ANL/MCS-TM-206-Rev 3

Users Manual for doctext: Producing Documentation from Source Code

by

William Gropp
Mathematics and Computer Science Division

MATHEMATICS AND
COMPUTER SCIENCE
DIVISION

Users Manual for doctext: Producing Documentation from Source Code

by

William Gropp

Abstract

One of the major problems that software library writers face, particularly in a research environment, is the generation of documentation. Producing good, professional-quality documentation is tedious and time consuming. Often, no documentation is produced. For many users, however, much of the need for documentation may be satisfied by a brief description of the purpose and use of the routines and their arguments. Even for more complete, hand-generated documentation, this information provides a convenient starting point. We describe here a tool that may be used to generate documentation about programs written in the C language. It uses a structured comment convention that preserves the original C source code and does not require any additional files. The markup language is designed to be an almost invisible structured comment in the C source code, retaining readability in the original source. Documentation in a form suitable for the Unix man program (nroff), LaTeX, and the World Wide Web can be produced. The output format is controlled by easily modified tables, permitting customization of the output. Support for other languages is also provided, though with fewer features.

1 Introduction

The tools described in this report are intended to help you create simple man-page-style documentation quickly and easily. Specifically, the program `doctext` takes C programs and generates nroff (Unix man page format), LaTeX [4], or HTML files. All of the information is embedded in structured comments, allowing the documentation to be maintained along with the code. The design of the special markup language commands emphasizes readability of the original source code (the structured comment). This approach differs from approaches such as `web` [3], where special processing must be applied to produce a readable version of the source code. Such approaches can do much more than `doctext` but are really aimed at documenting the source code of a program rather than the use of the program. Further, because those approaches are so intrusive, users tend to avoid them. The approach in `doctext` provides an effective comprimise between usability and functionality.

The `doctext` approach has been independently adopted by Java; the `javadoc` program provides HTML documentation from Java source. `doctext` predates Java and provides more formats (LaTeX and nroff as well as HTML) and formatting commands.

2 The doctext Program

The `doctext` program reads C programs and generates documentation. Command-line options provide control over the output. Fine tuning of the output can be accomplished by changing files that define the output form for each of the commands that `doctext` recognizes.

2.1 Getting Started

Using `doctext` is easy. For example, the command

```
doctext foo.c
```

will generate Unix man pages for all of the commented routines in the file '`foo.c`'. But before this will do you any good, you will need to add some structured comments to your file.

2.2 Structured Comments

The `doctext` program searches for C comments of the form `/*`*c* `...` *c*`*/`, where *c* is a single character indicating the type of documentation. The available types are `@` for routines, `M` for macros, `E` for enum definitions, `S` for struct definitions, and `D` for other miscellaneous documentation (such as introductions or descriptions of programs rather than routines). In all cases, the structured comment has this form:

```
/*@
    name - short description

    heading 1:

    heading 2:

    ...
@*/
```

The structured comment for a routine must immediately precede the declaration of the routine (either K&R or ANSI-style prototypes). Figure 1 shows the structured comment and the routine being documented.

The man (nroff-style) output of this is shown in Figure 2, and the LaTeX output is shown in Figure 3.

The body of the structured comment follows simple rules. Any line that ends in a colon (`:`) generates a section title with the line as the title. In the example of `Swap`, the line `Parameters:` generates a section in the man page with title **Parameters**. To end a line in a colon without generating a section heading, precede it with a backslash:

```
/*@
   Swap - Swaps two pointers

   Parameters:
.  ptr1,ptr2 - Pointers to swap
@*/
void Swap( ptr1, ptr2 )
void **ptr1, **ptr2;
{
void *tmp = *ptr1;
*ptr1 = *ptr2;
*ptr2 = tmp;
}
```

Figure 1: C code for a Swap routine with `doctext`-style structured comment

```
C Library Functions                                        Swap(3)



NAME
     Swap -  Swaps two pointers

SYNOPSIS
     void Swap( ptr1, ptr2 )
     void **ptr1, **ptr2;

PARAMETERS
     ptr1,ptr2
          - Pointers to swap

LOCATION
     swap.txt


                    Last change: 1/5/2000                  1
```

