GAP

Release 4.5
October 2011

The gapmacro.tex Manual Format

The GAP Group

https://www.gap-system.org

Contents

1 The gapmacro.tex Manual Format 3

1.1 The Main File . . . 3

1.2 Additional Typesetting Options . . . 6

1.3 Structuring the text: Chapters and Sections . . . 7

1.4 Suppressing Indexing and Labelling of a Section and Resolving Label Clashes . . . 8

1.5 Labels and References . . . 8

1.6 TeX Macros . . . 9

1.7 TeX Macros for Domains . . . 12

1.8 Examples, Lists, and Verbatim . . . 13

1.9 Tables, Displayed Mathematics and Mathematics Alignments . . . 16

1.10 Testing the Examples . . . 16

1.11 Usage of the Percent Symbol . . . 17

1.12 Catering for Plain Text and HTML Formats . . . 17

1.13 Umlauts . . . 18

1.14 Producing a Manual . . . 19

1.15 Using buildman.pe . . . 20

1 The gapmacro.tex Manual Format

This document describes a restricted TEX format, which is defined by the of macros in the file

```
GAPPATH/doc/gapmacro.tex
```

and how to create the final documents (which can be printed or used by GAP's online help) from it. Some GAP 4 package documentation is written in this format. Up to version 4.4, the same was true for the main GAP manuals.

See 1.6 and 1.8 for details on the restricted set of available TEX commands.

The first sections 1.1 and 1.3 describe the general layout of the files in case you need to write your own package documentation.

If you are planning to write new documentation for a GAP package, one alternative to using the format described in this document is to use the GAPDoc package, see Chapter "Introduction and Example" in the GAPDoc manual, for example type

```
gap> ?GAPDoc:chapters
```

in GAP's online help for a table of contents, or (if it is not available in your installation) see:

```
http://www.math.rwth-aachen.de/~Frank.Luebeck/GAPDoc/
```

If you want to use yet another document format you must provide certain information to the interface of GAP's online help. This is described in Chapter "Interface to the GAP Help System" of the GAP Reference Manual.

1.1 The Main File

The main TEX file is called `manual.tex`. This file should contain the following commands:

```
\input ../gapmacro
 ... (gapmacro.tex options, see "Additional Typesetting Options" below)
\Package{package-name}
\BeginningOfBook{name-of-book}
  \UseReferences{book1}
  ...
  \UseReferences{bookn}
  \TitlePage{title}
  \Colophon{text}
  \TableOfContents
  \FrontMatter
    \immediate\write\citeout{\bs bibdata{mybibliography}}
    \Input{file1}
    ...
    \Input{filen}
  \Chapters
    \Input{file1}
    ...
    \Input{filen}
```

```
\Appendices
  \Input{file1}
  ...
  \Input{filen}
  \Bibliography
  \Index
\EndOfBook
```

Now we describe what these commands do:

`\input` *path*`/gapmacro.tex`
: inputs the GAP "style" and macros file `gapmacro.tex`. If you are writing a GAP package either copy this file or use a relative path. The former method will always work but requires you to keep the file consistent with the system while the latter forces users to change the `manual.tex` file if they are installing a package in a private location. See also Section "GAP Root Directory" in the GAP Reference Manual.

`\Package{`*package-name*`}`
: defines a macro `\`*package-name* so that when you type `{\`*package-name*`}` (please include the curly braces) the text *package-name* is typeset in the right way for GAP packages, e.g. if you are writing a package MyPackage then you should include the line

```
\Package{MyPackage}
```

in your `manual.tex` file and then in your chapter files use `{\MyPackage}` when you refer to MyPackage by name. There is also the command `\package{`*pkg*`}` when you wish to refer to other GAP packages; don't confuse the two i.e. `\Package{`*package-name*`}` defines a macro `\`*package-name* but produces no text, and `\package{`*pkg*`}` produces *pkg* set in the font that is right for GAP packages.

`\BeginningOfBook{`*name-of-book*`}`
: starts the book *name-of-book*. It is used for cross-references, see 1.5. If you are writing a GAP package use the name of your package here.

`\UseReferences{`*booki*`}`, `\UseGapDocReferences{`*booki*`}`
: If your manual cross-refers to another manual, `\UseReferences` can be used to load the labels of the other books in case cross-references occur. *booki* should be the path of the directory containing the book whose references you want to load. However, as said above this requires changes to the `manual.tex` file if the package is not installed in the standard location. `\UseGapDocReferences` can also be used to load GAPDoc style references, but this function exists only for backward compatibility.
Just ensure you get the path to the other manual's directory correct **relative** to the directory in which your manual resides.
If your `manual.tex` file lives in `pkg/qwer/doc` and you want to use references to the GAP Tutorial use

```
\UseGapDocReferences{../../../doc/tut}
```

`\TitlePage`
: produces a page containing the **title**. Please see the example.

`\Colophon`
: `\Colophon` produces a page following the title that can be used for more explicit author information, acknowledgements, dedications or whatsoever.

`\TableOfContents`
: produces a table of contents in double-column format. For short manuals, the double-column format may be inappropriate; in this case, use `\OneColumnTableOfContents` instead.

`\FrontMatter`
: starts the front matter chapters such as a copyright notice or a preface.
The line

```
\immediate\write\citeout{\bs bibdata{mybibliography}}
```

is for users of BibTeX. It will use the file *mybibliography*`.bib` to fetch bibliography information.

`\Chapters`
: starts the chapters of the manual, which are included via `\Input`. `\Input{`*filei*`}` inputs the file *filei*`.tex`, i.e. *filei* should be the name of the file **without** the `.tex` extension. For the chapter format, see Section 1.3.

`\Appendices`
: starts the appendices, i.e. it modifies the `\Chapter` command to use uppercase letters to number chapters.

`\Bibliography`
: produces a bibliography, i.e. it reads and typesets the `manual.bbl` file produced by BibTeX.

`\Index`
: produces an index, i.e. it reads and typesets the `manual.ind` file produced by the external `manualindex` program.

`\EndOfBook`
: Finally `\EndOfBook` closes the book.

Example

Assume you have a GAP package `qwert` with two chapters `Qwert` and `Extending Qwert`, a copyright notice, and a preface, then your `manual.tex` would basically look like:

```
\input ../../../doc/gapmacro         % The right path from pkg/qwert/doc
\Package{Qwert}                      % Defines macro {\Qwert}
\BeginningOfBook{qwert}
  \TitlePage{
    \centerline{\titlefont Qwert}\medskip           % Package name
    \centerline{\titlefont ---}\medskip
    \centerline{\titlefont A GAP4 Package}\bigskip\bigskip
    \centerline{\secfont Version 1.0}\medskip
    % If the package interfaces with an external program ...
    \centerline{\secfont Based on qwert Standalone Version 3.14}\vfill
    \centerline{\secfont by}\vfill
    \centerline{\secfont Q. Mustermensch}\medskip   % Author
    \centerline{Department of Mathematics}\medskip  % Affiliation
    \centerline{University of Erewhon}\medskip
    \centerline{\secfont email: qmuster@erewhon.uxyz.edu.ut} % Email address
    \vfill
    \centerline{\secfont{\Month} \Year}
  }
  \TableOfContents
  \FrontMatter
    \Input{copyright}
    \Input{preface}
  \Chapters
    \Input{qwert}
    \Input{extend}
  \Appendices
    \Index
\EndOfBook
```

Occasionally there will be the need for additional commands over and above those shown above. The ones described below should be the **only** exceptions.

