via relative pathnames) to the directories `../LaTeX`. To run LaTeX on the resulting `.tex` file in any *other* directory, one must therefore edit the either the pathnames in this file or the pathnames in the script `mktex` itself. Furthermore, `mktex` makes use of the `sed` script `bin/doc-to-tex.sed` (see above), which therefore must be present in a subdirectory `bin` of the directory in which `mktex` is run.

Because of these pathname dependencies, it is intended that individual authors working on the reference manual entries will create their own versions of `mktex` in their own file space. The `mktex` file in `Reference/bin` should not itself be edited.

7 The format of `.doc` files

Documentation for individual ML bindings is written in a very rigid form. This is to enable both LaTeX source for the manual and a plain file for online documentation to be easily generated from a single source. The following sections describe this standard format for `.doc` files and explain the conventions adopted for the form and content of reference manual entries in general.

In fact, there are really two distinct formats for `.doc` files:

- the format used to document pre-proved *theorems*, and
- the format used to document other ML bindings (i.e. general-purpose ML functions, logic functions, inference rules, tactics, and so on).

These are explained below in sections 7.1 and 7.2 respectively. The format of `.doc` files for theorems is the simpler (and more rigid) of the two, so it is explained first.

7.1 The format of `.doc` files for theorems

The format of `.doc` files for theorems is simple. For each theorem, there is a corresponding file called ⟨*thmname*⟩`.doc`, where ⟨*thmname*⟩ is either the ML identifier to which the theorem is bound (or will be bound by autoloading), or the name under which the theorem is stored in a theory file. The format of this `.doc` file is as shown below:

```
\THEOREM␣⟨thmname⟩␣⟨thyname⟩
⋮
\ENDTHEOREM
```

where

- the two occurrences of ␣ are *required* single spaces,
- ⟨*thmname*⟩ is the name of the theorem (as discussed above),
- ⟨*thyname*⟩ is the name of the theory in which the theorem is stored,
- and the lines between `\THEOREM` and `\ENDTHEOREM` are the ascii representation of the theorem in question, as printed by the standard HOL theorem pretty-printer.

No indentation should be added to the text of the theorem as printed by HOL. The theorem itself should be no more than 75 characters wide on any line (use `set_margin` if necessary). Theorems should almost always be printed without type annotations (i.e. with the `show_types` flag off). An exception is made for theorems whose content is likely to be obscure unless type information is shown. An example is the built-in theorem:

```
one_axiom   |- !(f:* -> one) (g:* -> one). f = g
```

The meaning of this theorem is clear only when it is known that `f` and `g` are functions that yield values of type `one`. For theorems of this kind, the documentation should contain the result of pretty-printing the theorem with the ML `show_types` flag turned on.

There are certain theorems which are bound to ML identifiers when the system is built, but which are not stored in any theory file. For these theorems, the `.doc` format is:

```
\THEOREM␣⟨thmname⟩␣\none
⋮
\ENDTHEOREM
```

That is, the string '`\none`' takes the place of the theory name. This format should also be used in library documentation for theorems which are bound to ML identifiers when a library is loaded, but which are not stored in the library in a theory file (this situation is expected to be rare).

7.1.1 Examples

Shown below is the `.doc` file for the built-in theorem `PRIM_REC_THM`:

```
\THEOREM PRIM_REC_THM prim_rec
|- !x f.
    (PRIM_REC x f 0 = x) /\
    (!m. PRIM_REC x f(SUC m) = f(PRIM_REC x f m)m)
\ENDTHEOREM
```

The ⟨*thmname*⟩ field in this file is '`PRIM_REC_THM`', and the ⟨*thyname*⟩ field is the theory name '`prim_rec`'. When typeset in LaTeX, the reference manual entry generated from this `.doc` file looks like this:

```
PRIM_REC_THM  (prim_rec)
  |- !x f.
      (PRIM_REC x f 0 = x) /\
      (!m. PRIM_REC x f(SUC m) = f(PRIM_REC x f m)m)
```

Here is an example showing the `.doc` file for a built-in theorem in which the keyword '`\none`' is used in the ⟨*thyname*⟩ field:

```
\THEOREM NOT_CLAUSES \none
|- (!t. ~~t = t) /\ (~T = F) /\ (~F = T)
\ENDTHEOREM
```

When typeset, this entry looks like:

```
NOT_CLAUSES  (none)
  |- (!t. ~~t = t) /\ (~T = F) /\ (~F = T)
```

The theory name '*none*' indicates that the theorem is not stored in any built-in theory.

