

FicMake

by Timothy Groves

Page layout and chapter management software for stories, geared towards fan fiction.

This program and manual are licensed under the GNU Public License. A copy of the license should have accompanied the program in the file COPYING.

Table of Contents

Introduction	2
System Requirements	2
Installation	2
Features	3
Invoking FicMake	3
One: Using Ficmake	4
Creating a new story	4
Keys	4
Fic Options	4
PDF Options Page	6
HTML Options	7
Chapter Options and Titles	8
Two: Files	9
Input Files	9
Output Files	10
Three: Groff and MS	12
Text Filling	12
Paragraph Formatting	12
Character Formatting	13
FicMake macros	15
Appendix: Output Formats	17
HTML	17
Plain Text	17
Plain Text Bulk File	17
Postscript	17
PDF	17

Introduction

As a longtime fanfic writer, I have become used to the need to post in a wide variety of formats. As a Linux user, I've been using `groff` and `make` to manage these formats. But the amount of work required to set up the complex of files required to use `groff` and `make` to format fics soon became more than the amount of work required to write the fics in the first place.

But besides being a writer and a Linux user, I am also a programmer, and it quickly became habit for me to write a program to create the makefile. Then I expanded on that, and created programs to create various wrapper files, indices and so on. But upkeep of *these* programs began to eat into my time as well.

Finally, the day came when I said, "Enough is enough. I need one central program which can manage almost any fic, and can do all the jobs I need done."

That night, I wrote the core of what would become **FicMake**.

FicMake is not for everyone. It is intended for use by fanfic authors, and is geared towards fiction writing. (Though as this document proves, it is rather flexible.) It runs only under GNU/Linux at this time, and to be at all useful, you need several other programs. You need to know at least a smattering of `groff`, particularly the `ms` macros for such. I have included a tutorial on the most useful `groff/ms` tags in this document—far more than you need to use FicMake, really.

But within these limits, it is rather powerful, and I am already grateful I've written it.

System Requirements

FicMake itself has shockingly low system requirements—at least, to those who have been computing since the seventies, such as myself. I have successfully compiled it with as little hardware as a 386 with 8 megabytes of RAM. Recommended system requirements are a 486DX/33MHz or better, with at least 8 megabytes of RAM. Storage requirements for the output files are a bit higher; a 38-chapter epic of mine, converted using FicMake, required 7 megabytes of storage. The required software to go with FicMake can easily consume 40 to 60 megabytes.

On the software side, it has been tested with `groff` 1.19.2, GhostScript 8.62, `netpbm` 10.35.22, `sed` 4.1.5, and GNU `make` 3.81, and will probably work with earlier versions of any of the above. It requires `fpc` to compile, and has been tested under versions 2.2.2 and 2.4.0. A plain text editor is strongly recommended, but none is absolutely required; you are free to create the source documents in any program you wish, so long as it can output standard 7-bit ASCII `.txt` files.

Installation

To compile FicMake, you must have `fpc` and GNU `make` installed. From the `ficmake` directory, type **make**. Then copy the `ficmake` binary to anywhere in your path.

The `ficmake` binary is the only file required to use the software; no support files are required. It is a completely encapsulated file.

To compile the manual, enter the manual directory, and type **ficmake -a**. Then type **make**. This is precisely the same sequence that is used to run FicMake on your own stories.

Features

FicMake includes a console user interface that allows you to edit your chapter list, lay out your document, and set several options for file output.

The formats of outputted files include the following:

- Postscript and Portable Document Format (PDF);
- HTML suitable for a web page, with an automatically generated index file;
- Plain text, in separate chapters;
- Plain text, in one bulk file;
- Stripped-down HTML suitable for uploading to fanfiction.net, and probably useable on other archive sites (I haven't really tested anywhere else).

The various different layout options for each of the above are managed by creating wrapper files. FicMake handles this automatically; all that the author must provide are the individual chapter files. More on files can be found on page 9.

Layout Options

Each of the five different output styles has several options for formatting. Some of the formats allow for more options than others; the plain-text formats allow hardly any, but that's the limit of that style.

Among the options you can set for PDF and Postscript output, there are page size, orientation and columns; font family, weight and size for six different styles; and margins. FicMake will also generate a Table of Contents for Postscript, PDF and bulk text output. It allows for importing of JFIF/JPEG graphics, and performs cross-referencing. More details on these options are given in the next section.

