NAME

extract-sig — generate SML
extract-info, **merge-info** — generate intermediate files
html-gen, **html-index**, **html-toc** — generate HTML
latex-gen, **proof-latex** — generate LaTeX
filter-index, **mk-mldoc-makefile** — miscellaneous ml-doc tools

SYNOPSIS

```
extract-sig [-config config-file] [-dir output-dir] mldoc-files
extract-info [-config config-file] [-dir output-dir] mldoc-files
merge-info [-config config-file] [-o output-file] info-files
html-index [-config config-file] [-dir output-dir] [-o output-file]
            [-info path] [-cols num]
            [-all | -sig | -struct | -exn | -type | -val]
html-toc [-config config-file] [-dir output-dir] [-o output-file]
            [-info path] [-depth num]
html-gen [-config config-file] [-dir output-dir] [-info path-to-infofile]
            [-wid num] mldoc-file
latex-gen [-config config-file] [-dir output-dir] [-doc] [-index]
            [-info path-to-infofile] mldoc-file
proof-latex [-config config-file] [-dir output-dir] [-doc] [-index]
            [-info path-to-infofile] mldoc-file
filter-index
mk-mldoc-makefile [-help] [-bin dir] [-html dir] [-info dir] [-latex dir]
            [-proof dir] [-root file] mldoc-files
```

DESCRIPTION

ML-Doc is a system for producing reference manuals for SML libraries. It can produce high quality documentation with extensive indexing, like the published SML Basis Library reference manual, and also HTML for online use.

ML-Doc is different to systems like javadoc that extract documentation from source files. Rather, the documentation *is* the primary artifact—in fact **extract-sig** can be used to create source files from documentation. The `mkdoc`(1) utility, however, can produce first drafts from source files. ML-Doc is most useful for documenting stable library interfaces.

ML-Doc files, marked by the extension `.mldoc`, are SGML documents conforming to the ML-Doc document type definition.

This manual has four major sections besides the present one:

OPTIONS	describes command line options and configuration files.
USING ML-DOC	describes how the toolset is used to produce documentation and includes subsections on **HTML Templates** and **Entities**.
WRITING DOCUMENTATION	describes the markup used within ∗`.mldoc` files.
SGML VS HTML/XML	discusses some SGML features absent from HTML and XML.

Generating SML

extract-sig — Extracts SML signatures from ∗`.mldoc` documentation.

Generating intermediate files

extract-info — Produces intermediate files summarising interfaces and the section hierarchy.

merge-info — Combines intermediate files into a single master info file.

Generating HTML

`html-index` —Produces HTML index files from a master info file.

`html-toc` —Transform a master info file and HTML template into a table of contents.

`html-gen` —Generate HTML output given template, master info, and mldoc input files.

Generating LaTeX

`latex-gen` —Produce LaTex for printing given master info and mldoc input files.

`proof-latex` —Produce LaTeX for proofing given master info and mldoc input files.

Miscellaneous

`filter-index` —Reads a list of index entries, one per line, from standard input, removes any duplicates, and writes the sorted entries to standard output.

`mk-mldoc-makefile` —Create a Makefile that ties all the other tools together.

OPTIONS

Many command line options can also be specified in a configuration file, using the names given in brackets.

−config *config-file*
Defaults to `Config.cfg`. Contains global and per-tool settings.

−cols *num*
(`NumColumns`) Number of columns in the index. Defaults to 3.

−debug
(`Debug`) Currently unused.

−depth *num*
(`Depth`) Sections nested more deeply than this number will not be listed in the table of contents. Defaults to 3.

−dir *output-dir*
(`OutDir`) Specifies where to write output files. The default is different for each tool:

extract-sig	`Sigs`
extract-info	`Info`
html-index	`HTML`
html-toc	`HTML`
html-gen	`HTML`
latex-gen	`Hardcopy`
proof-latex	`Proof`

−doc (`Standalone`) Currently unused.

−index
Currently unused.

−info *path*
(`MasterInfoFile`) Locates the output of **merge-info**. Defaults to `Info/LaTeX.info` for the hardcopy and proof tools, and `Info/HTML.info` for the HTML tools.

−mldoc
(`MLDocDir`) Names the subdirectory of the `*.mldoc` files. Defaults to `ML-Doc`.

−o *output-file*
(`OutFile`) Specifies where to write output. For **html-index** the default depends on the index generated:

all	`index-all.html`
sig	`index-sig.html`
struct	`index-struct.html`
exn	`index-exn.html`
type	`index-type.html`
val	`index-val.html`

Otherwise the name depends on the tool:

`merge-info`	`Info/Master.info`
`html-toc`	`toc.html`

`−all` | **`−sig`** | **`−struct`** | **`−exn`** | **`−type`** | **`−val`**

Specify which index to generate, defaulting to **`−all`**:

`−all`	unifi ed identifi er index
`−sig`	signatures
`−struct`	structures
`−exn`	exceptions
`−type`	types
`−val`	values

`−wid` *`num`*

(`PreWid`). Width of the output device. Currently unused. Defaults to 60.