7.1.2 Automatically-generated `.doc` files for theorems

To make it easy to create documentation for theorems, two programs are provided which will generate `.doc` files for every axiom, definition, and theorem in any given theory. The main program is in the file `bin/create-doc-files.ml`. After loading this file into HOL, evaluating:

`create_doc_files` ⟨*thy*⟩`;;`

will create a `.doc` file in the standard format for every axiom, definition, and theorem stored in the theory ⟨*thy*⟩`.th`. Each file will have a name of the form ⟨*id*⟩`.doc`, where ⟨*id*⟩ is the name of one of the axioms, definitions, or theorems in the theory ⟨*thy*⟩`.th`. The `.doc` files are created in the directory in which the HOL session was started.

The program `create_doc_files` makes calls to LISP, and at present it works only with the Franz Lisp version of HOL. It does not work in the Common Lisp version. However, there is an alternative: the shellscript `bin/create-doc-files2` provides a similar function. Running

`create-doc-files2` ⟨*thy*⟩

from the shell will also create in the current directory all the `.doc` files for the theory ⟨*thy*⟩`.th`. This alternative has the merit of working in any version of HOL under Unix, regardless of the underlying Lisp, but is somewhat less flexible because it simply tries to load the given theory as an ancestor in a fresh HOL session. Unless the relevant theory file is on the search path by default (typically, this will include the basic theories and any theory files in the current directory), it may be necessary to set up a `hol-init.ml` file or edit the shellscript to get the desired behaviour.

7.2 The format of `.doc` files for other identifiers

In this section, the format and content of `.doc` files for ML identifiers other than those bound to theorems is explained. These identifiers are mostly the names of built-in ML functions. For each identifier ⟨*name*⟩, there will (eventually) be a corresponding file called ⟨*name*⟩`.doc`[1].

The general format of a `.doc` file is shown in Figure 1. The file has several fields, which must appear in the order shown below (though some of the fields are optional). The fields are identified by the keywords:

- `\DOC`
- `\TYPE` (or `\BLTYPE` and `\ELTYPE`)
- `\SYNOPSIS`

[1] Actually for some non-alphanumeric identifiers like `/`, it is necessary to call the file something else.

- `\KEYWORDS`
- `\LIBRARY` (present only in library documentation)
- `\DESCRIBE`
- `\FAILURE`
- `\EXAMPLE`
- `\USES`
- `\COMMENTS`
- `\SEEALSO`
- `\ENDDOC`

The format and content of each of these fields is explained in detail in the sections that follow.

7.2.1 The `\DOC` line (*required*)

The first line in each `.doc` file must have the form

`\DOC␣`⟨*name*⟩

where

- ␣ is a *required* single space, and
- ⟨*name*⟩ is the ML identifier being documented.

If the ML identifier in the ⟨*name*⟩ field stands for an infix ML function, it should *not* be preceded by the dollar sign '`$`'. For example, the ⟨*name*⟩ field for the tactical `THEN` should simply be '`THEN`' and not '`$THEN`'. The `\DOC` line is a required field in every `.doc` file. There must be exactly one blank line between it and the `\TYPE` line.

An example `\DOC` line is shown below:

```
\DOC ACCEPT_TAC
```

7.2.2 The `\TYPE` line (*required*)

The third line in each `.doc` file must have the form

`\TYPE␣{`⟨*name*⟩`␣:␣`⟨*type*⟩`}`

where

- the three occurrences of ␣ are *required* single spaces,
- ⟨*name*⟩ is the ML identifier being documented, and
- ⟨*type*⟩ is the type of the identifier being documented, usually precisely as printed by the system with ML type abbreviations in force (see below).