Invoking FicMake

FicMake may be run with any of the following flags:

- a Autogenerate wrapper files
- e Edit the desc.fic file in interactive mode
- h Display this help text

If ficmake is launched with no parameters, it defaults to interactive mode.

One: Using Ficmake

FicMake is a console program, and can only be used in one of two environments. For the interactive mode, you must be at the console, either a genuine Linux console or a reasonable hand-drawn facsimile such as xterm or rxvt. In automated mode, FicMake can be used within shell scripts, as well as from the console.

Creating a new story

FicMake is designed to automate the creation of its various formats as much as humanly possible. However, at least a bit of human interaction is required to produce any results at all.

To start a new fic, create an empty directory to hold it. Within this directory, type **ficmake**. This will launch FicMake in its interactive mode.

Keys

The following keyboard commands are available at any time when running FicMake:

- F1** Access the Help for the current page
- F2** Access the Page Menu, to select a different page or to exit FicMake
- F3** Access the Fic Menu
- F4** Auto-Generate all files for the fic
- F5** Access the Fic Options page
- F6** Access the PDF Options page
- F7** Access the HTML Options page
- F8** Access the Chapter Options page
- F9** Create the container directories for all formats
- F10** Reload the **desc.fic** file from disk
- F12** Exit FicMake

Fic Options

When launching FicMake for the first time, you will be presented with the Fic Options page. You can return to this page at any time by selecting it from the Page Menu, or by pressing F5. This page includes the following items:

- Story Title;
- Author;
- Long Filename;
- Short Filename;
- Chapter Name;
- Book Name;
- Prologue Name;
- Epilogue Name;
- Output Formats;
- Title Picture;
- A Taskbar containing the following items: Help, Page Menu, Fic Menu.

These options are explained in detail below.

Story Title and Author

These are largely self-explanatory. For most formats, these will be inserted in the title or index page. In Postscript and PDF format, the title of the story will be inserted in the header of every even-numbered page, and the author's name will be inserted in the footer of every page.

Long Filename and Short Filename

Of the options visible in the screen, these two are absolutely required. The short name is used to select the input files of your fic. The long name is used for the output of FicMake and groff.

For example, perhaps you wish to write a story entitled "The Shining One". You might select the long name "shining" and the short name "tso". FicMake would then select files starting with "tso" for its chapters, and would output files such as "shining.pdf", "shining.txt" and so on.

Despite the names, there is no requirement for a long name to be longer than a short name. The naming convention simply comes from my own writing habits. However, the short name will have the chapter number appended to it, and if you are using this program on a file system with limited directory sizes, you will need to keep these limits in mind when selecting file names.

Due to the fact that **FicMake** *must* have these two entries filled in, all new stories will default to a long name of 'newstory' and a short name of 'story'.

Chapter Name and Book Name

By default, these entries read "Chapter" and "Book". They are inserted before each book number and chapter number.

For example, you may decide that your story has Acts and Scenes instead of Books and Chapters. By changing these entries, your story would have entries such as Act One, Scene One and so forth, instead of Book One, Chapter One.

Prologue and Epilogue Name

Prologues and Epilogues are not numbered, but instead marked solely with the entries in these fields.

For example, this document used the alternate term "Introduction" instead of "Prologue", and "Appendix" instead of "Epilogue". The results should be quite obvious.

Output Formats

This will pop open a box allowing you to select which formats are available for output: PDF, HTML, HTML for Fanfiction.net, Text, and Bulk Text. Press the indicated letter to set any of these formats on or off. Press ENTER when done, or ESC to close and cancel changes.

Note that creating a PDF file will also create a Postscript file, and that creating HTML files will also create the appropriate index.html.

Title Picture

FicMake can import any JFIF/JPEG file as a title picture, which is used in place of a printed title in Postscript/PDF output, and in the index file for HTML output. The file must reside in the same directory as you shortname.so files; that is, in the root directory of your story.

The Page Menu

The Page Menu will allow you to switch to the PDF Options Page, the HTML Options Page, the Chapter List, or allow you to exit the program. You can also press F12 to exit at any time.

The Fic Menu

The Fic Menu offers three options: Create Directories, Auto-Generate, and Revert To Saved.

Create Directories

This will create the target directories for the webpage, fanfiction.net and plain text chapter files. This is done by default by FicMake when the fic is created, but if you run a 'make clean', these target directories will be deleted. Running 'ficmake -a' will re-create these directories as well.

