ROBODoc 4.99.19 User Manual

Ed. Users Guide $Revision: 1.46 $

COLLABORATORS

	TITLE : ROBODoc 4.99.19 User Manual		*REFERENCE* :
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Frans Slothouber and Petteri Kettunen	December 15, 2005	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Preface

ROBODoc is a API documentation tool for C, C++, Java, Assembler, Basic, Fortran, LaTeX, Postscript, Tcl/Tk, LISP, Forth, Perl, Shell Scripts, Makefiles, Occam, COBOL, DCL, Visual Basic, HTML, DB/C, XML, and many other languages. It can be made to work with any language that supports comments.

ROBODoc works by extracting specially formated headers from your source code and writes these to documentation files. These files can be formatted in HTML, ASCII, XML DocBook, or RTF; and indirect to PDF.

ROBODoc is similar to JavaDoc, though the idea is much older than JavaDoc. ROBODoc allows you to maintain a program and its documentation in a single file. This makes it easier to keep your documentation up-to-date.

ROBODoc can be used to document anything you like, functions, methods, variables, definitions, test cases, makefile entries, and anything else you can think of.

It can create documentation consisting of many small files. For instance in HTML format for easy browsing and publication on the web. It can also create a single file in LaTeX or RTF format for inclusion into bigger design documents. The RTF format is suited to be included in Word documents.

ROBODoc allows you to separate internal documentation from external documentation. In singledoc mode it can create a section layout based on the hierarchy of your modules.

ROBODoc is designed to work with a lot of different programming languages. It has no knowledge of the syntax of a programming languages. It only has some knowledge about how remarks start and end in a lot of programming languages. This means that you sometimes have to do a little more work compared to other tools that have detailed knowledge of the syntax of a particular language. They can use that knowledge to figure out some of the information automatically. This usually also means that they work only with one or two languages.

2 Installing ROBODoc

The easiest way to install ROBODoc is to use one of the packages. There are package for RedHat, Debian, OSX, and a precompiled executable for Windows.

You can also compile robodoc from the sources. On a system with **autoconfig** it is as simple as:

```
./configure
make
make install
```

On a Windows system with VC++ you have two options. Either use the following commands in the source directory (tested under NT):

```
vcvars32
nmake -f makefile.win32
```

Or use the supplied project file `robodoc.dws` in the `Win32` directory. This only works if the `.dsp` and `.dws` have the right format. Both files should use the windows convention for line-endings (cr/lf). Sometimes these get lost while zipping or unzipping the ROBODoc archive, and in that case the project will turn op empty.

There is also a makefile for Borland C, as well as for MINGW. For other compilers you might want to try `makefile.plain`.

You can test your executable, by going to the `Examples/PerlExample` directory in the archive, and running robodoc. This should create a directory called `Doc`. In there you should now find a file called `masterindex.html`.

3 Preparing your source code for ROBODoc

ROBODoc allows you to mix the program documentation with the source code. It does require though that this documentation has a particular layout so ROBODoc can recognize it. There are three key concepts: headers, items, and sections.

3.1 Headers

Headers are the building blocks of the documentation. Lets look at an example. The following header was taken from the documentation of the predecessor of ROBODoc, AutoDoc.

```
/****f* financial.library/StealMoney
 *  NAME
 *    StealMoney -- Steal money from the Federal Reserve Bank. (V77)
 *  SYNOPSIS
 *    error = StealMoney( userName, amount, destAccount, falseTrail )
 *  FUNCTION
 *    Transfer money from the Federal Reserve Bank into the
 *    specified interest-earning checking account.  No records of
 *    the transaction will be retained.
 *  INPUTS
 *    userName    - name to make the transaction under.  Popular
 *                  favorites include "Ronald Reagan" and
 *                  "Mohamar Quadaffi".
 *    amount      - Number of dollars to transfer (in thousands).
 *    destAccount - A filled-in AccountSpec structure detailing the
 *                  destination account (see financial/accounts.h).
 *                  If NULL, a second Great Depression will be
 *                  triggered.
 *    falseTrail  - If the DA_FALSETRAIL bit is set in the
 *                  destAccount, a falseTrail structure must be
 *                  provided.
 *  RESULT
 *    error - zero for success, else an error code is returned
 *           (see financial/errors.h).
 *  EXAMPLE
 *    Federal regulations prohibit a demonstration of this function.
 *  NOTES
 *    Do not run on Tuesdays!
 *  BUGS
 *    Before V88, this function would occasionally print the
 *    address and home phone number of the caller on local police
 *    976 terminals.  We are confident that this problem has been
 *    resolved.
 *  SEE ALSO
 *    CreateAccountSpec(),security.device/SCMD_DESTROY_EVIDENCE,
 *    financial/misc.h
 ******
 * You can use this space for remarks that should not be included
 * in the documentation.
 */
```

