Almost Free Text (AFT) Reference Manual

Todd A. Coram

September 10, 2010

Version 5.98

revised 09/09/2010

Contents

---

1 Introduction

This document exists as a simple reference manual for the AFT. It is not meant to be used as a tutorial. [1]

AFT is a *nearly* free format documentation system which can be typed in using any editor or wordprocessor that supports tabs or *hard spaces* (column-based spaces whose number doesn't shrink or grow based on formatting). By using AFT, you are no longer constrained to one wordprocessing file standard (such as Microsoft Word), nor do you have to enter a plethora of weird syntactical incantations of an *embedded* mark up language (such as LaTeX or HTML). [2]

Unlike other mark up languages, AFT is designed to parse and recognize *patterns* rather than formal commands. That is why there is no single *escape* or command sequence that tells AFT what to do. In this regard, AFT will process almost anything you throw at it.

An AFT document is easily converted into such popular formats as HTML, RTF and LaTeX. Because there are few commands, learning to write a document in AFT requires little effort. This doesn't mean that AFT is not powerful. In fact, this very document was conceived and written using AFT.

1.1 How to Read this Document

In this document, *user entered text* is set in `teletype`. Everywhere you see the sequence `[tab]`, this is just a way of showing that the user is expected to press a *tab* key (or enter 8 consecutive spaces). See Tab Modes (§ 6) for instructions on how you can set *tabs* to *any* number of consecutive spaces.

Each command introduced will begin with a description and then will be followed by one or more examples of the command as it would be entered. In most cases, the actually result of the command is shown.

2 Running AFT

Running AFT is as simple as typing:

```
aft NAME_OF_YOUR_FILE.aft
```

Replace `NAME_OF_YOUR_FILE` with the full path of your file. It should end with `.aft` although that is not a rule. You can also give multiple file names too. AFT will, by default, use the first supplied file name as a basis for the name of the output file. The name of the output file will have an

[1] That's not quite true. If you read this in a linear fashion, it's a bit like a tutorial!

[2] LaTeX and HTML are the major mark up languages supported; there is lesser support for other formats – mileage may vary.

extension that reflects the default designated output type. This default was chosen when you first installed AFT.

There are a few more parameters that can be supplied to AFT. Here is what AFT prints out if you don't give it any arguments:

```
aft [--verbose] [--autonumber] [--output=file | --output=-] [--type=output_type] infile ..
```

Everything enclosed in `[]` is optional. Where you see a bar '|' indicates that you can supply one of the two parameters on either side.

Here is a breakdown of each parameter:

–**verbose** Generate a lot of commentary. By default, AFT will just silently process files (unless errors occur). Using this option causes AFT to keep you informed about what it is doing.

–**autonumber** This switch tells AFT to automatically number your sections.

–**output=file** | –**output=-** This tells AFT where to send its processed output. You can supply a filename (file) or `-` which tells AFT to write to your standard output (your display or `stdout`).

–**type=output_type** This tells AFT what type of output to do. For example HTML output is used for `html` and DocBook output is used for `docbook`. As a side effect, this will also specify the file name extension for the output file if the `-output` option isn't specified.

infile .. One or more AFT documents to be processed.

It is very unlikely that AFT will display errors. Since this is a *mostly free* text parser, nothing in your source document will ever be syntactically incorrect. In most cases, AFT will produce perfectly parsable output. But, remember: *garbage in, garbage out.*

3 AFT Text: An Example

The following text represents typical input to the AFT parser:

```
 *Title: Musical Categorization
 *Author: Alfred Theodore Franti

[tab]# I am an optimist. From where it is, music is mostly alright,
[tab]  or at least in a healthy state for the future, in spite of
[tab]  the fact that it may sound as though it is being held
[tab]  hostage. --- Duke Ellington

There are many ''flavors'' of music to choose from. Indeed, music is a
lot like food: different tastes for different folks.  But, what are
the ''flavors'' that music comes in?  Here is a short, not nearly
comprehensive list:

[tab]* Jazz
[tab]* Classical
```

```
[tab]* Blues
[tab]* Country
[tab]* Folk
[tab]* Rock
[tab]* Rap
[tab]* R&B
[tab]* Soul

There are many others categories and sub-categories that I have failed
to mention.
```

