AutoDoc

Generate documentation from GAP source code

2016.02.16

16/02/2016

Sebastian Gutsche

Max Horn

Sebastian Gutsche
Email: gutsche@mathematik.uni-kl.de
Homepage: http://wwwb.math.rwth-aachen.de/~gutsche/
Address: Department of Mathematics
University of Kaiserslautern
67653 Kaiserslautern
Germany

Max Horn
Email: max.horn@math.uni-giessen.de
Homepage: http://www.quendi.de/math
Address: AG Algebra
Mathematisches Institut
JLU Gießen
Arndtstraße 2
D-35392 Gießen
Germany

Copyright

Contents

1 Getting started using AutoDoc **4**
- 1.1 Creating a package manual from scratch . . . 4
- 1.2 Documenting code with AutoDoc . . . 5
- 1.3 Using AutoDoc in an existing GAPDoc manual . . . 6
- 1.4 Automatic regeneration of the manual . . . 6
- 1.5 What is taken from `PackageInfo.g` . . . 6

2 AutoDoc documentation comments **8**
- 2.1 Documenting declarations . . . 8
- 2.2 Other documentation comments . . . 10
- 2.3 Title page commands . . . 13
- 2.4 Plain text files . . . 14
- 2.5 Grouping . . . 14
- 2.6 Level . . . 15
- 2.7 Some syntax for AutoDoc text . . . 15

3 AutoDoc worksheets **17**
- 3.1 Worksheets . . . 17

4 AutoDoc **18**
- 4.1 The AutoDoc() function . . . 18

Chapter 1

Getting started using AutoDoc

AutoDoc is a GAP package which is meant to aide GAP package authors in creating and maintaing the documentation of their packages. In this capacity it builds upon GAPDoc (and hence is not a replacement for it, but rather a complement). In this chapter we describe how you can get started using AutoDoc for your package. To this end, we will assume from now on that your package is called SomePackage.

1.1 Creating a package manual from scratch

Suppose your package is already up and running, but so far has no manual. Then you can rapidly generate a "scaffold" for a package manual using the `AutoDoc` (4.1.1) command like this, while running GAP from within your package's directory (the one containing the `PackageInfo.g`:

```
LoadPackage( "AutoDoc" );
AutoDoc( : scaffold := true );
```

This first reads the `PackageInfo.g` file from the current directory. It extracts information about package from it (such as its name and version, see Section 1.5). It then creates two XML files `doc/SomePackage.xml` and `doc/title.xml` insider the package directory. Finally, it runs GAPDoc on them to produce a nice initial PDF and HTML version of your fresh manual.

To ensure that the GAP help system picks up your package manual, you should also add something like the following to your `PackageInfo.g`:

```
PackageDoc := rec(
  BookName  := ~.PackageName,
  ArchiveURLSubset := ["doc"],
  HTMLStart := "doc/chap0.html",
  PDFFile   := "doc/manual.pdf",
  SixFile   := "doc/manual.six",
  LongTitle := ~.Subtitle,
),
```

Congratulations, your package now has a minimal working manual. Of course it will be mostly empty for now, but it already should contain some useful information, based on the data in your `PackageInfo.g`. This includes your package's name, version and description as well as information

about its authors. And if you ever change the package data, (e.g. because your email address changed), just re-run the above command to regenerate the two main XML files with the latest information.

Next of course you need to provide actual content (unfortunately, we were not yet able to automate *that* for you, more research on artificial intelligence is required). To add more content, you have several options: You could add further GAPDoc XML files containing extra chapters, sections and so on. Or you could use classic GAPDoc source comments (in either case, see Section 1.3 on how to teach the `AutoDoc` (4.1.1) command to include this extra documentation). Or you could use the special documentation facilities AutoDoc provides (see Section 1.2).

You may also wish to consult Section 1.4 for hints on automatically re-generating your package manual when necessary.