There are some additional options that can only be given in a confi guration fi le:

BaseURL	Applies to the `HTML` grouping (described below). If the RelativeLinks flag is true then intra-document links will be relative, otherwise the value of BaseURL, if present, is added as an HTML `BASE` tag.
Catalog	Global option. Names the catalog fi le. Defaults to `CATALOG`.
RelativeLinks	Applies to the `HTML` grouping. See BaseURL above. Defaults to false.
Root	Applies to **`html-gen`**. Currently unused.
SGMLS	Applies to the `Tools` grouping. Path to an SGML validating parser. Defaults to `/usr/local/bin/nsgmls`.
Template	Applies to **`html-index`**, **`html-toc`**, and **`html-gen`**. Names the template fi le.
TopLevelSection	Applies to **`latex-gen`** and **`proof-latex`**. Defaults to `Chapter`. Other permitted values are `Part` and `Section`. Controls the generation of LaTeX sections.

A confi guration fi le contains name/value pairs separated within and between by white space. Single-line comments begin with a ‘`#`’. Groupings are made by giving a name followed by contained options enclosed in braces, `{ ... }`. Values are either names, decimal or hexadecimal literals, the latter prefi xed with ‘`0x`’, strings enclosed in double quotes, or logical values (`TRUE` or `FALSE`). As a brief example:

```
# Catalog.cfg
Catalog "CATALOG"

HTML {
    BaseURL          "CML"
    RelativeLinks    TRUE
    PreWid           70
}
```

`mk-mldoc-makefile` does not currently respect the confi guration fi le, instead it uses command line options, which differ somewhat from the other tools:

`−help` Display a summary of options.

`-bin` *`dir`*
: Specifies where the ML-Doc tool binaries are installed. Defaults to `/usr/local/bin`.

`-html` *`dir`*
: Output directory for HTML files. Defaults to `HTML`.

`-info` *`dir`*
: Output directory for info files. Defaults to `Info`.

`-latex` *`dir`*
: Output directory for LaTeX files. Defaults to `Hardcopy`.

`-proof` *`dir`*
: Output directory for proof LaTeX files. Defaults to `Proof`.

`-root` *`file`*
: If present, the generated Makefile will run LaTeX against the given filename, which should not have a `.tex` extension.

USING ML-DOC

Directory structure

An ML-Doc *project* comprises a set of files and directories that, at a minimum, will include:

`CATALOG`	Typically links to `Entities.sgml` in the same directory and the installed ML-Doc `CATALOG` file. See `FILES`.
`Config.cfg`	Global and per-tool configuration options. See `OPTIONS`.
`Entities.sgml`	Contains entities local to the project. These entities are used for abbrevations and referencing external documents and files. See `Entities`.
`ML-Doc/`	The source ∗`.mldoc` SGML documentation.
`index.template`	Template used by **`html-index`**.
`page.template`	Template used by **`html-gen`**.
`toc.template`	Template used by **`html-toc`**.

Running **`mk-mldoc-makefile`** adds:

`Makefile`	Orchestrates the manifold programs to produce documentation.

The other tools place their output in additional subdirectories. Empty subdirectories and any child directories therein must be created before running the tools. The default names are:

`HTML/`	Output from the **`html-gen`**, **`html-toc`**, and **`html-index`** tools.
`Hardcopy/`	Output from **`latex-gen`**.
`Info/`	Output from **`extract-info`**, further augmented by **`merge-info`**, to be used by the other tools.
`Proof/`	Output from **`proof-latex`**.
`Sigs/`	SML code created by **`extract-sig`**.

HTML Templates

Templates are HTML files containing special entities. The **`html-`**∗ tools replace entities with details from ∗`.mldoc` and ∗`.info` files. Each tool works from a distinct template named in the configuration file.

Tool name (in configuration file)	**Typical Template value**
HTML-Gen	`page.template`
HTML-Index	`index.template`
HTML-TOC	`toc.template`

The entities are:

`&body;`	placeholder for document body
`&filename;`	document filename without extension
`&title;`	document title
`&version;`	document version
`&doc.date;`	document date in "month day, year" format
`&doc.year;`	document year (4 digit format)
`&doc.day;`	document day
`&doc.month;`	document month (as a string)
`&doc.monthnum;`	document month (as a number from 1-12)
`&today.date;`	current date in "month day, year" format
`&today.year;`	current year (4 digit format)
`&today.day;`	current day
`&today.month;`	current month (as a string)
`&today.monthnum;`	current month (as a number from 1-12)
`&base.url;`	URL of the root directory of the document
`&parent.url;`	URL of the parent document
`&root.url;`	URL of the document root
`&index.url;`	URL of the document index
`&toc.url;`	URL of the table of contents

Entities

Entities are used within ML-Doc to include mathematical and other specialised symbols, to abbreviate titles and other text, to reference files, and to name certain output files. The latter three purposes are served by including a customised `Entities.sgml` file within a project.

