# The LEDA Tools for Manual Production and Documentation

Evelyn Haak    Kurt Mehlhorn    Stefan Näher    Michael Seel
Christian Uhrig

July 14, 1999

## Contents

# 1 The Manual Page of the Documentation Tools

## 1.1 How to use the manual production tools

LEDA offers several tools for manual production and documentation. The following diagram shows the different extraction and conversion processes possible from a documented and manual commented file to the different intermediate files for further processing.

```
       __________ weave _______   LaTeX source
      |
      |   ------- Lman --------   LaTeX source for manual page
    foo
      |   ------- Ldoc --------   LaTeX source for full documentation
      |
      |   ------- Fman --------   ASCII version of manual page
      |
      |__________ tangle ______   C/C++ program


    foo's ------- lextract ----   LaTeX source of manual

    foo's ------- Mkdvi -------   dvi versions of all manual pages
                                  in directory MANUAL/DVI
```

Lman produces manual pages from header files suitably augmented by so-called manual comments. Many examples can be found in the LEDA include directory (LEDAROOT/incl/LEDA). A manual comment starts with `\*{\M` and ends with `}*/`. Try `Lman list` to see Lman in action. If it does not work the error is very likely to be one of the following (if not, you should refer to the LEDA installation guide):

- The environment variable LEDAROOT is not set to the root directory of the LEDA system.
- $LEDAROOT/Manual/cmd is not part of your PATH.
- $LEDAROOT/Manual/tex is not part of your TEXINPUTS.

```
               LEDAROOT
                   |
    -----------------------------------
   |               |                   |
 incl            Manual               DVI
   |          _____|_______________
 LEDA        |     |        |      |
            cmd  noweb     tex   MANUAL
```

shows the parts of the LEDAROOT directory relevant for manual production and documentation. The include directory contains all header files of the LEDA system Manual contains all the relevant commands and tex files.

**Lman** can be applied to LEDA data types and user-defined data types. Write `Lman T[.h] options` to apply it to the LEDA data type T and `Lman T[.ext] options` to apply it to file T.ext in the working directory. The extensions lw (Lweb file), nw (noweb file), w (a Cweb-file), and h (h-file) are possible. The extensions are tried in this order and the working directory is searched before the LEDA include directory.

The following options are available:

```
size={12,11,10}
constref={no,yes}
partypes={no,yes}
usesubscripts={no,yes}
```

```
numbered={no,yes}
xdvi={yes,no}
warnings={yes,no,log}
includefile={no,yes}
indexing={no,yes}
informational={yes,no}
ack={yes,no}
latexruns={1,2,0}
filter={all, signatures, definition, types, creation, operations, events,
implementation, example, operator, opname}
pid={"",string}
```

Options are given in assignment syntax. There must be no blank on either side of the equality sign.

The size-option selects the font-size. The option constref determines whether const-ref pairs are shown, the option partypes determines whether the type of a value parameter of an operator is suppressed if equal to the currently defined type, and the option usesubscripts determines whether indexed variables are typeset using subscripts.

A manual page can be numbered or not and the call of xdvi can be suppressed. Lman may be asked to issue warnings and warnings may require a user action or not. We recommend to run Lman with `warnings=yes` and `ack=yes` in the early stages of designing a manual page. Lman usually runs LaTeX twice in order to get cross-references correct. You may settle for a single run of LaTeX. Lman makes use of a preprocessor called ext. The preprocessor writes its output on file /tmp/pid-ext.tex where pid is either the process identification of the preprocessor run or specified by the pid-option. Mkman makes use of the pid-option. You will probably never have to use it.

There are four ways to specify options for Lman. You may either put them into the file itself by means of the Moptions command, or you can put them onto the command line, or you can put them into a configuration file Lman.cfg in either the working directory or the home directory.

The Lman-utility can also be told to extract only part of a manual page by the filter-option. The options definition, creation, operations, implementation, and example cause extraction of the corresponding section of the manual, and the option signatures extracts the signatures of all operations but suppresses the descriptions. One can also extract a single operation by using its name, e.g., `Lman stack filter=push` extracts the push operation of the stack type and `Lman point filter=operator+` extracts the + operator of the point type (write arrop to inquire about the array operator and funop to inquire about the function operator).