1.2 Documenting code with AutoDoc

To get one of your global functions, operations, attributes etc. to appear in the package manual, simply insert an AutoDoc comment of the form `#!` directly in front of it. For example:

```
#!
DeclareOperation( "ToricVariety", [ IsConvexObject ] );
```

This tiny change is already sufficient to ensure that the operation appears in the manual. In general, you will want to add further information about the operation, such as in the following example:

```
#! @Arguments conv
#! @Returns a toric variety
#! @Description
#!  Creates a toric variety out
#!  of the convex object <A>conv</A>.
DeclareOperation( "ToricVariety", [ IsConvexObject ] );
```

For a thorough description of what you can do with AutoDoc documentation comments, please refer to chapter 2.

Suppose you have not been using GAPDoc before but instead used the process described in section 1.1 to create your manual. Then the following GAP command will regenerate the manual and automatically include all newly documented functions, operations etc.:

```
LoadPackage( "AutoDoc" );
AutoDoc( : scaffold := true, autodoc := true );
```

If you are not using the scaffolding feature, e.g. because you already have an existing GAPDoc based manual, then you can still use AutoDoc documentation comments. Just make sure to first edit the main XML file of your documentation, and insert the line

```
#Include SYSTEM "_AutoDocMainFile.xml"
```

in a suitable place. This means that you can mix AutoDoc documentation comment freely with your existing documentation; you can even still make use of any existing GAPDoc documentation comments in your code. The following command should be useful for you in this case; it still scans the package code for AutoDoc documentation comments and the runs GAPDoc to produce HTML and PDF output, but does not touch your documentation XML files otherwise.

```
LoadPackage( "AutoDoc" );
AutoDoc( : autodoc := true );
```

1.3 Using AutoDoc in an existing GAPDoc manual

Even if you already have an existing GAPDoc manual, it might be interesting for you to use AutoDoc for two purposes:

First off, with AutoDoc is very convenient to regenerate your documentation.

Secondly, the scaffolding feature which generates a title package with all the metadata of your package in a uniform way is very handy. The somewhat tedious process of keeping your title page in sync with your `PackageInfo.g` is fully automated this way (including the correct version, release data, author information and so on).

There are various examples of packages using AutoDoc for only this purpose, e.g. IO and orb.

1.4 Automatic regeneration of the manual

You will probably want to re-run the `AutoDoc` (4.1.1) command frequently, e.g. whenever you modified your documentation or your `PackageInfo.g`. To make this more convenient and reproducible, we recommend putting its invocation into a file `makedoc.g` in your package directory, with content based on the following example:

```
LoadPackage( "AutoDoc" );
AutoDoc( : autodoc := true );
QUIT;
```

Then you can regenerate the package manual from the command line with the following command, executed from within in the package directory:

```
gap makedoc.g
```

1.5 What is taken from `PackageInfo.g`

AutoDoc can extract data from `PackageInfo.g` in order to generate a title page. Specifically, the following components of the package info record are looked at:

Version
: This is used to set the `<Version>` element of the title page, with the string "Version " prepended.

Date This is used to set the `<Date>` element of the title page.

Subtitle
: This is used to set the `<Subtitle>` element of the title page (the `<Title>` is set to the package name).

Persons
: This is used to generate `<Author>` elements in the generated title page.

PackageDoc

This is a record (or a list of records) which is used to tell the GAP help system about the package manual. Currently AutoDoc extracts the value of the `PackageDoc.BookName` component and then passes that on to GAPDoc when creating the HTML, PDF and text versions of the manual.

AutoDoc

This is a record which can be used to control the scaffolding performed by AutoDoc, specifically to provide extra information for the title page. For example, you can set `AutoDoc.TitlePage.Copyright` to a string which will then be inserted on the generated title page. Using this method you can customize the following title page elements: `TitleComment`, `Abstract`, `Copyright`, `Acknowledgements` and `Colophon`.

Note that `AutoDoc.TitlePage` behaves exactly the same as the `scaffold.TitlePage` parameter of the `AutoDoc` (4.1.1) function.

Chapter 2

AutoDoc documentation comments

You can document declarations of global functions and variables, operations, attributes etc. by inserting AutoDoc comments into your sources before these declaration. An AutoDoc comment always starts with `#!`. This is also the smallest possible AutoDoc command. If you want your declaration documented, just write `#!` at the line before the documentation. For example:

```
#!
DeclareOperation( "AnOperation",
                  [ IsList ] );
```

This will produce a manual entry for the operation `AnOperation`.