– There may be other packages that are referred to a lot, so that it's worthwhile to add more `\Package` commands. (There's nothing special about `\Package`, you can use it to define macros for other packages besides the package being documented.)

– Besides the macros `{\Month}` and `{\Year}`, which typeset the current month (as an English word) and the year (all four digits), respectively, there are also `{\Day}` and `{\Today}` which are mainly intended for drafts. `{\Day}` typesets the day of the month as a number and `{\Today}` is equivalent to: `{\Day} {\Month} {\Year}`.

– Sometimes one desires a chapter to be unnumbered in the TEX-produced manuals, e.g. the Tutorial manual has GAP's Copyright Notice as an unnumbered chapter. To achieve this one inputs the file containing the chapter via TEX's `\input` command rather than `\Input`. However, neither the on-line help browser nor the HTML converter "sees" such chapters. Thus if it is desired that the on-line help browser and the HTML manuals should also have such chapters, they must be "input" again via the `\PseudoInput` command (not necessarily in the same manual).

– For chapters that should only appear via the on-line help browser or in the HTML manuals, one may use the `\PseudoInput` command. Any `\PseudoInput` commands should come **after** all `\Input` commands; failure to do this will result in different numbering of `\Input` chapters for TEX-produced and HTML manuals. The syntax of this command is as follows:

`\PseudoInput{`*filename*`}{`*six-entry*`}{`*chaptername*`}`

where *filename* is the name of the file containing the chapter without the `.tex` extension, as for the `\Input` command, *six-entry* is the section-index-entry for the chapter (written to the `manual.six` file) and *chaptername* is the **actual** argument of the `\Chapter` command that appears at the beginning of *filename*`.tex`. The argument *six-entry* enables the on-line text browser to reference the chapter by a name other than *chaptername*. Thus a copyright chapter for the book with name *name-of-book* might have *chaptername* "`Copyright Notice`" but *six-entry* "`Copyright`", which would enable one to access the chapter "`Copyright Notice`" via ?*name-of-book*`:copyright` via the on-line browser. The HTML converter adds an index entry for both *six-entry* and *chaptername*.

Note

Usage of the commands `\input` and `\PseudoInput` in the way described above will necessitate special treatment of references to such chapters. For such purposes, there is a special variant of the `%display` environment (see 1.12), e.g. a copyright notice appearing via `\input` at the beginning of a TEX-produced manual and appearing in the non-TEX manuals – the on-line help browser or HTML manual – via a `\PseudoInput` command as described above, may be referenced via

```
%display{tex}
See the copyright notice at the beginning of this book.
%display{nontex}
%See "Copyright".
%enddisplay
```

1.2 Additional Typesetting Options

There are a number of additional options which you can activate/ deactivate by adding the following TEX just after the line

```
\input ../gapmacro.
```

Of course, you need not set options which are marked as default.

`\usepsfonts`
: (default) use the standard Postscript fonts for typesetting

`\usecmfonts`
: use the TeX standard (Computer Modern) fonts (this was the behaviour until GAP 4.4).

`\addlinks`
: (default) This inserts pdf links within the document, so that you can click on a reference or citation. This will only work with pdftex, otherwise this has no effect.

`\nolinks`
: This switches off additional pdf links within the document (this was the behaviour until GAP 4.4.

`\citebooksfalse`
: (default) References to external books will just print the chapter/section/subsection number(s).

`\citebookstrue`
: References to external books will be printed as `GAP Reference Manual, 2.7.12` instead of just `2.7.12`, similarly for the GAP tutorial. If you cite other books, you have to define for each book `XXX` a macro `\xxxManual` which expands to the text which you want to be inserted before the number for package `XXX`. (Note that the `xxx` in `\xxxManual` must always be lower case, regardless of the actual package name `XXX`.) If you want to cite from the manual of a package "OtherPackage", then you should add the line

  ```
  \gdef\otherpackageManual{The OtherPackage Manual}
  ```

  at the beginning of your main TEX input file.

`\biblitemfalse`
: (default, same as in GAP 4.4) In the bibliography, print abbreviations for papers right-aligned

`\biblitemtrue`
: Print abbreviations left-aligned in the bibliography. d Depending on the abbreviations you use, you may need more indentation. You may set `\bibindent` to another value (the present default is 3 pc) if you get overfull `\hboxes`. (This effect also exists if you use `\bibitemsfalse` but there is not very obvious because long abbreviations will protrude into the left margin only.)

`\casesensitivefalse`
: (default) labels are case insensitive

`\casesensitivetrue`
: makes labels case sensitive (this is still experimental and not currently supported by the html converter)

1.3 Structuring the text: Chapters and Sections

The contents of each chapter must be in its **own** `.tex` file. The command `\Chapter{`*chaptername*`}` starts a new chapter named *chaptername*; it should constitute the first non-comment (and non-blank) line of the file containing a chapter. A chapter begins with an introduction to the chapter and is followed by sections created with the `\Section{`*secname*`}` command. The strings *chaptername* and *secname* are automatically available as references (see Section 1.5).

There must be **no further commands** on the same line as the `\Chapter` or `\Section` line, and there **must** be an empty line after a `\Chapter` or `\Section` command. This means that `\index` commands referring to the chapter or section can be placed only after this empty line.

Finally, the HTML converter requires that each `\Section` line is preceded by a line starting with at least 16 percentage signs (conventionally, one actually types a full line of percentage signs). The HTML converter stops converting a section whenever it hits such a line; therefore do not add lines starting with 16 or more % signs which are **not** just before a `\Section` command. Failure to include the line of percentage signs before a `\Section` line will cause the converter to crash, due to the discovery of what it sees as two `\Section` commands within the one section.

1.4 Suppressing Indexing and Labelling of a Section and Resolving Label Clashes

Sometimes one does not wish a section to be indexed. To suppress the indexing of a section, simply add the macro `\null` after the `\Section` command, e.g.

`\Section{`*section-name*`}\null`

and then *section-name* will still generate a label (so that you can still refer to it via `Section~"`*section-name*`"`), but *section-name* will not appear in the index.

Occasionally, one has a dedicated section for the description of a single function. If the label generated for the section coincides with the label for a subsection (generated by a `\>` command) a multiply defined label results. In these cases, one would generally rather that the section did not generate a label or an index entry. To suppress the generation of both the label and the index entry of such a section, simply add the macro `\nolabel` immediately after the `\Section` command, e.g. for a section dedicated to the function *func*:

`\Section{`*func*`}\nolabel`

Note: Labels are generated by converting to lowercase and removing whitespace. So coincidences can occur when you might not have expected it. An alternative to index suppression to resolve label clashes is to include a sub-label for the function in the `\>` command (see Section 1.6).

1.5 Labels and References

Each `\Chapter`, `\Section` and `\>` command generates a (short) label *label*, which is extended by *name-of-book* (the argument of `\BeginningOfBook` mentioned earlier in Section 1.1), to create a "long label" *long-label*, and emitted to a file `manual.lab`. The construction of *long-label* is *name-of-book*`:`*label*, where the *label* generated by either of the commands `\Chapter` or `\Section` is just its *chaptername* or *secname* argument. For `\>`, there are a few cases to consider, and we'll consider them in Section 1.6, where we meet the various forms of the `\>` command. To see how to resolve problems with label clashes see Section 1.4.