If the ML identifier in the ⟨*name*⟩ field stands for an infix ML function, it must be preceded by the dollar sign '`$`'. For example, the ⟨*name*⟩ field for the tactical `THEN` should be '`$THEN`', not '`THEN`'. The `\TYPE` line is a required field in every `.doc` file. There must be exactly one blank line between it and the line beginning '`\SYNOPSIS`'.

Occasionally the type may be too long to fit on one line. In such circumstances the name and type should be placed on one or more lines, sandwiched between `\BLTYPE` and `\ELTYPE` ('begin-long-type' and 'end-long-type'), as follows:

`\BLTYPE`
⟨*name*⟩`␣:␣`⟨$type_1$⟩
⟨$type_2$⟩
⋮
⟨$type_n$⟩
`\ELTYPE`

where ⟨$type_1$⟩, ..., ⟨$type_n$⟩ represent the type broken onto n lines. The type may begin on the second line if that is more appropriate; that is the ⟨*name*⟩ field and the colon can appear on their own in the first line. When necessary, `\BLTYPE` and `\ELTYPE` should be used in place of `\TYPE`, and all other conventions associated with `\TYPE` should be adhered to when using them.

An example `\TYPE` line is shown below:

```
\TYPE {ACCEPT_TAC : thm_tactic}
```

The type quoted in the `\TYPE` should almost always be precisely the type shown by the system. An exception is made for types which if abbreviated by the built-in type abbreviations (e.g. `conv`) may be misleading as to the nature of the ML function in question. For example, the type of `AP_THM` is shown by the system as `thm -> conv`, but in this case it is clearer to write `(thm -> term -> thm)` since `AP_THM` is not really a conversion-generating function.

If the \TYPE field differs from the type printed by the system, then the fact should be noted in the \COMMENTS field. Unless there is a very good reason, changing the types quoted by the system should be done in the following circumstances only:

- Appearance of goal when a term list # term has no real correspondence with a goal, or anything to do with backwards proof. A good example is list_mk_forall.
- Appearance of conv when a term -> thm is not most naturally thought of as a conversion. ASSUME at least fits uncontroversially into this category.

Most \TYPE fields should agree precisely with the system (even up to bracketing and spacing). The shell script bin/typecheck may therefore be used to check exact correspondence with system types automatically. It is not as robust as it could be, but at least it pinpoints mistakes and deliberate typing changes. The script should be given as an argument the doc-files to be typechecked, for example '*.doc'. The following example shows the form of the output, which is from diff(1).

```
/homes/jrh$ cd doc
/homes/jrh/doc$ typecheck *.doc
2c2
< ALPHA : (term -> term -> thm)
---
> ALPHA : (term -> conv)
5,6c5,6
< AP_THM : (thm -> term -> thm)
< ASSUME : (term -> thm)
---
> AP_THM : (thm -> conv)
> ASSUME : conv
```

7.2.3 The \SYNOPSIS field (*required*)

The \SYNOPSIS field follows the blank line after the previous field, and must be present in every .doc file. It consists of a line containing only the keyword '\SYNOPSIS', followed by a *brief* description of the ML identifier being documented. The text of this description should start on the line immediately following the line beginning '\SYNOPSIS'. There must be exactly one blank line between the text of the synopsis (which is indicated by three dots in Figure 1) and the keyword of the field that follows.

The synopsis should consist of at most one or two short sentences (or phrases) which provide a concise but accurate indication of the purpose or function of the ML identifier being documented. The aim of the synopsis is not to give a full specification or description, but merely to allow the reader to decide at a glance whether this item may provide the functionality that he or she is looking for. If at all possible, this should fit on one line on an 80-column display. Unless clearly inappropriate, the synopsis should use the the third person singular present tense, for example "Solves any goal of the form ...", "Introduces an assumption ..." or "Performs the action corresponding to ...". Consistency in this regard makes reading a lot of synopsis lines displayed by a help tool less tiresome for the user.