A header consists of three different elements. A begin marker, a number of items, and an end marker. The begin marker in the example is example is:

```
 ****f* financial.library/StealMoney
```

It marks the that marks the begin of a header. It also tells ROBODoc the name of the element that is being documented, StealMoney, the module it is part of, financial.library, and the kind of element, `f`, which stands for function. ROBODoc always expects a module name and an element name separated by a `/`. So `ModFoo/funcBar` is a valid name, but `funcBar` is not. See Sections for more information.

The end marker:

```
 ******
```

marks the end of a header.

Items begin with an item name and are followed by the item's body. An example:

```
  *  FUNCTION
  *    Transfer money from the Federal Reserve Bank into the
  *    specified interest-earning checking account.  No records of
  *    the transaction will be retained.
```

In this case the item's name is FUNCTION.

Each line of an item starts with a remark marker. In this case `*`.

The above example is a C example. ROBODoc supports many more languages though. The following table shows all the markers that ROBODoc supports by default.

```

/****       C, C++
 *
 ***/

//****      C++
//
//***

(****       Pascal, Modula-2
 *
 ***
 *)

{****       Pascal
 *
 ***
 *}

;****       M68K assembler
;
;***

****        M68K assembler, COBOL
*
***

C     ****  Fortran
C
C     ***

REM ****    BASIC
REM *
REM ***

%****       LaTeX, TeX, Postscript
%
%***

#****       Tcl/Tk
#
#***

--****      Occam
--
--***

<!--****     HTML Code
*
***
```

```
<!---****    HTML Code
*
***

|****       GNU Assembler
|
|***

$!****      DCL
$!
$!***

'****       Visual Basic, Lotus script
'*
'***

.****       DB/C
.*
.***

!!****      FORTRAN 90
!!
!!***

!****       FORTRAN 90
!
!***
```

Any of these markers can be mixed, and they are not limited to the languages listed. So if you have a language that is not listed but that has remarks that start with a # you can use the Tcl/Tk markers, and create headers such as:

```
#****f* Foo/Bar
# FUNCTION
#   Bar snarfs the Foo input and mangles it.  Given the right settings
#   it might also do a bit of snu snu.
#***
```

3.2 Header Types

ROBODoc defines a number of header types. You don't need to use them but they can be useful for sorting information. The headertype tells ROBODoc what kind of object you are documenting. This information allows ROBODoc to create more useful index tables.

The type is identified by one or two characters. ROBODoc expects to find them after the fourth `*` in the begin marker. So `#****f` is a valid marker, but `#**f**` is not.

If a single character is given, the type is defined as listed in the following table

- c -- Header for a class.
- d -- Header for a constant (from define).
- f -- Header for a function.
- h -- Header for a module in a project.
- m -- Header for a method.
- s -- Header for a structure.

- t -- Header for a types.
- u -- Header for a unittest.
- v -- Header for a variable.
- * -- Generic header for every thing else.

If two characters are given, the first character should be `i` and the second can be any of the other characters from the table above. This creates an internal header of the type specified by the second character. Internal headers are special. They can be used to hide certain headers. They are only extracted if requested. You can use them to document internal functions, classes, etc. that you do not want clients to see, creating what might be a programmer's manual as opposed to a user's manual.

So `/****if* Module/func1` defines an internal function called `func1`.

Headers marked internal are by default not included in the generated documentation. If you want to include them use the option `--internal`. You can also generate the documentation from only the internal headers with the option `--internalonly`.

You can define your own headertypes using the ROBODoc configuration file, `robodoc.rc`. See Customizing ROBODoc. This way you can document anything you like, for instance makefile entries, system tests, or exceptions.