Auto-Generate

This will run the automated build section of FicMake, in the same manner as though you had typed **ficmake -a** from the command line. It will not build the output files; it merely creates the wrappers needed for the output files.

Revert To Saved

This will restore the currently-loaded version of the desc.fic from the version currently on disk. A word of warning: Performing an auto-generate will save the desc.fic file.

PDF Options Page

This page allows you to set up the options for PDF (and Postscript) output. It includes the following options:

- Page Size;
- Page Orientation;
- Columns;
- Section Divider;
- Heading Graphic;
- Margins and Gutters;
- Fonts.

Page Size

FicMake supports three different page sizes by default: Letter (8.5"x11"), Legal (8.5"x14") and Statement (5.5"x8.5"). Statement-size is included because it is precisely half the size of a standard sheet of Letter paper, which makes it handy if you wish to create documents that can be printed in booklet format, then folded and stapled.

To change the page size, press the **S** key to cycle between the available sizes.

Page Orientation

Press the **O** key to change between Portrait and Landscape orientation.

Columns

Press the **C** key to cycle between one, two and three columns per page. Bear in mind that FicMake will blindly create statement-sized pages with three columns, and the output may not be what you desire.

Section Divider

The Section Divider is used to split text with a convenient marker, in this manner:

This entry allows you to define the text to be used as the divider. This divider is also used in HTML format and plain text. Fanfiction.net HTML does

not use the divider; the rules of that site state that you are to use a horizontal rule instead.

The divider is inserted on its own standalone line, centered. If you are willing to enter some groff hackery here, you can pull off some impressive stunts, as escapes and macros can be used instead of plain text. However, if you do so, the divider may not display properly in the plain text modes.

Inserting Section Divider is covered in Chapter 3, on page 15.

Heading Graphic

If this entry is defined, FicMake will insert the requested graphic behind each Chapter Heading. The file declared here should be no more than a quarter inch in declared height, and must be a Postscript (.ps) file.

H1 Center Mode

This somewhat cryptic entry defines how the .H1 macro (see page 15) relocates chapter headings. The options are:

- Center over all columns;
- Center over first column;
- Left-Justify over first column;
- Left-Justify over all columns.

If only one column is being used, any entry that centers will produce identical output: headings will be centered over the (only) column. Similarly, Left-Justify will align the header with the left margin over the column.

Fonts

The five Heading styles and the Text style may be set up here. To select a Heading style, press the corresponding number key - 0 to 4. To set the body Text style, press T. You can also use the up and down arrow keys to switch between the fonts.

Once a font is selected, press F, N or P to adjust its settings - Family, Font and Point Size, respectively.

Resetting Default Fonts

Ctrl-D will reset all fonts back to the defaults as stored within FicMake. It will prompt you to verify this action before it completes it.

Margins and Gutters

The framing of the document on a page is important, and can make a considerable difference in whether or not anyone will read the PDF version. It also makes it a lot easier to print. Unlike most other entries, FicMake uses typographic points for these entries. 72 points equals one inch.

Left/Right Margins

The margins for PDF format may only be set equally; whatever you enter here will be used on either side of the page. The value is given in points; 72 points equals one inch. The default value is 36 points, which is half an inch.

Top/Bottom Margins

Similarly, the top and bottom margins may be set here. They should be set to no less than 36 points, as room must be left for headers and footers. The default is 54 points, which is 0.75 inches.

Gutter

The gutter is an empty column found at one edge of the document—the left edge for odd-numbered pages, and the right edge for even. This puts the gutter on the inside of duplex-printed pages, leaving room for binding, three-hole punching or whatever. If you can't think of why you might need this, it can be left at its default of 0 points.

HTML Options

This page allows you to set several options for HTML output. However, due to the limits of the **groff_www** macros, it is not as complex as one might wish.

Back Link and Back Text

At the bottom of the Index page, a link is inserted to return the viewer to whatever root documents they accessed the index from. The Back Link entry allows the user to choose this page, and the Back Text entry allows the user to alter the text that is displayed.

Include Disclaimer

These three check boxes allow you to determine where the disclaimer.so file should be inserted; into the Index file, into each chapter file, and into the fanfiction.net chapter files. They are not mutually exclusive.

Insert bookxx.so files in index

FicMake does not generate HTML files for book headings. However, if you select this option, it will insert any bookxx.so files into the index, just after the book heading. For more on bookxx.so files, see page 10.

Place all chapter links on one line for each book

Largely self-explanatory. The only real use for this is if your fic is divided into multiple books, but does not have chapter titles. It can help neaten up the index page, especially if you have a lot of chapters.