**Fman** is an alternative to Lman. It gives low quality ASCII output but works faster.

**Mkman** is the tool to produce manuals like the LEDA manual.

**Ldoc** is the utility that supports the production of full documentations. All remarks above apply equally well to Ldoc. Ldoc applies to Lweb, noweb, and Cweb files. For use with noweb and Lweb, the LEDA dialect of noweb, you must have installed noweb and for use with Cweb you must have installed Cweb. The subdirectories Cweb, noweb, and Lweb contain full documentations of parts of the LEDA system. Copy a file from one of these directories to your working directory and run Lman on it to see it in action. If it does not work check whether you have installed the proper literate programming system.

**lweave** is the weave tool for Lweb.

The production of manual pages and full documentations proceeds in phases.

```
Lman XXX.[h|lw|nw|w]  ----->  XXX.tex   -------> XXX.dvi
                       ext.pl            latex

lweave and lw2dvi XXX.lw ---------->  XXX.tex -------> XXX.dvi
           lweave              latex
```

```
Ldoc XXX.[lw|nw|w] ----> temp.[lw|nw|w] --------------> XXX.tex -------> XXX.dvi
       |           ext.pl              [l|no|c]weave       |    latex
       | ext.pl                                            |
       |                                                   |
     XXX.man ---------------------------------------------
                         \input

Fman XXX.[h|lw|nw|w]  ----------> ASCII
                          Fman
```

We use standard tools for all but the first phase. The first phase is accomplished by the perl program *ext.pl*. It realizes Lman, Ldoc and Fman in a very direct way.

## 1.2 How to create a Manpage

Starting from a C++ data type `DT` we give the documentation process and its syntactical elements in form of a context free grammar. Non-terminals are marked by `<...>`. Terminals are pseudocode statements like `text_text_text` which represent pieces of latex text apart from the C++ language constructs which should be easy to locate. Optional syntactical elements are put in square brackets `[...]`.

```
<Augmented C++-File> -> <Manpage Statement>
                        class DT {
                        <Mdefinition Statement>
                        <Mtypes Statement>
                        <Mcreation Statement>
                        <Moperations Statement>
                        };
                        <Moperations Statement>
                        <Mimplementation Statement>
                        <Mexample Statement>
```

The previous picture should give you an impression about the possible sectioning of the implementation of `DT`. Note that several of the cited sections are optional and are not necessarily part of the data types manpage specification. We now handle the different sections.

#### The Manpage Statement

```
<Manpage Statement> ->
        /*{\Manpage {DT} {T1,..,Tk} {short_description} {d}}*/
```

This comment is mandatory for the start of a manpage. It introduces the data type to the macro package and sets the main variable `\Mname = DT<T1,..,Tk>`. Thus `DT` the data types name. `T1,..,Tk` is a list of template parameters which can be empty. the `short_description` appears in the manpage header and can be used as a latex reference. Finally `\Mvar` is set to `d` as the default variable name of a class instance.

#### The Mdefinition Statement

```
<Mdefinition Statement> ->
        /*{\Mdefinition specification_text}*/
```

This comment is somehow mandatory as a manpage should always contain an accurate specification of a data type.

#### The Mtypes Statement

```
<Mtypes Statement> ->
        /*{\Mtypes [W]}*/
        <list of local type definitions>
```

This comment is optional. It should be placed in a public section of class `DT` where the user wants to specify local types in class scope. `W` is an optional width specifier in cm determining the width of the indendation of the following type specifications.

```
<list of local type definitions> ->
        typedef  TYPE1  LOCALTYPE1;
        /*{\Mtypedef typedef_documentation}*/
        <list of local type definitions>
```

This triggers a documenation line containing the whole typedef statement followed by `typedef_documentation`.

```
<list of local type definitions> ->
        typedef TYPE2   LOCALTYPE2;
        /*{\Mtypemember type_documentation}*/
        <list of local type definitions>
```

This triggers a documenation line containing only the defined type `LOCALTYPE2` followed by `type_documentation`.

```
<list of local type definitions> ->
        class LOCALTYPE3 {
        /*{\Mtypemember type_documentation}*/
          // local class implemenation
        };
        <list of local type definitions>
```

This triggers a documenation line containing only the defined type `LOCALTYPE3` followed by `type_documentation`.

```
<list of local type definitions> ->
        enum LOCALTYPE4 { T1, ... , Tk };
        /*{\Menum type_documentation}*/
        <list of local type definitions>
```

This triggers a documenation line containing the defined enum type `LOCALTYPE4` followed by `type_documentation`.

```
<list of local type definitions> -> <empty>
```

### The Mcreation Statement

```
<Mcreation Statement> ->
        /*{\Mcreation [MVAR] [W]}*/
        <list of constructors>
```