An example `\SYNOPSIS` field for the tactic `ACCEPT_TAC` is shown below:

```
\SYNOPSIS
Solves a goal if supplied with the desired theorem (up to alpha-conversion).
```

7.2.4 The `\KEYWORDS` field (*optional*)

This field is optional. It is intended simply to provide a list of relevant words which can be used by (existing or future) help tools to find an entry when you cannot remember the name, but do know the kind of thing you are looking for. The words should be written on the line(s) below, separated by commas and terminated by a period.

Here is an example:

```
\KEYWORDS
assumptions, rewriting.
```

7.2.5 The `\LIBRARY` field (*optional*)

This section is present *only* in library documentation, and should be followed by a single space and the name of the library. For example:

```
\LIBRARY reduce
```

7.2.6 The `\DESCRIBE` field (*optional*)

The `\DESCRIBE` field consists of a line containing only the keyword '`\DESCRIBE`', followed by a full description of the ML identifier being documented. The `\DESCRIBE` field is almost always a required field; it is omitted only when the item being documented is so simple that a complete description is already given by the `\SYNOPSIS` field. The text of the description should start on the line immediately following the line beginning '`\DESCRIBE`'. There must be exactly one blank line between the text of the description (which is indicated by three dots in Figure 1) and the keyword of the field that follows.

The aim of the `\DESCRIBE` field is to give a complete and accurate specification of what the ML function[2] being documented does. The description should not make reference to the ML source code, and it should not depend on specific examples in which the function

[2]... or other ML data object, but the vast majority of manual entries are for functions.

is used (these belong under \EXAMPLES). Neither should the description discuss specific situations in which one might use the function in question (this belongs under \USES). Comments of a general nature (e.g. 'this function really should be reimplemented to do so-and-so') belong under \COMMENTS. Failure may be mentioned when it plays a role in the function's behaviour when applied to arguments for which it is intended (e.g. when failure is used for backtracking, as in REPEAT and ORELSE). But a description of failure due to incorrect use of the function (e.g. applying dest_comb to a variable) does not belong here; this information should instead be given in the \FAILURE field.

As an example, here is the \DESCRIBE field for ACCEPT_TAC:

```
\DESCRIBE
{ACCEPT_TAC} maps a given theorem {th} to a tactic that solves any goal whose
conclusion is alpha-convertible to the conclusion of {th}.
```

The use of the braces '{' and '}' in this example is explained below in Section 7.3.

7.2.7 The \FAILURE field (*required*)

The \FAILURE field consists of a line containing only the keyword '\FAILURE', followed by a full (English language) specification of the conditions under which evaluation can fail when the ML identifier being documented is used. The \FAILURE field is required, even when the function being described cannot fail. The text of the failure specification should start on the line immediately following the line beginning '\FAILURE'. There must be exactly one blank line between the text of this specification (which is indicated by three dots in Figure 1) and the keyword of the field that follows.

The \FAILURE field should describe the conditions under which both failure and looping can occur. For some functions, it may be difficult to give both necessary and sufficient conditions for failure. For example, the first argument to the function INDUCT_THEN should be a theorem of the form returned by the built-in function prove_induction_thm. If the supplied theorem does *not* have this form, INDUCT_THEN will almost certainly fail. It is, however, a feature of the way INDUCT_THEN is implemented that this function may succeed, and even do the intended thing, when supplied with theorems whose form is not precisely that of a theorem proved by prove_induction_thm (users should not depend on this). The \FAILURE field for INDUCT_THEN should make this clear, perhaps by stating that this function succeeds when applied to theorems of the correct form, but 'may' fail on other theorems. When the function being documented cannot fail, the formula 'Never fails.' should be used.

Here is an example \FAILURE field for ACCEPT_TAC:

```
\FAILURE
{ACCEPT_TAC th (A,g)} fails if the term {g} is not alpha-convertible to the
conclusion of the supplied theorem {th}.
```

The use of the braces '{' and '}' in this example is explained below in Section 7.3.

7.2.8 The \EXAMPLE field (*optional*)

The \EXAMPLE field consists of a line containing only the keyword '\EXAMPLE', followed by an illustrative example or examples of the use of the ML identifier being documented. The text of the example should start on the line immediately following the line beginning '\EXAMPLE'. There must be exactly one blank line between the text of this specification (which is indicated by three dots in Figure 1) and the keyword of the field that follows.