Chapter Options and Titles

This options screen has the greatest potential to confuse newcomers to the software. It includes the following elements:

- Prologue;
- Epilogue;
- Entry Type;
- Chapter Title;
- Chapter List.

Prologue and Epilogue

These options declare whether or not a Prologue or Epilogue file exists. (For more on the required files, see page 9.) If Prologue is set, the first Chapter entry in the Chapters List will be the prologue. If Epilogue is set, the last Chapter entry in the Chapters List will be the epilogue.

Prologues and Epilogues are handled differently from normal chapters. They are outside the normal numbering structure, receiving "numbers" of 'pr' and 'ep' respectively.

Entry Type

This toggles the currently-selected Chapter Entry between Book and Chapter status. Books do not have associated files, and are not part of the normal numbering scheme. They have their own independent numbering.

Chapter Title

This is the title of the currently-selected Book or Chapter. By storing this here, there is no need to store it in the Chapter file.

Text Editor

By default, FicMake will launch kwrite to edit chapter files. This might not be your favourite editor; in fact, you might not have it installed at all. To select a different editor, change the value of this option.

Edit File

You can press F11 to edit the chapter file for the currently selected chapter. However, when the editor is running, FicMake will lock up, until you exit the editor. This is a *feature*; it keeps you from changing anything that might cause FicMake to lose data while editing a file.

The Chapter List

This section lists the titles of all current chapters. Use the up and down arrow keys to navigate the list.

On the left-hand side of the Chapter List is the current Book or Chapter Number. For Chapters, the number corresponds to the filename used by a Chapter file.

Press the **A** key to append a new chapter at the end of the list. This will also automatically select the Chapter Title so that you may edit the chapter immediately.

Press the **INS** key to insert a new chapter before the currently-selected chapter. As with appending a chapter, this will also automatically select the Chapter Title so that you may edit the chapter immediately. This will also automatically renumber all chapters below it automatically. Remember this when it comes to editing your source documents.

Press the **DEL** key to delete a chapter. As with inserting a chapter, this will automatically renumber all chapters below it.

Press **PgUp** or **PgDn** to move chapters up or down the list. This will renumber chapters accordingly.

With all the above, it is important to remember that the actual content of the chapters is determined by the chapter number, not by anything else!

There is a theoretical maximum of 99 chapters, plus prologue and epilogue, that this program can handle.

Two: Files

FicMake requires a fair number of files to work with in order to generate its output. It also spews forth a very large number of files as a result of its execution. The naming convention of these files is given below. Names are important! You must keep to the naming convention as much as humanly possible.

The Long Filename and Short Filename entries allow you to fine-tune your filenames to some degree, as described below.

Input Files

The various chapter files are the most important input files required by FicMake. They contain the bulk of your story, after all. They come in three flavours: the basic chapter files, teaser files, and omake files. However, several other files are used for many purposes as well.

Chapter Files

Chapter files are composed of the short filename of your story, followed by the chapter number (or 'pr' and 'ep' for prologues and epilogues), with the extension '.so'.

For example, if your story has the short name 'tso', and you have a prologue, three chapters and an epilogue, your chapters would be in the following files:

tsopr.so

tso01.so

tso02.so

tso03.so

tsoep.so

The Chapter Options page will list the required suffix for each chapter file in the left-hand margin, as described on page 8.

Teaser Files

Some authors - such as myself - make use of teasers. These are short segments of text, meant to be included before the chapter headings. Teaser files should be named **teaserXX.so**, where XX is the chapter number of the chapter in which the teaser will be included.

Teasers are handled differently according to the format in which you are using them. In Postscript and PDF format, as well as in bulk text format, they will be included after the chapter headings. In text chapters and html files of either flavour, they will be included before the chapter headings.

Omake Files

In the world of manga, an Omake is a short, amusing comic included at the end of a regular feature, typically one page in length. The concept has been pulled into fanfiction, where an Omake is a short story, not considered part of the regular story (and often conflicting badly with such). An Omake may be included at the end of a story by adding a file named **omakeXX.so**. Again, XX will be the chapter number of the chapter in which the Omake will be included.

Omake are added only in html and chapter text formats. They are ignored for Postscript, PDF and bulk text formats. HTML index files will not show the Omake in the list.