A reference to a label *any-label* (long or short) is made by enclosing *any-label* in a pair of double quotation marks: `"`*any-label*`"`; it is replaced by the number of the `\Chapter`, `\Section` or `\>` command that generated *any-label* in the first place. Generally, one only needs to make references to long labels when referring to other manuals. For references within the same manual, short labels are sufficient, except when the short label itself contains a colon.

Example

Since the `\BeginningOfBook` command for this manual defines *name-of-book* to be `gapmacro`, the long label for the current section is `gapmacro:Labels and References` and so a reference to this section within this manual might be: `Section "Labels and References"` (which is typeset as: Section 1.5). From another manual, a long label reference is required.

Another example

A section of this document has the title "Structuring the text: Chapters and Sections", which contains a colon. Hence, to refer to that section, one **must** use a long label:

```
Section "gapmacro:Structuring the text: Chapters and Sections"
```

produces: Section 1.3.

Note

In actual fact long labels are first sanitised by conversion to lower case and removal of superfluous white space (multiple blanks and new lines are converted to a single space). The same sanitisation process is applied to references. Thus,

```
Section "gapmacro:Structuring the text:                    Chapters
and Sections"
```

also produces: Section 1.3. So, don't worry about references to labels being broken over lines and think of them as being case-insensitive, except that the HTML converter requires that one respects case for the *name-of-book* component of a long label.

1.6 TeX Macros

As the manual pages are also used as on-line help, and are automatically converted to HTML, the use of special TEX commands should be avoided. The following macros can be used to structure the text, the mentioned fonts are used when printing the manual, however the on-line help and HTML are free to use other fonts or even colour. Since, the plain text on-line help, doesn't have special fonts, it leaves in much of the markup, including the left and right quotes that surround something intended to be displayed in typewriter type, the angle brackets that surround something intended to appear in italics, and the dollar-signs enclosing mathematics; you will need to keep that in mind when reading the following section.

‘*text*’
: sets *text* in `typewriter style`. This is typically used to denote GAP keywords such as `for` and `false` or variables that are not arguments to a function, e.g., ‘`for`’ produces `for`. See also <*text*>. Use `\<` to get a "less than" sign.

‘‘*text*’’
: encloses *text* in double quotes, e.g., ‘‘`double-quoted text`’’ produces "double-quoted text". In particular, ‘‘*text*’’ does **not** set ‘*text*’ in typewriter style; use ‘{‘`text`’}’ to produce ‘`text`’. Double quotes are mainly used to mark a phrase which will be defined later or is used in an uncommon way.

`\lq`
: sets a single left quote: ‘. For a phrase *text* that is to be defined later or is used in an uncommon way, please use ‘‘*text*’’ (which encloses *text* in double quotes rather than single quotes).

`\rq`, `\pif`
: each set a single apostrophe (right quote): ’. For the HTML and on-line manuals `\accent19{}` also sets an apostrophe; however the TEX-derived manuals produce an acute-d blankspace (what it in fact is).

`\accent127`
: sets an umlaut, e.g. `\accent127a` produces ä. Do not use the shorthand `\"` (otherwise the HTML converter will not translate it properly).

<*text*>
: sets *text* in italics. This can also be used inside `$...$` and ‘`...`’. Use `\<` to get a "less than" sign. `<...>` is used to denote a variable which is an argument of a function; a typical application is the description of a function:

```
\>Group( <gens> ) F
The function ‘Group’ constructs a group generated by <gens>.
```

  The `F` at the end of the first line in the above example indicates that `Group` is a function (see the `\>` entry, below).

text
: sets *text* in **emphasized style**.

`$a.b$`
: Inside math mode, you can use `.` instead of `\cdot` (a centred multiplication dot). Use `\.` for a full stop inside math mode. For example, `$a.b$` produces $a \cdot b$ while `$a\.b$` produces $a.b$.

`\cite{...}`
: produces a reference to a bibliography entry (the `\cite[...]{...}` option of LaTEX is **not** supported).

"*label*"
: produces a reference to *label*. Labels are generated by the commands `\Chapter`, `\Section` (see 1.5), and `\>` commands (see below).

`\index{`*index-entry*`}`
: defines an index entry *index-entry*. Besides appearing in the index, *index-entry* is also written to the section index file `manual.six` used by the on-line help. An exclamation mark (`!`), if present, is used to partition

index-entry into main entry (left part) and subentry (right part) components, in the index. TEX converts *index-entry* to lowercase and sets it in roman type, in the index. The HTML converter respects case and uses the default font, in producing the HTML manual index. *index-entry* must be completely free of special characters and font changing commands; if you need special fonts, characters or commands use one of `\indextt` or `\atindex`.

Note that for the HTML converter to process indexing commands (`\index`, `\indextt` and `\atindex`) correctly they **must** be on lines of their own. There can be several indexing commands on the same line, but there should be no horizontal whitespace before each indexing command, and if an indexing command needs to be broken over lines place a `%` at the point of the break at the end of the line to mark a "continuation".

For the HTML converter it works best to put indexing commands all together at the beginning of a paragraph, rather than strewn between lines of a paragraph. However, for the TEX-produced manuals after a maths display one gets a rogue space if you do this (this is a bug); you can work around the bug by putting at least one word of the paragraph followed by your line(s) of indexing commands.

Note also that indexing commands do **not** produce labels for cross-references; they **only** produce entries for the index. Labels are **only** produced by the chapter (`\Chapter`), section (`\Section`) and subsection (`\>`) commands.

`\indextt{`*index-entry*`}`

is the same as `\index{`*index-entry*`}`, except that *index-entry* is set by TEX in typewriter style, respecting case; the HTML converter sets *index-entry* in the default font. Again, *index-entry* should be completely free of special characters and font changing commands, and `!` may be used for sub-entries in the same way as for `\index`. Note that a sub-entry component, if present, is **not** set in typewriter style for the TEX-produced manuals; if you want that it is, use `\atindex`.

`\atindex{`*sort-entry*`}{|indexit}`

is simply a special form of the `\index` command that tells TEX to typeset the page number in italics.

`\atindex{`*sort-entry*`}{@`*index-entry*`}`

The HTML converter treats this command as if it was `\index{`*index-entry*`}`, except that it strips out any font information and sets it in the default font, but nevertheless respects case. *index-entry* may have `|indexit` at the end which is ignored by the HTML converter.

The TEX-produced manuals set the index entry as *index-entry* respecting font and case, and list it according to *sort-entry*. If a sub-entry is required then it should be present behind a `!` in **both** the *sort-entry* and *index-entry*; the only difference between the sub-entry in *sort-entry* and that in *index-entry*, is that the *sort-entry* sub-entry should be stripped of mark-up and font changing command. The *index-entry* component is ignored when constructing the `manual.six` files, and is also ignored by the HTML converter. Anything after an `!` in *sort-entry* is ignored when constructing the `manual.idx` file that is processed by MakeIndex. Macros like `{\GAP}` are allowed in *index-entry*. However, any `‘` that appears in *index-entry* **must** be preceded by `\noexpand`; *sort-entry* must be completely free of special characters and font changing commands.

