`aldoc` — ALDOR documentation class for LaTeX
`extract` — ALDOR to LaTeX utility
`aldoc2html` — `aldoc` to HTML utility

Niklaus Mannhart Institute for Scientific Computing
ETH Zürich
E–Mail: `mannhart@inf.ethz.ch`
url: `http://www.inf.ethz.ch/~mannhart`

Version: 3.0 Aug 7, 2001

Abstract

`aldoc`, `extract` and `aldoc2html` are utilities that help programmers document their ALDOR programs in an easy way. The LaTeX class file `aldoc` provides useful macros with a unique manual page layout and optional with hyper-references for `xdvi` and `pdf`-files if supported. `extract` is a utility that converts documented ALDOR programs to LaTeX code. `aldoc2html` is a utility that converts LaTeX code created from `extract` to a LaTeX format which is well prepared for `latex2html` conversion.

This manual is divided into two parts. The first part, the user's guide, describes the macros in detail and shows how TeX code is produced from documented ALDOR programs by using the `extract` and `aldoc2html` utilities. The second part, only compiled by LaTeX when a special flag is turned off, documents the class file itself (only useful for class file hackers).

1 User's Guide

1.1 Introduction

In order to keep documentation synchronized with code development, it is recommended for ALDOR developers to document all the exported functions from a type in the actual ALDOR source file for that type. Three tools help make this easier:

- `aldoc` is a LaTeX class file allowing you to write reference manual pages in a format independent way, even with hyper references if supported by your previewer (`xdvi`, pdf previewer), and HTML.
- `extract` is a documentation extractor that creates `.tex` files from the documentation contained in the ALDOR sources.

- `aldoc2html` is a documentation converter that creates `.tex` files which are optimized for `latex2html` conversion.

We first explain the `aldoc` macros before describing the `extract` and `aldoc2html` utilities in detail.
Note: Please report all bugs — yes, even this class file has bugs — to the author. Comments and suggestions are also welcome!

1.2 The page layout

Manual pages created with `aldoc` contain a header, a body and a footer. The header contains the name of the type and the function that is currently described, the footer holds the page number.
The underlined header looks as follows: On the left side, the type's name is printed; on the right side the function's name. On even sides of two sided documents, the name of the type is placed on the right side and the the function's name on the left side. In both cases, the page number is centered in the footer. Figure 1 shows the manual page of the function `apply` belonging to type `BinaryTreeCategory`.

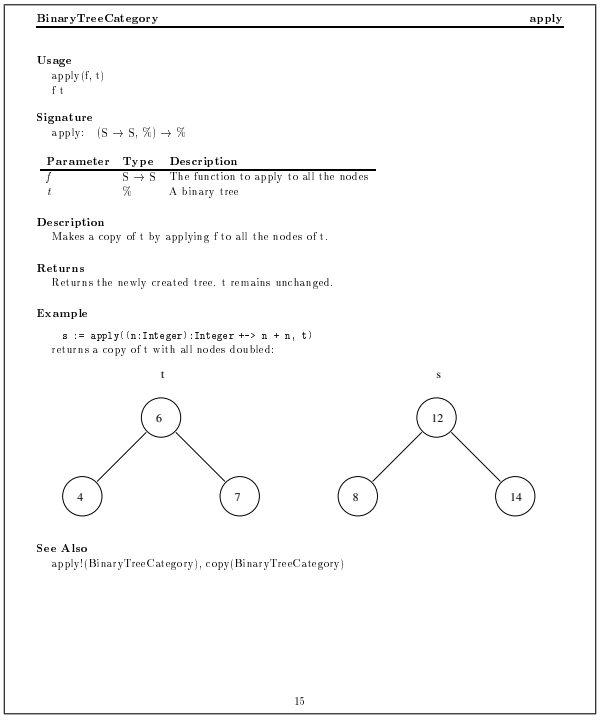


Figure 1: manual page layout of the function `apply` belonging to type `BinaryTreeCategory`

The `aldoc` class file doesn't specify a manual page structure, except that types are described before their functions. However, we recommend using the description structure described below so that all `aldoc` based manuals look similar. Types are like subsections and functions are subsubsections. Hence types are described first. If you don't follow this rule `aldoc` produces wrong index entries and some macros such as `\this` and `\name` return wrong results.
The description of a type has the following structure:

`\thistype[`*short form*`]{`*type name*`}`
[history section]
[usage]
[parameters]
[description]
[exports]
[other sections]

The *history* section contains information about the author of the type, respectively function, as well as the date and nature of any changes made to the type or function. This piece of information is useful for software maintenance, especially when several programmers are working on the same library. However, this section is not printed in the final document unless you turn on a special flag.[1]
Functions belonging to a type are described in a similar way.

`\alpage{`*function name*`}`
[history section]
[usage]
[signatures]
[parameters]
[description]
[return]
[example]
[see also]

1.3 `aldoc` macros in detail

`\thistype` `\thistype[`*shortform*`]{`*type*`}` starts the description of a type (called *type*). The macro starts a new page and puts the type's name into the index, the table of contents and the header. The optional argument *shortform* allows you to define a short form for the type's name.

```
\thistype{Quotient}
```

```
\thistype[BinTree]{BinaryTreeCategory}
```

[1] The current version of `aldoc` (v3.0) doesn't print the history section at all.

The first example starts the description of the type `Quotient`, the latter of the type `BinaryTreeCategory` with the short form `BinTree`.

`\this` `\shortthis` During the description of a type and its functions, the macros `\this` and `\shortthis` return the type's name and its abbreviated form. If no abbreviated form was given, then `\shortthis` returns the same result as `\this`.

```
\shortthis\ is the short form for \this.
```
BinTree is the short form for BinaryTreeCategory.