The \EXAMPLE field is optional, and should be present only when an example will genuinely add to the reader's understanding. In most cases, examples should be accompanied by a running commentary, with the actual ML code displayed (see section 7.3) when necessary. Examples should be constructed such that the reader can run them in a standard virgin HOL session.

Examples should avoid using the subgoals package if possible (an exception is documenting the package itself). It is still preferable to give complete ML sessions, but if that is impractical, then give a slightly higher-level description of what is going on. An example is given below:

```
\EXAMPLE
The goal:
{
   ?- ?x. x=T
}
\noindent can be solved by:
{
   EXISTS_TAC "T" THEN REFL_TAC
}
```

Please remember the `noindent` directive on intermediate non-verbatim lines, if there are any, and that all verbatim displays should be indented by 3 spaces.

7.2.9 The \USES field (*optional*)

The \USES field consists of a line containing only the keyword '\USES', followed by a statement of common or typical uses of the ML identifier being documented. The text of the example should start on the line immediately following the line beginning '\USES'. There must be exactly one blank line between the text of this specification (which is indicated by three dots in Figure 1) and the keyword of the field that follows.

The \USES field is optional, and is likely to be present only for complex functions such as certain tactics and inference rules. In many cases, the \SYNOPSIS will already have said everything that needs to be said about the uses of a function. A \USES field will not be needed in these cases. The \USES field should not simply state what the function in question does, but should describe general contexts in which it might be used. For example, suppose that the `\SYNOPSIS` for `SPEC` is:

```
\SYNOPSIS
Specializes the leading universally-quantified variable of a theorem to a
supplied value.
```

In this case, the `\USES` field should *not* be something like:

```
\USES
Used for specializing universally-quantified theorems.
```

since this adds no information at all. If anything, the `\USES` field for `SPEC` should be something like:

```
\USES
Obtaining specific instances of already proved theorems; implementing
efficient derived inference rules by appeal to pre-proved general theorems.
```

But, in the case of `SPEC`, no `\USES` field at all is probably more appropriate (in the above, the first 'use' is still a paraphrase of the function of `SPEC`; the second 'use' is rather specific, and using `SPEC` is only a very small part of this kind of activity).

7.2.10 The `\COMMENTS` field (*optional*)

The `\COMMENTS` field consists of a line containing only the keyword '`\COMMENTS`', followed by any general remarks about the ML identifier being documented that need to be made but do not belong elsewhere. The text of the example should start on the line immediately following the line beginning '`\COMMENTS`'. There must be exactly one blank line between the text of this specification (which is indicated by three dots in Figure 1) and the keyword of the field that follows.

The `\COMMENTS` field is optional, and should be used for general remarks about implementation matters, promises of future reimplementation, style (e.g. 'don't use it' in the entry on `POP_ASSUM`), and so on.

Validity checking should not be mentioned in the comments section. The composition of validity checking steps simply follows the composition of the justification functions created by tactics, so need not be mentioned separately.

7.2.11 The \SEEALSO field (*optional*)

The \SEEALSO field consists of a line containing only the keyword '\SEEALSO', followed by list of manual entries relevant to the ML identifier being documented. The list of related entries should start on the line immediately following the line beginning '\EXAMPLE'. The list itself should consist of a sequence of ML identifiers in alphabetical order, separated by commas and (always) terminated by a full stop. There must be exactly one blank line between this list (which is indicated by three dots in Figure 1) and the keyword of the field that follows.

When deciding whether to add a cross-reference to the SEEALSO field, follow the rule: *if in doubt, add it.* Note that cross-references must always be ordered alphabetically.

Here is a sample \SEEALSO field for REWRITE_TAC:

```
\SEEALSO
FILTER_ASM_REWRITE_TAC, FILTER_ONCE_ASM_REWRITE_TAC, ONCE_ASM_REWRITE_TAC,
ONCE_REWRITE_TAC, PURE_ASM_REWRITE_TAC, PURE_ONCE_ASM_REWRITE_TAC,
PURE_ONCE_REWRITE_TAC, PURE_REWRITE_TAC, REWRITE_TAC, SUBST_TAC.
```

7.2.12 The \ENDDOC line (*required*)

The \ENDDOC line, which consists of a single line containing only the keyword '\ENDDOC', signals the end of a manual entry. It must be present in every .doc file.