In general, one should make *sort-entry* the same as *index-entry* modulo fonts and other mark-up, e.g.,

```
\atindex{Fred!Nerk}{@\noexpand‘Fred’!\noexpand‘Nerk’}
```

`{\GAP}`

typesets GAP.

`\package{`*pkg*`}`

typesets *pkg* in the font correct for GAP packages (respecting case). This is intended for cross-referencing other GAP packages. There is also the command `\Package{`*mypkg*`}` command which defines a macro `\`*mypkg* so that when you type `{\`*mypkg*`}` (please include the curly braces) the text *mypkg* is typeset in the right way for GAP packages. The `\Package` command should normally be included in one's `manual.tex` file (see 1.1) and just allows one to type `{\`*mypkg*`}` rather than the longer `\Package{`*mypkg*`}` as one is frequently likely to do when formulating one's own GAP package documentation. So, just to be clear about the difference between `\Package` and `\package`, `\Package{`*mypkg*`}` defines a macro `\`*mypkg* but produces no text, and `\package{`*pkg*`}` produces *pkg* set in the font that is right for GAP packages.

`\>`

produces a subsection. The line following the `\>` entry must either contain another `\>` entry (in which case the further entries are assumed to be variants and do not start a new subsection) or must be empty. The description text will follow this empty line.
There are several forms of the `\>` command. In all forms, a label and index entry are generated; the HTML converter uses the label to form an index entry, respecting case and setting in roman type. If the next non-space character is not a left quote (`‘`) it is assumed that the subsection is for a "function"; we exhibit these forms first.

`\>`*func*

While this form is supported; it is discouraged. If *func* is a 0-argument function, *func* should be followed by an empty pair of brackets (see `\>`*func*`(`*args*`)` below). If *func* is actually a global variable then `\>‘`*global-var*`’ V` should be used instead (see below). In order for this form to be parsed correctly the remainder of the line to the right of *func* must be empty. It generates *func* as both a label and index entry; *func* appears as is, in typewriter type in the TEX-derived manual index.

`\>`*func*`(`*args*`)`

The macro uses the brackets after *func* to parse the arguments *args*. Thus, it is necessary for the function to use brackets and for the arguments to have none. (We use the term "function" loosely here to mean "a GAP command with arguments"; we really mean an object that GAP knows as a: "Function", "Property", "Operation", "Category", or "Representation" — but not "Variable", since a "Variable" does not have arguments.) The label and index entry generated consists of the text between the `>` and opening bracket. The index entry is set as is (i.e. without conversion to lowercase) in typewriter type in the TEX-derived manual index. Here is an example of how to use `\>`; the index entry is "`Size`" (in typewriter type, with mixed case preserved).

```
\>Size( <obj> ) A
```

The `A` indicates that `Size` is an "Attribute". Instead of `A` there can be `F`, `P`, `O`, `C`, or `R` which indicate that a command is a "Function" (probably the most common), "Property", "Operation", "Category", or "Representation", respectively. For the forms of the `\>` command followed by a left quote, `V` indicating "Variable" (an object without arguments), is also possible. (See Section "Manual Conventions" and Chapter "Types of Objects" in the GAP Reference Manual).

`\>`*func*`(`*args*`)!{`*sub-entry*`}`

This is a special form of the previous command, that forms a label *func*`!`*sub-entry* and an index entry with main entry *func* (set in typewriter type and respecting case) and sub-entry *sub-entry* (set in roman type but also respecting case).
The remaining forms of the command `\>` expect to be followed by a `‘`.

`\>‘`*command*`’{`*label*`}`

works as `\>` without `‘...’`, but will not use bracket matching; it simply displays *command* as a header, which appears in typewriter type. It will use *label* as both the label and index entry, and the index entry is set in roman type. Whenever *label* contains a `!`, it is used to partition *label* into main entry (left part) and subentry (right part) components, in the index.

```
\>‘<a> + <b>’{addition}
\>‘Size( <obj> )’{size} A
```

In the first of the examples immediately above, the first form of `\>` cannot be used because no brackets occur. Also, observe that there is no command type (it's not appropriate here); TEX does not need it to correctly parse a `\>` entry, in general. The second example differs from our previous `Size` example, in that the index entry will be typeset as "size" (in roman type), rather than "`Size`". Also, the index entry is always converted to lowercase, no matter what case or mixed case was used.

`\>‘`*command*`’{`*label*`}!{`*sub-entry*`}`

is equivalent to: `\>‘`*command*`’{`*label*`!`*sub-entry*`}`.

`\>‘`*command*`’{`*label*`}@{`*index-entry*`}`
: works as `\>‘`*command*`’{`*label*`}`, except that it uses *label* for sorting the index entry and the index entry itself is printed as *index-entry*. References to the subsection use *label*. (Note that the HTML converter ignores everything after an `@` symbol in these commands, essentially treating the command as if it were `\>‘`*command*`’{`*label*`}`. However, the HTML converter also always preserves the case in a label.) Here are two examples.

```
\>‘Size( <obj> )’{size}@{‘Size’} A
\>‘Size( GL( <n>, <q> ) )’{Size!GL( n, q )}@{‘Size’! ‘GL’( \noexpand<n>, \noexpand<q> )} A
```

: The first of these examples is equivalent to “`\>Size( <obj> )`”. For the second example, it was necessary to use ‘ and ’, since the argument contained brackets. Note that `\noexpand` is needed before < here, but not needed before ‘ in the *index-entry* argument. Otherwise, the rules for sub-entries are the same as for `\atindex`.

`\>‘`*global-var*`’ V`
: This is actually a short-hand for: “`\>‘`*global-var*`’{`*global-var*`}@{‘`*global-var*`’} V`” to save you some typing when creating subsections for global variables, i.e., *global-var* is the label and the index entry appears in typewriter type, with mixed case preserved.

`\){\fmark ...}`
: is like `\>` except that it produces no label and index entry. It is `\fmark` that produces the filled in right arrow. Omitting it produces a line in typewriter type.

`\){\kernttindent ...}`
: is useful for producing a line in typewriter type, that you might otherwise have typed between `\begintt` and `\endtt`, but where you actually want the TEX macros and variables `<...>` to be interpreted.

`\URL{`*url*`}`
: prints the WWW URL *url*. In the HTML version this will be a HREF link.

`\Mailto{`*email*`}`
: prints the email address *email*. In the HTML version this will be a `mailto` link.

Note: When a TEX macro is followed by a space, TEX generally swallows up the space; one way, and it is the GAP-preferred way, of preventing the space being swallowed up, is by enclosing the macro in `{...}`. When a macro is often followed by a space, it’s a good habit to get into to **always** enclose that macro in `{...}` (the braces do nothing when the macro is not followed by a space, and prevent TEX from swallowing up the space, otherwise). Thus the macro for GAP should **always** be typed `{\GAP}`. Similarly, macros like `\lq`, `\rq` and `\pif` should probably always appear in braces; moreover the word “don’t” typeset via “`don{\pif}t`” will actually be interpreted correctly by the on-line browser.