Specialised symbols include various mathematical symbols (described under **Mathematics**) , the HTML 2.0 standard entities, e.g. `&nbsp;`, `&copy;`, and these SGML/LaTeX symbols:

```
&LT;       <             &LTE;      <=
&GT;       >             &GTE;      >=
&NEQ;      !=            &AMP;      &
&DQUOTE;   "             &BAR;      |
&DASH;     -
```

Abbreviations encourage consistency and facilitate certain types of updates. These definitions, in an `Entities.sgml` file, are a good example:

```
<!ENTITY SMLBASIS SDATA "SML Basis Library">
<!ENTITY SMLNJ    SDATA "SML/NJ">
```

References, which will be expanded in the output text, may then be made from ∗`.mldoc` files:

    This feature requires the &SMLNJ; libraries.

Documentation will sometimes need to reference the ML-Doc descriptions of other libraries; such as those of the SML Basis, SML/NJ, or Concurrent ML. The reference tags described under **References** (e.g. `SIGREF`, `AREF` and `DOCREF`) provide this facility via a `DOCUMENT` attribute whose value must be an entity listed in `Entities.sgml`, a bracketing ampersand and semicolon are not used for such values. The entity resolves to an identifying string, which is sought within the configuration file, `Config.cfg`, to find an *external document entry*. As an example, consider a constructor reference:

```
<CONREF DOCUMENT=SML-BASIS-DOC STRID="Option"/NONE/
```

The `DOCUMENT` attribute value, `SML-BASIS-DOC`, is defined in the project entity file, `Entities.sgml`:

```
<!ENTITY SML-BASIS-DOC SDATA "SML-Basis-Doc">
```

The entity value, `SML-Basis-Doc,` in turn refers to an entry in the configuration file:

```
...
SML-Basis-Doc {
    InfoFile   "/usr/local/smlnj/smlnj-lib/Doc/BasisInfo/HTML.info"
    BaseURL    "www.standardml.org/Basis"
    RootURL    "www.standardml.org/Basis/index.html"
}
...
```

This example refers to a master info file installed with SML/NJ and directs hyperlinks to the online documentation.

Entities are also used within ML-Doc to specify values for the `FILE` attribute, which specifies an input file for the (currently unsupported) `FIGURE` tag, and names an output file for the `SIGBODY` tag. **`extract-sig`** expands the given entity to name the source code it produces, e.g. given an entity declaration:

```
<!ENTITY CML-SIG SDATA "cml-sig.sml">
```

The specifications in

```
<SIGBODY SIGID="CML" FILE=CML-SIG>
```

would be extracted to a file named `cml-sig.sml`.

General Use

A simplified sequence of steps for creating ML-Doc documentation:

1. Create and populate the basic directory structure, as per **Directory Structure**.
2. Use `mkdoc`(1) to produce skeleton ∗`.mldoc` files. Add explanatory text to each.
3. Add other ∗`.mldoc` files to link together into a continuous whole, and to explain the various interfaces.
4. Create a `Makefile` with **`mk-mldoc-makefile`**, as shown under **EXAMPLES**.
5. Use the make targets to generate documentation.

`HTML`	Default. Produce HTML pages.
`Hardcopy`	Create LaTeX files for generating the final document.
`Proof`	Create LaTeX files for generating a review document.
`clean`	Run the individual `clean-html`, `clean-info`, `clean-latex`, and `clean-proof` targets to remove generated files.

6. LaTeX documentation requires further processing as described below.
7. Any changes to the library interface should be made against the ∗`.mldoc` files, **`extract-sig`** is able to extract an updated SML version.

LaTeX documents require a *root file* to specify the document class and indexes. For example, if the first file is called `intro.mldoc` then `manual.tex` might resemble:

```
\documentclass{mldoc-book}
\mldMakeTopicIndex\mldMakeIdIndex\mldMakeRaisesIndex
\newcommand{\kw}[1]{\textbf{#1}}

\begin{document}
  \input{intro}
  \mldPrintTopicIndex\mldPrintIdIndex\mldPrintRaisesIndex
\end{document}
```

The `kw` command formats identifiers. The mldoc class and style files must be in a path searched by LaTeX. Updating the `TEXINPUTS` environment variable is one way of ensuring this. E.g. under bash:

```
export TEXINPUTS=".:/usr/local/share/ml-doc/lib/LaTeX:".
```

The **EXAMPLES** section shows how to run the LaTeX tools.

WRITING DOCUMENTATION

This section essentially elaborates on the `ml-doc.dtd` file. An ML-Doc file begins with the declaration:

```
<!DOCTYPE ML-DOC SYSTEM>
```

and contains header elements followed by one or more, potentially nested, sections.

There are four types of header element. Only `TITLE` is mandatory. They may be given in any order.

`VERSION` The version of the documentation, with attributes:

`VERID`	e.g. `1.4`.
`YEAR`	Year of release.
`MONTH`	(optional) Month of release.
`DAY`	(optional) Day of release.

E.g. `<VERSION VERID="1.4" YEAR=2007 MONTH=8 DAY=12>`

`COPYRIGHT` Identify a copyright holder with attributes:

`OWNER`	The copyright owner.
`YEAR`	Year of assertion.

E.g. `<COPYRIGHT OWNER="Mega Corp" YEAR=2003>`
Multiple `COPYRIGHT` elements may be present.