`\shortheader` Sometimes, type names are very long and use too much space in the header. Therefore, you might want to put a short form into the header.
The `\shortheader` command forces `aldoc` to put the short form of the type into the header. If no short form was given then `\this` is put into the header.

```
\thistype[BinTree]{BinaryTreeCategory}
\shortheader
```

`\alpage` The command `\alpage{`*function*`}` starts the description of a new function (called *function*). The macro clears the old page and puts the name of the function into the header and the index with the name of the type as a subentry. Note: the names of the functions are not added to the table of contents.

```
\alpage{apply}
```

`\name` Similar to the `\this` command, the macro `\name` returns the name of the functions that is currently described.

```
\name
```
apply

Usually there are two versions of the `aldoc` command. The first version, starting with a lowercase letter, is a LaTeX environment, the second version, starting with an uppercase letter, is a LaTeX command. Both versions produce the same output.

```
\begin{descr}
   Makes a copy of....
\end{descr}
```

```
\Descr{Makes a copy of....}
```

Of course, the second version is faster to write, but if a long text has to be printed, then the environment version is easier to debug and uses less memory during LaTeX compilation.

`\History` The history section is not used by `aldoc` yet. Nevertheless we encourage you to use the command already.
The `\History` command takes three arguments. Argument 1 contains the name of the author who created or changed the code. The date of the change is passed in argument 2 and comments are passed in argument 3.

```
\History{A. Einstein}{1912/3/13}{creation of the Relativity Theory}
\History{G. Lagaffe}{1995/2/28}{improved the theory}
\History{G. Lagaffe}{1995/3/1}{deleted the improvements}
```

`usage` `\Usage` The usage part describes the usage of a function or a type, i.e. how the function,

respectively the type is called by the user. Both commands print the title *Usage* in boldface and indent the left margin for the text that follows.

```
\Usage{apply(f,t)}
```

Usage

apply(f,t)

`params` `\Params` Parameters are put in a tabular environment that is created automatically. The tabular contains three columns called *Parameter*, *Type* and *Description* which are printed in boldface. The & symbol is used as the column separator.

```
\begin{params}
S & Order & The type of the node \\
t & \% & A binary tree \\
\end{params}
```

Parameter	**Type**	**Description**
S	Order	The type of the nodes
t	%	A binary tree

`descr` `\Descr` In the description section the type or function is described in detail. The title *Description* is printed in boldface and the left margin is indented for the text that follows.

```
\Descr{Makes a copy of t by applying f
to all the nodes of t.}
```

Description

Makes a copy of t by applying f to all the nodes of t.

`exports` The *exports* section is only used for types. The environment creates a three column tabular. The first column contains the name of the exported function, the second column contains the signature of the function, and the last column contains notes describing the exported function. The export description of the type `BinaryTreeCategory` might look as follows:

```
\begin{exports}
apply:  & (S $\to$ S, \%) $\to$ \% &
Apply a function to all the nodes \\
$\ldots$ & & \\
\end{exports}
```

Exports

apply:	(S → S, %) → %	Apply a function to all the nodes
...		

When types have arguments then the export list might depend on them, i.e. some functions are only exported if the argument has a specific category. Thus, the `exports` environment has an optional parameter that contains the condition under which the functions are exported. The parameter immediately follows the `\begin{exports}` environment. (`\begin{exports}[`*condition*`]`) The following example shows a conditional export list (see also `example.as` on page 12):

```
\begin{exports}[if R has
FiniteCharacteristic then]
\category{FiniteCharacteristic}\\
$\ldots$ & & \\
\end{exports}
```

Exports

if R has FiniteCharacteristic then
FiniteCharacteristic
...

For each conditional export list an `exports` environment has to be used.

`hyperlinks` Imagine, you can click on types and functions in order to move to related top-

ics? For example you look up a type or function in the index section, find the word and would like to click on it in order to jump to the corresponding page. Sounds like a web page. Well, formulas are not supported in html pages yet but we still can offer a similar way by using the LaTeX package `hyperlink`. It offers link support for `xdvi`, pdf previewer and `latex2html`. So with `xdvi` or Acrobat Reader[2] you are able to link pages. `aldoc` supports four commands for hyperlinks: `\alfunc`, `\alexp`, `\altype` and `\altarget`. We encourage to use this functions always, even if you don't plan to use hyperlinks. They can be turned on or off by a hyperlinks flag. See section 1.4.

`\alalias` With `aldoc2html` ALDORfiles, documented with `aldoc` can be converted to HTML. In order to support links in HTML documents `aldoc`provides the command \alalias{*type*}{*alias*}{*function*}. It creates a link to *type:alias* and prints the name *function*. \alalias is the basic command other commands relay on.

`\alfunc` \alfunc{*type*}{*function*} prints *function* and makes it a hyperlinks to the man page for *function* in the type *type*. The command is simply an abbreviation of \alalias{*type*}{*function*}{*function*}.

```
\begin{exports}
\alfunc{Quotient}{apply}: & (S $\to$ S,
\%) $\to$ \% & Apply a function to all
the nodes \\
$\ldots$ & & \\
\end{exports}
```

Exports

apply:	(S → S, %) → %	Apply a function to all the nodes
...			

Note: the printed output does not differ from the output of the above example (`\exports`). If `hyperref` is turned on then links for `xdvi`, pdf previewers and of course in HTML are put into it.

`\alexp` In the previous example you would have to write the type `Quotient` for each export field. You could also write \alfunc{\this}{*function*} instead of the type name itself. `aldoc` provides a shorthand for this command: \alexp{*function*} is the abbreviation of \alfunc{\this}{*function*}.