After we run this text through AFT, we get:

Musical Categorization

Alfred Theodore Franti

> I am an optimist. From where it is, music is mostly alright, or at least in a healthy state for the future, in spite of the fact that it may sound as though it is being held hostage. — Duke Ellington

There are many *flavors* of music to choose from. Indeed, music is a lot like food: different tastes for different folks. But, what are the "flavors' that music comes in? Here is a short, not nearly comprehensive list:

- Jazz
- Classical
- Blues
- Country
- Folk
- Rock
- Rap
- R&B
- Soul

There are many others categories and sub-categories that I have failed to mention.

---

4 Syntax Overview

AFT processes your input file one line at a time. A line is a chunk of text terminated by a newline, carriage return or whatever combination of the two that is native to your operating system (don't worry, your text editor handles it all behind the scenes for you).

4.1 Line Continuations

If the last character on your line is a backslash \ , then the next line is taken as a continuation of your current line and all of the AFTish things that are done on a line-by-line basis are now applied to to all of the lines *continued* with backslashes.

This is terribly useful when you present AFT with text that may have been *wrapped* by your text editor (because the line was too long):

```
Here is a long URL typed across 2 lines: http://www.maplefish\
.com/todd/aft.html
```

produces

Here is a long URL typed across 2 lines: http://www.maplefish.com/todd/aft.html

After a full line (continuations included) has been parsed, AFT determines whether the line is part of a special mode. The modes of AFT are:

1. Paragraph - the normal mode.
2. List - a tab mode.
3. Verbatim - a tab mode.
4. BlockVerbatim - a special mode between ^<< and ^>>.
5. Quote - a tab mode.
6. Centered - a tab mode.
7. Table - a tab mode.

4.2 Modes

Sections are treated as temporary *one shot* modes. After a sectional change occurs, you re-enter paragraph mode. Other modes can be thought of as *sticky* (you can enter a mode and stick indefinitely in that mode).

An empty line terminates most modes and takes you back to paragraph mode.

4.3 Special characters

Any line beginning with a `#` probably signifies that Comments (§ 9.4), SET (§ 9.5.2) controls or Pragmas (§ 9.5) follow.

A line beginning with a `*` is sectional. That is, Sections (§ 5), TOC (§ 5.2.2), or Images (§ 5.2.3) follow.

4.4 Line Filtering

AFT tries to apply *prefilters* from the rule file (`.dat` file), then it searches and replaces Targets and References (§ 8), and performs [Font Faces] tricks. After the line has been completely brutalized by AFT, *postfilters* are applied.

5 Sections

All sections start with at least 1 (but no more than 7) stars * in the leftmost column of a line. Each star represents a section level. With the exception of special sections (Special Sections (§ 5.2)), a space can follow the final star before the section name. The section name is terminated by a line break. **Warning:** *Trailing spaces are considered part of the section name.*

```
* This is a Top Level Section
**This is a Second Level Section
```

Here we can see how the various section levels nest:

5.1 Section 2

While HTML supports 6 section levels, we can't expect this level of support from every target.

5.1.1 Section 3

Here we see Section 3.

Section 4 And now Section 4. This is pretty deep nesting. Do you want to go lower?

Section 5 Starting with Section 5, the text body begins to indent. This carries until the end of the section. [3]

1. Lists are indented too!
2. Everything is indented.

Section 6
We are starting to get really nested here. Beware!

1. You really should consider restructuring your text.
2. Do you really need this many nested sections?

Section 7
This is as deep as we go. This shouldn't show up in the table of contents.