Figure 2: Unix-style man page for the Swap routine

Swap — Swaps two pointers

Synopsis

```
void Swap( ptr1, ptr2 )
void **ptr1, **ptr2;
```

Parameters

ptr1,ptr2 Pointers to swap

Location

'swap.txt'

Figure 3: LaTeX page for the Swap routine

```
   This is not a new section\:
```

The first column within a structured comment has a special meaning. A period followed by a space indicates that the line begins the description of an argument (ended by another argument or a blank line). This line has a special format. The first space-separated token is taken as the argument. The next character should be a dash (-). After the dash comes the text. The `Swap` example shows this for the arguments `ptr1,ptr2`. Other commands are described below in Section 3.

2.3 C Routines

C routines are indicated by the structured comment `/*@ ... @*/`. The `doctext` program provides a synopsis automatically by reading the declarations of the routine. Arguments are specified as described in See section 3.1 [Describing Arguments], page 6.

2.4 C Macros

C macros are indicated by the structured comment `/*M ... M*/`. Unlike the case of C routines, macro definitions do not provide any information on the types of the arguments. Thus, the body of a structured comment for a C macro should include a *synopsis* section, containing a declaration of the macro as if it were a C routine. For example, if the `Swap` example were implemented as a macro, the structured comment for it would look like

```
/*M
   Swap - Swaps two pointers

   Parameters:
.  ptr1,ptr2 - Pointers to swap

   Synopsis:
   void Swap( ptr1, ptr2 )
   void **ptr1, **ptr2;
M*/
```

It is important that the word *Synopsis* be used; `doctext` and related programs (`bfort` and `doc2lt`) use this name to find the C-like declaration for the macro.

2.5 Enums and Structs

A C enum may be documented using the `E` command:

```
/*E
  Read_t - Enum for read modes
E*/
typedef enum  Read, ReadBack  Read_t;
```

A C struct definition uses the `S` command:

```
/*S
  File_t - Structure defining a file type
S*/
typedef struct
    Read_t mode;
    int    fd;
    int (*readfn)( int, void *, int );
    int (*writefn)( int, void *, int );
     File_t;
```

2.6 Miscellaneous Documentation

In addition to routines, a library will often have a few additional manual pages, for example, an overview of the members of the library or instructions on installing or debugging the library. In order to allow the same tools to be used for all of the documentation, a comment of the form `/*D ... D*/` may appear anywhere and will generate a manual page.

2.7 Indicating Special Limitations

Two modifiers to the structured comments indicate special behavior of the function. The modifiers must come after the character that indicates a routine, macro, or documentation. The modifier `C` indicates that this routine is available only in C (and not from Fortran). For example, the `Swap` program cannot be used in Fortran, so its structured comment should be

```
/*@C
  ....
@*/
```

The modifier `X` indicates that the routine requires the X11 Window System. This is intended primarily for the program `bfort` [1], which is used to generate Fortran interfaces for systems that do not have X11.

The modifiers `C` and `X` may be used together and may be specified in either order (i.e., `CX` or `XC`).

2.8 Indicating Include Files

It is often very important to indicate what include files need to be used with a particular routine. This may be accomplished with a special structured comment of the form `/*I`*include-file-name*`I*/`. For example, to indicate that the routine requires that '`<sys/time.h>`' has been included, use

```
#include <sys/time.h>          /*I <sys/time.h> I*/
```

in the C file. A user-include can be specified as

```
#include "system/nreg.h"      /*I "system/nreg.h" I*/
```

This approach of putting the structured include comment on the same line as the include of the file ensures that if the source file is changed by removing the include, the documentation will reflect that change. Includes are added to the synopsis of all routines in the file that contains the include comment.

3 Special Formatting

The structured comment format for `doctext` was designed to have a small impact on the appearance of the C source code. As such, it provides minimal formatting for the generated manual pages. Three important cases are provided: describing arguments, verbatim output, and common blocks of text. In addition, there are some additional formatting controls for things like emphasis (italics text) and fixed-width fonts. This section describes all of these in more detail.