3.3 Items

By default ROBODoc recognizes the following items:

- NAME -- Item name plus a short description.
- COPYRIGHT -- Who own the copyright : "(c) <year>-<year> by <company/person>"
- SYNOPSIS, USAGE -- How to use it.
- FUNCTION, DESCRIPTION, PURPOSE -- What does it do.
- AUTHOR -- Who wrote it.
- CREATION DATE -- When did the work start.
- MODIFICATION HISTORY, HISTORY -- Who has done which changes and when.
- INPUTS, ARGUMENTS, OPTIONS, PARAMETERS, SWITCHES -- What can we feed into it.
- OUTPUT, SIDE EFFECTS -- What output is made.
- RESULT, RETURN VALUE -- What do we get returned.
- EXAMPLE -- A clear example of the items use.
- NOTES -- Any annotations
- DIAGNOSTICS -- Diagnostic output
- WARNINGS, ERRORS -- Warning and error-messages.
- BUGS -- Known bugs.
- TODO, IDEAS -- What to implement next and ideas.
- PORTABILITY -- Where does it come from, where will it work.
- SEE ALSO -- References to other functions, man pages, other documentation.
- METHODS, NEW METHODS -- OOP methods.
- ATTRIBUTES, NEW ATTRIBUTES -- OOP attributes
- TAGS -- Tag-item description.

- COMMANDS -- Command description.
- DERIVED FROM -- OOP super class.
- DERIVED BY -- OOP sub class.
- USES, CHILDREN -- What modules are used by this one.
- USED BY, PARENTS -- Which modules do use this one.
- SOURCE -- Source code inclusion.

You can define your own items using the ROBODoc configuration file, `robodoc.rc`. See Customizing ROBODoc.

3.4 Sections

The structure of source code for an project is usually hierarchical. A project might consists of several applications, an application of several modules, a module of several functions or even submodules. ROBODoc allows you to show this hierarchy in your documentation. For this you specify the hierarchy in the header name. For instance, you have a project that is going to create a new language called D. The D Language project might consists of three applications: a preprocessor, a compiler, and a linker. The compiler consists of two modules, a parser and a generator. The parser module consists of several functions.

The following three headers show how this hierarchy can be defined in the header name.

```
#****h* D-Language/Compiler
# FUNCTION
#   The compiler takes a preprocessed source file and
#   turns it into an object file.
#***
```

```
#****h* D-Language/Linker
# FUNCTION
#   The linker module contains functions that scan a
#   object file and build the executable.
#***
```

```
#****h* Compiler/Parser
# FUNCTION
#   The parser module contains functions that scan a
#   preprocessed source file and build the syntax tree.
#***
```

```
#****f* Parser/ReadToken
# FUNCTION
#   ReadToken reads the next token from the input
#   file.
#***
```

When you generate documentation with the option `--section`, ROBODoc uses the hierarchical information when generating the table of content and document section information. For instance in HTML sections are started with <H1>, <H2>, <H3> depending on the level in the hierarchy. The table of will also contain levels. The table of contents for the above example will be:

```
1. D-Language/Compiler
1.1 Compiler/Parser
1.1.1 Parser/ReadToken
2. D-Language/Linker
```

3.5 Text Formatting

By default ROBODoc creates preformatted text in the output documentation for all the text it finds in an item. This means that the formatting of the output looks the same as the formatting of the text of an item. Line-breaks and indentation stay the same. This is easy but does not always creates the best looking output.

ROBODoc can also try to deduce the formatting of your text based on the indentation of your text and on special characters in the text. It works a bit similar to the input method of Wikis.

You switch this on with the option `--nopre`. ROBODoc now tries to find three kind of elements: paragraphs, lists, and preformatted text.

Paragraphs are separated by empty lines. So the following item has two paragraphs.

```
FUNCTION
  This function does something.

  And it then does something else
  and a bit more.
```

A List starts with a line that ends with a ':' which is then followed by one or more list items. List items should start with '*', '+', or 'o'. So the following item contains a valid list:

```
FUNCTION
   This function does:
   * a lot of foo
   * some snafuing
   * and a bit of foobarring.
```

A list item can span than more line of the second and following lines are indented. So this is also a valid list:

```
FUNCTION
   This function does:
   * a lot of foo and preprossesing of the
     raw input with the aid of the some magic
   * some snafuing
   * and a bit of foobarring.
```

If listitems directly follow the name of a robodoc item they also form a valid list. So this is a valid list:

```
INPUTS
  * inputname -- the name of the input file
  * outputname -- the name of the output file
  * n -- the number of characters to be written
```

Preformatted text is indicated by indenting it more that the surrounding text. The first non-empty line in an item determines the base indenting. Any lines with an indentation larger than this are preformatted. An example:

```
FUNCTION
  The following lines are preformatted
     +--------+
     | A box  |
     |        |
     +--------+
  End of the preformatted text.
```

The following is complete example and its translation into HTML.

```
This is some example text.
And some more.
```

```
This is even more, and we start a list:
* a list item
* a list item
* a list item

And we can also do preformatted stuff
by indenting
    +--------+
    |        |
    +--------+
The box will stay.
```

will be turn into

```
<p>This is some example text.
And some more.</p>

<p>This is even more, and we start a list:</p>
<ul>
<li>a list item</li>
<li>a list item</li>
<li>a list item</li>
</ul>

<p>And we can also do preformatted stuff
by indenting</p>
<pre>
    +--------+
    |        |
    +--------+
</pre>
<p> The box will stay.</p>
```

4 Extracting Documentation with ROBODoc

Now that you have prepared your source code for use with ROBODoc you are ready to extract the documentation. There are several choices to be made.