`AUTHOR` Identifies the document author, with attributes:

`NAME`	Name of author.
`EMAIL`	Email address of author.
`YEAR`	Year written.
`MONTH`	(optional) Month written.
`DAY`	(optional) Day written.

E.g. `<AUTHOR NAME="J. Doe" EMAIL="doej@mega.co" YEAR=2006>`

`TITLE` The document title. E.g. `<TITLE>Superhash Structure</TITLE>`

Sections begin with a `HEAD` element, followed by zero or more paragraphs (`PP` and `FLOAT` elements), and then zero or more nested sections, included files and/or interfaces. The `SECTION` tag has three optional attributes:

`LABEL`	A string used for making cross-references (from `SECREF`).
`NONUMBER`	If present, the section will not be numbered in LaTeX output.
`NOTOC`	Excludes the section from the table of contents. Only valid if `NONUMBER` is present.

`HEAD` and `PP` elements have no attributes. For `FLOAT` elements see **GENERAL MARKUP**. E.g.

```
<SECTION>
  <HEAD>General Markup</HEAD>
  <PP>
  Various tags provide...
  <SECTION>
    <HEAD>Formatting</HEAD>
    <PP>
    Tags available for inline...
  </SECTION>
  ...
</SECTION>
```

Typically a separate `*.mldoc` file will be used for each major topic, such as introductory or background text, and interface (signature, structure or functor). A corresponding `*.html` file, for HTML output, and/or `*.tex` file, for LaTeX output, is generated for each ML-Doc source file. A complete document is constructed by including, with `INCLFILE` tags, individual files within a hierarchy of sections. Inclusions become hyperlinks in HTML output and inserted pages in LaTeX output. An `INCLFILE` has a single, mandatory `FILE` attribute specifying a relative or absolute file path. The `.mldoc` extension is implied. E.g.

```
<INCLFILE FILE="lib/hash-sig">
```

SGML comments, e.g. `<!-- check source code. -->`, are ignored.

SML objects are described between `INTERFACE` tags which have optional `LABEL` attributes for making cross-references. An `INTERFACE` element contains, in order: a `HEAD` element (as per `SECTION`), an optional `SEEALSO` element containing one or more references (see **References**), optional paragraphs of introductory text, an *object description*, and optional paragraphs of concluding text.

Object Descriptions

Object description elements may have a `STATUS` attribute, with value `REQUIRED`, `OPTIONAL`, or `PROPOSED`. There are three types of element for describing objects provided by a library:

`SIGNATURE`	With mandatory `SIGID` attribute.
`STRUCTURE`	With mandatory `STRID` attribute.
`FUNCTOR`	With mandatory `FCTID` attribute.

Example outline of an interface:

```
<INTERFACE>
  <HEAD>The <CD/Hash/ structure</HEAD>
  <SEEALSO>
    <STRREF DOCUMENT=SML-BASIS-DOC TOPID/Option/
    <STRREF DOCUMENT=SML-BASIS-DOC TOPID/Time/
  </SEEALSO>

  <PP>
  Optional text after synopsis.

  <STRUCTURE STRID="CML">
  ...
  </STRUCTURE>

  <PP>
  Optional final discussion.
</INTERFACE>
```

It is usually easier to generate object descriptions directly from SML source files using `mkdoc`(1).

The `SIGBODY` element is central to describing objects. It must be included in `SIGNATURE`s, may be included in `STRUCTURE`s and `FUNCTOR`s, and has two optional attributes:

`SIGID` Identifies the signature being described. It is used for cross-referencing. When a `SIGBODY` is given within a `SIGNATURE`, both may have identical `SIGID` values. Within a `STRUCTURE` the `SIGID` attribute is usually the ‘signature version’ of the outer `STRID`. It is a succinct way of describing both interface and implementation, e.g.

```
<STRUCTURE STRID="ControlSet">
  <SIGBODY SIGID="CONTROL_SET" FILE=CONTROL-SET>
  ...
  </SIGBODY>
</STRUCTURE>
```

`FILE` The `mkdoc`(1) utility uses the value of this attribute to name extracted SML files. `SIGID` and `FILE` are typically omitted on `SIGBODY`s used as functor arguments.

A `SIGBODY` contains one or more `SPEC` and/or `SPECBREAK` elements. The former is described under **Specifications**. The latter is used to form sub-groups of related specifications, when marked with a `NEWLINE` attribute, blank lines are placed in both extracted signatures and generated output.

A `SIGNATURE` contains a `SIGBODY` which may be followed by a series of `SIGINSTANCE` elements that list structures implementing the signature, each containing an `ID`, zero or more `WHERETYPE` elements, and optionally a `COMMENT`. A `WHERETYPE` element states which signature types are instantiated in an implementation, it consists of an optional `TYPARAM` binding, the `ID` of the type, and a `TY` type constraint. The `COMMENT`, `TYPARAM`, and `TY` elements are described below. The `SIGINSTANCE` tag has two optional attributes:

`STATUS` Having a value of `REQUIRED`, `OPTIONAL`, or `PROPOSED`.
`OPAQUE` Indicates an opaque signature binding.