5.2 Special Sections

AFT has a few reserved section names. These are used to provide special services that can be logically considered *section* based. All special sections consist of one star * followed immediately by the section name.

[3] From Section 5 down to Section 7, won't show up in the table of contents... thank goodness.

5.2.1 Titles and Authors

The title of the document can be signified by using the section name `*Title:` followed (optionally) by a space and the title name.

```
*Title: Almost Free Text (AFT) Reference Manual
```

The author of the document follows a similar format. Instead of `*Title:`, you enter `*Author`

```
*Author: Todd A. Coram
```

If `*Title:` and `*Author:` are the first two (non-comment) lines in the document, they will be used in the document preamble. This is useful for formats such as HTML which would like to have the title of the document in the 'header' rather than 'body'.

5.2.2 TOC

`*TOC:`, or Table of Contents, is used to automatically generate a table of contents section for your document. The table of contents is populated by names from Sections (§ 5).

If need to collect your own table of contents information (for output that doesn't automatically generate it – such as HTML), then you should run the `aft` command on your source files twice. The first pass produces a special table of contents file and the second pass inserts that special file.

```
*TOC
```

5.2.3 Images

A very simple image importing facility is available in AFT. The `*Image` command is followed by the name of an image file that is to be incorporated into the document, replacing the line where the command appears.

```
*Image: %aftimage%
```

The above line will insert the file `aft.jpg`:

There are a few variations on `*Image:`. You can suggest how the image should be placed in the document with the following commands:

- `*Image-left:`

- `*Image-center:`
- `*Image-right:`

The results are as follows:

```
*Image-left: %aftimage%
```

```
*Image-center: %aftimage%
```

```
*Image-right: %aftimage%
```

You can also use `*Image:` to cascade images:

```
*Image: %aftimage%
*Image: %aftimage%
*Image: %aftimage%
```

6 Tab Modes

While processing your document, AFT may enter certain *modes*. The most common mode is *Tab Mode*. Requesting tab mode is requested by entering a *tab* ([tab]) character as the first character in a line. A tab mode ends at the first line encountered that *does not* begin with a tab character. This includes blank (empty) lines.

If you cannot type *tabs*, you can use spaces instead. By default, AFT will interpret every 8 consecutive spaces as tabs. You can change this default by using the command:

```
#---TABSTOP=N
```

where `N` is the number of spaces you wish to represent tabs.

Examples:

```
#---TABSTOP=4
#---TABSTOP=8
```

6.1 Lists

Lists are the most popular tab modes. A list element is identified as any line containing a initial tab and followed immediately by one of the following character sequences :

Enumerated (§ 6.1.2) Element Any number of digits followed by a `.` or `)`.

Bullet (§ 6.1.3) Element A single star `*`.

Named (§ 6.1.4) Element A left bracket `[` followed by text and ending with a right bracket `]`.

Nested (§ 6.1.5) One or more tab characters.

You may mix and match list element types. However, each time you change type, you are effectively starting a new *list group*.

In addition, each list element in a list group must immediately follow the last element. There cannot be any blank lines between elements.

```
[tab]1. This is the first element.

[tab]2. This is NOT the second element.
```

produces

1. This is the first element.

1. This is NOT the second element.

6.1.1 Tab Continuations

If a tab element doesn't fit physically on the same line you are entering, it may be continued by entering a new line, a tab and any character not present in the list sequences (§ 6.1) described above. However, it is recommended that you simple use a single space in order to keep your lists consistent.

```
[tab]* This is the first physical line in the element.
[tab]  This is the second physical line in the element.
[tab]* This is a new element.
```

produces

- This is the first physical line in the element. This is the second physical line in the element.
- This is a new element.

You can also take advantage of AFTs parsing syntax described in Syntax Overview (§ 4) and do the following:

```
[tab]* This is the first physical line in the element.\
This is the second physical line in the element.
[tab]* This is a new element.
```

produces

- This is the first physical line in the element.This is the second physical line in the element.
- This is a new element.

but looks rather awkward...