Trailer Files

Similar to teasers and Omake files, a trailer file is tacked onto the end of the story, and is intended to serve as a "sneak preview" of the next chapter. For this reason, it is included only in the fanfiction.net html files and the chapter text files, as these are the ones intended for serial posting.

A Trailer may be added to a chapter by adding a file named **trailXX.so**. As always, XX is the chapter number of the chapter in which the Trailer will be added.

Book Files

If you are using Books in your fic, and you create a file named **bookXX.so**, with a number corresponding to the Book number, it will be included just under the book heading in PDF format. This can be used for anything from adding illustrations, blurbs or introductions to creating custom macros specifically for that Book. As usual, XX is the Book number of the book in which the book file is to be included.

If you do not create a book file, FicMake will display only the Book number and title.

Blurb

A blurb is a short description of the story in its entirety. If the file **blurb.so** is present, it will be inserted into the title page of PDF, Postscript and bulk text output, and the index file of html output.

Disclaimer

Fanfiction is, by definition, a violation of copyright. The fact that many authors don't care - or actively support fanfiction - doesn't change the fact that it's still technically illegal. So it has become standard practice to include a disclaimer at the beginning of each fic, if only to salve the conscience of the fanfic writer.

If the Disclaimer file **disclaimer.so** exists, it can be added to the following locations:

- The title page of the story in bulk text, Postscript and PDF;
- The Index page of the story in html format;
- The top of each chapter page in chapter text, html and fanfiction.net format.

Credits

If the file **credits.so** exists, it will be inserted into the title page of Postscript, PDF and bulk text output, and at the bottom of the index file of html output.

Output Files

FicMake generates a horde of output files. These include the bulk wrappers, the chapter wrappers, and files needed to format for various outputs.

Chapter Wrappers

FicMake creates chapter wrappers with .ms and .ff extensions. These are used by groff to produce html (both varieties) and plain text formats. The chapter wrappers call the file **top.so**, also created by FicMake. When **make** is run in the story directory, you will likely see errors produced from trying to call .HR and .HTML macros when generating plain text. Ignore them; groff does.

Bulk Wrappers

FicMake also creates index.ms, which is used to generate the index page for html mode, and .ms files for bulk plain text and Postscript. These files do not call top.so, and each is individually formatted. However, the bulk plain text and Postscript files must be called three times each to resolve all cross-referencing. FicMake will handle this for you. (It's why I *wrote* the blasted program, after all.)

Table of Contents and Cross-Reference Files

FicMake creates files with .toc and .ref extensions, to contain all document cross-references. The .toc file contains the Table of Contents, and the .ref file contains the logic needed to create bookmarks and cross-reference data. These are overwritten every time you run **make**, so if you delete them, it doesn't matter.

Fic Descriptor File

The file **desc.fic** is created by FicMake, to store information on the story. It is plain text, and can be hand-edited, though there is no real reason to do so.

If you create a directory in your home called **.ficmake** (**~/ficmake**) and place a **desc.fic** file within it, everything in that file will be loaded as a default for any new fic. You can use FicMake itself to edit this file, though FicMake will also create wrappers and such.

Directories

On its first run for any one story, FicMake creates a number of empty directories, called **htm**, **txt** and **ffn**. It also creates a subdirectory of **htm**, named with the Long Filename alone, to hold the chapter files. HTML output, plain text chapter output, and HTML for fanfiction.net output are distributed to these directories. The PDF, Postscript, and bulk plain text files, however, are created in the root directory of the story.

Makefile

Finally, the Makefile is created. This contains all the rules that GNU make requires in order to create the various output files. Once FicMake has built the wrapper files, you can simply type **make** to create the output files.

You can also do a 'make clean' to remove all generated files, including the container directories. Typing 'make' will *not* work after this, despite the fact that the Makefile is not wiped out. You will need to re-generate the story information by running **ficmake -a**.

Three: Groff and MS

FicMake uses groff for all formatting, and uses the ms macros as its default. However, it also expands on the default groff/ms combination, as needed by the output formats.

The files found in the **manual** subdirectory were prepared using only the escapes and macro requests outlined in this document. You may use them as an example of what you can do with just a tiny bit of groff knowledge, and the FicMake program.