`\altype` `\altypes` Besides making links to functions, one likes to link to types as well. `aldoc` offers the command \altype{*type*} for that purpose. It prints the name *type* and makes it a hyperlinks to the manual page of that type, if defined. The command \altypes{*header*} puts the type name *header* to the table of contents. This is usefull when types are grouped under a name that should only appear in the table of contents.

```
\begin{exports}
$\ldots$ & & \\
\alexp{order}: & (R,Integer) $\to$
\altype{Partial} \altype{Integer} &
bounded ... \\
\end{exports}
```

Exports

...		
order:	(R,Integer) → Partial Integer	bounded ...
...		

[2] Acrobat Reader is a free pdf previewer from Adobe Systems Incorporated `http://www.adobe.com`

`\albuiltin` `\alexttype` Both macros are used for external references. \albuiltin{*type*} is a reference to the builtin type *type*, \alexttype{*library*}{*type*} is a reference of *type* in the library *library*.

```
\albuiltin{Integer}
\alexttype{Sumit}{Fraction}
```

`\altarget` \altarget`{`*name*`}` creates an alternative hyper-target name. The command is useful when you have to link a page under several names. Example: If `\alpage{map}` documents both `map` and `map!`, then add the line `\altarget{map!}` right after it. This way, if both `\alexp{map}` and `\alexp{map!}` are in the exports list, they will both point to that page.

`\category` Sometimes types export categories. Of course, you could write the category's name in the first column, but usually the names are very long and, thus, enlarge the first column of the tabular in such a way that the output looks ugly. To overcome this problem the macro `\category` is introduced. It takes a name as argument and prints it on one line, i.e. three columns are combined to one column. In fact, `\category` is an abbreviation for `\multicolumn{3}{l}`.

```
\begin{exports}
$\ldots$ & & \\
\category{\altype{CommutativeRing}} \\
$\ldots$ & & \\
\end{exports}
```

Exports

...
CommutativeRing
...

`alwhere` If names in export tabulars are too long, then one might whish to abbreviate the name and explain it in detail below the tabular. Therefore, `aldoc` contains the environment `alwhere` which usually follows after an *export* environment. It prints the name *where* and creates a three column tabular.

```
\begin{exports}
$\ldots$ & & \\
order:  & (R,Z) $\to$ Partial Z &
bounded order at the place \\
\end{exports}
\begin{alwhere}
Z & == & \altype{Integer} \\
$\ldots$ & & \\
\end{alwhere}
```

Exports

...
order: (R,Z) → Partial Z bounded order at place
...
where
Z == Integer
...

`signatures` `\Signatures` A signature describes the types of the arguments and the type of the return value. Obviously a signature only makes sense to functions and, hence, you will never use such a command in a type description. The title *Signatures* is printed in boldface and the left margin is indented. Both commands open a tabular environment with two columns (function's name, signature).

```
\begin{signatures}
apply:  & (S $\to$ S, %) $\to$ \% \\
\end{signatures}
```

Signatures

apply: (S → S, %) → %

`\Signature` Often only one signature has to be printed. Therefore, `aldoc` has a special signature command called `\Signature`. It prints the title *Signature* and indents the left margin. Unlike the commands `\Signatures` and `signatures`, `\Signature` prints the name of the function automatically. Hence, the macro has only two arguments, the parameter signature (argument 1) and the return type signature (argument 2). The signature is printed in the following way: *name of the function*: #1 → #2 where #1 is the first argument and #2 the second.

```
\signature{(S $\to$ S, \%)}{\%}
```

Signature

apply: (S → S, %) → %

`\alconstant` Sometimes a constant is defined only. The command \alconstant{*type*} prints the function name implicitly and puts the type of the constant to the right.

```
\alconstant{\albuiltin{SingleFloat}}
```

Signature

PI: SingleFloat

`retval` `\Retval` This describes the return value of a function. Again, the title *Returns* is printed in boldface and the left margin indented for the text that follows.

```
\begin{retval}
Returns the newly created tree.
t remains unchanged.
\end{retval}
```

Returns

Returns the newly created tree. t remains unchanged.

`alex` It's sometimes useful to add examples in a manual page. Therefore, `aldoc` supports the command `\alex`. The command prints the title *Example* in boldface and indent the left margin.

```
\begin{alex}
\begin{ttyout}
s:= apply((n:Integer):Integer +-> n+n, t)
\end{ttyout}
returns a copy of t with all nodes
doubled.
\end{alex}
```

Example

```
s:= apply((n:Integer):Integer +-> n+n, t)
```
returns a copy of t with all nodes doubled.

In example sections, you often use the verbatim environment for source code or output produced by the system. `aldoc` doesn't have its own verbatim macros, but it includes the verbatim package `ttyverb`[3] which has the following advantages over the original verbatim package included in LaTeX:

- font and size can by changed,
- the vertical space is narrower than the LaTeX version of verbatim
- tabs are converted to as many spaces as you like (default 8)

[3]Note: `ttyverb` is the name of the package and `ttyout` is the name of the environment defined in `ttyverb`.

Of course, you can use your own verbatim macros, but using `ttyverb` makes life easier when you distribute documented ALDOR code to other people. A detailed description of `ttyverb` is found in the `ttyverb.sty` style file.

`\alseealso` Often one would like to give references to other functions or types containing useful information. Therefore `aldoc` contains the `\alseealso` command that takes the references as argument.

```
\alseealso{\alexp{apply!}}
```

See Also

apply!

Note: \alseealso is also a place where one likes to have hyper references, ie. links to the mentioned functions.

`remarks` `\Remarks` Sometimes you want to add remarks to a manual page. It's especially useful when limitations in the use of a function or type or bugs that are not fixed yet. The commands `remarks`, `\Remarks` print the title *Remarks* in boldface and indent the left margin.

```
\Remarks{\this\ still crashes when
$\ldots$
```

Remarks

BinaryTree still crashes when ...