4.1 Single document or many smaller documents

First of all, ROBODoc can be used in three modes.

- multidoc -- in this mode ROBODoc scans all the source files in your source directory and creates a separate document file for each of these in a document directory. The document directory is created automatically. Its structure is a mirror of the structure of your source directory.
- singledoc -- in this mode ROBODoc scans all the source files in your source directory and creates a single documentation file that contains all the documentation extracted from your source files.
- singlefile -- in this mode ROBODoc scans a single source file and creates a single documentation file.

4.2 multidoc

The multidoc mode is useful to create browsable documents. For instance many small HTML files that can be viewed with a web-browser. This mode requires the following arguments:

`robodoc` --src *`source directory`* --doc *`document directory`* --multidoc [other options]

An additional option that is useful with this mode is `--index`, this creates a series of index files, one for each header type.

4.3 singledoc

The singledoc mode is useful to create bulk documentation that can be incorporated in other documents, or that can be delivered to a client as a single document. For instance a file created in RTF format can be included into a larger design document written in Word format. This mode requires the following arguments:

`robodoc` --src *`source directory`* --doc *`document file without extension`* --singledoc [other options]

An additional option that is useful with this mode is `--sections`, this causes the headers to follow a section layout based on the module element hierarchy defined in the header name.

4.4 singlefile

The singlefile mode is not very useful. It is mainly used for debugging purposes. This mode requires the following arguments:

`robodoc` --src *`source file`* --doc *`document file`* --singlefile [other options]

4.5 Output formats

Your next choice is the output format. ROBODoc can create documentation in several formats:

- HTML, option `--html`
- RTF, option `--rtf`
- LaTeX, option `--latex`
- XML DocBook, option `--dbxml`

What format to use depends on your wishes. If you want a single printable document, use LaTeX or XML DocBook. If you want a document that can be included into a larger (Word) document use RTF. If you want something that is browsable use HTML, or use XML DocBook and then convert it to HTML.

4.6 Options

The behavior of ROBODoc can be further fine-tune with a large number of options.

4.6.1 -c

Show the copyright message.

4.6.2 --cmode

Use ANSI C grammar in SOURCE items and use this for some syntax highlighting (HTML only).

4.6.3 --css

Use to content of the specified file to create the `robodoc.css`. The content of the file is copied into `robodoc.css`.

4.6.4 --dbxml

Generate documentation in XML DocBook format.

4.6.5 --debug

Works like --tell, bug gives a lot more information.

4.6.6 --doc

Define the path to the documentation directory or documentation file. A path can start with `./` or `/`. Do not use `..` in the path. The documentation directory can be a subdirectory of the source directory, or be parallel to the source directory, however they can not be equal. So **--src ./sources** together with **--doc ./documents** is fine, but **--src ./Everything** together with **--doc ./Everything** is not.

4.6.7 --folds

Use fold marks to split a big document into smaller ones.

4.6.8 --headless

Do not create the head of a document. This allows you to create documents that can be included in other documents, or to which you can add your own style.

For html output this means that no `<html><head> ..... <body>` is generated.

For LaTeX output this means none of the document initialization code is generated, such as `\documentclass{article}` or `\begin{document}` is generated. If you use this option in combination with `--footless` you can use `\include` or `\input` commands to include the ROBODoc generated documents in a larger document.

For XML DocBook output this means no `<!DOCTYPE>`, `<article>`, and `<articleinfo>` is generated.

4.6.9 --footless

Do not create the foot of a document. This allows you to create documents that can be included in other documents, or to which you can add your own style.

For html output this means that no `</body></html>` is generated.

For LaTeX output this means no `\end{document}` is generated.

For XML DocBook output this means no `</article>` is generated.

4.6.10 --html

Generate documentation in HTML format.

4.6.11 --ignore_case_when_linking

Ignore differences in case when creating crosslinks. This is handy for languages such as Fortran or Pascal, but in most cases it is better not to use it.