7.3 Using typesetting commands

The \SYNOPSIS, \DESCRIBE, \FAILURE, \EXAMPLE, \USES, and \COMMENTS fields of a .doc file all consist of a keyword line followed by general text. Within this text, only two kinds of typesetting command are allowed: '\noindent␣' at the beginning of a line, and {⟨*text*⟩}. These are interpreted as follows:

1. '\noindent␣' at the beginning of a line prevents indentation of that line.
2. An inline {⟨*text*⟩} results in ⟨*text*⟩ being typeset in small verbatim font. In particular, '{⟨*text*⟩}' is converted to '{\small\verb%⟨*text*⟩%}'. This means that ⟨*text*⟩ should not contain the character '%'.
3. A sequence of lines of the following form, where '{' and '}' occur at the beginning of otherwise empty lines:

```
{
␣␣␣⋮
}
```

gets typeset to displayed verbatim text.

When using displayed verbatim text, the lines between '{' and '}' should be manually indented by three spaces. Furthermore, there should be no blank line between the preceding line of text and the open bracket '{' and no blank line between the closing bracket '}' and the following line of text (even if otherwise required by virtue of being the last line in a field). Here is an example of the correct use of the displayed verbatim format:

```
from this, we can see that the tactic:
{
   FIRST_ASSUM (\th g. ACCEPT_TAC (SYM th) g ? NO_TAC g)
}
will search the assumption list for an equation whose symmetric form will
solve the current goal.  This kind of tactic is very useful when dealing with
```

Here, there are three spaces manually inserted at the beginning of the displayed line in order to give the finished product the appropriate appearance.

The following exception may made to the general rule of indenting displayed text by three spaces: when an example is presented without accompanying commentary, the 'displayed' text is not indented. For example, in:

```
\EXAMPLE
{
#length [];;
0 : int

#length [1;2;7;1;1];;
5 : int
}
```

no indentation is used, since the 'displayed' text is not surrounded by a commentary of ordinary text.

7.4 General conventions

In addition to the specific conventions laid down in the preceding sections for the text of the various fields in a reference manual entry, the following general conventions should also be observed.

7.4.1 Spelling and terminology.

The '-ize' spelling for words like 'generalize', 'specialize' is preferred. Use 'analyze' rather than 'analyse'. The following preferred spellings and hyphenations should also be noted: left-hand side, right-hand side, premise (not premiss), and theorem-tactic.

The preferred terms for the parts of theorems, goals, implications and rules are as follows:

- An implicative term has an *antecedent* and a *consequent*
- A theorem or goal has *assumptions* and a *conclusion*
- A rule has *premises* and *conclusions*

It is permissible to use *hypotheses* instead of *assumptions* if preferred: the two should be regarded as strictly synonymous. The overloading of the word *conclusion* is unfortunate, but no viable alternative has been proposed.

7.4.2 Substitution

Substitution of term `u` for a variable `x` in term `t` should be shown as a transition from `t` to `t[u/x]`, rather than from `t[x]` to `t[u]`. Although perhaps slightly less intuitive, it is both more formal and more flexible. Multiple sequential substitutions should be shown as `t[u1/t1]...[un/tn]`, and multiple parallel substitutions as `t[u1,...,un/t1,...,tn]`. The use of `t[u1/t1,...,un/tn]` should be avoided because of the scope for ambiguity.

7.4.3 Goals

Goals with assumptions should be represented by using `?-` by exact analogy with `|-` for theorems. Thus the goal corresponding to the theorem:

```
x = 0 |- SUC x = 1
```

would be:

```
x = 0 ?- SUC x = 1
```

Assumption lists of goals should be shown, even when they do not play a major role. Braces are now free to be used for set-notation without confusion. To get braces into a verbatim section, just double them up: {{`A`}} etc.