This comment is somehow mandatory as most classes have constructors which have to be documented. `MVAR` can be used to set `\Mvar` to the string `MVAR`. `W` is an optional width specifier in cm determining the width of the indendation of the following construction operations.

```
<list of constructors> ->
        DT(paramlist);
         /*{\Mcreate constructor_specification }*/
        <list of constructors>
```

creates the documenation for the constructor `DT(paramlist)`.

```
<list of constructors> ->
        ~DT();
         /*{\Mdesctruct desctructor_specification }*/
        <list of constructors>
```

creates the documenation for the destructor `~DT()`.

```
<list of constructors> -> <empty>
```

### The Moperations Statement

```
<Moperations Statement> ->
        /*{\Moperations [W1] [W2]}*/
        <list of operation definitions>
```

This comments starts the operations section of the manpage which usually is dominating the manpage. `W1` is an optional width specifier determining the indendation used for the return type of all following operations. `W2` is an optional width specifier determining the width of the additional indendation up to the textual specification block. In the following we give all possible kinds of manual comments documenting member operations and friend operations of a class.

```
<list of operation definitions> ->
        <operation definition> <list of operation defintions> |
        <empty>
```

The following entries should be rather self explaining. The first section provides documentation elements concerning member operations. Top down we have standard named member operations, static operations, conversion operations, binary operators, unary operatos and the special selection operator known from arrays.

```
<operation definition> ->
        return_type operation_name(paramlist);
        /*{\Mop semantic_description }*/                |
        static return_type static_operation_name(paramlist);
        /*{\Mstatic semantic_description }*/            |
        operator return_type ( );
        /*{\Mconversion semantic_description }*/        |
        return_type operator binary_op_symbol (paramlist);
        /*{\Mbinop semantic_description }*/             |
        return_type operator unary_op_symbol (paramlist);
        /*{\Munop semantic_description }*/              |
        return_type operator [] (paramlist);
        /*{\Marrop semantic_description }*/
```

The next section gives documentation elements suitable for friends in classes or for global functions.

```
<operation definition> ->
        return_type friend_operation_name(paramlist);
        /*{\Mfunc semantic_description }*/              |
        return_type friend_operator binary_op_symbol (paramlist);
        /*{\Mbinopfunc semantic_description }*/         |
        return_type friend_operator unary_op_symbol (paramlist);
        /*{\Munopfunc semantic_description }*/
```

Finally there's a function object documentation element.

```
<operation definition> ->
        constructor(paramlist);
        /*{\Mfunobj{return_type} semantic_description }*/
```

### The Mimplementation Statement

```
<Mimplementation Statement> ->
        /*{\Mimplementation implemenation_information }*/
```

This section can be used to give the user certain implemenation details like running time, space requirements or implementation specific informations which are helpfull for the user of the data type.

**The Mexample Statement**

```
<Mexample Statement> ->
        /*{\Mexample (example_description [<verbatim_example_block>])* }*/
```

In this section you should provide a small example demonstrating the usage of `DT`. `example_description` is just regular latex description.

```
<verbatim_example_block> ->
        \begin{Mverb}
        verbatim_multi_line_example_code
        \end{Mverb}
```

This is just an optional block structure enabling code citation within the manual comment.

## 1.3 A Manpage Example

This section gives a short example of a manpage following the above specifications.

Figure 2 shows the intermediate TEX-file produced by Lman for the data type stack. The TeX file basically contains one TeX-macro call for each manual command. It also has a short preamble and postamble. The preamble reads MANUAL.pagesize and MANUAL.mac, opens the document and enters the manual environment, and the postamble closes the manual environment and the document. The files MANUAL.pagesize and MANUAL.mac are both contained in the LEDAMAN directory; they contain the definition of the pagesize used for the LEDA manual and the definition of the manual macros respectively.