1.4 A word about Hyper-References

We strongly recommend to use the hyper-reference features provided in `aldoc`, ie. \alfunc, \alexp, \altype, \albuiltin, \alexttype and \altarget. Even when the `hyperref` package is not installed on your system or you do not want to use `latex2html` yet, you can produces printed output without any problems. By default hyper-reference is turned off and the hyper-reference commands produces the usual output. To turn on hyper-reference you have to set the *hyperref* option in the latex file.

```
\documentclass[hyperref]{aldoc}
```

will turn on hyper-reference

1.5 Putting everything together

In the previous sections we explained the `aldoc` macros in detail but said nothing about ALDOR files. Before you can create the TEX files using the `extract` utility, you must be sure that all documentation lines in your source files are bracketed between `#if` *marker* and `#endif` statements where *marker* can by any name. `extract` takes as input a ALDOR file and the *marker* and produces the LATEX file by discarding the code that is not between `#if` *marker* and `#endif` statements. In each ALDOR source file you must add the `#unassert` *marker* statement before the first `#if` *marker* statement starts.
The $\sum^{IT}$ library for which this class file originally was written uses `ASDOC` as its marker, ALDOR uses `ALDOC` as its marker. We encourage you to use the

same marker depending on the library you are working with, because whenever `sumit.as` or `aldor.as` is included, `ASDOC`, `ALDOC` respectively is already unasserted and hence, the `#unassert ASDOC` or `#unassert ALDOC` statement can be omitted.

```
#unassert ALDOC   % can be omitted if you include aldor.as
...
#if ALDOC
This is some documentation which will be ignored by
the A# compiler.
...
#endif
[A# code]
#if ALDOC
This is again documentation.
#endif
[A# code again]
```

In the documentation parts, i.e. between `#if` *marker* and `#endif` statements, you can use any `aldoc` or LaTeX command.

1.6 `extract` option

The general usage of `extract` is:

`extract` -m *marker* [-h] [-o *outputfile*] [-r] [-t] [-v] *sourcefile*[.as]

where the various options are:

-h Help: displays a short help text (causes all the other arguments to be ignored).

-m *marker* Defined marker in the *sourcefile*. Note: No default value is defined.

-o *outputfile* Names the output file, default is *sourcefile.tex*

-r Reverse operation: the output file contains all the lines which are not bracketed between `#if` *marker* and `#endif` statements.

-t Test: adds an appropriate LaTeX prologue and epilogue so that the output file can be run through LaTeX independently.

-v Verbose: shows progress (number of documentation blocks processed.)

1.7 The manual LaTeX file

When your Aldor programs are documented and `extract` has created the LaTeX files then you are ready to write the final "manual" file. This LaTeX file turns some `aldoc` features on or off and contains text you want to add to the manual pages. A "manual" file might look like this:

```
\documentclass[options]{aldoc}
packages
\pagestyle{fancyhdr}      % header creation turned on
\makeindex                % create index
\begin{document}
introduction, table of contents, table of figures and so on
\input{...}
other imports
\printindex               % print index
\end{document}
```

options can be any valid option under LaTeX such as twoside, 12pt, a4paper and so on. If you want to use the hyper-reference features from `aldoc` then you have to set the `hyperref` option! In the *packages* part you can include additional packages that are used in your document. However, the packages `fancyhdr`, `epsfig`, `supertabular`, `ttyverb` and `makeidx` don't have to be loaded because `aldoc` loads them automatically.
If you want to print your document using the standard `aldoc` header then you have to call pagestyle with fancyplain. If you omit this command then no header is printed. An index is created by LaTeX when `makeindex` is called in the preamble. If you comment out the command no index is generated. A detailed description of the `makeindex` and the `fancyhdr` packages is found in [1].
One last thing you have to keep in mind is that the command `\thistype` is the same as a subsection but without a subsection number. Of course, you can create sections and subsections, but the latter does not get a subsection number.

1.8 Aldoc2HTML

Version 3 of `aldoc` has better html support. The utility `aldoc2html` converts `.tex` files created with the `extract` utility to new `.tex` files which are optimized for the `latex2html` utility.
The general usage of `aldoc2html` is:

`aldoc2html` [-h] [-o *outputfile*] [-x] [-v] *sourcefile*[.tex]

where the various options are:

- -h Help: displays a short help text (causes all the other arguments to be ignored).
- -o *outputfile* Names the output file, default is standard output.
- -x Input: expands only \input statements.
- -v Verbose: shows progress.

In order to create a HTML version of the documentation do the following:

1. Create ALDOR source files documentend with the `aldoc` style file.
2. Extract the documentation `.tex` by using the `extract` utility.
3. Create HTML optimized LaTeX files by using the `aldoc2html` utility.
4. Run `latex2html` to create the HTML files.