2.1 Documenting declarations

In the bare form above, the manual entry for `AnOperation` will not contain much more than the name of the operation. In order to change this, there are several commands you can put into the AutoDoc comment before the declaration. Currently, the following commands are provided:

2.1.1 @Description *descr*

Adds the text in the following lines of the AutoDoc to the description of the declaration in the manual. Lines are until the next AutoDoc command or until the declaration is reached.

2.1.2 @Returns *ret_val*

The string *ret_val* is added to the documentation, with the text "Returns: " put in front of it. This should usually give a brief hint about the type or meaning of the value retuned by the documented function.

2.1.3 @Arguments *args*

The string *args* contains a description of the arguments the function expects, including optional parts, which are denoted by square brackets. The argument names can be separated by whitespace, commas or square brackets for the optional arguments, like "grp[, elm]" or "xx[y[z]]". If GAP options are used, this can be followed by a colon : and one or more assignments, like "n[, r]: tries := 100".

2.1.4 @Group *grpname*

Adds the following method to a group with the given name. See section 2.5 for more information about groups.

2.1.5 @Label *label*

Adds label to the function as label. If this is not specified, then for declarations that involve a list of input filters (as is the case for `DeclareOperation`, `DeclareAttribute`, etc.), a default label is generated from this filter list.

```
#! @Label testlabel
DeclareProperty( "AProperty",
                 IsObject );
```

leads to this:

2.1.6 AProperty (testlabel)

▷ `AProperty(`*arg*`)` (property)

Returns: `true` or `false`

while

```
#!
DeclareProperty( "AProperty",
                 IsObject );
```

leads to this:

2.1.7 AProperty (for IsObject)

▷ `AProperty(`*arg*`)` (property)

Returns: `true` or `false`

2.1.8 @ChapterInfo *chapter, section*

Adds the entry to the given chapter and section. Here, *chapter* and *section* are the respective titles.

As an example, a full AutoDoc comment for with all options could look like this:

```
#! @Description
#! Computes the list of lists of degrees of ordinary characters
#! associated to the <A>p</A>-blocks of the group <A>G</A>
#! with <A>p</A>-modular character table <A>modtbl</A>
#! and underlying ordinary character table <A>ordtbl</A>.
#! @Returns a list
#! @Arguments modtbl
#! @Group CharacterDegreesOfBlocks
#! @FunctionLabel chardegblocks
#! @ChapterInfo Blocks, Attributes
DeclareAttribute( "CharacterDegreesOfBlocks",
        IsBrauerTable );
```

2.2 Other documentation comments

There are also some commands which can be used in AutoDoc comments that are not associated to any declaration. This is useful for additional text in your documentation, examples, mathematical chapters, etc..

2.2.1 @Chapter *name*

Sets a chapter, all functions without seperate info will be added to this chapter. Also all text comments, i.e. lines that begin with #! without a command, and which do not follow after @description, will be added to the chapter as regular text. Example:

```
#! @Chapter My chapter
#!  This is my chapter.
#!  I document my stuff in it.
```

2.2.2 @Section *name*

Sets a section like chapter sets a chapter.

```
#! @Section My first manual section
#!  In this section I am going to document my first method.
```

2.2.3 @EndSection

Closes the current section. Please be careful here. Closing a section before opening it might cause unexpected errors.

```
#! @EndSection
#### The following text again belongs to the chapter
#! Now we could start a second section if we want to.
```

2.2.4 @Subsection *name*

Sets a subsection like chapter sets a chapter.

```
#! @Subsection My first manual subsection
#!  In this subsection I am going to document my first example.
```

2.2.5 @EndSubsection

Closes the current subsection. Please be careful here. Closing a subsection before opening it might cause unexpected errors.

```
#! @EndSubsection
#### The following text again belongs to the section
#! Now we are in the section again
```

2.2.6 @BeginAutoDoc

Causes all subsequent declarations to be documented in the manual, regardless of whether they have an AutoDoc comment in front of them or not.