6.1.2 Enumerated

An enumerated list element is indicated by any number of digits followed by a `.` or `)`. The actual number is ignored. The list is always enumerated starting at `1`. If you find that choosing numbers is too much trouble (especially since AFT ignores them), you can replace the number with `#` (i.e. `#)` or `#.`).

```
1. First thing is first.
3. Third thing is always second.
#) Oh, I give up. You chose the numbering.
```

produces

1. First thing is first.
2. Third thing is always second.
3. Oh, I give up. You chose the numbering.

If you want to control the numbering, then use a Named (§ 6.1.4) list with your numbers as the names.

6.1.3 Bullet

A bullet list element is indicated by the presence of a single star `*`.

```
[tab]* My gawd. It's full of stars.
[tab]* Actually, sir, it is just a single star.
```

produces

- My gawd. It's full of stars.
- Actually, sir, it is just a single star.

6.1.4 Named

A named list element is indicated by a left brack `[` followed by text (the *name*) and ending with a right bracket `]`. The text following this indicator is supplemental.

```
[tab][Aardvark] A small mammal that munches ants.
[tab][Ant] A small creature munched upon by aardvarks.
```

produces

Aardvark A small mammal that munches ants.

Ant A small creature munched upon by aardvarks.

6.1.5 Nested

Lists can nest. Nesting modes are managed automatically by AFT. You can freely enter and exit nesting levels at will. A Nested list is indicated by an additional tab followed by any of the list sequences (§ 6.1). The same rule for Tab Continuations (§ 6.1.1) apply.

```
[tab] * List, at level 1.
[tab] * Another item at level 1
[tab][tab] with a continuation.
[tab][tab] 1. a level 2 item.
[tab][tab] 2. another level 2 item.
```

```
[tab][tab][tab] * A level 3 item.
[tab][tab] 3. the third level 2 item.
[tab][tab][tab] * A special level 3 item.
[tab][tab][tab] * More special level 3 items.
[tab] * Back to level 1.
[tab][tab] * A new level 2 list.
[tab][tab][tab] 1. A new level 3.
[tab] 1. A new level 1.
[tab] 2. Got another level 1.
```

produces

- List, at level 1.
- Another item at level 1 with a continuation.
    1. a level 2 item.
    2. another level 2 item.
        - A level 3 item.
    3. the third level 2 item.
        - A special level 3 item.
        - More special level 3 items.
- Back to level 1.
    - A new level 2 list.
        1. A new level 3.

1. A new level 1.
2. Got another level 1.

6.2 Other Tab Modes

Aside from Lists (§ 6.1), there are a couple of other tab modes.

6.2.1 Quoted Text

Want to enter simple verse or a quote? This is a very simple mode. It is indicated by a tab immediately followed by `#`. Any following lines (preceded by one or more tabs) are considered part of the quote.

```
[tab]# You're _damned_ if you do;
[tab]You're _damned_ if you don't. - Bart Simpson
```

produces

> You're **damned** if you do; You're **damned** if you don't. - Bart Simpson

6.2.2 Verbatim Modes

Verbatim mode is the *catch all* tab mode. Any entry into a tab mode that doesn't look like Lists (§ 6.1), Centered Lines (§ 7.3), Tables (§ 9.3) or Quoted Text (§ 6.2.1) is considered *verbatim* text and will appear in the output in a `teletype` font with limited processing of text. Line breaks are always honored in this mode.

There are two flavors of verbatim text: Tabbed (§ 6.2.2) and Literal (§ 6.2.2). Each of these flavors will, in general, cause text to appear as it was written.