3.1 Describing Arguments

Arguments to routines and command-line options for programs are described by using a plus (`+`), period (`.`), and minus (`-`) in the first column, followed by one or more spaces. The name of the argument or command line variable is next. Several arguments may be placed together, separated with spaces and/or commas. To include a hyphen (`-`) in the argument, prefix it with a backslash (`\`). Follow this with one or more spaces, then a dash (`-`), optional spaces, and finally the text of the description. The description may use multiple lines. The `+` should be used to start an argument list and `-` should be used to end the list. The period (`.`) should be used for all other arguments in the list. For example,

```
+ a - Pointer to
      the data item
. i, j - Sizes of data item
. str - Another argument value
- \-arg value - Sample command-line argument
```

The `+` and `-` are optional but recommended. They permit better formatting of lists of arguments.

3.2 Common Blocks of Text

In some cases, it is useful to repeat a body of text in many man pages. For example, there may be a common argument that has a lengthy description. A common block of text is defined with

```
/*N name
   text
N*/
```

The *name* may be any blank-delimited string.

To insert this block of text into a man page, use the `.N` format command

```
/*@
    ...

.N name

    ...
@*/
```

The definition must precede all uses. Once defined, the definition is remembered; if multiple files are processed and the first file contains a definition of a named block, all subsequently processed files may refer to that named block. This feature allows different replacement texts for different situations. For example, the replacement text for a man page should contain the entire text, whereas the replacement text for a manual (LaTeX) may reference a common section.

For example, the MPICH implementation of MPI uses this to define all of the error conditions. A file '`errnotes`' contains lines like

```
/*N Errors
Errors:
```

```
 ... text about MPI errors ...

N*/
/*N MPI_SUCCESS
. MPI_SUCCESS - No error; MPI routine completed successfully.
N*/
/*N MPI_ERR_BUFFER
. MPI_ERR_BUFFER - Invalid buffer pointer.  Usually a null buffer where
  one is not valid.
N*/
```

Each MPI man page ends with references to the name blocks for the errors that the particular MPI routine may return. `MPI_Send` has the following commands at the end of its structured comment:

```
.N Errors
.N MPI_SUCCESS
.N MPI_ERR_COMM
.N MPI_ERR_COUNT
.N MPI_ERR_TYPE
.N MPI_ERR_TAG
.N MPI_ERR_RANK
```

3.3 Line Breaks

A new line may be begun with the `.n` command at the beginning of a line. The rest of the line is read. For example, to provide a list of items, use

```
.n This is the first item
.n This is the second item
.n This is the third item
.n
```

Note that a final line containing only `.n` is used to ensure that the next line of text begins on a new line rather than at the end of the third line.

3.4 Verbatim Blocks of Text

To display a block of text in a fixed-width font, use the `.vb` command to begin the block and `.ve` command to end the block. These commands, like the other dot commands, must begin in the first column. For example, to display two lines of shell commands, use this text in the structured comment:

```
.vb
  cd .
  echo "I'm here"
.ve
```

3.5 Emphasis

To emphasize some text, include it between back quotes: `‘this text is emphasized‘`. Emphasized text may appear in italics (*this text is emphasized*) or in bold face (**this text is emphasized**). (Whether italics or bold face is used depends on the implementation and the output format). This effect requires the use of the `-quotefmt` command-line option; if the option is not used, back quotes are intrepreted as a simple character. To get a single back-quote when `-quotefmt` is used, double the quote: `‘‘`.

3.6 Code and Filename Font

To display some text in a font appropriate for code and filenames, include it between single forward quotes: `'this text is code'`. This will appear in a form such as `this text is code`. This effect requires the use of the `-quotefmt` command-line option. To get a single quote when `-quotefmt` is used, double the quote: `''`.

3.7 Pictures

This section described a proposed feature; comments are welcome.