2.2.7 @EndAutoDoc

Ends the affect of `@BeginAutoDoc`. So from here on, again only declarations with an explicit AutoDoc comment in front are added to the manual.

```
#! @BeginAutoDoc

DeclareOperation( "Operation1", [ IsList ] );

DeclareProperty( "IsProperty", IsList );

#! @EndAutoDoc
```

Both, `Operation1` and `IsProperty` would appear in the manual.

2.2.8 @BeginGroup *[grpname]*

Starts a group. All following documented declarations without an explicit `@Group` command are grouped together in the same group with the given name. If no name is given, then a new nameless group is generated. The effect of this command is ended when an `@EndGroup` command is reached.

See section 2.5 for more information about groups.

2.2.9 @EndGroup

Ends the current group.

```
#! @BeginGroup MyGroup
#!
DeclareAttribute( "GroupedAttribute",
                  IsList );

DeclareOperation( "NonGroupedOperation",
                  [ IsObject ] );

#!
DeclareOperation( "GroupedOperation",
                  [ IsList, IsRubbish ] );
#! @EndGroup
```

2.2.10 @Level *lvl*

Sets the current level of the documentation. All items created after this, chapters, sections, and items, are given the level *lvl*, until the `@ResetLevel` command resets the level to 0 or another level is set.

See section 2.6 for more information about groups.

2.2.11 @ResetLevel

Resets the current level to 0.

2.2.12 @BeginExample and @EndExample

@BeginExample inserts an example into the manual. The syntax is like the example enviroment in GAPDoc. This examples can be tested by GAPDoc, and also stay readable by GAP. The GAP prompt is added by AutoDoc. @EndExample ends the example block.

```
#! @BeginExample
S3 := SymmetricGroup(5);
#! Sym( [ 1 .. 5 ] )
Order(S3);
#! 120
#! @EndExample
```

2.2.13 @BeginLog and @EndLog

Works just like the @BeginExample command, but the example wont be testet. See the GAPDoc manual for more information.

2.2.14 @DoNotReadRestOfFile

Prevents the rest of the file from being read by the parser. Useful for not finished or temporary files.

```
#! This will appear in the manual

#! @DoNotReadRestOfFile

#! This wont.
```

2.2.15 @BeginChunk *name*, @EndChunk, and @InsertChunk *name*

@BeginChunk causes the next documentation parts not to be inserted in the documentation at it's point in the file, but at the point where the @InsertChunk *name* command is. This can be used to insert examples from different files at a specific point in the documentation. A normal chunk ends at the end of the file. You can also end a system with @EndChunk.

```
#! @BeginChunk MyChunk
#! This is some text.
#! @EndChunk

#! @InsertChunk MyChunk
## Text is inserted here.
```

```
#! @BeginChunk Example_Symmetric_Group
#! @BeginExample
S3 := SymmetricGroup(5);
#! Sym( [ 1 .. 5 ] )
```

```
Order(S3);
#! 120
#! @EndExample
#! @EndChunk
```

```
#! @InsertChunk Example_Symmetric_Group
```

2.2.16 @BeginSystem *name*, @EndSystem, and @InsertSystem *name*

Same as @BeginChunk etc. This command is deprecated. Please use chunk instead.

2.2.17 @BeginCode *name*, @EndCode, and @InsertCode *name*

Inserts the code between @BeginCode and @EndCode verbatim at the point where @InsertCode is called. This is useful to insert your programm code directly into the manual.

```
#! @BeginCode Increment
i := i + 1;
#! @EndCode

#! @InsertCode Increment
## Code is inserted here.
```

2.2.18 @LatexOnly *text*, @BeginLatexOnly , and @EndLatexOnly

Code inserted between @BeginLatexOnly and @EndLatexOnly or after @LatexOnly is only inserted in the PDF version of the manual or worksheet. It can hold arbitrary LaTeX-commands.

```
#! @BeginLatexOnly
#! \include{picture.tex}
#! @EndLatexOnly

#! @LatexOnly \include{picture.tex}
```

2.3 Title page commands

The following commands can be used to add the corresponding parts to the title page of the document, in case the scaffolding is enabled.