4.6.12 --internal

Also include headers marked internal.

4.6.13 --internalonly

Only include headers marked internal.

4.6.14 --index

Also create a master index file.

4.6.15 --lock

Per source file robodoc locks on the first headermarker it finds and will recognize only that particular headermarker for the remaining part of the file. In addition it locks on the first remark marker in each header and will recognize only that particular remark marker for the remaining part of the header.

4.6.16 --multidoc

Generate one document per source file, and copy the directory hierarchy.

4.6.17 --nosource

Do not include the SOURCE items.

4.6.18 --nodesc

Do not scan any subdirectories, scan only the top level directory of the source tree.

4.6.19 --nosort

Do not sort the headers when generating the documentation. The headers will appear in the same order in the documentation as they appear in the source code.

4.6.20 --nopre

With this option ROBODoc does not generate preformatted text when creating the item documentation. (Using the `<PRE>` and `</PRE>` construction in HTML format for instance). Instead ROBODoc tries to deduce the formatting from the indentation and special characters. See Text Formatting. This creates much better looking documentation.

4.6.21 --nogeneratedwith

Do not add the "generated with robodoc" message at the top of each documentation file.

4.6.22 --rc

Use the specified file instead of `robodoc.rc`.

4.6.23 --rtf

Generate documentation in RTF format.

4.6.24 --sections

Create sections based on the module hierarchy.

4.6.25 --sectionnameonly

ROBODoc generates the section headers with names only, no chapter numbers, no parent section names.

4.6.26 --singledoc

Define the documentation directory or documentation file.

4.6.27 --singlefile

Generate a single document from a single file

4.6.28 --src

Define the path for the source directory or source file. The path can start with `./` or `/`. Do not use `..` in the path.

4.6.29 --tabsize

Lets you specify the tabsize.

4.6.30 --toc

Add a table of contents. This works in multidoc mode as well as singledoc mode.

4.6.31 --latex

Generate documentation in LaTeX format.

4.6.32 --tell

ROBODoc tells you what steps it is taking.

5 Customizing ROBODoc

ROBODoc can be configured with a configuration file called `robodoc.rc`. With it you can define item names, frequently used options, and translations for English terms. One should note that if configuration file is specified, its definitions over-ride ROBODoc internal (i.e. built-in) settings. This is a feature; some arbitrary langugage may include syntax which conflicts with ROBODoc's internal markers. By taking use of a configuration file, these sort of issues and conflicts can be circumvented. An example is shown below.

```
# Example robodoc.rc
#
items:
    NAME
    SYNOPSIS
    INPUTS
    OUTPUTS
    SIDE EFFECTS
    HISTORY
    BUGS
ignore items:
    HISTORY
    BUGS
source items:
    SYNOPSIS
options:
    --src ./source
    --doc ./doc
    --html
    --multidoc
    --index
    --tabsize 8
headertypes:
    e  "Makefile Entries"  robo_mk_entries
    x  "System Tests"      robo_syst_tests
    q  Queries             robo_queries
ignore files:
    README
    CVS
    *.bak
    *~
    "a test_*"
accept files:
    *.c
    *.h
    *.pl
header markers:
    /****
    #****
remark markers:
    *
    #
end markers:
    ****
    #****
remark begin markers:
    /*
remark end markers:
    */
```

The configuration file consists of a number of blocks. Each block starts with a name followed by a `:`. There are currently eight blocks: items, ignore items, options, header types, ignore files, header markers, remark markers, and end markers. In each block you define a number of values. Each value must start with at least one space.

5.1 items block

In this block you can define the names of items that ROBODoc should recognize. Item names can consist of more than one word but should be written in all uppercase characters. Define one item name per line. You do not need to put quotes around them if they contain spaces.

If you do not define an items block ROBODoc uses its default list of item names. If you define an items block ROBODoc uses only those item names, any of the default items names (except SOURCE) are no longer recognized.

5.2 ignore items block

In this block you can define the names of items that ROBODoc should ignore when generating documentation. This can be useful if you want to create documentation for a client, but for instance do not want to include the history items and bugs items.

5.3 source items block

In this block you can define the names of items that ROBODoc handle in the same way as the SOURCE item.

5.4 options block

In this block you can define frequently used options. The options you specify here are added to any options you specify on the command line.

5.5 headertypes block

In this block you can define your own headertypes. These are added to the existing headertypes. Each new headertype requires three parameters: the character used to indicate a header of this type, the title for this type as used in the master index, the name of the file in which the index this type is stored. If you use a character of an existing headertype, this headertype is overwritten.