1.9 Example

In this section we create the manual pages of the `example.as` file. The described type is called `Quotient`. The following code fragment shows the documented ALDOR file. Note: `example.as` is shipped with this package but it cannot be compiled with ALDOR because it contains a category not provided by the standard library. However, it can be used as a template for your own documentation (`extract` can create the LaTeX file, which can then be processed).

```
#include "aldor.as"
#unassert ALDOC

#if ALDOC

\thistype{Quotient}
\History{G.~Lagaffe}{1/12/94}{created}

\Usage{import from \this~R}

\Params{ \emph{R} & \altype{IntegralDomain}
        & an integral domain\\
}

\Descr{\this~R forms the quotient field of the
integral domain \emph{R}}

\begin{exports}
\category{\altype{Field}}\\
\category{\altype{DifferentialExtension} R}\\
$/$: & (R,R) $\to$ \% & take the quotient of two
ring elements\\
\alexp{coerce}: & R $\to$ \% & coercion from R to \this\\
\alexp{denominator}: & \% $\to$ R & get the denominator of a quotient\\
\alexp{numerator}: & \% $\to$ R & get the numerator of a quotient\\
\end{exports}
\begin{exports}[if R has \altype{FiniteCharacteristic} then]
\category{\altype{FiniteCharacteristic}}\\
\end{exports}
\begin{exports}[if R has \altype{GcdDomain} then]
\alexp{normalize}: & \% $\to$ \% & normalize a quotient\\
```

```
\end{exports}
#endif

Quotient(R: IntegralDomain):
 Join(Field, DifferentialExtension R) with {
     /:      (R, R) -> %;

#if ALDOC
\alpage{/}

\Usage{n~\name~d}
\Signature{(R,R)}{\%}
\Params{
\emph{n} & R & An element of the ring.\\
\emph{d} & R & An element of the ring.\\
}
\Retval{Returns the quotient \emph{n} over \emph{d}.}
#endif

     if R has FiniteCharacteristic then
         FiniteCharacteristic;
     coerce:     R -> %;

#if ALDOC
\alpage{coerce}

\Usage{\name~x}
\Signature{R}{\%}
\Params{ \emph{x} & R & An element of the ring\\ }
\Retval{Returns the quotient with numerator \emph{x} and
        denominator 1.
}
#endif

     denominator:    % -> R;

#if ALDOC
\alpage{denominator}

\Usage{\name~x}
\Signature{\%}{R}
\Params{ \emph{x} & \% & A quotient\\ }
\Retval{Returns the denominator of a quotient.}
\alseealso{\alexp{numerator}
#endif
```

```
     if R has GcdDomain then
         normalize:  % -> %;

#if ALDOC
\alpage{normalize}
\Usage{\name~x}
\Signature{\%}{\%}
\Params{ \emph{x} & \% & An quotient\\ }
\Descr{Normalize $x$ by eliminating common divisors of
       the numerator and denominator.
}
#endif

     numerator:  % -> R;

#if ALDOC
\alpage{numerator}

\Usage{\name~x}
\Signature{\%}{R}
\Params{ \emph{x} & \% & A quotient\\ }
\Retval{Returns the numerator of a quotient.}
\alseealso{\alexp{denominator}}
#endif
} == add { implementation }
```

First we produce the corresponding LaTeX file by using the `extract` utility.

```
extract -m ALDOC example.as
```

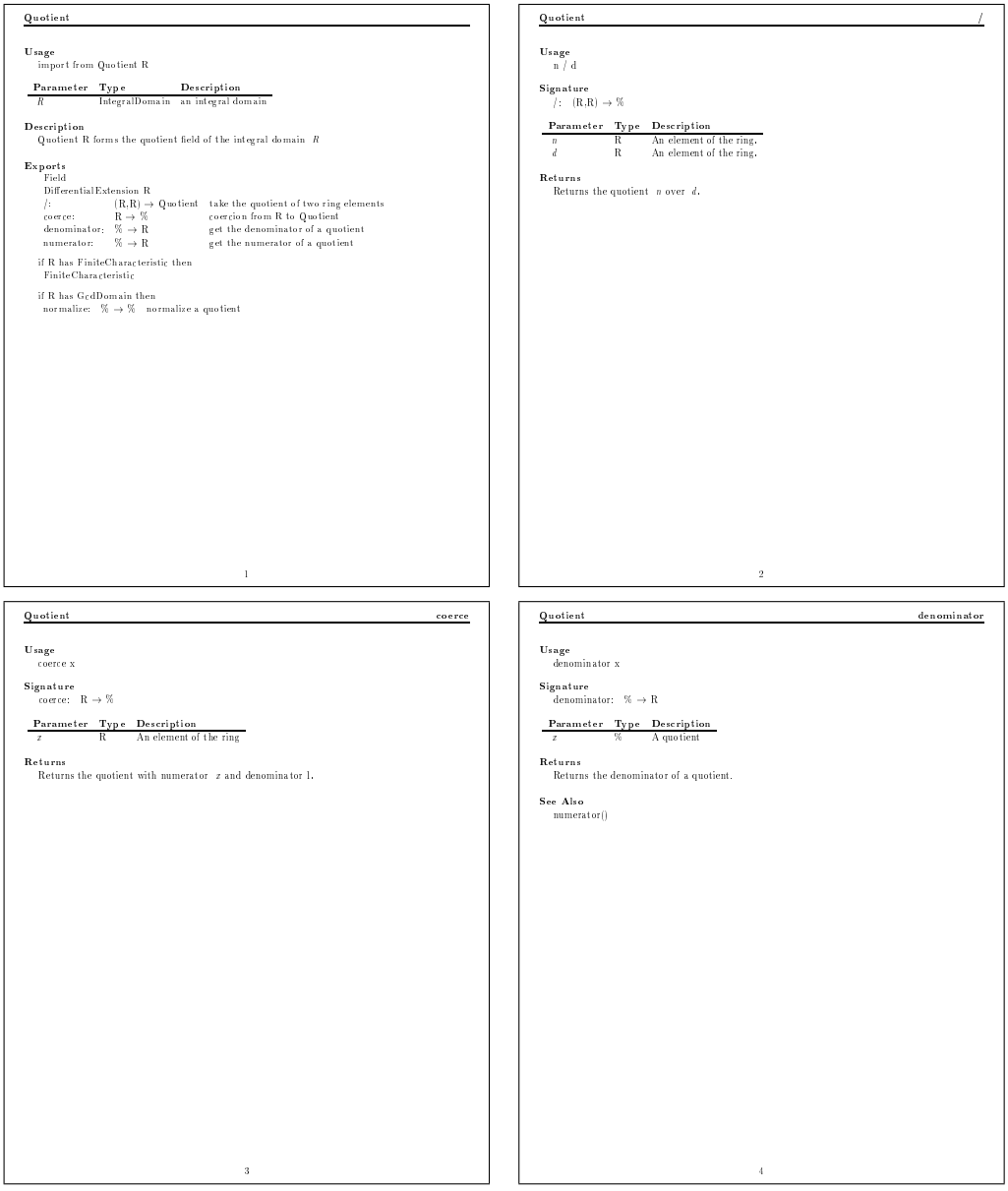
The extract utility creates the file `example.tex`. Now we have to create the "manual" LaTeX file that looks as follows:

```
% manual.tex
\documentclass[12pt,hyperref]{aldoc}
\pagestyle{fancyplain}
\makeindex
\begin{document}
\input{example.tex}
\printindex
\end{document}
```