- @Title
- @Subtitle
- @Version
- @TitleComment
- @Author
- @Date

- @Address
- @Abstract
- @Copyright
- @Acknowledgements
- @Colophon
- @URL

Those add the following lines at the corresponding point of the titlepage. Please note that many of those things can be (better) extracted from the PackageInfo.g. In case you set some of those, the extracted or in scaffold defined items will be overwritten.

2.4 Plain text files

AutoDoc plain text files work exactly like AutoDoc comments, except that the #! is unnecessary at the beginning of a line which should be documented. Files that have the suffix .autodoc will automatically regarded as plain text files while the commands @AutoDocPlainText and @EndAutoDocPlainText mark parts in plain text files which should be regarded as AutoDoc parts. All commands can be used like before.

2.5 Grouping

TODO: explain more about groups and what they do, how they look in the generated output etc.

Note that group names are globally unique throughout the whole manual. That is, groups with the same name are in fact merged into a single group, even if they were declared in different source files. Thus you can have multiple `@BeginGroup` / `@EndGroup` pairs using the same group name, in different places, and these all will refer to the same group.

Moreover, this means that you can add items to a group via the `@Group` command in the AutoDoc comment of an arbitrary declaration, at any time. The following code

```
#! @BeginGroup Group1

#! @Description
#!  First sentence.
DeclareOperation( "FirstOperation", [ IsInt ] );

#! @Description
#!  Second sentence.
DeclareOperation( "SecondOperation", [ IsInt, IsGroup ] );

#! @EndGroup

## .. Stuff ..

#! @Description
#!  Third sentence.
```

```
#! @Group Group1
KeyDependentOperation( "ThirdOperation", IsGroup, IsInt, "prime );
```

produces the following:

2.5.1 FirstOperation (for IsInt)

▷ `FirstOperation(`*arg*`)` (operation)
▷ `SecondOperation(`*arg1, arg2*`)` (operation)
▷ `ThirdOperation(`*arg1, arg2*`)` (operation)

Returns:
First sentence. Second sentence. Third sentence.

2.6 Level

Levels can be set to not write certain parts in the manual by default. Every entry has by default the level 0. The command `@Level` can be used to set the level of the following part to a higher level, for example 1, and prevent it from being printed to the manual by default. However, if one sets the level to a higher value in the autodoc option of `AutoDoc`, the parts will be included in the manual at the specific place.

```
#! This text will be printed to the manual.
#! @Level 1
#! This text will be printed to the manual if created with level 1 or higher.
#! @Level 2
#! This text will be printed to the manual if created with level 2 or higher.
#! @ResetLevel
#! This text will be printed to the manual.
```

2.7 Some syntax for AutoDoc text

AutoDoc offers some useful syntax extensions which can be used in AutoDoc plain text files and AutoDoc comments. The syntax is inspired by the Markdown language, but it does not implement all features, neither is strict Markdown, but tries to be as straight as possible. The following constructions are possible right now:

2.7.1 Lists

One can create items lists by beginning a new line with *, +, -, followed by one space. The first item starts the list. When items are longer than one line, the following lines have to be indented by at least two spaces. The list ends when a line which does not start a new item is not indented by two spaces. Of curse lists can be nested. Here is an example:

```
#! The list starts in the next line
#! * item 1
#! * item 2
#!   which is a bit longer
#!   * and also contains a nested list
```

```
#!   * with two items
#! * item 3 of the outer list
#! This does not belong to the list anymore.
```

This is the output:
The list starts in the next line

- item 1
- item 2 which is a bit longer
  - and also contains a nested list
  - with two items
- item 3 of the outer list

This does not belong to the list anymore.
The *, -, and + are fully interchangeable and can even be used mixed, but this is not recommended.