1.7 TeX Macros for Domains

The following macros are required for the following common domains:

`\N` the natural numbers (you should probably indicate whether by your convention $\mathbb{N}$ includes zero or not, when using this);

`\Z` the integers;

`\Q` the rational numbers;

`\R` the real numbers;

`\C` the complex numbers;

`\F` a field; and

`\calR` a general domain e.g. a ring.

1.8 Examples, Lists, and Verbatim

In order to produce a list of items with descriptions use the `\beginitems`, `\enditems` environment, i.e. this is a "description" environment in the parlance of LaTeX and HTML.

For example, the following list describes `base`, `knownBase`, and `reduced`. The different item/description pairs must be separated by blank lines.

```
\beginitems
‘base’ &
    must be a  list of points ...

‘knownBase’ &
    If a base for <G> is known in advance ...

‘reduced’ (default ‘true’) &
    If this is ‘true’ the resulting stabilizer chain will be ...
\enditems
```

This will be printed as

`base`
: must be a list of points ...

`knownBase`
: If a base for G is known in advance ...

`reduced` (default `true`)
: If this is `true` the resulting stabilizer chain will be ...

In order to produce a list in a more compact format, use the `\beginlist`, `\endlist` environment.

An example is the following list.

```
\beginlist
\item{(a)}
    first entry
\item{(b)}
    second entry
\itemitem{--}
    a sub-item of the second entry
\itemitem{--}
    another sub-item of the second entry
\item{(c)}
    third entry
\endlist
```

It is printed as follows.

(a) first entry

(b) second entry

- a sub-item of the second entry
- another sub-item of the second entry

(c) third entry

The above example will take advantage of the ordered and unordered list environments in the HTML version, with the addition of slightly more mark-up. First, we present the example again with that additional mark-up, and then we explain how it works.

```
\beginlist%ordered{a}
\item{(a)}
    first entry
\item{(b)}
    second entry
\itemitem{--}%unordered
    a sub-item of the second entry
\itemitem{--}
    another sub-item of the second entry
\item{(c)}
    third entry
\endlist
```

It is printed as follows (of course, you should see no difference in the TEX-produced and on-line versions of this manual).

(a) first entry

(b) second entry

- a sub-item of the second entry
- another sub-item of the second entry

(c) third entry

In the HTML version the above example is interpreted as a nested list. The outer list is interpreted as an **ordered** list. The HTML standard provides 5 different types of ordered list, and these mirror the types provided by the `enumerate` LaTEX package. To signify that the outer list was **ordered** the comment `%ordered` was added after `\beginlist`. If there is no further markup the list is numbered in the default manner, namely with integers. Otherwise, following `%ordered` there should be one of the following:

`{1}` indicates the list should be numbered with integers (the default obtained when there is nothing following `%ordered`);

`{a}` indicates the list should be numbered with lowercase letters (`a`, `b`, ...);

`{A}` indicates the list should be numbered with uppercase letters (`A`, `B`, ...);

`{i}` indicates the list should be numbered with lowercase roman numerals (`i`, `ii`, ...); and finally

`{I}` indicates the list should be numbered with uppercase roman numerals (`I`, `II`, ...).

The `\beginlist` of the above example was followed by `%ordered{a}` and so the list is numbered using lowercase letters in the HTML version and using the ordered list environment (rather than the description environment).

Occasionally, it is necessary to break from a list, add some explanatory text and then restart the list, and resume numbering the items from where you left off. To do this follow the comment mark-up already mentioned by an **integer** in curly braces, i.e. if the outer list should actually start at `c` then you would need to have `%ordered{a}{3}` after `\beginlist` because `c` is the 3rd letter of our alphabet. Note that, for an integer-numbered list not starting at 1, you must have the full markup; you cannot omit the `{1}` after `%ordered` in this case.

The inner list of the above example is an **unordered** list (this corresponds to a plain `itemize` environment in LaTEX). To indicate this the first `\itemitem` above was followed by `%unordered`.

Of course, to get an unordered outer list, one would start the list with `\beginlist%ordered`, and to get an ordered inner list numbered with lowercase letters the first `\itemitem` would need to be followed by `%ordered{a}`, i.e. the

same syntax is used for the comment added after a `\beginlist` and after the first `\itemitem` in a sequence of `\itemitems`.

Notes

1. Only lists to a maximum depth of two are supported.
2. You cannot change the type of a sublist halfway through. Only the comment after the first `\itemitem` in a sequence is interpreted.

There are two types of **verbatim** environments. Example GAP sessions are typeset in typewriter style using the `\beginexample`, `\endexample` environment. Make sure that`\beginexample`, `\endexample`, `\begintt` and `\endtt` are on lines of their own.

```
\beginexample
gap> 1+2;
3
\endexample
```

typesets the example

```
gap> 1+2;
3
```

All examples in a chapter will also be written to files with the extension `.example-`*chapno*`.tst`, where *chapno* is the chapter number. These `.tst` files can be used to verify the examples in the manual. See 1.10 below for details.

Examples whose output may vary can be excluded from these test files, by using the command `\testexamplefalse`, e. g.

```
\testexamplefalse
\beginexample
gap> Exec("date");
Sun Oct  7 16:23:45 CEST 2001
\endexample
```

`\testexamplefalse` is only valid for the example immediately following.

Non-GAP examples should be typeset in typewriter style using the `\begintt`, `\endtt` environment.

Notes

1. The manual style will automatically indent examples. It also will break examples which become too long to fit on one page. If you want to encourage breaks at specific points in an example, end the example with `\endexample` and immediately start a new example environment with `\beginexample` on the next line.
2. To typeset a pipe symbol `|` in the `\begintt`, `\endtt` environment or `\beginexample`, `\endexample` you need to actually type `||`.

1.9 Tables, Displayed Mathematics and Mathematics Alignments

Tables should normally be set using the `\begintt`, `\endtt` environment. This means that one should enter the appropriate white space so that columns line up. Note that to get a vertical line `|` in the `\begintt`, `\endtt` environment one must actually type `||`. The reason for setting tables this way is so that both the HTML converter and GAP's built-in text browser have no trouble in displaying them correctly.

The HTML converter when used with its `-t` option (which causes it to use TtH to translate mathematics) usually does a reasonable job of converting mathematics displays and mathematics alignments. To help GAP's built-in text browser, however, one should follow a few rules:

– Place the `$$`s that begin and end the mathematics display on lines of their own. (If you don't do this it will be displayed in the same way as ordinary in-line mathematics.)

– Use only the `\matrix{ .. }` environment for mathematics alignments. The `\matrix{` starting the alignment should be on a line on its own, (flush left and no trailing whitespace). The `}` closing the environment should also be on a line of its own. The built-in browser doesn't do anything special to line things up; you must insert the whitespace where it's needed. Any `\hfill` macros you add to help the line things up in the TEX and HTML formats is ignored by the GAP's built-in text browser. The `\matrix{ .. }` environment should be used even when one might like to use TEX's `\cases{ .. }` environment.

The following example shows a typical usage of the `\matrix{ .. }` environment (in particular, it shows how one can use it to avoid using the `\cases{ .. }` environment). Observe, how sufficient whitespace has been added in order that alignment is maintained by GAP's built-in text browser. (Recall that `\right.` which produces nothing is required to match `\left\{`.)

```
From a theorem of Gauss we know that
$$
b_N = \left\{
\matrix{
\frac{1}{2}(-1+\sqrt{N})    &{\rm if} &N \equiv 1  &\pmod 4\cr
\frac{1}{2}(-1+i \sqrt{N})  &{\rm if} &N \equiv -1 &\pmod 4\cr
}
\right.
$$
```

The example produces . . .