5.6 ignore files block

In this block you can define the names of files or directories that ROBODoc should ignore while scanning the directory tree for source files. You can use the wildcard symbols `*` and `?`. If you use spaces in the expression enclose the experssion in quotes.

For instance, the rc file above causes ROBODoc to skip all `README` files, all files with the name `CVS` (these are usually directories). It also skips all files with a name that ends with `.bak` or or that start with `a test_`

Normally you specify the names of directories here and do the filtering of file names with a accept files block.

5.7 accept files block

In this block you can define the names of files that robodoc should accept. This test is carried out after the ’ignore files’ filtering is performed. Any files that do not match the patterns in this block are ignored by ROBODoc.

If there is no accept files block all files are accepted.

5.8 header markers

In this block you define what ROBODoc will recognize as start of a header. If you use this block ROBODoc only recognizes these markers, any of the inbuild markers will be ignored.

5.9 remark markers

In this block you define what ROBODoc will recognize as start of remark. If you use this block ROBODoc only recognizes these markers, any of the inbuild markers will be ignored.

5.10 end markers

In this block you define what ROBODoc will recognize as end of a header. If you use this block ROBODoc only recognizes these markers, any of the inbuild markers will be ignored.

5.11 remark begin markers

In this block you define what ROBODoc will recognize as the begin of a remark block. This is used in combination with the source items block. This allows robodoc to recognize that a header ends and source begins. For example it makes it possible that ROBODoc interprets `int foo( float correction )` as code in the following header.

```
/****f* Bar/foo
 * FUNCTION
 *   foo computues the foo factor.
 * SYNOPSIS
 */
int foo( float correction )
/*
 * BUGS
 *   no bugs.
 *****/
{
    return correction * 42.0;
}
```

5.12 remark end markers

In this block you define what ROBODoc will recognize as the begin of a remark block. See the previous block.

5.13 Configuration file location

ROBODoc searches the your current directory for the `robodoc.rc` file. If it can't find it there it will search in `$HOME/`, then `$HOMEDRIVE$HOMEPATH/`, and finally in `/usr/share/robodoc/`.

With the `--rc` option can tell ROBODoc to use a different file then `robodoc.rc` as configuration file. The is handy if you want to create documentation in different formats. For instance:

```
robodoc --rc  htmlsingle.rc
robodoc --rc  rtfsingle.rc
robodoc --rc  htmlmulti.rc
```

6 Examples

6.1 HTML Example

For this you need a web browser, say FireFox or Mozilla. You can try this in the robodoc root directory. It creates a document called `HDocs/masterindex.html` plus a lot of smaller documents from all the source files in the directory `Source`.

```
robodoc --src ./Source --doc ./HDocs --multidoc --index --html
```

6.2 RTF Example

For this you need an rtf reader, for instance **Word**. You can try this in the robodoc root directory.

```
robodoc --src ./Source --doc api --singledoc --rtf --sections
```

This will create a document called `api.rtf`.

By default the document looks pretty plain. There is no chapter numbering or a table of contents, even if you asked for it. All the information for this is included but not visible. This is because chapter numbering and a table of contents are generated by Word based on formatting information that is part of a Word document but not part of a RTF document.

To make it visible you include the generated document into a bigger document with the right formatting options. This is best done with a cut-and-paste operation. Use the cut-past-paste special menu, and paste it as RTF formatted text into your Word document.

6.3 LaTeX Example

For this you need **latex** and **makeindex**. You can try this in the robodoc root directory. It creates a single document called `api.dvi` from all the source files in the directory Source.

```
robodoc --src ./Source --doc api --singledoc --latex --sections
latex api.tex
latex api.tex
makeindex api.idx
latex api.tex
xdvi api.dvi
```

6.4 XML DocBook Example

For this you need **xmlto** You can try this in the robodoc root directory. It creates a single document called `api.xml` from all the source files in the directory Source. The **xmlto** then creates a browsable HTML document from this.

```
robodoc --src ./Source --doc api --singledoc --dbxml --sections
xmlto html api.xml
```

To create a PDF document use:

```
xmlto pdf api.xml
```

7 Tips and Tricks

7.1 Using ROBODoc under Windows

When you use ROBODoc under windows, don't forget that it is a command line tool. ROBODoc relies on the console window to inform you about problems and errors.

An easy mistake to make it to create a shortcut to `robodoc.exe` and then click on the icon to generate the documentation each time you made some changes to your source code. If you have a fast machine a console window pops up quickly and after that your documentation is ready.