In the HTML and LaTeX formats, pictures (Postscript files) can be included by using the `.p` command:

```
.p filename
```

The file must be a Postscript file. A caption for the picture can be included with the `.cb` and `.ce` command. This caption is not displayed in the Unix man-page document but does appear in the LaTeX and HTML versions. For example,

```
.p sample-out.ps
.cb
 This image shows the output of the 'foo' command
 Note particularly ...
.ce
```

(not yet implemented)

3.8 Verbatim

A dollar (`$`) in the first column indicates a verbatim line. In most situations, this can be used for forcing a line break at the end of the line. An example is

```
$ This is a test
$
$    Here is a sample command line
$
$ And the start of the description about
  it.
```

This version of verbatim is deprecated and supported for backward compatibility.

3.9 Keywords

Keywords may be specified with the command `.k`*list of keywords*. For example, to indicate that keywords *test, example, help* belong to the current routine or item, use

```
.k test, example, help
```

To improve the readability of the source, you may use `.keywords` instead of just `.k`.

The command line option `-keyword filename` causes the keywords to be appended to the specified file; this can be used to produce a keyword list.

3.10 Other Formatting Options

The program `doctext` is designed so that it is easy to add additional formatting commands of the form `.`*command*. In fact, you can define your own formatting commands. This is done in two steps. First, you add formatting commands using a command definition file (See See section 8 [Customizing the Output Format], page 14) and include those definitions with the `-defn filename` command-line argument. You can then invoke your new commands by using `.f<command name>` in the source file. For example, if the file '`indent.def`' contains

```
in \begin{quotation}%n
out \end{quotation}%n
```

you can use

```
.fin
  This text will be within a LaTeX quotation environment
.fout
```

in the source file.

Other commands beyond those described in this document may be added in the future, so you should not rely on any particular behavior of *.character*. For example, `.seealso` has been added to provide a common reference for related information. If you need some particular formatting option that you cannot achieve with the `.f` approach, please send the suggestion to `gropp@mcs.anl.gov`.

4 Command Line Arguments

To use `doctext`, you need only to give it the name of the files to process:

```
doctext *.[ch]
```

Command-line options to `doctext` allow you to change the details of how `doctext` generates output.

A complete list of the command line options follows. Some of these will be used often (e.g., `-ext` and `-mpath`); others are needed only in special cases (e.g., `-outfile`).

The choice of output format is selected with these switches:

-latex Generate LaTeX output rather than man (nroff) format files.

-html Generate HTML (WWW) output rather than man (nroff) format files.

If neither of these is used, the output is in man (nroff) format.

The following options control some aspects of the appearance and content of the generated page, as well as the name of the output file.

-mpath path Set the path where the man pages will be written.

-ext n Set the extension (1-9,l, etc.). The default value is 3. This is the man page chapter and is used as the extension on the filename. For example, `doctext` creates the file '`Swap.3`' for the `Swap` routine.

-heading name Name for the heading (middle of top line). The default value is `PETSc`. This is used only for Unix man-page (nroff) format.

-nolocation Suppress the generation of a line that indicates the source file that contained the structured comment.

-quotefmt Enable the use of single quotes and back quotes to use code and emphasis fonts.

-dosnl End every line with return-newline instead of just newline. This makes that file easier to read on Windows systems, and is quite helpful for HTML output.

-defn filename Load the output definitions in file '`filename`', in addition to the defaults.

-I filename `filename` contains the public includes needed by these routines; it will automatically be added to the synopsis of the routines. The file should contain exactly the text to be added to the synopsis. For example, to specify that the include `<stdio.h>` be included, use a file with only the text `#include <stdio.h>` in it.

The following options control the generation of additional data about HTML-format output that may be used by other programs to automatically generate links to the files created by `doctext`.

-index filename Generate an index file appropriate for `tohtml` [2] (for generating WWW files). This *appends* to any existing file; make sure that you don't add duplicate entries.

-indexdir dirname Set the root directory for the index file entries. This allows you to specify an absolute URL for the generated files to be used in the generated index, for example,

```
-index foo.cit -indexdir "http:/www.mcs.anl.gov/home/gropp/man"
```

-mapref filename Use file '`filename`' as a source of names that should be automatically mapped to URLs. One source of a properly formatted mapref file is the output from `doctext` in the file specified with the `-index filename` command.

The following options control the output filenames and the list of files to process.