This example from the SML/NJ Library:

```
<SIGNATURE SIGID="ORD_MAP">
  <SIGBODY SIGID="ORD_MAP" FILE=ORD-MAP>
  ...
  <SIGINSTANCE OPAQUE><ID>IntBinaryMap
    <WHERETYPE><ID>Key.ord_key<TY>Int.int
    <COMMENT>
    ...
  <SIGINSTANCE OPAQUE><ID>IntListMap
    <WHERETYPE><ID>Key.ord_key<TY>Int.int
    <COMMENT>
    ...
</SIGNATURE>
```

would yield a synopsis:

```
signature ORD_MAP
structure IntBinaryMap :> ORD_MAP
structure IntListMap :> ORD_MAP
```

A `STRUCTURE` contains either a full `SIGBODY`, as described above, or simply the name of (`ID`), or reference to (`SIGREF`) a signature described elsewhere. A name or reference may be followed by `WHERETYPE` type realisations. An `OPAQUE` element can be placed before the signature body, name or reference, to signify an opaque binding, e.g.

```
<STRUCTURE STRID="Store">
  <OPAQUE>
  <SIGREF/ORD_SET/
  <WHERETYPE><ID>Key.ord_key<TY>String.string
</STRUCTURE>
```

The contents of a `FUNCTOR` are identical to those of a `STRUCTURE`, except that they must begin with an argument element, which takes one of two forms:

short An argument name (`ID`) followed by the name of (`ID`), or reference (`SIGREF`) to a signature.
general A `SIGBODY` element containing specifications, notably of `SUBSTRUCT` and `TYPE` elements.

E.g. `functor ListSetFn (ORD_KEY): ORD_SET` could be written:

```
<FUNCTOR FCTID="ListSetFn">
  <ID/K/<ID>ORD_KEY</ID>   <!--argument-->
  <ID>ORD_SET              <!--result-->
</FUNCTOR>
```

The second `ID` could have been written as a `SIGREF`, the third could have been given in full using `SIGBODY`.

Specifi cations

A `SIGBODY` comprises a list of `SPEC` elements, each containing either a single substructure specifi cation, or multiple specifi cations of the same kind (though types and eqtypes may be mixed), and optionally followed by a comment. The specifi cation element types are:

`INCLUDE` An `ID` or `SIGREF` naming an included signature, optionally followed by `WHERETYPE` realisations. E.g.

```
<SPEC><INCLUDE><SIGREF DOCUMENT=SML-BASIS-DOC/OS_IO/
```

`SUBSTRUCT` The contents are identical to those of `STRUCTURE` except that the fi rst contained element must be an `ID` naming the substructure, e.g.

```
<SPEC><SUBSTRUCT>Key<SIGREF/ORD_KEY/</SUBSTRUCT>
```

`SHARING` An `ID` followed by one or more pairings of `EQU` and `ID` elements. The `SHARING` element may include a `TYPE` attribute. E.g.

```
<SPEC>
  <SHARING TYPE><ID>A.vector<EQU><ID>V.vector
  <SHARING TYPE><ID>A.elem<EQU><ID>V.elem
  ...
```

`EXN` An `ID` and optional `TY`, e.g.

```
<EXN><ID>Fail<TY>string
```

or taking advantage of SGML short-hand (the `ID` tag is optional):

```
<EXN>Fail<TY>string
```

`TYPE` / `EQTYPE` An optional `TYPARAM`, an `ID`, and an optional `TY`, e.g.

```
<TYPE><TYPARAM>'a<ID>queue
```

`DATATYPE` An optional `TYPARAM`, an `ID`, and a list of `CONS` elements, each containing an `ID` followed by optional `TY` and `COMMENT` elements, e.g.

```
<SPEC>
  <DATATYPE><TYPARAM>'a<ID>option
            <CONS>NONE
            <CONS>SOME<TY>'a
  </DATATYPE>
```

A `DATATYPE` element may be marked with a `COMPACT` attribute that directs the output generators to place all of the constructors on a single line if possible.

`DATATYPEDEF` An `ID` followed by another `ID` or a `TYREF`. Adds a datatype replication, e.g.

```
<DATATYPEDEF><ID>access_mode</ID>
  <TYREF STRID="OS.FileSys" DOCUMENT=SML-BASIS-DOC>
    OS.FileSys.access_mode</TYREF>
</DATATYPEDEF>
```

`VAL` An `ID` followed by a `TY` and optionally a `RAISES` element containing one or more `EXNREF`s, e.g.

```
<SPEC>
  <VAL>fact<TY>int -> int
           <RAISES><EXNREF/Domain/
                   <EXNREF/Overflow/
```

Comments

Comments may be included as the last item within a SIGINSTANCE, SPEC, or CONS element. A COMMENT contains one or more paragraphs (PP), each optionally preceded by a PROTOTY element that contains one or more PROTO elements showing, in SML, how an object might be called or used with argument names. To take an example from the SML Basis Library:

```
<SPEC>
  <VAL>before<TY>('a * unit) -> 'a
  <COMMENT>
    <PROTOTY>
      <PROTO>
        <ARG/a/ before <ARG/b/
    <PP>
    returns <ARG/a/. It provides a notational shorthand for
    evaluating <ARG/a/, then <ARG/b/, before returning the
    value of <ARG/a/.
```

The specifi c argument names make it easier to write a description of what the function does.