7.4.4 Alpha-equivalence

Alpha-convertibility of terms should not, except in obvious special cases like `ALPHA`, be shown by using different variable names. It is usually sufficient to assume by default that we are dealing with alpha equivalence classes of terms. If important, these issues can be mentioned in the description or comments field.

The union (of assumption lists) should be written as an infix lowercase `u`, for example '`A1 u A2`', whether or not alpha-equivalent terms are deemed to be the same when taking the union. If this is an important consideration, it can be discussed separately. The addition or subtraction of an assumption should be written as `A u {t}` and `A - {t}` respectively, rather than `A` becoming `A,t` or vice versa.

7.4.5 Typesetting inference rules and tactics

The following formats should be followed for rule-style specifications of inference rules and tactics:

```
   <premises>                          <goal>
 --------------  <name & args>     ============  <name & args>
  <conclusion>                      <subgoals>
```

More precisely, displayed inference rules and tactics in this 'rule-notation' should be typeset as follows:

- Everything in verbatim mode should be manually indented by 3 spaces.
- Use a line consisting of minus-signs for inference rules, and a line consisting of equal-signs for tactics.
- The line should overlap by exactly one space on either side of the wider of the lines of text above and below.
- The other line of text should be centred within the rule bar as closely as possible.
- Above the line put the theorem(s) or goal prior to application of the rule or tactic (may be empty in the case of theorems).
- Below the line put the theorem(s) or subgoal(s) resulting from application of the rule or theorem. In the case of a tactic which completely solves a goal, this should be a single `-` placed centrally on the bottom line, rather than a completely empty field.
- To the right of the line, after 2 spaces, write the name of the rule or tactic.
- After the name, give any arguments (e.g. terms) that it is applied to in order to produce the illustrated effect, except for those which actually appear in the top line of the rule. Sometimes this convention does not make manifest the required order of the arguments; in this case the arguments which appear in the rule line should be restated (see the example of `AP_THM` given below). When they are written as arguments (only), terms should be put in quotation marks and theorems in parentheses, since this corresponds more closely to an ML expression.

Some examples of rule notation as described above are:

```
   A |- x = y
----------------  AP_TERM "f"
 A |- f x = f y


   A |- f = g
----------------  AP_THM (A |- f = g) "x"
 A |- f x = g x


 A ?- f x = f y
================  AP_TERM_TAC
   A ?- x = y


 A ?- ?x. t
=============  EXISTS_TAC "u"
 A ?- t[u/x]
```

7.5 Format of a `.doc` file

```
\DOC␣⟨name⟩                              ( required )

\TYPE␣{⟨name⟩␣:␣⟨type⟩}                  ( required )

\SYNOPSIS                                ( required )
⋮

\KEYWORDS                                ( optional)

\LIBRARY                                 ( optional)

\DESCRIBE                                ( optional, but usually present )
⋮

\FAILURE                                 ( required )
⋮

\EXAMPLE                                 ( optional )
⋮

\USES                                    ( optional )
⋮

\COMMENTS                                ( optional )
⋮

\SEEALSO                                 ( optional )
⋮

\ENDDOC                                  ( required )
```

Figure 1: Format of a `.doc` file.

7.6 Example of a .doc file

Here is a sample .doc file for ACCEPT_TAC:

```
\DOC ACCEPT_TAC

\TYPE {ACCEPT_TAC : thm_tactic}

\SYNOPSIS
Solves a goal if supplied with the desired theorem (up to alpha-conversion).

\KEYWORDS
tactic.

\DESCRIBE
{ACCEPT_TAC} maps a given theorem {th} to a tactic that solves any goal whose
conclusion is alpha-convertible to the conclusion of {th}.

\FAILURE
{ACCEPT_TAC th (A,g)} fails if the term {g} is not alpha-convertible to the
conclusion of the supplied theorem {th}.

\EXAMPLE
{ACCEPT_TAC} applied to the axiom
{
   BOOL_CASES_AX = |- !t. (t = T) \/ (t = F)
}
\noindent will solve the goal
{
   ?- !x. (x = T) \/ (x = F)
}
\noindent but will fail on the goal
{
   ?- !x. (x = F) \/ (x = T)
}
\USES
Used for completing proofs by supplying an existing theorem, such as an axiom,
or a lemma already proved.

\SEEALSO
MATCH_ACCEPT_TAC.

\ENDDOC
```

7.7 Example of a typeset .doc file

Here is the typeset documentation for the ACCEPT_TAC .doc file shown above:

ACCEPT_TAC

ACCEPT_TAC : thm_tactic

Synopsis

Solves a goal if supplied with the desired theorem (up to alpha-conversion).