-outfile filename Put the man pages in the indicated file. Normally, a separate file is created for each routine. This may be appropriate for HTML output. The output filename is normally made up of the name of the routine, followed by the extension (`-ext n`) for nroff, `.tex` for LaTeX, or `.html` for HTML files.

-keyword filename Append the keywords (from `.k` commands) to the given file. If the filename is '`-`', use standard output.

The following options control the input format of the file, allowing `doctext` to be used with documentation in source files for Fortran or shell scripts.

-skipprefix name Set the characters to be skipped at the beginning of each line. The default is the empty string. Use

```
-skipprefix C
```

for Fortran 77 programs. You can use `-skipprefix #` for shell scripts and `-skipprefix \!` for Fortran 90 programs.

For example, with `-skipprefix C`, a Fortran program can use the D (not @) mode:

```
C/*D
C fortranfoo - An example of using Fortran with doctext
C
C Input Parameters:
C
C+ foo - bar
C. bar - stool
C- stool - pidgeon
CD*/
      function fortranfoo( foo, bar stool )
      integer foo, bar stool
      end
```

This option is new and has not been extensively tested.

-ignore name Skip over `name` when processing a function definition. This is particularly useful when a special (non-standard) keyword is needed to build a dynamic link library (DLL) (e.g., Microsoft C/C++) or to simplify the construction of export lists (IBM's AIX). For example, say that `EXPORT` is used to indicate a routine that will be exported as part of a DLL. The source code then looks like

```
/*@ myroutine - a routine visible as part of a DLL
 @*/
EXPORT void myroutine( int a )
```

Using the command line option `-ignore EXPORT` will cause `doctext` to skip the `EXPORT` keyword; the synopsis will contain only

```
void myroutine( int a )
```

After the command-line arguments come the names of the files from which documents are to be extracted.

5 Man Pages for Hypertext Documents

One of the most difficult tasks in creating extensive hypertext is generating the initial documents and providing an easy way to link to them. For the the hypertext to be useful, there must be an easy way to create links to the generated documents. This section describes how to do this. The information produced by `doctext` may be used by `tohtml` [2] to generate documents that have extensive cross-links to hypertext versions of their man pages. Special formatting options for HTML pages are also discussed.

5.1 Generating Collections of Linked Web Pages

To generate HTML man pages for a collection of source files in '`/home/me/foo`', do the following:

```
cd
mkdir www
mkdir www/man3
cd foo
doctext -html -index ../foo.cit \
        -indexdir http://www.mcs.anl.gov/me/foo/www/man3 \
        -mpath ../www/man3 *.[ch]
cd ..
```

This puts the HTML files into the directory '`www/man3`' and the index (in the correct format for the `-mapman` option of `tohtml`) into the file '`foo.cit`'. The `-indexdir` option is used to specify the ultimate location for the files (in this case, the directory `me/foo/www/man3` at the Web site `www.mcs.anl.gov`). Once you are sure that the files are correct, you can move them into the Web area with

```
cp -r www /mcs/www/home/me
```

(assuming that '`/mcs/www`' corresponds to `http://www.mcs.anl.gov` in the `-indexdir` argument).

To generate an HTML listing of the routines, you can execute the following script, with, of course, the appropriate changes to the text:

```
#! /bin/sh
echo '<TITLE>Web pages for My Routines</TITLE>' >> www/index.html
echo '<H1>Web pages for My Routines</H1>' >> www/index.html
echo '<H2>My Routines</H2>'  >> www/index.html
echo '<MENU>' >> www/index.html
ls -1 www/man3 | \
    sed -e 's%^\(.*\).html$%<LI><A HREF="man3/\1.html">\1</A>%g' \
        >> www/index.html
echo '</MENU>' >> www/index.html
```

This example may be found in the file '`mkhtml.sam`' in the source directory for `doctext`.

If you have only a few routines to document, you can dispense with the second directory level above (the '`man3`'). However, you might find it valuable to follow (at least loosely) the Unix man page format, with commands and installation instructions in '`man1`' and routines spread across '`man2`' through '`man8`'.