The fi nal element within a PROTO can be EVALTO, which should contain SML showing the result of evaluating the expression.

References

There are several ways to cross-reference:

SML objects — Signatures, structures, functors, and their contents. Further details follow this list.

External objects — An IDREF element references an external non-SML identifi er. An optional KIND attribute describes what is being referred to, e.g. a C function or a system call. If the NOINDEX attribute is present the identifi er instance is not noted in an index. E.g. `<IDREF KIND="POSIX" NOINDEX/printf/`

HTML anchors — Cross-linking for HTML output. An ADEF with mandatory TAG attribute defi nes an anchor name for a run of text. AREF elements, also with TAG attributes, link back to the matching name. Reference is made to anchors in other documents by setting the DOCUMENT attribute to an entity value, see Entities. E.g.

```
<ADEF TAG="FLUSHWARNING">Disaster results if a flush
is attempted after closing the pipe</ADEF>.
   ...
Heed the <AREF TAG="FLUSHWARNING">earlier</AREF> warning.
```

URLs — A URL tag links to an external resource specifi ed by an HREF attribute. This tag is ignored when generating LaTeX output.

External ML-Doc — Reference is made to other ML-Doc documents with a DOCREF element that specifi es an entity with the DOCUMENT attribute, e.g.

```
 ...features of the <DOCREF
  DOCUMENT=SML-BASIS-DOC/SML Basis Library/...
```

Sections/floats — Cross-references may be made to SECTIONs or FLOATs that have LABEL attributes. The former using SECREF and the latter with FLOATREF. Both require a LABEL attribute of their own. Neither contain text, which comes instead from the referenced object. E.g.

```
<SECTION LABEL="Files">
. . .
<SECTION LABEL="Storage">
  <HEAD>Sockets</HEAD>
```

```
<PP>
The <SECREF LABEL="Files"> section describes...
```

Citations	A `CITE` element refers to a bibliography entry. It is not supported by the HTML generators. The value of the `KEY` attribute is passed directly into LaTeX.

The remainder of this section discusses references to SML objects. These references are included in the document index, if not marked with a `NOINDEX` attribute, and become hyperlinks in HTML output, unless marked with a `NOLINK` attribute. A `DOCUMENT` attribute is used to reference another ML-Doc project.

The SML object reference tags are:

`SIGREF`	Refer to a complete signature, e.g. `<SIGREF/MONO_ARRAY_SORT/`
`FCTREF`	Refer to a functor, e.g. `The <FCTREF NOLINK/ArrayQSortFn/ implements...`
`FCTARGREF`	Refer to the argument of a functor.
`STRREF`	Refer to a structure or substructure.
`EXNREF`	Refer to an exception, e.g. `<RAISES>` `<EXNREF STRID="General" DOCUMENT=SML-BASIS-DOC/Size/`
`TYREF`	Refer to a type, e.g. `...constructor <TYREF SIGID="UREF">uref</TYREF> with...`
`CONREF`	Refer to a datatype constructor, e.g. `...then it returns <CONREF STRID="Option"/NONE/`
`VALREF`	Refer to a value, e.g `<VALREF>randMin</VALREF>`.

A context attribute should be given with the latter fi ve tags when they refer to a specifi cation in another part of the module hierarchy. There are four, mutually exclusive, possibilities:

`SIGID`	For specifi cations in `SIGNATURE`s.
`STRID`	For specifi cations in `STRUCTURE`s or `SUBSTRUCT`s.
`FCTID`	For specifi cations in `FUNCTOR`s.
`TOPID`	For specifi cations available at the top-level without qualifi cation. No value is given.

Examples may be seen throughout this document. The partial path given in the ∗`ID` attribute and the reference text are merged when there is overlap between the rightmost components of the former and the leftmost components of the latter. For example, in cases like:

```
<TYREF STRID="OS.FileSys">OS.FileSys.access_mode</TYREF>
```

An ∗`ID` attribute is not required when the reference is to another element in the same `SIGBODY`.

Formatting

Tags available for inline formatting:

tag	**effect**
`EM`	emphasis
`IT`	italics
`BF`	bold
`TT`	typewriter
`CD`	code
`ARG`	function argument
`KW`	keyword

Program text may also be displayed, rather than inlined with the tag `CODE`. Both `CD` and `CODE` tags may carry a `LANG` attribute having a value of either "sml" or "c".

Tables

Tables are as in HTML but initial `COL` tags are mandatory:

```
<TABLE>
  <COL ALIGN=LEFT PARBOX="5em"><COL ALIGN=CENTER><COL ALIGN=RIGHT>
  <TR><TH>Colour<TH>Code<TH>Comment
  <TR><TD>Red<TD><CD/#FF0000/<TD>Plain red
  <TR><TD>Blue<TD><CD/#0000FF/<TD>Plain blue
</TABLE>
```

Table heading (TH) and cell (TD) elements have optional ALIGN and COLSPAN attributes. Tables that span multiple pages should be marked LONG.