Escapes and Macros

Groff uses escapes and macros to format text. Escapes begin with ' or \ characters, and macros begin with . characters. As a result, there are a few rules that *must* be followed:

1. Never start a line with a period (.) unless it is a macro request. If you absolutely must start a line with a period, put a space before it, and the space will be stripped.
2. Never start a line with a ' character, as this character also signals a macro request. If you must, prepend it with \, and it will display properly.
3. A \ character signals an escape, and will cause groff to complain, and the rest of your line to be swallowed. If you must use a backslash, insert two in a row, in this manner: \\

Text Filling

Unless otherwise interrupted, groff will fill lines of text with as much as they can hold automatically. Interrupting text fill can be done with a blank line, or with a paragraph formatting macro, as covered in the next section.

There is no need to worry about word wrapping your text files. Lines may be as long or as short as you wish, and groff will cheerfully combine, fill and word wrap them. As an

example, if you examine the **chapter03.so** file in the **manual** subdirectory, you will discover that each word of this paragraph was on its own line.

Paragraph Formatting

Chapter files may use groff tags to organize content, but it is not strictly needed. At its very minimum, a file may consist of paragraphs separated by blank lines. All lines within a group not separated by a blank line will be wrapped and filled.

This paragraph gives an example of the sort of output you will get with just a blank line between it and the preceding paragraph. Note that the first line of this paragraph is not indented, and the spare blank line above it is left in place.

The **ms** macros include a number of ways of separating paragraphs other than with blank lines. These macros are automatically loaded by FicMake for each file in its output.

Most of this document uses a **.PP** macro. Insert **.PP** by itself on a blank line, and the next paragraph will be indented. In PDF or HTML format, a small amount of space, less than a full line, is inserted; in plain text format, a full line is inserted between paragraphs.

This paragraph had the **.LP** macro called before it. Note that while it is otherwise identical to the above paragraph, the first line is not indented.

This paragraph had the **.IP** macro called before it. As a result, the entire paragraph is indented.

1. This paragraph also had the **.IP** macro called before it, but with an argument, as follows: **.IP 1**. This inserted 1. before the indented paragraph, and is a good way of creating numbered points.

 - This paragraph used **.IP \(\bu** to create a bullet before the indented paragraph. Note that in plain text, the bullet will be replaced with a lower-case o.

† Another useful character to use is `\(dg`, which produces a dagger (†). Note, however, that the dagger is not available in plain text. Several other useful glyphs for the `.IP` macro include the double dagger `\(dd` (‡), the square `\(sq` (□) and the line-centered asterisk `\(**` (*).

??? In fact, you can use almost any text after an `.IP` macro call, and it will be displayed in the left margin. However, if there is insufficient room in the inserted margin for the text, it will be placed before the paragraph begins. For this paragraph, we used `.IP ???`

Further paragraph types are available through groff calls and ms macros, but they tend to break things when used in html and text output, so I shall not detail them here. The above should be good for all your fanfic needs.

Character Formatting

Bold and *italics* are available, as well as ***bold italics***. There are two ways to alter the character's format:

1. Use the `\fB` call inline for bold, or `\fI` call for italics. `\fR` signals a return to normal text. `\f[BI]` is used to select bold italics.
2. Use the `.B` and `.I` macros to signal that the next line is to be bold or italics, or `.BI` for bold italics. The `.R` macro will signal a return to normal text.

Here are some examples of each. The following line:

This line will have `\fBbold\fR` text and `\fIitalics\fR` text.

will produce:

This line will have **bold** text and *italics* text.

The following block of code:

This line will have

```
.B
bold
```

```
.R
text,
.I
italics
.R
text and
.BI
bold italics
.R
text.
```

will produce:

This line will have **bold** text, *italics* text and ***bold italics*** text.

Obviously, using escapes is far more compact than using macro requests.

Font Size

Font size can also be changed, by inserting a `.ps` request. The `.ps` request accepts one parameter, being the point size of the font requested. Relative changes can also be made, by including a `+/-` sign. The `.ps` request does not break filling, so requests can be made on the fly.

The following block of code:

```
.ps 14
This is an example
will produce the following output:
```

This is an example

An example of on-the-fly and relative font size changes:

```
We need to make
.ps -2
these words
.ps +2
smaller than the rest
```

will produce the following output:

We need to make these words smaller than the rest

Font family

Finally, you can change the font family. In Postscript format, there are seven font families available:

```
A  Avant-Garde
BM Bookman
C  Courier
H  Helvetica
N  New Century Schoolbook
P  Palatino
T  Times New Roman
```

The bulk of this document was written in Times New Roman. To request a different font family, use the **.fam** request, followed by the letter associated with the family you want.