In the last step we have to compile `manual.tex`, create the index with `makeindex` and compile the manual again.

```
latex manual.tex
makeindex manual.idx
latex manual.tex
```

Figure 2 and figure 3 show the manual pages produced.
Run `aldoc2html -o manual-html.tex" manual.tex`. Next, run `latex2html manual-html.tex` which creates the HTML files.



Quotient

Usage
import from Quotient R

Parameter	Type	Description
R	IntegralDomain	an integral domain

Description
Quotient R forms the quotient field of the integral domain *R*

Exports
Field
DifferentialExtension R

/:	(R,R) → Quotient	take the quotient of two ring elements
coerce:	R → %	coercion from R to Quotient
denominator:	% → R	get the denominator of a quotient
numerator:	% → R	get the numerator of a quotient

if R has FiniteCharacteristic then
FiniteCharacteristic

if R has GcdDomain then
normalize: % → % normalize a quotient

1

Quotient /

Usage
n / d

Signature
/: (R,R) → %

Parameter	Type	Description
n	R	An element of the ring.
d	R	An element of the ring.

Returns
Returns the quotient *n* over *d*.

2

Quotient coerce

Usage
coerce x

Signature
coerce: R → %

Parameter	Type	Description
x	R	An element of the ring

Returns
Returns the quotient with numerator *x* and denominator 1.

3

Quotient denominator

Usage
denominator x

Signature
denominator: % → R

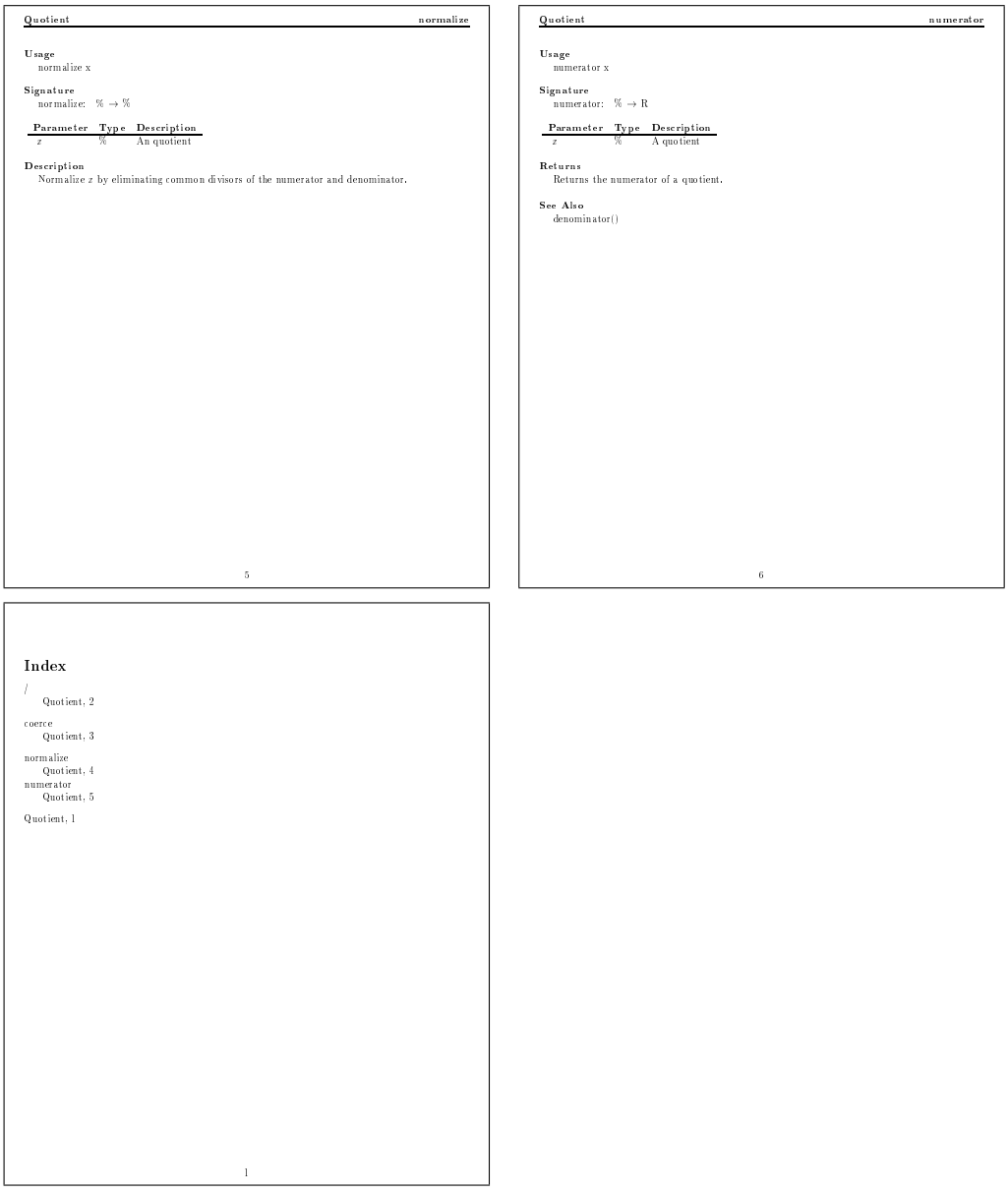
Parameter	Type	Description
x	%	A quotient

Returns
Returns the denominator of a quotient.