2.7.2 Math modes

One can start an inline formula with a $, and also end it with $, just like in LaTeX. This will translate into GAPDocs inline math enviroment. For display mode one can use $$, also like LaTeX.

```
#! This is an inline formula: $1+1 = 2$.
#! This is a display formula:
#! $$ \sum_{i=1}^n i. $$
```

produces the following output:
This is an inline formula: $1+1=2$. This is a display formula:

$$\sum_{i=1}^{n} i.$$

2.7.3 Emphasize

One can emphasize text by using two asteriks (**) or two underscores (__) at the beginning and the end of the text which should be emphasized. Example:

```
#! **This** is very important.
#! This is __also important__.
#! **Naturally, more than one line
#! can be important.**
```

This produces the following output:
This is very important. This is *also important*. *Naturally, more than one line can be important.*

Chapter 3

AutoDoc worksheets

3.1 Worksheets

3.1.1 AutoDocWorksheet

▷ `AutoDocWorksheet(`*`list_of_filenames: options`*`)` (function)

Returns:

This function works exactly like the AutoDoc command, except that no package is needed to create a worksheet. It takes the sampe optional records as the AutoDoc-command, so please refer this command for a full list. It's only optional argument is a (list of) filenames, which are then scanned by the AutoDoc parser.

Chapter 4

AutoDoc

4.1 The AutoDoc() function

4.1.1 AutoDoc

▷ `AutoDoc(`*`[package[, option_record]]`*`)` (function)

Returns: nothing

This is the main function of the AutoDoc package. It can perform any combination of the following three tasks:

1. It can (re)generate a scaffold for your package manual. That is, it can produce two XML files in GAPDoc format to be used as part of your manual: First, a file named `doc/PACKAGENAME.xml` (with your package's name substituted) which is used as main XML file for the package manual, i.e. this file sets the XML doctype and defines various XML entities, includes other XML files (both those generated by AutoDoc as well as additional files created by other means), tells GAPDoc to generate a table of content and an index, and more. Secondly, it creates a file `doc/title.xml` containing a title page for your documentation, with information about your package (name, description, version), its authors and more, based on the data in your `PackageInfo.g`.

2. It can scan your package for AutoDoc based documentation (by using AutoDoc tags and the Autodoc command. This will produce further XML files to be used as part of the package manual.

3. It can use GAPDoc to generate PDF, text and HTML (with MathJaX enabled) documentation from the GAPDoc XML files it generated as well as additional such files provided by you. For this, it invokes `MakeGAPDocDoc` (**GAPDoc: MakeGAPDocDoc**) to convert the XML sources, and it also instructs GAPDoc to copy supplementary files (such as CSS style files) into your doc directory (see `CopyHTMLStyleFiles` (**GAPDoc: CopyHTMLStyleFiles**)).

For more information and some examples, please refer to Chapter 1.

The parameters have the following meanings:

`package`

This is either the name of package, or an `IsDirectory` object. In the former case, AutoDoc uses the metadata of the first package with that name known to GAP. In the latter case, it checks whether the given directory contains a `PackageInfo.g` file, and extracts all needed metadata

from that. This is for example useful if you have multiple versions of the package around and want to make sure the documentation of the correct version is built.

If this argument is omitted, AutoDoc uses the `DirectoryCurrent()`.

option_record

option_record can be a record with some additional options. The following are currently supported:

dir This should be a string containing a (relative) path or a Directory() object specifying where the package documentation (i.e. the GAPDoc XML files) are stored.
Default value: "doc/".

scaffold

This controls whether and how to generate scaffold XML files for the package documentation.

The value should be either `true`, `false` or a record. If it is a record or `true` (the latter is equivalent to specifying an empty record), then this feature is enabled. It is also enabled if *opt.scaffold* is missing but the package's info record in `PackageInfo.g` has an `AutoDoc` entry. In all other cases (in particular if *opt.scaffold* is `false`), scaffolding is disabled.

If scaffolding is enabled, and *PackageInfo.AutoDoc* exists, then it is assumed to be a record, and its contents are used as default values for the scaffold settings.

If *opt.scaffold* is a record, it may contain the following entries.

includes

A list of XML files to be included in the body of the main XML file. If you specify this list and also are using AutoDoc to document your operations with AutoDoc comments, you can add `_AutoDocMainFile.xml` to this list to control at which point the documentation produced by AutoDoc is inserted. If you do not do this, it will be added after the last of your own XML files.

index

By default, the scaffold creates an index. If you do not want an index, set this to `false`.

appendix

This entry is similar to *opt.scaffold.includes* but is used to specify files to include after the main body of the manual, i.e. typically appendices.

bib

The name of a bibliography file, in Bibtex or XML format. If this key is not set, but there is a file `doc/PACKAGENAME.bib` then it is assumed that you want to use this as your bibliography.