Ldoc applied to the same file produces (in stack.man) the subfile starting with `\begin{manual}` and ending with `\end{manual}`. Ldoc also produces a temporary file temp.w which is obtained from the input file by deleting all manual comments. It applies cweave to this file to obtain XXX.tex.

```
#ifndef LEDA_STACK_H
#define LEDA_STACK_H

#include <LEDA/basic.h>
#include <LEDA/impl/slist.h>

/*{\Manpage {stack} {E} {Stacks} {S}}*/

template<class E> class _CLASSTYPE stack : private SLIST
{
/*{\Mdefinition
An instance |S| of the parameterized data type |\Mname| is a sequence of
elements of data type |E|, called the element type of |S|. Insertions or
deletions of elements take place only at one end of the sequence, called
the top of |S|. The size of |S| is the length of the sequence, a stack
of size zero is called the empty stack.}*/

public:

/*{\Mcreation}*/

  stack() {}
/*{\Mcreate creates an instance |\Mvar| of type |\Mname| and initializes
it to the empty stack.}*/

  stack(const stack<E>& S) : SLIST(S) {}
 ~stack() { clear(); }
  stack<E>& operator=(const stack<E>& S)
  { return (stack<E>&)SLIST::operator=(S); }

/*{\Moperations 2.5 4}*/

E top()   const { return ACCESS(E,SLIST::head());}
/*{\Mop      returns the top element of |\Mvar|.\\
             \precond $S$ is not empty.}*/

void push(E x)  { SLIST::push(Copy(x)); }
/*{\Mop      adds $x$ as new top element to |\Mvar|.}*/

E pop()         { E x=top(); SLIST::pop(); return x; }
/*{\Mop      deletes and returns the top element of |\Mvar|.\\
             \precond $S$ is not empty.}*/

\*{\Mimplementation
Stacks are implemented by singly linked linear lists. All operations take
time $O(1)$.}*/

#endif
```

Figure 1: The header file stack.h

```
\documentclass[12pt,a4paper]{article}
\input MANUAL.pagesize
\input MANUAL.mac

\begin{document}
\begin{manual}

\section*{Stacks(stack)}\label{stack}

\definition

An instance $S$ of the parameterized data type \mbox{$\mathit{stack<E>}$} is a
sequence of elements of data type $E$, called the element type of
$S$. Insertions or deletions of elements take place only at one end of the
sequence, called the top of $S$. The size of $S$ is the length of the sequence,
a stack of size zero is called the empty stack.


\creation

\create {\mbox{$\mathit{stack\<E\>}$}}
{\mbox{$\mathit{S}$}}
{creates an instance \mbox{$\mathit{S}$} of type \mbox{$\mathit{stack\<E\>}$}
and initializes it to the empty stack.
}

\setlength{\typewidth}{2.5cm}
\setlength{\callwidth}{4cm}
\computewidths
\operations

\function {\mbox{$\mathit{E}$}}
{\mbox{$\mathit{S.}$}top\mbox{$\mathit{()}$}}
{returns the top element of \mbox{$\mathit{S}$}.\\ \precond $S$ is not empty.
}

\function {\mbox{$\mathit{void}$}}
{\mbox{$\mathit{S.}$}push\mbox{$\mathit{(E\ x)}$}}
{adds $x$ as new top element to \mbox{$\mathit{S}$}.
}

\function {\mbox{$\mathit{E}$}}
{\mbox{$\mathit{S.}$}pop\mbox{$\mathit{()}$}}
{deletes and returns the top element of \mbox{$\mathit{S}$}.\\ \precond $S$ is not empty.
}

\function {\mbox{$\mathit{int}$}}
{\mbox{$\mathit{S.}$}empty\mbox{$\mathit{()}$}}
{returns true if \mbox{$\mathit{S}$} is empty, false otherwise.
}

\end{manual}
\end{document}
```

Figure 2: The intermediate TEX-file produced from stack.h

# 2 Implementation of Toplevel Callers

## 2.1 The ASCII View

⟨*Fman*⟩≡

```
#!/bin/sh -f
perl $LEDAROOT/Manual/cmd/ext.pl Fman "$@"
```

## 2.2 The LaTeX View

⟨*Lman*⟩≡

```
#!/bin/sh -f
perl $LEDAROOT/Manual/cmd/ext.pl Lman "$@"
```

⟨*Ldoc*⟩≡

```
#!/bin/sh -f
perl $LEDAROOT/Manual/cmd/ext.pl Ldoc "$@"
```

⟨*lextract*⟩≡

```
#!/bin/sh -f
if [ "$1" = "" -o "$2" = "" ]
then
  echo ""
  echo "Usage is"
  echo "          lextract infile outfile [options]"
  echo ""
  echo "Extracts the manual comments from infile into outfile."
  exit
fi
infile=$1
outfile=$2
shift
shift
perl $LEDAROOT/Manual/cmd/ext.pl lextract $infile outfile=$outfile "$@"
```

⟨*Mkman*⟩≡

```
#!/bin/sh

if [ "$1" = "" ]; then
  source=$LEDAROOT/incl/LEDA
  ext=h
else
  source=$1
  if [ "$2" = "" ]; then
    ext=h
  else
    ext=$2
  fi
fi

rm -r -f extract

mkdir extract

echo Extracting manual pages ...
echo " "

for f in $source/*.$ext; do
   echo $f
   lextract $f  extract/`basename $f .$ext`.tex \
   informational=no indexing=yes includefile=yes warnings=log
done

if [ -d $source/generic ]; then
  for f in $source/generic/*.$ext; do
   echo $f
   lextract $f  extract/`basename $f .$ext`.tex \
   informational=no indexing=yes includefile=yes warnings=log
  done
fi
```