From a theorem of Gauss we know that

$$b_N = \begin{cases} \frac{1}{2}(-1+\sqrt{N}) & \text{if} \quad N \equiv 1 \pmod 4 \\ \frac{1}{2}(-1+i\sqrt{N}) & \text{if} \quad N \equiv -1 \pmod 4 \end{cases}$$

1.10 Testing the Examples

Ideally, the GAP examples (the text between `\beginexample` and `\endexample`) should be chosen such that every user obtains the same output (up to line breaks and whitespace) when typing in your example.

This is often difficult to achieve, or can only be achieved at the cost of writing unnecessarily complicated examples. Therefore, it is recommended that you choose examples in such a way that when a user starts GAP, loads your package and types the examples of a chapter in the given order, then (s)he will see the same output as in the manual examples. (This will ensure that the global random number generator is initialized to the **same** values. For more details, see the last paragraph of "Starting and Leaving GAP" in the GAP Tutorial.) In cases where this is impossible, you may use `\testexamplefalse` before `\beginexample`, see 1.8.

As mentioned above, a TEX run of the manual produces files `manual.example-`*chapno*`.tst`, one for each chapter containing at least one GAP example (the text between `\beginexample` and `\endexample`). These files can be read into GAP using `ReadTest` (see `?ReadTest`), to ensure that the GAP output for the examples hasn't changed.

The test also requires that examples are not indented in the TEX files; in the typeset manual, the lines between `\beginexample` and `\endexample` and the lines between `\begintt` and `\endtt` are automatically indented.

1.11 Usage of the Percent Symbol

The % symbol has a number of very specific uses. Take care that you use it correctly. These uses are:

1. A line **beginning** with 16 (or more) % symbols marks the **end** of a section, or the **end** of a chapter introduction (which may be empty). Such a line **must** precede **every** `\Section` (see 1.3).
2. A % at the beginning of a line tells TEX that the line is a comment and is to be ignored by TEX, **except** in the verbatim environments: `\begintt..\endtt` and `\beginexample..\endexample`. However, `%display` or `%enddisplay` commands have special meaning for the on-line text help browser and for the HTML converter and may temporarily alter the meaning of an initial % for these (see 1.12 for details); otherwise the meaning of an initial % is the same as for TEX.
3. A % at the end of a line marks a "continuation", **except** in the situation mentioned in item 4. A "continuation" may be needed for lines of indexing commands (`\index`, `\indextt` or `\atindex`). Such commands **must** occur on lines of their own (see 1.6), **not** mixed with text, and there must not be any superfluous whitespace (modulo the next statement). Occasionally an indexing command is too long to easily fit on a line; this is where a continuation is desirable; a % at the end of such a line indicates that the line is to be joined with the next line after removal of the % symbol and any initial whitespace on the next line (this is what TEX does! . . . and we mimic this behaviour for both the on-line text help browser and the HTML manuals).

   A "continuation" may also be necessary for subsections, i.e. lines beginning with `\>` or `\)` (again see 1.6); the usage is as for indexing line continuations.
4. A line ending with a % that is not an indexing command line or a subsection line that after any initial whitespace is removed matches **exactly** `{%` or `}%`, or begins with `{\` or `\` and is followed by a letter, is ignored by both the on-line browser and the HTML converter. This is intended to screen the on-line browser and HTML converter from TEX commands such as `\obeylines`, `\begingroup`, `\def` etc., without having to resort to using the `%display{tex}..%enddisplay` environment.

Warning. In view of items 3. and 4. above, avoid using a % at the end of a line unless you really need it, and it fits into those categories. In particular, do **not** put a % at the end of an indexing command line that is immediately followed by a line of text; otherwise, the text line will not appear in the HTML manual or on-line via the text help browser. Similarly, do not put a % line at the end of a text line that is immediately followed by an indexing command line; this causes the indexing command line to be ignored by the HTML converter. For the HTML converter it works best to put indexing commands all together at the beginning of a paragraph, rather than strewn between lines of a paragraph. However, for the TEX-produced manuals after a maths display one gets a rogue space if you do this (this is a bug); you can work around the bug by putting at least one word of the paragraph followed by your lines(s) of indexing commands.

1.12 Catering for Plain Text and HTML Formats

As described in 1.6, the use of macros should be restricted to the ones given in the previous sections. By doing so, you should find that the documentation you write will still look ok in GAP's on-line help (plain text format) and in the translated HTML. However, in rare situations one might be forced to use other TEX macros, for example in order to typeset a lattice. In this case you should provide an alternative for the on-line help, and possibly also for the HTML version. This can be done by putting in guiding commands as TEX comments:

```
%display{tex}
TeX version (only used by TeX manual)
%display{html}
%HTML version (only used by HTML manual)
%display{text}
%Text version (only used by the built-in manual browser)
%enddisplay
```

Observe that the lines that should appear only in the TEX-produced manuals do not begin with a %. For the HTML (resp. text) version the lines begin with a %; each line of a %display{html} (resp. %display{text}) environment is printed verbatim, after removing the initial % symbol. The above example produces:

TeX version (only used by TeX manual)

(Note the above example will vary according to whether you are viewing it as a TEX-produced manual, or as an HTML manual, or via the built-in manual browser — as it should!)

Sometimes one needs a %display environment to be not seen by TEX, but still interpreted normally (i.e. not printed verbatim). The following variant of the above provides this capability.

```
%display{tex}
TeX version (only used by TeX manual)
%display{nontex}
%HTML and Text version (interpreted normally, after removing the \% symbol)
%enddisplay
```

The above example produces:

TeX version (only used by TeX manual)

It is permissible to abbreviate any of the above by omitting %display{tex}, %display{html}, or %display{text} if that portion of the environment would be empty.

There are yet two more variants of conditional display. Firstly,

```
%display{nonhtml}
%Text version (interpreted normally by built-in browser, after removing the
%\% symbol)
%enddisplay
```

is normally used to ensure text only appears via the on-line help browser. If there is no initial % it also appears in the TEX-produced manuals. The above example produces:

Finally, there is

```
%display{nontext}
%HTML version (interpreted normally by HTML converter, after removing the
%\% symbol)
%enddisplay
```

which excludes text from the on-line help browser. Like the %display{nonhtml} environment, if there is no initial % it also appears in the TEX-produced manuals. The example produces:

However, the use of these special environments should be avoided as much as possible, since it is much more difficult to maintain such pseudo-duplicated documentation.

1.13 Umlauts

To produce umlauts, use \accent127 and not the shorthand \" (otherwise the HTML converter will not translate it properly).

1.14 Producing a Manual

To produce a manual you will need the following files:

`manual.tex`
: contains the body of the manual (as described in Section 1.1) and an `\Input` command for each chapter/appendix file.

file1`.tex`, *file2*`.tex`, ...
: the chapter/appendix files. There must be one file for each chapter or appendix, and each such file should have a `\Chapter` or `\PreliminaryChapter` command. Alternatively, one can write `.msk` files and use `buildman.pe` to generate the corresponding `.tex` files (see 1.15).