This works very fine until you make a mistake in one of your headers. The console window still pops up, but before you have a chance to read any of the error messages it is gone again. Most likely you won't even have noticed there were error messages. You will end up with empty documentation or old documentation.

Better is to create a batch file with the following commands and to store all the options in a `robodoc.rc` file:

```
robodoc.exe
pause
```

Now the console window stays open and you have the opportunity to read the error messages.

While the window is open, right click on the title bar, go to properties->layout and set the buffer size to something like 2500. That way you can scroll back too the next time you run it.

7.2 The SOURCE Item

With a little extra work you can include part of your source code into your documentation to. The following example shows how this is done:

```
/****f* Robodoc/RB_Panic [2.0d]
 * NAME
 *   RB_Panic - Shout panic, free resources, and shut down.
 * SYNOPSIS
 *   RB_Panic (cause, add_info)
 *   RB_Panic (char *, char *)
 * FUNCTION
 *   Prints an error message.
 *   Frees all resources used by robodoc.
 *   Terminates program.
 * INPUTS
 *   cause    - pointer to a string which describes the
 *              cause of the error.
 *   add_info - pointer to a string with additional information.
 * SEE ALSO
 *   RB_Close_The_Shop ()
 * SOURCE
 */

  void RB_Panic (char *cause, char *add_info)
  {
    printf ("Robodoc: Error, %s\n",cause) ;
    printf ("         %s\n", add_info) ;
    printf ("Robodoc: Panic Fatal error, closing down..\n") ;
    RB_Close_The_Shop () ; /* Free All Resources */
    exit(100) ;
  }

/*******/
```

You add a SOURCE item as the last item of your header. Then instead of closing your header with an end marker, you close it normally. The end marker is instead placed at the end of the fragment of source code that you want to include.

Note that ROBODoc will skip the first line after the line that contains `SOURCE`. In this case the line with `*/`. This is to ensure that this line does not end up in the documentation. For C sources this is obvious, but for Tcl sources for instance, it is not. Tcl does not need a seperate line to close of a remark. So a Tcl programmer would be tempted to write:

```
#****f* ModA/foobar
# FUNCTION
#   foobar computes the foo factor of a given file.
# SOURCE
sub foobar( f )
{

}
#****
```

and end up without `sub foobar( f )` in the documentation.

SOURCE items are included by default. If want to create a document without the SOURCE items use the option `--nosource`.

7.3 Minimizing Duplicate Information

It is always good to avoid having the same information in several different locations. It is easy to create headers that have a lot information duplication. Take for instance the following header.

```
/****f* Robodoc/RB_Panic [2.0d]
 * NAME
 *   RB_Panic - Shout panic, free resources, and shut down.
 * SYNOPSIS
 *   RB_Panic (cause, add_info)
 *   RB_Panic (char *, char *)
 * FUNCTION
 *   Prints an error message.
 *   Frees all resources used by robodoc.
 *   Terminates program.
 * INPUTS
 *   cause    - pointer to a string which describes the
 *              cause of the error.
 *   add_info - pointer to a string with additional information.
 * SEE ALSO
 *   RB_Close_The_Shop ()
 * SOURCE
 */

  void RB_Panic (char *cause, char *add_info)
  {
    printf ("Robodoc: Error, %s\n",cause) ;
    printf ("         %s\n", add_info) ;
    printf ("Robodoc: Panic Fatal error, closing down..\n") ;
    RB_Close_The_Shop () ; /* Free All Resources */
    exit(100) ;
  }

/*******/
```

The name `RB_Panic` occurs five times. This is tedious to type and difficult to maintain. However with a the right `robodoc.rc` this can be changed to:

```
/****f* Robodoc/RB_Panic [2.0d]
 * SUMMARY
 *   Shout panic, free resources, and shut down.
 * SYNOPSIS
 */

void RB_Panic (char* cause, char *add_info)

/*
 * FUNCTION
 *   Prints an error message.
 *   Frees all resources used by robodoc.
 *   Terminates program.
 * INPUTS
 *   cause    - pointer to a string which describes the
 *              cause of the error.
 *   add_info - pointer to a string with additional information.
 * SEE ALSO
 *   RB_Close_The_Shop ()
 * SOURCE
 */
  {
    printf ("Robodoc: Error, %s\n",cause) ;
    printf ("         %s\n", add_info) ;
    printf ("Robodoc: Panic Fatal error, closing down..\n") ;
    RB_Close_The_Shop () ; /* Free All Resources */
    exit(100) ;
  }

/*******/
```

`RB_Panic` occurs only twice now. In addition changes to the function definition only have to be done once.