5.2 Alternate Formatting for HTML

The appearance of arguments can be improved in HTML by using an HTML table. The command line argument `-defn htmlargtbl.def` will cause a table to be used in HTML mode for argument lists that begin with `+` and end with `-`.

You can also completely control the type of HTML that is generated. `doctext` uses definition files that define what HTML is produced for each `doctext` command. These files, for HTML, are '`html.def`' in the directories specified by `TEXTFILT_PATH` (for basic operations) and `DOCTEXT_PATH` (for doctext-specific operations). These are defined when `doctext` is built; the defaults are '`/usr/local/share`' and '`/usr/local/share/doctext`'. You can override any of these definitions by creating your own file and specifying it with the `-defn filename` argument. In addition, you can prepend or postpend your own commands to the predefined commands. For example, to add an index link to the bottom of every page, put the following definition in a file:

```
+eop    %n<BR><A HREF="http://www.mcs.anl.gov/myindex">Index</A>
```

This says to prepend (the `+` is before the end-of-page command `eop`) the HTML code given. The `%n` generates a newline. Placeing a `+` after the commands postpends the definition.

6 Making a Reference Manual

With the `-latex` option of `doctext`, it is easy to create a reference manual containing the manual pages produced by `doctext`. The style file `refman.sty` (in '`share/refman.sty`') provides the necessary LaTeX definitions. Below is an example skeleton for a reference manual:

```
\documentstyle[refman]{article}

... page size commands ...

\begin{document}

... title page commands ...

\pagenumbering{roman}
\tableofcontents
\clearpage

\pagenumbering{arabic}
\pagestyle{headings}

\section{Introduction}
... regular text ...

\section{My commands}
\input ref/ref1

... additional sections ...
\end{document}
```

To generate the LaTeX files for the individual man pages using `doctext`, you can use the following commands:

```
doctext -latex -mpath ref/ref1 *.c
```

followed by

```
cd ref
ls ref1 | sort | sed 's/^/\ input /g' > ref1.tex
```

If named replacement blocks of text (`.N name`) are used, it may be appropriate to use a different definition for the manual from that used for the man page. For example, if the file '`common.txt`' contains

```
/*N defs
    Definitions are described in Section 2.
N*/
```

then changing the `doctext` command to

```
doctext -latex -mpath ref/ref1 common.txt *.c
```

will cause each `.N defs` command to include the text "Definitions are described in Section 2."

7 Making the man Pages Available

In order to make the regular man pages available, a directory tree containing directories '`man`', '`man/man1`', etc. (for all of the man page extensions used) needs to be created. In addition, a "whatis" database should be built within the primary man directory so that `man -k topic` will find the command or routines with `topic` in their description. This is done with the program '`/usr/etc/catman`' or '`makewhatis`'. If either of these programs is not available, the "whatis" database cannot be created. The following C-shell code will create the database:

```
unalias cd
if (-e /usr/etc/catman) then
    (cd man ; /usr/etc/catman -w -M . )
else if (-e `which makewhatis`) then
    (cd man ; makewhatis -M `pwd`)
endif
```

Some `catman` programs may behave erroneously; at least one system, instead of using the `.SH NAME` entry, looks for `.SH NAME ... .SH`. What this means is that if the first text after the routine's description is not a section title (i.e., a line ending with a colon), the `catman` program will generate an incorrect whatis database.

It is easy to provide an X11 interface to the man pages by using `xman` and the `MANPATH` variable. Here is a simple shell script that provides access to the man pages in '`/home/me/man`':

```
#! /bin/sh
MANPATH=/home/me/man
export MANPATH
xman -notopbox -helpfile /home/me/man/me.help "$@" &
```

In order to get `man` and `xman` to display the correct names for the various sections (corresponding to the directories '`man/man1`', '`man/man2`', etc.), a file '`man/mandesc`' is required. Here is the '`man/mandesc`' for the PETSc package:

```
r(r)Introduction
1(1)Sparse Matrix Routines
2(2)Vector Routines
3(3)Simplified Solvers
4(4)Iterative Methods
5(5)High Level Communications
6(6)Low Level Communications
7(7)System Calls
8(8)Miscellaneous
9(9)Domains and Grids
b(b)BLAS
x(x)X Window System Tools
n(n)Nonlinear Solvers
no default sections
```

Finally, `xman` displays a help page when it starts. To change the file, use the `-helpfile` argument and provide a simple text file (not a man) page.