`gapmacro.tex`
: contains the macros for the manual. It must be input by an `\input` statement (**not** and `\Input` statement, which creates a Table of Contents entry) in `manual.tex`. You can either use the version in the `doc` directory of GAP (use a relative path then) or make a copy.

`manual.mst`
: is a "configure" file used by `makeindex` when processing index information in a TEX-generated and `manualindex`-preprocessed `manual.idx` file. It must reside in your manual directory.

`GAPDOCPATH/manualindex`
: is used to call `makeindex`. `GAPDOCPATH` is the path of the `doc` directory of your GAP distribution.

For bibliography information you will need a file `manual.bbl`. If you intend to create it with BibTEX, you will need to indicate the appropriate `.bib` file (as described in section 1.1). Then after running TEX once over the manual, run BibTEX to create the `manual.bbl` file.

Assuming that all necessary files are there (a `manual.lab` file for each *book* argument of a `\UseReferences` command, `mrabbrev.bib` and `manualindex` in the GAP `doc` directory), on a Unix system the following calls will then produce a file `manual.dvi` as well as a file `manual.six` which is used by the GAP help functions. If you are missing some of the needed files and don't have `CVS` access to GAP, just send an email request for them to `support@gap-system.org`.

Go to the directory holding the manual. Call

```
tex manual
```

to produce bibliography information. Unless you provide a `manual.bbl` file which is not produced by BibTEX, call

```
bibtex manual
```

to produce the `manual.bbl` file. Then run TEX twice over the manual to fill all references and produce a stable table of contents:

```
tex manual
tex manual
```

If you have sections which are named like commands, you may get messages about redefined labels. At this point you can ignore these.

Now it is time to produce the index. Call

```
GAPDOCPATH/manualindex manual
```

which preprocesses the `manual.idx` file and then runs `makeindex`. Provided that `manual.mst` exists, this produces a file `manual.ind`. Finally, once again run

```
tex manual
```

to incorporate the index. The manual is ready.

1.15 Using buildman.pe

Rather than write the chapter/appendix `.tex` files directly, one may incorporate one's documentation in comments in one's GAP code. To do it this way, there are four ingredients:

`.gd` files
: GAP files with `.gd` suffixes that have the documentation in comments (actually files with `.g` or `.gi` or any other extension are also possible, but files with extension `.gd` are the default);

`.msk` files
: which are just like the `.tex` files, and must obey all the rules given for `.tex` files previously, but additionally may have `\FileHeader` or `\Declaration` commands at places where text should be inserted from a `.gd` file, and with `{{`*variable*`}}` patterns which are replaced by *replacement* when written to the `.tex` file, if the configuration file *configfile* has a line of form: *variable*=*replacement*;

configfile
: a file which defines `msfiles` (the list of `.msk` files), `gdfiles` (the list of `.gd` files), `LIB` (the directory containing the `.gd` files), `DIR` (the directory in which to put the constructed `.tex` files, one `.tex` file for each `.msk` file), and optionally a line `check` (see below) and *variable*=*replacement* lines; and

`buildman.pe`
: a perl program (in the `etc` directory for those with `CVS` access to GAP), which strips the comments from the `.gd` files according to the `\FileHeader` or `\Declaration` commands in the `.msk` files, translates any `{{`*variable*`}}` patterns defined by the file *configfile* and constructs the `.tex` files.

If you don't have `CVS` access and want to use `buildman.pe`, just email `support@gap-system.org` and ask for it. Please note that there is no obligation for package authors to `buildman.pe`; nor does it attract the same level of support as the rest of GAP; in general, bugs can be expected to be fixed (eventually), but no new features will be added. Also, note that the GAPDoc package provides a similar facility.

The perl program `buildman.pe` is called as follows:

`buildman.pe -f` *configfile*

The form of *configfile*

There is no restriction on how to name *configfile*, but by convention it is of form `config.`*something* or `buildman.config`; *configfile* should contain lines of form:

`msfiles=`*msfile1*`,`*msfile2*`,...,`*msfilem*`;`
`gdfiles=`*gdfile1*`,`*gdfile2*`,...,`*gdfilen*`;`
`LIB=`*gdfile_dir*`;`
`DIR=`*TeX_dir*`;`

Optionally, as mentioned above, one may also have:

`check;`

which says to construct a `notfound` file that lists missing expected data, and any number of lines of form

variable=*replacement*

The file *configfile* should obey the following syntactic rules:

- After `msfiles=` there should be a comma-separated and semicolon-terminated list of `.msk` files with the `.msk` extensions removed; `buildman.pe` assumes that the `.msk` files are all in, or at least have path relative to, the directory in which `buildman.pe` is called.
- Similar to the `msfiles` definition, after `gdfiles=` there should be a comma-separated and semicolon-terminated list of "`.gd`" files. If a "`.gd`" file really does have a `.gd` extension, it may be listed without extension; otherwise the extension **must** be included. All the "`.gd`" files must be listed with path relative to the directory defined by `LIB`.
- For both the `msfiles` and `gdfiles` definitions, the lists following the `=` may continue over several lines if necessary, and any whitespace, parentheses (round brackets) or double-quotes characters are ignored.
- The paths after `LIB=` and `DIR=` are assumed relative to the "current directory", i.e. the directory in which `buildman.pe` is executed. For each *msfilei* listed after the `msfiles` keyword, `buildman.pe` constructs from *msfilei*`.msk` a corresponding *msfilei*`.tex` in *TeX_dir*. The `LIB` and `DIR` definitions must be on a single line.
- The terminating `;` is optional on the lines containing the keywords `LIB`, `DIR` or `check`.
- Superfluous characters around any of the keywords `msfiles`, `gdfiles`, `LIB`, `DIR` or `check`, but before the `=` on the lines where `=` is required, are ignored. Whitespace and double-quotes characters are ignored, everywhere.
- The *variable*=*replacement* lines (if there are any) should have no other punctuation or whitespace. These lines direct `buildman.pe` to replace any string of form `{{`*variable*`}}` in a `.msk` file with *replacement*.

Special `.msk` file commands

Now we describe the special (non-TEX) commands that direct `buildman.pe` to extract text from "`.gd`" files.

`\FileHeader[`*n*`]{`*gdfile*`}`
: This command is replaced by the text following a `#`*n* line (for positive integer *n*) in file *gdfile*`.gd` (or *gdfile* if *gdfile* already contains a suffix). The argument `[`*n*`]` of `\FileHeader` is optional; if it is omitted *n* is taken to be 1. See below for the typical form of a fileheader extracted by the `\FileHeader` command; the comments in the example describe its required format.