See Also
numerator()

4

Figure 2: Manual pages from "manual.tex".



Quotient normalize

Usage
normalize x

Signature
normalize: % → %

Parameter	Type	Description
z	%	An quotient

Description
Normalize *z* by eliminating common divisors of the numerator and denominator.

5

Quotient numerator

Usage
numerator x

Signature
numerator: % → R

Parameter	Type	Description
z	%	A quotient

Returns
Returns the numerator of a quotient.

See Also
denominator()

6

Index

1

Figure 3: Manual pages from "manual.tex". (continued)

2 Reference

This section describes the macros alphabetically. `#1`, `#2` refer to argument 1, argument 2 respectively.

`\alalias` \alalias{*type*{{*alias*}{*function*} creates a link to *type:alias* and prints the name *function*. \alalias is the basic command where other commands rely on.

`\albuiltin` `\albuiltin{#1}` make a reference to the builtin type `#1`. Not implemented in the current version of `aldoc`.

```
\albuiltin{SingleFloat}
```

`alex` Starts the *example* part of the manual page. The title *Example* is printed in boldface and the left margin is indented for the text that follows.

```
\begin{alex}
\begin{ttyout}
s:= apply((n:Integer):Integer +-> n+n, t)
\end{ttyout}
returns a copy of t with all nodes
doubled.
\end{alex}
```

Example

```
s:= apply((n:Integer):Integer +-> n+n, t)
```
returns a copy of t with all nodes doubled.

`\alexp` `\alexp{#1}` is shorthand for `\alfunc{\this}{#1}`.

```
\alexp{start!}
```

`\alexttype` `\alexttype{#1}{#2}` makes a link to type `#2` in library `#2`. Not implemented in the current version of `aldoc`.

```
\alexttype{Sumit}{Fraction}
```

`\alfunc` `\alfunc{#1}{#2}` prints function `#2` and makes it a hyperlink to the aspage for function `#2` in the type `#1`. It is a shorthand for `\alalias{#1}{#1}{#2}`

```
\alfunc{Timer}{start!}
```

`\alpage` `\alpage{#1}` Starts a new manual page, i.e. the description of a new function. The function's name `#1` is stored in an internal variable (see also `\name`) and the old page is cleared.

```
\alpage{apply}
```

`alwhere` `\alwhere` prints *where* and opens a three column tabular. The environment is usually used after an export environment.

```
\begin{exports}
$\ldots$ & & \\
order:  & (R,Z) $\to$ \altype{Partial} Z
& bounded order at the place \\
\end{exports}
\begin{alwhere}
Z & == & \altype{Integer} \\
$\ldots$ & & \\
\end{alwhere}
```

Exports

...			
order:	(R,Z) → Partial Z		bounded order at place
...			

where

Z	==	Integer
...		

`\altarget` `\altarget{#1}` creates an alternative hyper-target name. The command is useful when you want to link a page under serveral names.
Example: If `\alpage{map}` documents both `map` and `map!`, then add the line `\altarget{map!}` right after it. This way, if both `\alexp{map}` and `\alexp{map!}` are in the exports list, they will both point to that page.

`\altype` `\altype{#1}` prints type `#1` and makes it a hyperlink to the main page of that type, if defined.

```
\altype{Timer}
```

`\altypes` `\altypes{#1}` puts `#1` in the table of contents. This is useful when types are grouped and a title should appear in the table of contents.

`\category` `\category{#1}` Combines 3 columns of a tabular. The command is an abbreviation for `\multicolumn{3}{l}` and can be used in any tabular environment but is mainly used in *exports* sections.

```
\begin{exports}
$\ldots$
\category{\altype{CommutativeRing}} \\
$\ldots$
\end{exports}
```

Exports

...	&	& \\
CommutativeRing		
...	&	& \\

`\Descr` `\Descr{#1}` Short form for the `descr` environment.

```
\Descr{Makes a copy of t by applying f
to all the nodes of t.}
```

Description

Makes a copy of t by applying f to all the nodes of t.

`descr` Starts the *description* part of the manual page. The title *Description* is printed in boldface and the left margin is indented for the text that follows.

```
\Descr{Makes a copy of t by applying f
to all the nodes of t.}
```

Description

Makes a copy of t by applying f to all the nodes of t.

`\Errors` `\Errors{#1}` Short form for the `errors` environment.

```
\Errors{none}
```

Errors

none

`errors` Starts the *errors* part of the manual page. The title *Errors* is printed in boldface and the left margin is indented for the text that follows.

```
\begin{errors}
none
\end{errors}
```

Errors

none

`exports` `\begin{exports}[#1]` The environment opens a tabular environment with three column called export name, signature and description. Optionally, a condition under which the functions are exported can be given.

```
\begin{exports}
apply:  & (S $\to$ S, \%) $\to$ \% &
Apply a function to all the nodes \\
$\ldots$ & & \\
\end{exports}
```

Exports

apply:	(S → S, %) → %	Apply a function to all the nodes
...		

```
\begin{exports}[if R has
\altype{FiniteCharacteristic} then]
\category{\alstype{FiniteCharacterisic}}\\
$\ldots$ & & \\
\end{exports}
```

Exports

if R has FiniteCharacteristic then

FiniteCharacteristic

...