Description

ACCEPT_TAC maps a given theorem th to a tactic that solves any goal whose conclusion is alpha-convertible to the conclusion of th.

Failure

ACCEPT_TAC th (A,g) fails if the term g is not alpha-convertible to the conclusion of the supplied theorem th.

Example

ACCEPT_TAC applied to the axiom

```
BOOL_CASES_AX = |- !t. (t = T) \/ (t = F)
```

will solve the goal

```
?- !x. (x = T) \/ (x = F)
```

but will fail on the goal

```
?- !x. (x = F) \/ (x = T)
```

Uses

Used for completing proofs by supplying an existing theorem, such as an axiom, or a lemma already proved.

See also

MATCH_ACCEPT_TAC.

8 Writing library documentation

Libraries should conform to a rigid format, to allow easy interaction with the standard building, typesetting and help tools. The intention is that each library should be organized in the same way as the main HOL directory, with a special `Manual` directory for documentation and a `help` directory for reference entries.

8.1 Directory organization

The standard HOL distribution has a toplevel directory called 'Library', which contains one subdirectory for each library, whose name corresponds to the relevant library name. Within this, the following files should exist:

1. `Makefile`: this should be a standard Unix Makefile, with at least the following targets.

   - `all`: this should make object code and theory files (if any) for the entire library
   - `clean`: this should remove all object code, but not theory files, if any.
   - `clobber`: this should remove object code and theory files.

2. `READ-ME`: this should be a simple text file giving at least:

   - The name of the library
   - A brief description of what the library does (not more than a few lines)
   - The author, preferably including an email address
   - The date at which the library was written

   Other items may be included, but the `READ-ME` file should be kept brief.

3. `help` A help directory, containing one or both of the following subdirectories

   - `ENTRIES`: standard `'.doc'` files for all the ML functions provided by the library, if any
   - `THEOREMS`: standard `'.doc'` files for the theorems provided by the library, if any.

   These help files should be in the respective standard formats specified above. Note that help files for ML functions should have a `'LIBRARY'` field giving the name of the relevant library.

4. `Manual`: a subdirectory for the library documentation. This is described in more detail in the next section.

5. Other files, such as ML source, compiled object code, theory files, and any other files the library may need for some reason. Everything connected with documentation should go in the Manual directory.

8.2 The Manual directory

The '`Manual`' directory should contain the following files.

1. `Makefile`: this should be a standard Unix Makefile to build a dvi version of the library documentation. It should include at least the following targets.

   - `all`: make the dvi for the entire library documentation
   - `clean`: remove dvi files and unnecessary intermediate or auxiliary files produced by a run of LaTeX.

   The reference section for ML functions and theorems will be created automatically from the help directory. Examining a Makefile from an existing library (such as '`string`' or '`reduce`') will show the general format to make this procedure work.

2. `title.tex`: this is a standard file containing a title page for library documentation. The simplest way to make such a file is to copy one from an existing library (such as '`string`' or '`reduce`') and edit it with the new library name and author's name and address.

3. `description.tex`: this is the main documentation file written by the user. As its name suggests, it corresponds to a miniature *DESCRIPTION* document for the library, explaining in reasonable detail its purpose and usage, probably mentioning some of the ML functions and/or theorems provided, but not embarking on an exhaustive list.

4. `entries-intro.tex`: a brief paragraph to act as an introduction to the reference section of ML functions. Apart from the library name, this can again be copied from an existing library.

5. `theorems-intro.tex`: a brief paragraph to act as an introduction to the reference section of theorems. Apart from the library name, this can again be copied from an existing library.

6. `references.tex`: a standard LaTeX list of any references made in the description section.