⟨*Mkdvi*⟩≡

```
#!/bin/sh
# this script must be called in $LEDAROOT/Manual/MANUAL

cd ../..

if [ ! -d Manual/MANUAL ]; then
  echo ""
  echo "Mkdvi must be called from \$LEDAROOT/Manual/MANUAL"
  exit 0
fi;

lroot=`pwd`

LEDAROOT=$lroot
export LEDAROOT

# check for perl and latex

echo " "

touch t.pl

if perl t.pl; then
   rm t.pl
   echo using `which perl`
else
   rm t.pl
   echo Cannot execute perl.
   echo " "
   exit 0
fi


source=$lroot/incl/LEDA
dvidir=$lroot/Manual/DVI

cd $dvidir

cp $lroot/Manual/MANUAL/intro.tex ./Introduction.tex
cp $lroot/Manual/MANUAL/Title.tex   .
cp $lroot/Manual/MANUAL/Preface.tex .
cp $lroot/Manual/MANUAL/Basics.tex  .


if latex Introduction > /dev/null; then
   echo using `which latex`
else
   echo Cannot execute latex.
   echo " "
   exit 0
fi


echo " "
echo Updating DVI-files in `pwd`  ...
echo " "
```

```
if [ -x /usr/bin/test ]; then
  tcmd=/usr/bin/test
elif [ -x /bin/test ]; then
  tcmd=/bin/test
else
  tcmd=test
fi


for f in $source/*.h $source/generic/*.h; do
  dvi=`basename $f .h`.dvi
  if grep Manpage $f > /dev/null; then
     if $tcmd ! $dvi -nt $f; then
        echo $dvi
#       perl ../cmd/ext.pl Lman $f xdvi=no dvioutfile=$dvi
        perl ../cmd/ext.pl Lman $f xdvi=no dvioutfile=$dvi >> mkdvi.log 2>&1
     fi
  fi
done

rm -f *.log

cd ..
```

$\langle Mkdvisfrom \rangle\equiv$

```
#!/bin/sh

# check for perl and latex

echo " "

if perl $LEDAROOT/Manual/cmd/ext.pl; then
   echo "perl  ok"
else
   echo Cannot execute perl.
   echo " "
   exit 1
fi

if noweave < /dev/null > /dev/null; then
   echo "noweb ok"
else
   echo Cannot execute noweave.
   echo " "
   exit 1
fi

echo '\\documentclass{article}' > dummy.tex
echo '\\begin{document} hw \\end{document}' >> dummy.tex
echo '' >> dummy.tex
if latex dummy.tex > /dev/null; then
   echo "latex ok"
   rm -rf dummy.*
else
   echo Cannot execute latex.
   echo " "
   exit 1
fi


source=$1

echo " "
echo Constructing dvi-files for xlman in `pwd`  ...
echo " "

if [ -x /usr/local/bin/grep ]; then
  grep_cmd=/usr/local/bin/grep
elif [ -x /usr/bin/grep ]; then
  grep_cmd=/usr/bin/grep
else
  grep_cmd=grep
fi

for f in $source/*.h; do
  if $grep_cmd -q Manpage $f; then
    ln -s $f .
  fi
done

for f in *.h; do
```

```
  fbase=`basename $f .h`
  fbase=`basename $fbase _decl`
  perl $LEDAROOT/Manual/cmd/ext.pl Lman $f xdvi=no dvioutfile=$fbase.dvi
done

rm -f *.h *.log
```

## 2.3 The HTML View

The command **Mkhtml** makes the html-version of the LEDA-manual. It can be called in any directory. It gets the necessary tex and header files via LEDAROOT. The tasks are done by the two phases Toplevel and Lowlevel described below.

The top level is determined by

- It copies all tex-files from LEDAROOT/Manual/MANUAL
- It applies some replacements to the tex files latex2html insists that the command names of LaTeX commands consist of letters only. We used \<, \>. We remove them.
- We want the crossreferences to appear nicely in the html-document. latex2html replaces all **\ref** by hyperlinks but an ordinary ref or pageref appears only as a box. Named links are nicer. **handle_refs** replaces occurrences of **\ref{label}** by **\htmlref{label'}{label}** where label' is obtained by quoting underscores.