`\History` `\History{#1}{#2}{#3}` Stores history information belonging to the type or function that is currently described. The author's name is put in `#1`, the date of the change in `#2` and any comment in `#3`.[4]

```
\History{A. Einstein}{1912/3/13}{creation of the Relativity Theory}
```

`\name` Returns the name of the described function, i.e. the name that was stored in the `\alpage` macro. (see also `\alpage`)

```
\name
```

apply

`\Params` `\Params{#1}` Short form for the `params` environment.

```
\Params{S & Order & The type of the
nodes}
```

Parameter	Type	Description
S	Order	The type of the nodes

`params` Starts the *Parameter* part of the manual page. The title *Parameter* is printed in boldface and the left margin is indented for the text that follows. Next, a tabular environment is opened with the columns parameter, type and description. This column header is printed in boldface.

```
\begin{params}
S & Order & The type of the node \\
t & \% & A binary tree \\
\end{params}
```

Parameter	Type	Description
S	Order	The type of the nodes
t	%	A binary tree

`\Remarks` `\Remarks{#1}` Short form for the `remarks` environment. (see also `remarks`)

```
\Remarks{\this\ still crashes when
$\ldots$}
```

Remarks

BinaryTree still crashes when

...

`remarks` Starts the *remarks* part of the manual page. The title *Remarks* is printed in boldface and the left margin is indented for the text that follows.

[4] The current version of `aldoc` doesn't process this information yet.

```
\begin{remarks}
\this\ still crashes when $\ldots$
\end{remarks}
```

Remarks

BinaryTree still crashes when ...

`\Retval` `\Retval{#1}` Short form for the `retval` environment. (see also `retval`)

```
\Retval{Returns the newly created tree.
t remains unchanged.}
```

Returns

Returns the newly created tree. t remains unchanged.

`retval` Starts the *returns* part of the manual page. The title *Returns* is printed in boldface and the left margin is indented for the text that follows.

```
\begin{retval}
Returns the newly created tree.
t remains unchanged.
\end{retval}
```

Returns

Returns the newly created tree. t remains unchanged.

`\alseealso` `\alseealso{#1}` The title *See Also* is printed in boldface and the left margin is indented for the text that follows.

```
\alseealso{\alexp{apply}}
```

See Also

apply!

`\shortheader` Puts the information in `\shortthis` instead of `\this` into the header. If no short form was given, then `\this` is put into the header. The command must follow immediately after `\thistype`.

```
\thistype[BinTree]{BinaryTreeCategory}
\shortheader
```

`\shortthis` Returns the short form of the type name. `\shortthis` is set by `\thistype` command. (see als `\thistype`)

```
\shortthis
```

BinTree

`\Signature` `\Signature{#1}{#2}` This macro is used if only one signature is defined. The title *Signature* is printed in boldface and the left margin is indented for the signature that is written in the following way: `\name: #1 → #2`

```
\Signature{(S $\to$ S, %)}{%}
```

Signature

apply: (S → S, %) → %

`\Signatures` `\Signatures{#1}` Short form for the `signatures` environment. (see also `signatures`) Note: if only one signature exists, then use the short form `\Signature`. (see also `\signature`)

```
\Signatures{apply:  & (S $\to$ S, %)
$\to$ \% \\}
```

Signatures

apply: (S → S, %) → %

`signatures` Starts the *signatures* part of the manual page. The title *Signatures* is printed

in boldface and the left margin is indented for the signatures that follow. Additionally, a tabular environment is opened with two columns. (name, signature)

```
\begin{signatures}
apply:  & (S $\to$ S, %) $\to$ % \\
\end{signatures}
```

Signatures

apply: (S → S, %) → %

`\this` Returns the name of the type that is described. `\this` is set by `\thistype` command. (see also `\thistype`)

```
\this
```

BinaryTreeCategory

`\thistype` `\thistype[#1]{#2}` Starts the description of type `#2`. The name of the type (`#2`) is stored in `\this`. Optional a short form of the name (`#1`) that is stored in `\shortthis` can be given. `\thistype` starts a new page and puts the type name into the header, the table of contents and the index. Note: if `\shortheader` is following immediately after a `\thistype` command, then the short form (`\shortthis`) is put into the header.

```
\thistype[BinTree]{BinaryTreeCategory}
```

`\Usage` `\Usage{#1}` Short form for the `usage` environment. (see also `usage`)

```
\Usage{apply(f,t)}
```

Usage

apply(f,t)

`usage` Starts the *usage* part of the manual page. The title *Usage* is printed in boldface and the left margin is indented for the text that follows.

```
\begin{usage}
apply(f,t)
\end{usage}
```

Usage

apply(f,t)

3 How to print this manual

There are two ways to print this document. The first one is to compile this file with LaTeX i.e. `latex aldoc.dtx`, the second one is by compiling a new LaTeX file that looks as follows:

```
\documentstyle[]{article}
\usepackage{doc}              % include doc package
\usepackage{epsfig}           % include epsfig package
\EnableCrossrefs              % full index
\CodelineIndex                % by line numbers
\RecordChanges                % make change history
\OnlyDescription              % no code documentation
\setlength{\parindent}{0pt}  % no indents
\begin{document}
  \DocInput{aldoc.dtx} \PrintIndex \PrintChanges
\end{document}
```

Remove the command `\OnlyDescription` if you want to print the `aldoc` source code. Note: the documentation of the code is only needed if you are going to change the class file.

In both compilation methods you have to run the following commands in order to get the `aldoc` manual:

1. `latex aldoc.dtx`
2. `makeindex -s gind.ist aldoc`
3. `makeindex -s gglo.ist -o aldoc.gls aldoc.glo`
4. `latex aldoc.dtx`

References

[1] Michael Gossens, Frank Mittelbach, Alexander Samarin, *The LaTeX Companion*, Addison–Wesley, 2nd printing 1994, ISBN 0-201-54199-8.

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Change History