adoC documenting system*

June 10, 2002

Contents

1 Modules **3**

1.1 adoC . . . 3

2 Variables **6**

2.1 indexname . . . 6

2.2 itemname . . . 6

2.3 outype . . . 6

3 Functions **7**

3.1 printargs . . . 7

3.2 cook_str . . . 7

3.3 print_header_begin . . . 8

3.4 print_header_end . . . 8

3.5 print_item_begin . . . 9

3.6 print_item_end . . . 9

3.7 print_file_section . . . 10

3.8 print_index . . . 10

3.9 print_label . . . 11

3.10 untilchar . . . 11

3.11 BEGIN . . . 12

3.12 "find_start_sequence" . . . 12

3.13 "find_end_sequence" . . . 12

3.14 "process_doc_block" . . . 13

3.15 END . . . 13

* Generated by adoC, awk documenting C, June 10, 2002

Index

Alphabetical

- adoC, 3
- BEGIN, 12
- cook_str, 7
- END, 13
- indexname, 6
- itemname, 6
- outype, 6
- print_file_section, 10
- print_header_begin, 8
- print_header_end, 8
- print_index, 10
- print_item_begin, 9
- print_item_end, 9
- print_label, 11
- printargs, 7
- untilchar, 11

Files

- adoc
  - adoC, 3
  - BEGIN, 12
  - cook_str, 7
  - END, 13
  - indexname, 6
  - itemname, 6
  - outype, 6
  - print_file_section, 10
  - print_header_begin, 8
  - print_header_end, 8
  - print_index, 10
  - print_item_begin, 9
  - print_item_end, 9
  - print_label, 11
  - printargs, 7
  - untilchar, 11

Sorted

- Functions
  - BEGIN, 12
  - cook_str, 7
  - END, 13
  - print_file_section, 10
  - print_header_begin, 8
  - print_header_end, 8
  - print_index, 10
  - print_item_begin, 9
  - print_item_end, 9
  - print_label, 11
  - printargs, 7
  - untilchar, 11
- Modules
  - adoC, 3
- Variables
  - indexname, 6
  - itemname, 6
  - outype, 6

1 Modules

1.1 adoC

NAME

adoC

DESCRIPTION

This is a documenting system written in awk. The system can be used with almost any programming language. It tries to provide an easy to use format which does not cluter the source code itself and do not demand too much cooperation from the programmer. The initial idea is taken from the project ROBODoc.

Every documentation block appears as a comment in the source code Comments start with a "/****_*" signature and end with "******/" signature. The signature should be the last character set in the line, nothing should be after them, but they do not have to be at the beginning of the line. The underline character in the starting signature is actually one of the following characters: "f" for functions, "m" for modules, "s" for structures, "v" for variables, "t" for types, "d" for macros, "e" for enums and "o" for other features of the program which should be documented. From the above list it can be seen, that this documenting system was mainly designed for the C language. All lines between the signatures should have a star ("*") character somewhere at the beginning of the line. It does not have to be the first character, but the first word (character set which does not contain white spaces) in the line should contain the star character in it. For example in a shell script the "#*" word is acceptable. In this way indentation is also allowed.

The documentation block is divided into sections by keywords. The available keywords are:

- NAME
- SYNOPSIS
- DESCRIPTION
- ARGUMENTS
- MEMBERS
- RETURN VALUE
- NOTES
- WARNINGS
- EXAMPLE
- COPYRIGHT
- SEE ALSO
- IGNORE

When one of these words appears as the last word in a line then and only then it is recognised as a keyword. In all other cases it is just a word to print. Different documented program features require different keywords to be specified. All documentation block should have a DESCRIPTION section. They should also have a NAME section except the module documentation in which case it can be omitted. The documentation of a function also requires an ARGUMENTS section and the program will issue a warning message if no RETURN VALUE section is specified. The SYNOPSIS section can be given by the user or automatically derived. In the case of the automatic generation of a SYNOPSIS section the program reads the source code after the documentation block and takes all text until a "{" character is encountered. This means, that if your programming language uses a different convention for functions, you have to specify the SYNOPSIS section explicitly. On the other hand SYNOPSIS can be used only with functions. For structures and typedefs the MEMBERS section should be specified otherwise the program prints a warning message. The IGNORE section is never printed. The definition order of the keywords can be random, but they will be printed in the order as they are listed above. On the other hand keywords cannot be repeated in one documentation block.