For example, to request the Helvetica font (also known as Arial), use the following request:

```
.fam H
```

Font changes of any sort are wiped out next time you make a paragraph request - that is, call any of the **.PP**, **.LP** or **.IP** macros.

In plain-text format, font sizes cannot be adjusted, and there is no way to show bold or italics without using ANSI or ASCII escapes. Most programs that allow you to view plain-text on platforms other than Linux will barf when they see these escapes. Therefore, groff strips them out. No font shifts will occur in plain text.

If you used inline **\fB**, **\fI** or **\fR** requests, FicMake will call **sed** to replace them with asterisks. **sed** will *not* remove **\f[B]** requests—in fact, it will mangle them badly—nor will it affect the **ms** font requests.

In addition, in any HTML format, font families will be ignored and stripped out.

Reserving Space

The **.ne** request is a handy way of checking to see if a column break is needed. It is called with one parameter, the amount of space to reserve. This number is in Postscript points, which isn't always useful. Append an **i** to the number to express it in inches instead. This call was used at the beginning of this section, right above the header macro, to force a new column for this section.

For example, the following line:

```
.PP
.ne 0.5i
```

will check to see if there is half an inch left in the column, and force a column break if there is not.

Note that the **.ne** request is called *after* the paragraph request. This is very important. If it is called before the paragraph request, the last line of the previous paragraph will appear at the start of the next column, which is probably *not* what you want.

Boxes

It is possible to wrap text in boxes, if you so desire. Put a **.B1** request right before the text to be boxed up, and a **.B2** request right after it.

For example, the following code:

```
.PP
.B1
This output is contained within a
  box
.B2
```

will produce the following output:

This output is contained within a box

Note that the boxed text is unbreakable; if there is insufficient room remaining on the page for the box, groff will shift it to the top of the next page.

Note also that the **.B1** request is called after the **.PP** request. The type of paragraph request immediately preceding the **.B1** request will still

set the paragraph style for the boxed data. In addition, paragraph requests can be called within the box itself, without having to end the box.

Boxes in plain text mode are drawn with +, - and | characters, and work relatively well.

Boxes in HTML output are not yet working, due to limitations in groff itself. Maybe next version. If you're viewing this manual in html format, you'll see how badly it's broken. In fact, every time you run make, any boxes will appear in PNG files starting with 'grohtml' in the fic root directory. You could probably manually relocate them into the htm/longname directory, but this is a pain.

FicMake macros

This covers all the escapes and macros that you really need to know for groff and ms. However, FicMake includes a few additional macros that are handy to know.

Headers

FicMake supports five styles of headers, called using the .H0 through .H4 macros.

The **.H0** macro is called only internally, and you should never call it unless you *really* know what you're doing. This macro is defined only if a Book entry exists in the chapter list. If defined, it also inserts an entry in the Table of Contents. In html mode, this macro is never called, nor even defined, and calling it will simply produce an error.

The **.H1** macro is called internally, and there is little reason to call it manually. It places the heading at the top of a new page, centered above any columns. If a Heading Graphic is set, this is centered behind the output. The .H1 macro also inserts an entry in the Table of Contents, indented if the .H0 macro is defined. In html mode, it increases the font size three steps and sets the colour to blue.

The **.H2** macro is not called internally. It will insert an indented entry in the Table of Contents if the .H0 macro is *not* defined. In html

mode, it increases the font size two steps and sets the colour to blue.

The **.H3** and **.H4** macros are not called internally. They left-align their text. In html mode, they increase font size one step and set the colour to blue. They never create a Table of Contents entry.

All Header macros are called with one parameter, the text to be displayed. If the text includes spaces, you must wrap it in quotes. The .H0 macro accepts a second parameter, which is printed below the first on the page.

Font changes, including family, weight and point size, are set within the program. The defaults are suitable for almost any document.

For example, the following line:

```
.H3 "This is an example"
```

will produce the following output:

This is an example

Dividers

The **.SEP** macro can be used to insert a Divider. The exact appearance of the Divider can be set from within FicMake, as detailed on page 6. Place the .SEP macro on its own line. In fanfiction.net output, the Divider is replaced with a horizontal rule, as this conforms to the requirements of fanfiction.net postings.

It is usually worthwhile to make a .SEP request its own paragraph, as was done in the source code above, by placing it after an .LP request, and following it with your next paragraph request. The Divider above was generated with the following code:

```
.LP
.SEP
.PP
```

Cross-References.

Cross-references may be created with pairs of **.BOOKMARK** and **.XREF** calls. These macros only work with Postscript and PDF files.