Floats

Source code and (non-LONG) tables may be floated:

```
<FLOAT LABEL="modlist" CAPALIGN=TOP>
  <CAPTION>Module List
  <TABLE>...</TABLE>
</FLOAT>
```

The LABEL attribute is obligatory and may be referenced from a FLOATREF.

Lists

There are three types of lists:

- itemized (bulleted items), e.g.
  ```
  <ITEMIZE>
    <ITEM>First item
    <ITEM>Second item
  </ITEMIZE>
  ```
- enumerated (numbered), e.g.
  ```
  <ENUM>
    <ITEM>First item
    <ITEM>Second item
  </ENUM>
  ```
- descriptive, e.g.
  ```
  <DESCRIP>
    <DTAG/first/<ITEM>text about first item
    <DTAG/second/<ITEM>text about second item
  </DESCRIP>
  ```

Miscellaneous Text Blocks

Paragraphs may be grouped based on intent:

EXAMPLE	For extended examples.
QUESTION	Presents a note on, or discussion of, open design issues.
IMPLNOTE	Implementation notes, e.g. expected effi ciency.
RATIONALE	Explain the reasons behind design decisions.
SYSNOTE	Comments specifi c to an architecture (named with the ARCH attribute) or operating system (the OPSYS attribute) .

As an example:

```
<PP>
<RATIONALE>
  <PP>
```

```
          These functions...
          <PP>
          An alternative...
        </RATIONALE>
```

Regular Expressions

Regular expressions may be included within text (`RE`) or displayed on their own line (`REGEXP`). Literal characters are tagged as `GRAM.LIT` and displayed in typewriter font (other possible atoms are `GRAM.NONTERM` in italics, `GRAM.TERM` in roman, and `GRAM.KW` in bold). E.g.

```
        <RE><GRAM.LIT/i/<GRAM.LIT/f/</RE>
```

gives: `if`

Characters sets, displayed between square brackets, are built with `GRAM.CSET`. Sets may include character atoms and ranges, e.g.

```
        <REGEX><GRAM.CSET><GRAM.RANGE><GRAM.LIT/A/<GRAM.LIT/F/<GRAM.RANGE>
                          <GRAM.RANGE><GRAM.LIT/a/<GRAM.LIT/f/<GRAM.RANGE>
                          </GRAM.CSET></REGEX>
```

gives: `[A-Fa-f]`

Closures are expressed by grouping characters and specifying a count, e.g.

```
        <RE><GRAM.GRP ONE-OR-MORE><GRAM.LIT/a/<GRAM.LIT/b/</GRAM.GRP></RE>
```

gives: `(ab)+`

Possible counts are: `ONE` (the default), `ZERO-OR-ONE`, `ZERO-OR-MORE`, `ONE-OR-MORE`.

Alternation, regular expressions separated by bars, is also possible, e.g.

```
        <RE><GRAM.ALT>
            <GRAM.LIT/A/
            <GRAM.GRP ZERO-OR-MORE><GRAM.LIT/B/</GRAM.GRP>
            </GRAM.ALT></RE>
```

gives: `A | B*`

Mathematics

Some support is provided for type-setting mathematics. Formulas may be placed in running text (`MATH`), displayed (on a separate line, `DISPLAYMATH`) or made into a table of equations, e.g.

```
        <EQNARRAY>
          <EQN>x + y<EQNREL/=/2</EQN>
          <EQN>y<EQNREL/>=/0</EQN>
        </EQNARRAY>
```

gives:

```
        x + y   =    2
        y       >=   0
```

Other features:

subscripts	`<MATH>x<SUB/i+1/</MATH>`
superscripts	`<MATH>x<SUP/2*3/</MATH>`
modulo	`<MATH>x<MOD/y+z/</MATH>`
	displayed as: `x (mod (y+z))`
fractions	`<MATH><FRAC>1<OVER>2</FRAC></MATH>`
absolute value	`<MATH><NORM>x + y</NORM></MATH>`
ceiling	`<MATH><CEILING>x + y</CEILING></MATH>`
floor	`<MATH><FLOOR>x + y</FLOOR></MATH>`

sets	`<MATH><SET>x	x > 2</SET></MATH>`

Argument variables, e.g. `<ARG/lset/`, may be included in formulas, as may normal (roman) text, e.g.

```
<MTEXT/if not empty/
```

The predefi ned mathematical entities are:

```
&CUP;     &CAP;        &EXISTS;    FORALL
&GREATER; &GREATEREQ;  &IN;        &LESS;
&LESSEQ;  &NOTEQ;      &NOTIN;     &OMINUS;
&OPLUS;   &OTIMES;     &PI;        &PLUSMINUS;
&MINUSPLUS;            &INF;
```

Indexing

A hardcopy document contains up to three separate indexes:

topic	General index (by topic)
id	SML identifi er index
raises	Raised exception index

In HTML, separate indexes for signatures, structs, and types are possible.

Entries are extracted automatically from interface descriptions. They may also be inserted into text manually with the `INDEX` tag, e.g.

```
This module handles <INDEX KEY="Error Reporting">error reporting.
```

creating an entry in the General Index, by default, with appropriate page number:

```
Error Reporting, 14
```

The index—one of "topic, id" or "raises"—is specifi ed with the `WHICH` attribute. The `SEE` attribute replaces the page number with the given text.