TitlePage

A record whose entries are used to embellish the generated titlepage for the package manual with extra information, such as a copyright statement or acknowledgments. To this end, the names of the record components are used as XML element names, and the values of the components are outputted as content of these XML elements. For example, you could pass the following record to set a custom acknowledgements text:

```
rec( Acknowledgements := "Many thanks to ..." )
```

For a list of valid entries in the titlepage, please refer to the GAPDoc manual, specifically section (**GAPDoc: TitlePage**).

MainPage

If scaffolding is enabled, by default a main XML file is generated (this is the file which contains the XML doctype and more). If you do not want this (e.g. because you have a hand written main XML file), but still want AutoDoc to generate a title page for you, you can set this option to `false`

document_class

Sets the document class of the resulting pdf. The value can either be a string which has to be the name of the new document class, a list containing this string, or a list of two strings. Then the first one has to be the document class name, the second one the option string (contained in [ ]) in LaTeX.

latex_header_file

Replaces the standard header from GAPDoc completely with the header in this LaTeX file. Please be careful here, and look at GAPDoc's latexheader.tex file for an example.

gapdoc_latex_options

Must be a record with entries which can be understood by SetGapDocLaTeXOptions. Each entry can be a string, which will be given to GAPDoc directly, or a list containing of two entries: The first one must be the string "file", the second one a filename. This file will be read and then its content is passed to GAPDoc as option with the name of the entry.

autodoc

This controls whether and how to generate addition XML documentation files by scanning for AutoDoc documentation comments.

The value should be either `true`, `false` or a record. If it is a record or `true` (the latter is equivalent to specifying an empty record), then this feature is enabled. It is also enabled if *opt.autodoc* is missing but the package depends (directly) on the AutoDoc package. In all other cases (in particular if *opt.autodoc* is `false`), this feature is disabled.

If *opt.autodoc* is a record, it may contain the following entries.

files

A list of files (given by paths relative to the package directory) to be scanned for AutoDoc documentation comments. Usually it is more convenient to use *autodoc.scan_dirs*, see below.

scan_dirs

A list of subdirectories of the package directory (given as relative paths) which AutoDoc then scans for .gi, .gd and .g files; all of these files are then scanned for AutoDoc documentation comments.

Default value: ["gap", "lib", "examples", "examples/doc"].

level

This defines the level of the created documentation. The default value is 0. When parts of the manual are declared with a higher value they will not be printed into the manual.

gapdoc

This controls whether and how to invoke GAPDoc to create HTML, PDF and text files from your various XML files.

The value should be either `true`, `false` or a record. If it is a record or `true` (the latter is equivalent to specifying an empty record), then this feature is enabled. It is also enabled if *opt.gapdoc* is missing. In all other cases (in particular if *opt.gapdoc* is `false`), this feature is disabled.

If *opt.gapdoc* is a record, it may contain the following entries.

main

The name of the main XML file of the package manual. This exists primarily to support packages with existing manual which use a filename here which differs from the default. In particular, specifying this is unnecessary when using scaffolding.
Default value: PACKAGENAME.xml.

files

A list of files (given by paths relative to the package directory) to be scanned for GAPDoc documentation comments. Usually it is more convenient to use *gapdoc.scan_dirs*, see below.

scan_dirs

A list of subdirectories of the package directory (given as relative paths) which AutoDoc then scans for .gi, .gd and .g files; all of these files are then scanned for GAPDoc documentation comments.
Default value: ["gap", "lib", "examples", "examples/doc"].

maketest

The maketest item can be true or a record. When it is true, a simple maketest.g is created in the main package directory, which can be used to test the examples from the manual. As a record, the entry can have the following entries itself, to specify some options.

filename

Sets the name of the test file.

commands

A list of strings, each one a command, which will be executed at the beginning of the test file.

Index

AutoDoc, 4

AProperty
    for IsObject, 9
    testlabel, 9
AutoDoc, 18
AutoDocWorksheet, 17

FirstOperation
    for IsInt, 15

SecondOperation
    for IsInt, IsGroup, 15

ThirdOperation
    for IsGroupIsGroup, , 15