Tabbed A tabbed verbatim mode is entered under the rules of Tab Modes (§ 6) and are limited to any sequence that doesn't fall under Lists (§ 6.1).

```
 [tab] 1. This line enters tabbed verbatim mode and not list
 [tab]    mode because it starts with a space.
 [tab]
 [tab]Tabbed verbatim mode retains
 [tab][tab]hard carriage returns, spaces,
 [tab][tab]tabs and other drudge.
```

produces

```
  1. This line enters tabbed verbatim mode and not list
     mode because it starts with a space.

 Tabbed verbatim mode retains
         hard carriage returns, spaces,
                 tabs and other drudge.
```

Literal This is a very special kind of verbatim mode. It isn't legally a tab mode, but allows you to forgo all of the initial tabs that you must type in order to stay in tab mode. Literal mode begins with the character sequence ^<< as the first characters in a line and ends with the character sequence ^>> as the first characters in a line.

```
^<<
while (<RF>) {
  chop;
  $lcnt++;                # increment the line count
  /^(\s*\#|\Z)/ && next; # Skip comment lines
  doStuff();
}
^>>
```

produces

```
while (<RF>) {
  chop;
  $lcnt++;                # increment the line count
```

```
  /^(\s*\#|\Z)/ && next; # Skip comment lines
  doStuff();
}
```

A special modifier on the literal sequence allows us to apply *filtering* to the literal text. That means that the actual layout of the entered text is retained, but we can apply commands such as font changes to the text. This is indicated by typing the word `Filter` or `filter` immediately after ^<<.

```
^<<Filter
_while_ (<RF>) {
  _chop_;
  $lcnt++;                ''# increment the line count''
  /^(\s*\#|\Z)/ && _next_; ''# Skip comment lines''
  doStuff();
}
^>>
```

produces

```
while ($<$RF$>$) {
  chop;
  $lcnt++;                # increment the line count
  /^(\s*\#|\Z)/ && next; # Skip comment lines
  doStuff();
}
```

7 Presentation

AFT offers the user very little control over the explicit layout of the produced document. This limitation is a trade-off for ease of use. AFT is not meant to produce *flyers* or *slick sheets* or documents with strong page layout requirements.

With this in mind, there are a few niceties provided to help you with some level of control over your document layout.

7.1 Paragraphs

All paragraphs are ended by a single *blank* (empty) line.

7.2 Page breaks

The traditional form-feed character ^L (Control L) can be used to explicitly break pages.

7.3 Centered Lines

A line of text can be centered by modifying the Tab Modes (§ 6) rule a bit. If you are not in the middle of a Lists (§ 6.1), Tables (§ 9.3), Verbatim (§ 6.2.2), or Quoted Text (§ 6.2.1), then you could enter a *centered* line.

A centered line is a line of text preceded by 2 or more tabs.

```
[tab][tab][tab]Hey, I am the Center of the Universe.
[tab][tab]I am pretty self-centered too.
[tab] Keep in mind.
[tab][tab]You can't center while in a tab mode.
```

produces

Hey, I am the Center of the Universe.

I am pretty self-centered too.

```
Keep in mind.
       You can't center while in a tab mode.
```

7.4 Separator Lines

A single horizontal line can be produced by typing at least 4 dashes on a line by itself. You should add a blank line before and after the dashes in order to insert line breaks.

```
What is above.

-------------------------

Must never go below.
```

produces

What is above.

Must never go below.

7.5 Font Faces

AFT doesn't allow you to change your fonts, that is left up to configuration of your Web browser (for HTML) or TEX installation (for LATEX). However, AFT will allow you to change your font's *face*.

AFT's typeface scanner works on a line by line basis. In order to change the face of a piece of text, you must surround it with AFT font command sequences. The beginning and ending sequences must appear on the same physical line.

```
He was quite _bold and daring_ in his criticism.
He was quite _bold and
daring_ in his criticism.
```

produces

He was quite **bold and daring** in his criticism. He was quite _bold and daring_ in his criticism.

An exception to this rule is when you want to change a paragraph (or multiple lines) of text. In this case, you place the beginning font command sequence as the first characters in the first line and place the ending command sequences as the last characters in the last line.

```
_Warning. Do not touch here. Touching here could cause
severe pain and possibly death.
You have been warned. So don't touch here._
```

produces

Warning. Do not touch here. Touching here could cause severe pain and possibly death. You have been warned. So don't touch here.