8 Customizing the Output Format

The text formating commands that `doctext` uses for its output are controlled by several files. For each output format (nroff, html, and latex) there are two sets of definitions files. The first set are the ones common to all of the tools in the sowing package, and are in the directory '`share`'. The files are

html.def HTML commands

latex.def LaTeX commands

nroff.def Nroff (man) commands

In addition, the directory '`share/doctext`' contains special versions of these same files. `doctext` uses the files in '`share/doctext`' by default (see the file '`sowing/src/doctext/docpath.h`'). You can override or add to these commands with the `-defn filename` argument. The environment variable `DOCTEXT_PATH` may be used to point to a different set of definition files.

Small changes can be made by using the `-defn filename` switch to load a file that redefines a few of the commands. The file '`htmlargtbl.def`', mentioned above for using the HTML table commands for arguments, is an example. Similarly, the file '`latexargtbl.def`' can be used to place the argument definitions in a LaTeX `tabular` environment. Using the `-defn` argument to override a few definitions is usually preferable to replacing all of the definitions.

8.1 The Doctext Commands

`Doctext` uses named commands for all of its output. For each of these commands, there is a definition that tells `doctext` what to do for each of these. For example, when `doctext` outputs a section (from a line ending in a `:`), `doctext` uses the `section` command. The default definition for this when generating HTML is

```
section    %n<H2>%1</H2>%n
```

This instructs `doctext` to output a newline if it isn't already at the start of a line (the `%n`); followed by standard HTML for a level 2 heading, followed by another newline. The `%1` in the argument string stands for the first argument to the `section` command. To change the output style, you need only replace the command definitions. The next two sections detail the commands and the command language.

8.2 Commands

blank Generate a blank space. This is usually just a blank space.

bof Beginning of file command. This is called when a new file is created. There is one argument: the name of the directory.

bop Beginning of page command. This is called when starting a new manual page.

definition There is one argument, the one-line description of the routine. Used only by `doc2lt`.

em Begin emphasis (or italic) style text. See the command syntax for the discussion of how font modes like this are exited.

em_dash Generate an em dash (—).

end_par Generate an end-of-paragraph.

eof End of file command. This is called just before `doctext` closes a file.

eop End of page command. This is called at the end of a man page.

key There is one argument, the name of the routine that a man page is being generated for. Used only by `doc2lt`

linebreak Generate a forced line break.

location Show the location of the file the generated this man page. The argument is the filename.

mantitle Generate the man page title. This takes four string arguments. They are, in order

name Name of the routine

level Man page section (e.g., `3`)

date Date of last change to file

heading man section name (from the `-heading` argument to `doctext`).

picture This command takes the name of a file as argument and will be used in future versions of `doctext` to allow the inclusion of graphics in man pages.

postamble Used only by `doc2lt`

preamble Used only by `doc2lt`

rm Change font style to roman.

section Generate a man page section header. There are two arguments. The string argument is the name of the section; the integer argument is the level of the section. The default section level is two.

synopsis Generate the `Synopsis` section of the man page. The argument is the synopsis text. Used only by `doc2lt`.

tt Change font style to fixed width.

In addition to the above commands, there are a number of commands that come in pairs, with a start (`s_xxx`) and an end (`e_xxx`) term.

s_arg Begin an argument (`.` format command).

s_defn Begin the definition of an argument.

s_arg_list Begin a list of argument (the `+` format command). This doesn't include the argument itself; that is begun with the `s_arg_inlist` command.

s_arg_inlist Begin an argument within an argument list (started with `s_arg_list`).

s_defn_inlist Begin the definition of an argument in an argument list.

s_caption Begin the text for a caption. This is intended for use with the `picture` command, and is not supported.

s_doctext_verbatim Begin the single-line verbatim mode (the `$` command).

s_synopsis Begin the synopsis section.

s_verbatim Begin a multiline verbatim section (`.vb`).