The key-view tag specifi es alternate text or formatting for an index entry. Extra sub-index tags can be added, with optional key-views, to provide one or two extra levels of specifi city, e.g.

```
This module handles <INDEX KEY="Error Reporting">
  <KEY-VIEW><IT/Error Reporting/</KEY-VIEW>
  <SUBINDEX KEY="Module">
</INDEX>error reporting.
```

Gives:

Error Reporting

```
    Module, 14
```

Topics that span bodies of text are delimited with `START` and `STOP` attributes, e.g.

```
<INDEX KEY="Displays" START> This chapter discusses display
characteristics of...
  ...which concludes our discussion.<INDEX KEY="Displays" STOP>
```

Could produce:

```
Displays, 19—25
```

SGML vs HTML/XML

Although the basics of editing ML-Doc will be familiar to most authors of HTML and XML, SGML has some peculiarities that are designed to make editing ‘by hand’ easier.

Many end-tags are optional, as in HTML, but unlike in XML, e.g.

```
<PP>
No explicit end-tag is given for this paragraph...
<PP>
  ...before moving into the next
```

The usual start and end-tags are acceptable:

```
<CD>'a</CD>
```

But *null end-tags* can also be used for the same effect:

```
<CD/'a/
```

Attribute names are often optional when an enumerated value is required, i.e. instead of `<GRAM.GRP COUNT=ONE>` one can also write `<GRAM.GRP ONE>`. some attribute values can be stated without a name, i.e. instead of:

```
<SPECBREAK NEWLINE=NEWLINE>
```

one can write:

```
<SPECBREAK NEWLINE>
```

Attribute values need not always be enclosed in double quotes.

These details are well described in Chapter 9 of *SGML and HTML Explained* (refer **SEE ALSO**) .

FILES

System-wide ML-Doc fi les and directories are stored at:

```
/usr/local/share/ml-doc
```

Notably:

`lib/catalog`	Master catalog fi le.
`lib/ml-doc.dtd`	DTD of ML-Doc language.
`lib/entities.sgml`	Entity defi nitions (see `Entities`) .
`lib/LaTeX/`	LaTeX class and style fi les for Hardcopy and Proof output.

EXAMPLES

After creating the directory structure (see **Directory Structure**) , and writing up the ML-Doc fi les (see **WRITING DOCUMENTATION**) , create a `Makefile`:

```
find ML-Doc -name '*.mldoc' -print | mk-mldoc-makefile
```

Then produce HTML documentation:

```
make
```

and/or LaTeX:

```
make Hardcopy
```

Creating/updating a signature fi le:

```
extract-sig ML-Doc/sync-var.mldoc
```

Processing LaTeX output:

```
latex manual
makeindex -o manual.ind manual.idx    # topic index
makeindex -o manual.nnd manual.ndx    # identifier index
makeindex -o manual.rnd manual.rdx    # exception index
latex manual
```

BUGS

The utilities do not provide helpful error messages, usually just uncaught exceptions.

The toolset does not seem, in totality, to handle sub-directories under `ML-Doc` very well. Soft links provide a rudimentary work around.

The `FIGURE` tag is not implemented.

The `GRAMMAR` and `GRAM.SEP` tags are not supported.

Neither `SUM`, `PROD`, `UNION`, nor `INTERSECT` are implemented.

`SUB` and `SUP` do not work well together, i.e. squaring the ith x will not be typeset properly:

```
<MATH>x<SUB/i/<SUP/2/</MATH>
```

It is not clear what `MGROUP` does. The contents are wrapped in brackets () in HTML output and invisible parentheses {} in LaTeX output.

The target added by the **`−root`** option of **`mk-mldoc-makefile`** does not run `makeindex`(1).

It is not clear how to write mutually recursive `DATATYPE` or `VAL` specifi cations, nor is it clear what the `REC` attribute signifi es .

The `DOCREF` tag is not supported. There is no way of referencing `INTERFACE` elements (i.e. the `LABEL` attribute is redundant) .

SEE ALSO

`mkdoc`(1), `/usr/local/share/ml-doc/lib/ml-doc.dtd`.

John H. Reppy, *Concurrent Programming in ML*, *Cambridge University Press*, 1999.

Emden R. Gansner and John H. Reppy, *The Standard ML Basis Library*, *Cambridge University Press*, 2004.

Martin Bryan, *SGML and HTML Explained*, *Addison Wesley Longman*, 1997, available online: http://www.is-thought.co.uk/book/home.htm.

AUTHORS

John H. Reppy ⟨jhr@cs.uchicago.edu⟩ wrote and maintains ML-Doc.
Andrew Appel ⟨appel@princeton.edu⟩ wrote the fi rst version of **`latex-gen`**.
Lal George ⟨lg@network-speed.com⟩ wrote the **`proof-latex`** tool.

Timothy Bourke ⟨timbob@bigpond.com⟩ wrote this man page for the FreeBSD port based on documentation and source fi les from the distribution.