The program writes the resulting document to the standard output. The format of the document is a Latex source. This means, that anything after the keywords in a documentation block will be interpreted by Latex at a later stage, therefore not all characters are acceptable. In short the following characters cannot be used freely, only in special circumstances: "$&\#". On the other hand if you know Latex you can use Latex formatting in the text.

Large part of the awk script is Latex independent. The other part of the program can be replaced easily. The main formatting of the document is done by Latex, but it should not be too difficult to write an HTML version of this program. Currently it is not done, since a marvellous Latex2HTML program exists.

The script has two modes. In the default mode the documentation is written per file, while in the second mode all source code features, like functions, variabled, macros, etc. are collected under separate sections. In both modes the program writes the index and the table of contents as well.

There are three major pattern+action pairs in the awk program. The first pattern recognizes the starting sequence and the action part of it determines the type of the documentation block. The second pattern allows the collection of all text in the documentation block. The text under each keyword is collected into a separate variable. The text is preserved in the format as the user typed it in, except the initial comment marker (the first word containing the * character) and one white space. When the third pattern recognizes the end of the documentation block the action part of it carries out a syntax checking and prints the sections for Latex.

NOTES

The name of the program was suggested by Jelle Muylle. This documentation was prepared by adoC itself.

EXAMPLE

```
/****f*
* NAME
*   my_function
* DESCRIPTION
*   This function is described here. The synopsis is automatically
*   generated
* ARGUMENTS
*   arg1 - First argument
```

```
*   arg2 - Second argument
* RETURN VALUE
*   None.
******/
void my_function(int arg1,
                 double arg2) {
```

COPYRIGHT

2 Variables

2.1 indexname

NAME

indexname

DESCRIPTION

This variable specifies the name of the different program features which can be documented with adoC. By default these are: "Functions", "Structures", "Variables", "Typedefs", "Macros", "Enums", "Modules" and "Other".

2.2 itemname

NAME

itemname

DESCRIPTION

This array holds the name of the sections in a documantation block.

2.3 outype

NAME

outype

DESCRIPTION

This variable specifies the type of output. When its value is an empty string the documentation blocks are outputed per file. When its value is one of the type characters ("f", "m", "v", etc.) then only those documentation blocks are outputed which corresponds to the specified type.

3 Functions

3.1 printargs

NAME

printargs

SYNOPSIS

```
function printargs(str)
```

DESCRIPTION

This function to formats the body of the ARGUMENTS. It searches for the "-" character in the text and if it finds one then the whole text section will be formatted as described below. If there is no "-" character in the text then the text is outputed without any formatting.

The word before the "-" character is the name of the argument and the section after the "-" character is the description of it. They are put into a table where the argument is outputed in bold style and the description is typeset on the right hand side. Technically the "-" character is replaced by the "&" character and two backslash characters with a new line character are added to the word before the name of the argument, aa(k-2).

ARGUMENTS

str – all text describing the arguments

RETURN VALUE

None.

NOTES

Current implementation is ugly, it should be done with "gsub" and regular expressions.

3.2 cook_str

NAME

cook_str

SYNOPSIS

```
function cook_str(str)
```

DESCRIPTION

This function searches the given text for the underline character and replaces it with the "\e2Under" macro. This macro is able to differentiate between mathematics and normal mode in Latex therefore it puts the backslash character before the underline character in normal mode and leaves it untouched in mathematics mode.

ARGUMENTS

str – The text which should be checked for the underline character.

RETURN VALUE

Returns the modified text.

SEE ALSO

\e2Under macro

3.3 print_header_begin

NAME

print_header_begin

SYNOPSIS

```
function print_header_begin(str)
```

DESCRIPTION

This function prints the starting formatting sequences for a documented section. When the type of the documentation block is not variable "v" a new page is started in the output document. In the mode when all documented sections are grouped this function prints the head of the group as well, but only once, for the very first time.

ARGUMENTS

str – The name of the documentation block which was given in the NAME section.