Making the lower level is simple. We only need to call ext.pl in mode HTML to extract the appropriate Tex-file.

Having set up the necessary tex-files we call **latex MANUAL** twice and then **latex2html** once. This will construct the html-version of the manual in the subdirectory MANUAL.

⟨*Mkhtml*⟩≡

```
#!/bin/sh

if [ "$1" = "top" -o "$1" = "" ]; then
echo " "
echo "Preparing all tex-files in top level of the Manual"
files="$LEDAROOT/Manual/MANUAL/*.tex"
for f in $files
do
   core=`basename $f .tex`
   echo "preparing $core"
   cp $f $core.mkhtml
   perl -p -i.bak -e 's/\\</</g;s/\\>/>/g;' $core.mkhtml
   perl $LEDAROOT/Manual/cmd/handle_refs.pl $core.mkhtml > $core.tex
done
\rm -f *.mkhtml *.mkhtml.bak
fi # $1 != low

if [ "$1" = "low" -o "$1" = "" ]; then
\rm -r -f extract
mkdir extract
cd extract
notangle -Rnamereplacementtable $LEDAROOT/Manual/noweb/ext.nw \
  > namereplacementtable

if [ -x /usr/local/bin/grep ]; then
  grep_cmd=/usr/local/bin/grep
elif [ -x /usr/bin/grep ]; then
  grep_cmd=/usr/bin/grep
else
  grep_cmd=grep
fi

echo "Extending namereplacementtable by Manpage entries"
files="$LEDAROOT/incl/LEDA/*.h $LEDAROOT/incl/LEDA/generic/*.h"
for f in $files
do
```

```
  core=`basename $f .h`
  # printf "$core "
  printf "."
  perl $LEDAROOT/Manual/cmd/extract_mtype.pl $f >> namereplacementtable
done

echo " "
echo "Preparing all tex-files for LEDA types"
files="$LEDAROOT/incl/LEDA/*.h $LEDAROOT/incl/LEDA/generic/*.h"
for f in $files
do
  if $grep_cmd -q Manpage $f; then
   core=`basename $f .h`
   echo "preparing tex-file for $core"
   perl $LEDAROOT/Manual/cmd/ext.pl HTMLext $f outfile=$core.tex \
   informational=no warnings=log indexing=yes includefile=yes
  fi
done
cd ..

fi #

echo "calling latex MANUAL.tex"
echo " "

latex MANUAL.tex
latex MANUAL.tex
dvips MANUAL

echo "please ensure that your path does not contain dots!!!"
echo "calling latex2html MANUAL.tex"
echo ""

\rm -rf MANUAL

if [ "$1" = "low" -o "$1" = "top" ]; then
  shift
fi

notangle -R.latex2html-init $LEDAROOT/Manual/noweb/ext.nw \
  > .latex2html-init # this gives flags
latex2html -debug MANUAL.tex $*

exit
```

We use the following addidional settings in the dotfile:

⟨*.latex2html-init*⟩≡

```
# This number will determine the size of the equations, special characters,
# and anything which will be converted into an inlined image
# *except* "image generating environments" such as "figure", "table"
# or "minipage".
# Effective values are those greater than 0.
# Sensible values are between 0.1 - 4.
$MATH_SCALE_FACTOR = 1.4;

# This number will determine the size of
# image generating environments such as "figure", "table" or "minipage".
# Effective values are those greater than 0.
# Sensible values are between 0.1 - 4.
$FIGURE_SCALE_FACTOR = 1.4;

# If this is set then the resulting HTML will look marginally better if viewed
# with Netscape.
$NETSCAPE_HTML = 1;

# We want a friendly white backround for the manual
$BODYTEXT = 'BGCOLOR="#FFFFFF" BACKGROUND="ledabg.jpg"';
$HTML_VERSION = "3.2,math";
$NO_SIMPLE_MATH = 1;
$MAX_SPLIT_DEPTH = 4;
$CUSTOM_TITLES = 3;
# $LONG_TITLES = 3;
# $SHOW_SECTION_NUMBERS = 1;
$SHORT_INDEX = 1;
$LOCAL_ICONS = 1;
$NAV_BORDER = 0;
$ALTERNATIVE_ICONS = $ENV{'LEDAROOT'}."/Manual/noweb/icons";
$EXTERNAL_UP_LINK = "http://www.mpi-sb.mpg.de/LEDA";
$EXTERNAL_UP_TITLE = "LEDA home page";

$iconsizes{'up'} = 'WIDTH="44" HEIGHT="44"';
$iconsizes{'next'} = 'WIDTH="44" HEIGHT="44"';
$iconsizes{'previous'} = 'WIDTH="44" HEIGHT="44"';
$iconsizes{'contents'} = 'WIDTH="44" HEIGHT="44"';
$iconsizes{'index'} = 'WIDTH="44" HEIGHT="44"';


# the following custom title production procedure makes almost
# the same as the LONG_TILTLES version, only it finds doubles
# and takes our short section names in the brackets if there
sub custom_title_hook  {
    local($_)= @_;
    local($num_words) = $CUSTOM_TITLES;
    $_ =~ s/(^|\s)\s*($GENERIC_WORDS)(\'|\b)//ig;
    #remove leading numbering, unless that's all there is.
    local($sec_num);
    if (!(/^\d+(\.\d*)*\s*$/)&&(s/^\s*(\d+(\.\d*)*)\s*/$sec_num=$1;''/e))
    { $num_words-- };
    #revert entities, etc. to TeX-form...
    &remove_markers; s/<[^>]*>//g; #remove tags
    $_ = &revert_to_raw_tex($_);
```

```
    # get $CUSTOM_TITLES number of words from what remains
    if ( /[^\(]*\(([^\)]*)\)/ ) { $_ = $1; }
    else { $_ = &get_first_words($_, $num_words) if ($num_words); }
    if ( $ledaschondrin{$_} )
    { my $tmp=$_; $_ .= "$ledaschondrin{$_}"; ++$ledaschondrin{$tmp}; }
    else { $ledaschondrin{$_} = 1; }
    # ...and cleanup accents, spaces and punctuation
    s/\\\W\{?|\}//g; s/^\s*//g; s/\s*$//g; s/\s/_/g; s/\W/_/g; s/__+/_/g; s/_+$//;
    $_;
}

&ignore_commands( <<_IGNORED_CMDS_);
setopdims # {} # {}
_IGNORED_CMDS_

1;      # This must be the last line
```

The following macro extracts the data type symbol from a manpage:

⟨*extract_mtype.pl*⟩≡

```
$INPUT = $ARGV[0] && shift;
open (INPUT)  || die "Error:extract_mtype: can't find input file $INPUT: $!\n";
while (<INPUT>) {
  if ( /\/\*\{\\Manpage\s*\{([^\{\}]*)\}/ )
  { print "$1\n"; }
}
```

I am handling refs and pagerefs.

⟨*handle_refs*⟩≡

```
#!/bin/sh -f

for f
do
   core=`basename $f .tex`
   mv $core.tex $core.temp
   perl $LEDAROOT/Manual/cmd/handle_refs.pl $core.temp > $core.tex
   rm $core.temp
done
```

⟨*handle_refs.pl*⟩≡

```
$INPUT = $ARGV[0] && shift;
open (INPUT)  || die "Error: Can't find input file $INPUT: $!\n";

while (<INPUT>)
{ if (/\\ref\{([^\}]*)\}/)
  { $key = $1; $key1 = $key; $key1 =~ s/_/\\_/g;
    s/\\ref\{$key\}/\\htmlref\{$key1\}\{$key\}/g;
  }
  if (/\\pageref\{([^\}]*)\}/)
  { $key = $1; $key1 = $key; $key1 =~ s/_/\\_/g;
    s/\\pageref\{$key\}/\\htmlref\{$key1\}\{$key\}/g;
  }
  print;
}
```

The command `HTMLman T` makes the html version of the manual page for type T. The command looks for files `T.lw`, `T.nw`, `T.w`, `T.h` in this order and produces a tex-file `T.tex`. The tex-file is sent through latex twice and then through latex2html. The entry node to the html manual page is `T/T.html`.

⟨*HTMLman*⟩≡

```
#!/bin/sh -f

if [ $1 = "" ]; then
    echo HTMLman file[.h]
    echo   produce a HTML manual page from a header file
    exit
fi

echo " extracting file $1.tex"
echo " "

notangle -Rnamereplacementtable $LEDAROOT/Manual/noweb/ext.nw \
  > namereplacementtable

if [ -d $1 ]; then
  rm -rf $1
fi

perl $LEDAROOT/Manual/cmd/ext.pl HTMLman $1 outfile=$1.tex \
  informational=no warnings=log indexing=yes includefile=yes

latex $1.tex
latex $1.tex

latex2html -split 0 -local_icons -html_version 3.2,math -no_math $1.tex
  # we require version > 98.1
base=`basename $1`
here=`pwd`;
netscape -raise -remote "openURL($here/$base/$base.html)"
  # we use netscape version > 4.0

rm -f namereplacementtable
```