`\Declaration{`*func*`}[`*gdfile*`]{`*label*`}!{`*sub-entry*`}@{`*index-entry*`}`
: This command is replaced by a `\>` subsection declaration or block of `\>` declarations, and their description extracted from a block in a "`.gd`" file that starts with a line matching `#`*X* *func*, for some letter *X* in `F`, `M`, `A`, `P`, `O`, `C`, `R` or `V`. The line "matches" if there is a `(`, space, or newline after *func*. The argument *func* (in `{..}`) is the only mandatory argument.
If present, `[`*gdfile*`]`, says that *func* is to be found in the file *gdfile*`.gd` (or *gdfile* if *gdfile* already contains a suffix); it is required only if *func* appears in more than one of the "`.gd`" files listed in the file *configfile*. The *gdfile* argument is typically required for distinguishing methods of operations.
The remaining arguments (if present) have exactly the purpose that they have in subsection declarations, i.e. lines of the following forms:

  `\>`*func*`!{`*sub-entry*`}`
  `\>'`*command*`'{`*label*`}`
  `\>'`*command*`'!{`*sub-entry*`}`
  `\>'`*command*`'@{`*index-entry*`}`

  (see Section 1.6), and are used to build subsection declaration lines of these forms. Note that the *label*, *sub-entry* and *index-entry* arguments, if needed, should follow the `\Declaration` command (and **not** be in the "`.gd`" file `#`*X* *func*`...` lines, where they will be indistinguishable from comments). If in the "`.gd`" file the `#`*X* *func* line is followed by other `#`*Xi* *funci* lines, then each `\>` subsection declaration formed has the same *label*, *sub-entry* and *index-entry* arguments appended.

Corresponding to `\FileHeader[`*n*`]{`*gdfile*`}`, in the "`.gd`" file denoted by *gdfile*, there should be:

```
#n
##  Text for \FileHeader[n]{gdfile}. Each line
##  should have two # characters followed by 2 blank
##  space characters at the left margin. The text
##  can and should include any necessary {\TeX}
##  mark-up and indexing commands.
##
##  A fileheader may consist of any number of paragraphs.
##  It is terminated by a totally empty line (i.e.~a
##  line devoid even of # characters).
##
```

Corresponding to each `\Declaration{`*func*`}...` line of a `.msk` file there should be in one of the "`.gd`" files, a block of form:

```
#X  func( args ) comment
#Y  func2( args2 ) comment2
.
.
#Z  funcn( argsn ) commentn
##
##  description of func, func2, ..., funcn.
##
Declare...( "func" ...);
Declare...( "func2" ...);
.
.
Declare...( "funcn" ...);
```

The above block should comply with the following syntactic rules. Below we use the term "function" in a general sense to mean any one of function (in the strict sense), attribute, category, method, representation, operation, property or variable.

- $X, Y, \ldots, Z \in \{\mathtt{A}, \mathtt{C}, \mathtt{F}, \mathtt{M}, \mathtt{R}, \mathtt{O}, \mathtt{P}, \mathtt{V}\}$. If the letter is `V` then no parentheses or arguments should follow the "function name" *funci*.
- The letters, X, Y, ..., Z are printed in the manual. If a letter is `A` or `P`, then also the letters `S` and `T` are printed if the setter and the tester are available. If the letter is `A`, then the letter `M` is printed if the attribute is mutable.
- The comments *comment*, *comment2*, ..., *commentn* (by convention starting with spaced dots) which do not appear in the manual, are optional.
- The X, Y, ..., Z "function name" lines must appear on consecutive lines, i.e. not intermingled with text lines.
- After the "function name" lines there should be text lines describing the "functions". As with fileheader text these text lines should contain any TEX mark-up and indexing commands that are necessary, and there should be two blank space characters between the `##` and the text. Lines starting with `#T` (or some other non-`#` character in place of `T`) are ignored.
- It is assumed that for each "function name" *func*, *func2*, ..., *funcn* there is a corresponding GAP declaration (which need not be via a `Declare...` command, e.g. it might be `BindGlobal`) after the `##` text lines (and comment lines), **and** that they appear in the **same** order.

Example

Suppose we have a manual whose `.msk` files are in the directory `doc/build`, whose `.tex` files are created in the directory `doc/ref`, and whose GAP code files are in the directory `lib`, one of them being the file `lib/algebra.gd`, which contains the following declaration:

```
#############################################################################
##
#O  DirectSumOfAlgebras( <A1>, <A2> )
#O  DirectSumOfAlgebras( <list> )
##
##  is the direct sum of the two algebras <A1> and <A2> respectively of the
##  algebras in the list <list>.
##
##  If all involved algebras are associative algebras then the result is also
##  known to be associative.
##  If all involved algebras are Lie algebras then the result is also known
##  to be a Lie algebra.
##
##  All involved algebras must have the same left acting domain.
##
##  The default case is that the result is a structure constants algebra.
##  If all involved algebras are matrix algebras, and either both are Lie
##  algebras or both are associative then the result is again a
##  matrix algebra of the appropriate type.
##
DeclareOperation( "DirectSumOfAlgebras", [ IsDenseList ] );
```

Further suppose that the file `doc/build/algebra.msk` contains the line:

```
\Declaration{DirectSumOfAlgebras}
```

The "config" file `doc/build/config.alg`:

```
@msfiles = ("algebra","algfp","alglie","mgmring");
@gdfiles = ("algebra","alghom","alglie","object","liefam","mgmring","algrep",
            "lierep");
DIR = "../ref";
LIB = "../../lib";
```

specifies `algebra.msk` via the first entry of `msfiles` and `lib/algebra.gd` via the first entry of `gdfiles` and (its directory by) the definition of `LIB`. Observe that there are `@` and `"` symbols, as well as parentheses and whitespace, in the above "config" file; **none** of these is necessary, but they don't do any harm either. Generally, one calls `buildman.pe` in the same directory that contains the `msfiles` (which is why one doesn't need to specify the directory containing the `msfiles`) and the "config" file. Since `DIR = "../ref"`, `buildman.pe` constructs `algebra.tex` from `algebra.msk` in directory `doc/ref`. The subsection generated in `algebra.tex` by the above `\Declaration` command starts with the header:

```
\>DirectSumOfAlgebras( <A1>, <A2> ) O
\>DirectSumOfAlgebras( <list> ) O
```

and is followed by its description, i.e. the lines beginning with two hashes and two blanks, but with the hashes and blanks stripped away, so that when it is processed the resulting subsection appears as:

▶ `DirectSumOfAlgebras(` *A1*`,` *A2* `)` O

▶ `DirectSumOfAlgebras( list )` O

is the direct sum of the two algebras *A1* and *A2* respectively of the algebras in the list *list*.

If all involved algebras are associative algebras then the result is also known to be associative. If all involved algebras are Lie algebras then the result is also known to be a Lie algebra.

All involved algebras must have the same left acting domain.

The default case is that the result is a structure constants algebra. If all involved algebras are matrix algebras, and either both are Lie algebras or both are associative then the result is again a matrix algebra of the appropriate type.

Variable replacement

As mentioned above the "config" file may also contain lines that assign variables, e.g.

```
versionnumber=4.3
versionsuffix=4r3
```

Occurrences of these variables in double curly braces will be replaced by their value. For example the lines

```
When ‘unzoo -x’ is applied to {\GAP}~{{versionnumber}}’s ‘zoo’ file
‘gap{{versionsuffix}}.zoo’ a directory ‘gap{{versionsuffix}}’ is formed.
```

in a `.msk` file will be replaced by:

```
When ‘unzoo -x’ is applied to {\GAP}~4.3’s ‘zoo’ file
‘gap4r3.zoo’ a directory ‘gap4r3’ is formed.
```

in the corresponding `.tex` file. This feature is very handy for information that changes over time.

Final note

There is a document for version `0.0` of `buildman.pe` that describes features that have either never been used or have since been disabled. Only the features described in this section can be relied upon to have currency.