RETURN VALUE

None.

3.4 print_header_end

NAME

print_header_end

SYNOPSIS

```
function print_header_end()
```

DESCRIPTION

This function prints the ending formatting sequences for a documented section.

ARGUMENTS

None.

RETURN VALUE

None.

SEE ALSO

print_header_begin

3.5 print_item_begin

NAME

print_item_begin

SYNOPSIS

```
function print_item_begin(str)
```

DESCRIPTION

This function prints the starting formatting sequences for a section in a documentation block. In Latex the formatting sequences are environments which have the same names as the keywords, except that the spaces are removed from the keywords if there is any.

ARGUMENTS

str – The keyword for which the formatting sequence is printted.

RETURN VALUE

None.

3.6 print_item_end

NAME

print_item_end

SYNOPSIS

```
function print_item_end(str)
```

DESCRIPTION

This function prints the ending formatting sequences for a section in a documentation block.

ARGUMENTS

str – The keyword for which the formatting sequence is printted

RETURN VALUE

None.

SEE ALSO

print_item_begin

3.7 print_file_section

NAME

print_file_section

SYNOPSIS

```
function print_file_section(str)
```

DESCRIPTION

This function is used when the source code is documented on a per file basis. It starts a new page for every file and prints the name of the file as a header.

ARGUMENTS

str – The name of the file

RETURN VALUE

None.

3.8 print_index

NAME

print_index

SYNOPSIS

```
function print_index(itype, name)
```

DESCRIPTION

This function prints a Latex index. The documentation blocks indexed by their type and by the file which contains it.

ARGUMENTS

itype – The name of the type of the documentation block, for example ”function”, ”macros”. The currently documented program feature will appear under this section in the index.
name – The name of the program feature given in the NAME section.

RETURN VALUE

None.

3.9 print_label

NAME

print_label

SYNOPSIS

```
function print_label(ltype, name)
```

DESCRIPTION

This function prints a Latex label.

ARGUMENTS

ltype – The name of the type of the documentation block.
name – The name of the program feature given in the NAME section.

RETURN VALUE

None.

NOTES

Not used at the moment.

3.10 untilchar

NAME

untilchar

SYNOPSIS

```
function untilchar(uc)
```

DESCRIPTION

This function searches the text after a documentation block for a specified character and returns all text between the current position and the found character.

ARGUMENTS

uc – The character which the function is looking for.

RETURN VALUE

Returns the text until the specified character.

3.11 BEGIN

NAME

BEGIN

SYNOPSIS

```
BEGIN
```

DESCRIPTION

awk executes this function first, before any other part of the program. This part initialises the most important global variables.

ARGUMENTS

None.

RETURN VALUE

None.

3.12 ”find_start_sequence”

NAME

”find_start_sequence”

SYNOPSIS

```
/\/\*\*\*\*[fsvtedmo]\*$/
```

DESCRIPTION

This pattern recognizes the beginning of a documentation block only when it appears at the end of a line. It switches the ”proc” variable, determines the type of the documentation block, stores it in ”type” and deletes the storage array ”store”.

ARGUMENTS

None.

RETURN VALUE

None.

3.13 ”find_end_sequence”

NAME

”find_end_sequence”

SYNOPSIS

```
proc == 1 && /\*\*\*\*\*\*\/$/
```

DESCRIPTION

This pattern recognizes the end of a documentation block when it appears at the end of the line and when the "proc" variable is set. It switches the "proc" variable off, carries out a syntax checking, and prints the stored sections of the current documentation block.

ARGUMENTS

None.

RETURN VALUE

None.

3.14 "process_doc_block"

NAME

"process_doc_block"

SYNOPSIS

```
proc == 1
```

DESCRIPTION

This action is executed, when the "proc" variable is true. It means, that the program should process the lines as either keywords or text which belongs to the keyword. The text is stored per section in the "store" array.

ARGUMENTS

None.

RETURN VALUE

None.

3.15 END

NAME

END

SYNOPSIS

```
END
```

DESCRIPTION

awk executes this function last, after any other part of the program. At the moment it only checks that all documentation blocks has been closed properly.

ARGUMENTS

None.

RETURN VALUE

None.