The **.BOOKMARK** tag takes one argument, being the name of the bookmark being generated. This name can be any alphanumeric characters, upper- or lower-case, but it is case sensitive and cannot include spaces.

The **.XREF** tag looks up the saved bookmark by name, and returns its page number. It accepts two arguments. The first argument is the bookmark to look up. The second, if given, is immediately appended to the bookmark output.

For example, the following call:

```
.BOOKMARK story-arc
```

will store in the reference file a lookup for the current page named 'story-arc'. To call this back up, one might use the following code:

```
Recalling the story-arc bookmark  
gives us page
```

```
.XREF story-arc .
```

Note the trailing period as the second parameter. Were we to omit that, and put the period on the next line, the output would be:

```
Recalling the story-arc bookmark gives us  
page 11 .
```

Note the gap between the period and the page number. But with that trailing period on the request call, we instead get

```
Recalling the story-arc bookmark gives us  
page 11.
```

Image Support

Image support is at best rudimentary in FicMake. The program supports importing of JPG/JFIF files. The extension of the file does not matter, as long as it is a genuine JFIF file.

To insert an image, use the **.JPEG** macro, followed by the name of the image. If the name of the image includes any spaces, be sure to wrap the image name in quotation marks.

As an example, this document comes with a graphic header, called 'title.jpg'. To include this graphic, type:

```
.JPEG title.jpg
```

This will embed a graphic in the output file, resulting in something like this:



Using **.JPEG** in html format will copy the image to the htm/longname directory, so that the web pages can find it. The graphic will be automatically sized to 50% of your screen width, and adjusted to the right. It will also adjust the paragraph immediately after it (this one, if you are looking at the web page version of this documentation) to wrap around the picture.

Using **.JPEG** in a PDF file will cause groff to create a postscript file for each picture, which it will *not* clean up. It will cause groff to output a couple of message lines for each pic, telling you what is going into the pic. If something goes wrong with your image, you will be able to see it here. The image will be shrunk, if necessary, to fit the available page width. Text will *not* be wrapped in PDF output.

Using **.JPEG** in fanfiction.net format, or any text format, will cause groff to complain.

Appendix: Output Formats

Each of the output formats has its own notes that need to be addressed.

HTML

HTML output is intended for posting to a web page. All Headings are in blue, and all font controls are supported. Individual pages for each chapter are generated, as is an index file that links to all of them. Navigation bars at the top and bottom of each chapter allow the reader to navigate from one chapter to the next or previous, or back to the index. Disclaimers are included in each chapter, and teaser files are relocated to before the titles.

Page layout details are ignored entirely, save that the **.IP** request will still indent a paragraph.

The Index file is placed in **./htm**, and the chapter files are placed in **./htm/longname**.

HTML for fanfiction.net

Since fanfiction.net strips most tags out, several requests will not function as described above. Header requests, colour changes and size changes of all sorts will appear in the generated files, but fanfiction.net will strip them out when you upload the file. Only font changes—bold, italics and regular—will remain untouched.

Other changes will be made by FicMake itself. Separators are replaced with horizontal rules. Disclaimers are inserted at the top of every chapter file. Teasers appear before the chapter headings, trailers and omake files are included.

The individual chapter files are placed in **./ffn**.

Plain Text

Plain text is the most limited of the available formats. All font escapes are replaced with asterisks, font-related requests are discarded, headers and footers are removed, and the only page layout available is portrait letter, one column.

Individual chapters are placed in **./txt**

Plain Text Bulk File

As with the individual plain text files, all formatting is discarded, and page layout is limited to portrait letter, one column. Headers and footers are included, but are very minimalist. One large bulk text file, containing a Table of Contents and all chapters, is placed in **./**.

Postscript

Postscript format is the most involved of the various formats. All font escapes and headers are supported, as is page size, Table of Contents, margins, credits.so, disclaimer.so and blurb.so. Headers and footers are created, as is a title page and a two-level Table of Contents. Book entries, if any exist, generate title pages as well.

However, colour is not supported, and teaser files are located after the titles, instead of before them. Trailer and omake files are omitted completely.

PDF

The Postscript file is translated *verbatim* into a PDF file. Therefore, all that holds for Postscript also holds for PDF. The PDF file will *not* include indexing, as support for this in groff is still rudimentary and buggy. (Not to mention confusing.) In addition, the PDF file is not encrypted, and can be edited at will.