A tool to make HTML documents from general Lweb documents would be nice:

⟨*HTMLdoc*⟩≡

```
#!/bin/sh

if [ $1 = "" ]; then
    echo HTMLdoc file[.lw]
    echo   produce a HTML manual page from a header file
    exit
fi

base=`basename $1 .lw`
if [ -d $base ]; then
  rm -rf $base
fi

perl $LEDAROOT/Manual/cmd/ext.pl HTMLdoc $1
shift
if [ ! -f HTMLdoc-latex2html.cfg ]; then
  notangle -R.latex2html-init $LEDAROOT/Manual/noweb/ext.nw \
    > HTMLdoc-latex2html.cfg
fi
latex2html -init_file HTMLdoc-latex2html.cfg "$@" $base.tex
  # we require version > 98.1
perl $LEDAROOT/Manual/cmd/htmlweave.pl $base AFTER
rm -rf $base/*.html.bak
  # this script repairs some ill sitting crossrefs created by
  # l2h when distributing the HTML output over various files

echo You can change the settings for latex2html by editing the file
echo
echo "  HTMLdoc-latex2html.cfg"
echo

here=`pwd`;
netscape -raise -remote "openURL($here/$base/$base.html)"
  # we use netscape version > 4.0
```

# 3 The Main Engine: ext

## 3.1 An Overview

Ext.pl operates in phases. The first phase processes the command line, takes care of initialization, and produces the preamble. The second phase does most of the work. It reads the input file and produces the body of a TeX-file. The third phase adds the postamble and calls cweave and/or latex. In Ldoc-mode the third phase also removes all With the option mode=Mkman ext carries out only the first two phases, the resulting file can then be included in some larger document, and with the option Fman ext produces ASCII-output on standard output.

The production of ext's output is controlled by the so-called manual comments in the input file, see the chapter on Manual Production in the LEDA-book for further details. You must be familiar with that chapter in order to understand this program.

The details of the behavior of ext are directed by options. The options are processed in the initialization phase of ext.

⟨*ext.pl*⟩≡
  ⟨*required packages*⟩
  ⟨*command line processing and initialization*⟩
  ⟨*main program*⟩
  ⟨*generate postamble and call LaTeX or Cweb*⟩

  ⟨*subroutines*⟩

## 3.2 Command Line Processing and Initialization

⟨*command line processing and initialization*⟩≡
  ⟨*determine mode and set options to default values*⟩
  ⟨*determine actual options and derived variables*⟩
  ⟨*set up input and output files*⟩
  ⟨*generate preamble*⟩

The first argument tells ext whether it works for Lman, Ldoc, Fman or Mkmanual. If ext receives only one argument it prints usage information for its first argument, see section **??**. We have a variable for each option. The variables are initialized to their default values.

⟨*determine mode and set options to default values*⟩≡
```
$mode = $ARGV[0] && shift;
$filearg = $ARGV[0] && shift;

if ($mode eq "Mkman" || $mode eq "lextract")
{ $numbered = "yes"; }
else
{ $numbered  = "no";}
  # Usually manual pages are not numbered
$ack = "yes";
  # ext asks for acknowledgments when it is confused
$constref = "no";
  # const-& pairs are usually suppressed
$partypes = "no";
  # operators suppress argument types that are identical to Mname
$usesubscripts = "no";
  # usually we do not write indexed variables as subscripts
$size = 12;
  # default size is 12pt
$xdvi = "yes";
$warnings = "yes";
$informational = "yes";
  # default is to give warnings, to give progress report
$debugging = "no";
  # no debugging information
$indexing = "no";
$indexentry = "";
  # create no indexing output, indexentry is global index entry buffer
$includefile = "no";

$filter = "all";
  # usually we show everything

$pid = "";
  # we do not know the process id yet
$latexruns=1;
@substlist = ();
  # a list L1,R1,L2,R2,....  of all substitutions
$delman = "yes";
  # Ldoc usually removes manual comments
$outfile = "";
  # the default is that we determine the outfile
$dvioutfile = "";
  # only relevant for Mkdvi
$section = "section";
  # each manual page is a LaTeX section
$nextwarning = "yes";
  # default is to show the next warning
$justset = "no";
  # $nextwarning was not just set to no.
$noweaveoptions = "";
  # we start with no options.
if ($mode eq "Lman" || $mode eq "Mkman" || $mode eq "lextract" ||
```