The `robodoc.rc` required for this is:

```
# robodoc.rc file
items:
    SUMMARY
    SYNOPSIS
    INPUTS
    OUTPUTS
    SEE ALSO
    BUGS
source items:
    SYNOPSIS
remark begin markers:
    /*
remark end markers:
    */
```

7.4 Advanced formating with raw HTML and LaTeX code

By default an item's body shows up in your documentation in the same way as it is formatted in your source code. All special characters for the output mode are escaped. For instance an < is translated to a &lt; in HTML mode. Sometimes however you like to have some more control of what goes into the documentation. This is possible with the piping. If a line of your item's body starts with on of the special piping markers, the text after this marker is copied verbatim into your documentation. The following example shows how this is done, and how to add equations to your documentation.

```
/****m* pipe/pipetest
 * NAME
 *   pipetest
 * NAME
 *   Simple header to show "piping" features in items.
 * EXAMPLE
 *   Only "pipes" which match selected output style are picked up.
 *   |html <CENTER>This will be included in <B>HTML</B> output.</CENTER>
 *   |latex \centerline{This will be included in \LaTeX output}
 *   Space is mandatory following the pipe marker. The following is not a
 *   valid pipe marker:
 *   |html<B>Moi!</B>
 *   You should see an equation on the following line:
 *   |html y = x^2 (sorry, plain HTML is not very powerful)
 *   |latex \centerline{$y = x^2$}
 *   How does this look like?
 *   Here comes a multi-line equation array:
 *    |latex \begin{eqnarray}
 *    |latex \frac{\partial u}{\partial \tau} & = & D_u {\nabla}^2 u +
 *    |latex \frac{1}{\epsilon}
 *    |latex \left ( \hat{u}-{\hat{u}}^2-f\, {v} \, \frac{\hat{u}-q}{\hat{u}+q}
 *     |latex \right ) \; ,  \label{diffspot:1} \\
 *     |latex \frac{\partial v}{\partial \tau} & = & \hat{u}-v \; ,
 *     |latex \label{diffspot:2} \\
 *     |latex \frac{\partial r}{\partial \tau} & = & D_r {\nabla}^2 r \; .
 *     |latex \label{diffspAot:3}
 *     |latex \end{eqnarray}
 *     |html <I>TODO: write this in html</I>
 *   End of the example.
 ******
 */
```

7.5 Linking to external documents (href, file, mailto, images)

In HTML mode ROBODoc recognizes the following links to external documents.

- `href:body` -- This is replaced with `<a href="body">body</A>`
- `file:/body` -- This is replaced with `<a href="file:/body">file:/body</A>`
- `mailto:body` -- This is replaced with `<a href="mailto:body">body</A>`
- `http://body` -- This is replaced with `<a href="http://body">http://body</A>`
- `image:body` -- This is replaced with `<image src="body">`

7.6 Linking from an external document.

To link from an external document to one of the HTML documents generated by ROBODoc you need a label. ROBODoc creates two labels for each header. The first one starts with `robo` followed by a number. You can not use this one because the numbers will change each time you run ROBODoc. The second label is an escaped version of the whole header name. In this label all the non alphanumeric characters of the name are replaced by their two digit hexadecimal code.

An example, if your header name is `Analyser/RB_ToBeAdded` the label is `Analyser2fRB5fToBeAdded`. Here `/` was replaced by `2f` and `_` was replaced by `5f`. As long as you do not change the header name, this label stays the same each time you run ROBODoc.

7.7 ROBODoc-ing an existing project.

ROBODoc package includes also a standalone binary named `robohdrs`. This helper program can take clean source file and insert ROBODoc headers to functions, global variables, and macros. There are issues with this tool but it saves lots of cumbersome typing when starting on documenting an existing code-base with ROBODoc. Type

```
man robohdrs
```

or

```
robohdrs -h
```

for help. Example:

```
robohdrs -s -p testproj -i "MODIFICATION HISTORY" -i IDEAS testproj.c
```

Note that `robohdrs` is supported on UNIX-like platforms only. It requires `fork()` and Exuberant Ctags 5.3.1 or newer.

8 Suggestions and Bugs

If you find any bugs, catch them, put them in a jar, and send them to: Frans Slothouber at rfsber {at} xs4all.nl. Suggestions are also welcome at this address. Flames can be directed to the sun.