8.3 The Definition Command Language

The command language for the definitions is very simple. Each line has the form

```
command-name replacement-text
```

The `command-name` is any of the names in Section 8.2. Comment lines may be included anywhere and start with `#`. The `replacement-text` is almost literal text, except for `%` escapes. These escapes are almost all single letters; for example, `%n` and `%1`. The complete list of escapes is

% Output the % character

1 Insert the first string argument

2 Insert the second string argument

3 Insert the third string argument

4 Insert the fourth string argument

i Insert the first integer argument

n Insert a newline only if not at an newline

u1 Insert the first string argument, but in uppercase. Similarly for `u2` through `u4`.

p Insert end-of-paragraph string only if not at an end-of-paragraph.

f Insert the end of font string. This string is defined by the `%e=` command.

N="..." Replace newlines with the string given in quotes. The string can be 32 characters or less.

e="..." Define the end of font string. The string can be 32 characters or less.

m="..." Define the output mode. This is a string that is understood by the particular output format. For example, the LaTeX output routines understand the mode `verbatim` and use that to change how characters are output.

a0="..." Define the value of string register zero. The string may be up to 63 characters long. There are 10 registers, commands `a1` through `a9` access the other nine. The value of `a0` is set to the name of the function or macro by `doctext`, and normally should not be changed.

r1 Insert the value of string register zero. Similarly for `r1` through `r9`.

To continue `replacement-text` to another line, end the line with the `\` character. A single space at the end of a line may be represented with `\<blank>`.

Sometimes you will not want to replace a command; instead, you will want to insert your replacement text either before or after the standard text. You can do this by placing a `+` either before or after the `command-name`. An example of this is in Section 5.2.

8.4 Examples of Command Definition

This first example shows how to place the argument names and definitions into a table when generating HTML output. This works only with the argument list commands (start a list with `+` and end it with `-`). Here are the definitions:

```
# You can get this by specifying -defn htmlargtbl.def to doctext.
s_arg_list    %n<TABLE BORDER=0>
s_arg_inlist  %n<TR><TD WIDTH=20></TD><TD ALIGN=LEFT VALIGN=TOP><B>
e_arg_inlist  </B></TD>
s_defn_inlist <TD VALIGN=TOP>
# The br is used to break the line in the event that the browser doesn't
# support tables.  It is better than nothing.
e_defn_inlist <BR></TD></TR>
e_arg_list    </TABLE>%n
```

This example uses only the `%n` escape. The definitions are ordered as they would be used by `doctext`.

This second example makes `tt` text one size larger when generating HTML:

```
tt     %f<TT><FONT SIZE=+1>%e="</FONT></TT>"
```

The `%f` command is used to terminate any previous font style command. The `%e` command defines how to terminate this font style.

9 Installing doctext

The `doctext` program is part of the a sowing package. (An earlier version was part of the PETSc package of tools for scientific computing.) The program is available from ftp://ftp.mcs.anl.gov/pub/sowing/sowing.tar.gz. Directories and files such as '`refman.sty`' referred to in this manual are provided in this implementation and may be found in the installation. To build,

```
gunzip -c sowing.tar.gz | tar xf -
cd sowing
./configure
make
```

The `doctext` program is in the '`sowing/src/doctext`' directory.

`doctext` requires a C++ compiler; most testing has been done with `gcc`.

Please send any comments to `gropp@mcs.anl.gov`.

Acknowledgment

The author thanks Lois Curfman McInnes and Barry Smith for their careful reading and vigorous use of the `doctext` manual and program and Ewing Lusk for valuable suggestions about additional functionality.

References

[1] William Gropp. Users manual for `bfort`: Producing Fortran interfaces to C source code. Technical Report ANL/MCS-TM-208, Argonne National Laboratory, March 1995.

[2] William Gropp. Users manual for `tohtml`: Producing true hypertext documents from LaTeX. Technical Report ANL/MCS-TM-207, Argonne National Laboratory, March 1995.

[3] Donald E. Knuth. The `web` system of structured documentation. Technical Report 980, Computer Science Department, Stanford University, September 1983.

[4] Leslie Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, 1986.