7.5.1 Bold

Bolded text is surrounded by the character `_`. If you wish to include this character literally in your text, then you must break your physical line so that it only appears once, or you can double up the `_` character.

```
_This is your brain on bold._  This__is__not__your__brain__on__bold.
```

produces

This is your brain on bold. This_is_not_your_brain_on_bold.

7.5.2 Italics

Italicized text is surrounded by the character sequence ■. If you wish to include this sequence literally in your text, then you must break your physical line so that it only appears once, or you can double up the ■ sequence.

```
''It was quite droll''.  ‘‘This isn't droll at all’’’’.
```

produces

It was quite droll. "This isn't droll at all".

7.5.3 Teletype and Small Text

Teletyped text is surrounded by the character `|`. If you wish to include this character literally in your text, then you must break your physical line so that it only appears once, or you can double up the `|` character or precede the pipe with a backslash \ . Small text works the same way, except it uses `~`.

```
|int small__number = 1 \| 2;| ~# this holds a small number \~ (1 or 2).~
```

produces

`int small_number = 1 | 2;` # this holds a small number ~ (1 or 2).

8 Targets and References

HyperText is supported for HTML and References are supported for printed output such as LaTeX. Both use the same command sequences.

8.1 Targets

Targets are anchor points in a document, often referenced by References (§ 8.2). Targets take the form:

```
=[text]=
```

or

```
=[(text)]=
```

The difference being that the former *inserts* the text at the point of the target while the latter (with parenthesis), provides just a reference point without inserting the text.

The `text` is used by References (§ 8.2).

Here are a couple of examples:

```
=[Does your dog bite?]=  No, my dog doesn't bite.

=[(Ow, he bit me!)]= That... is not my dog.
```

produces

Does your dog bite? No, my dog doesn't bite.

That... is not my dog.

8.2 References

A reference takes one of three forms:

```
Form #1 [text]
Form #2 [text (reference_text)]
Form #3 [text (url:reference_text)] or [text (:reference_text)]
```

where `text` references target (§ 8.1), unless specific `reference_text` is is supplied (and in that case, `reference_text` refers to target (§ 8.1).

The first form (#1) is the simplest. It refers to a target in the same document file. The second form (#2) allows arbitrary text to be used. The third form (#3) allows arbitrary text to refer to outside URLs. `url` may be `ftp`, `http`, `https`, `mailto` just omit it (leaving the ':') for relative http references. If you want to embed URLs directly into your document, you can dispose of the brackets altogether. (See URL Targets (§ 8.3)).

Here we refer to targets set up in Targets (§ 8.1):

```
 [Ask about the dog (Does your dog bite?)]!
```

```
 [Ow, he bit me!]

See [how to avoid dog bites (http://www.hsus.org/ace/11858)].
```

produces

Ask about the dog (§ 8.1)!

Ow, he bit me! (§ 8.1)

See how to avoid dog bites http://www.hsus.org/ace/11858.

But, what if you want to include plain old bracketed text like [Coram97] that isn't a link? You can break the bracketed text into two lines:

```
 [Coram97
 ]
```

but that many introduce unwanted space after 97 (i.e. [Coram97]).

So, you probably want to *escape* the bracket with a backslash \ :

```
\[not a link nor does it have any unwanted spaces]
```

Because, this is [not a link nor does it have any unwanted spaces].

8.3 URL Targets

If you want to just want to supply plain old `URL` targets, you don't need to use any special markup. You just type in the address. AFT doesn't recognize sophisticated URL naming when using this feature. You need to keep it simple (e.g. URLs containing parens or complex http parameters should be avoided).

For example:

```
 * http://www.maplefish.com/todd is my home page.
 * (http://www.maplefish.com/todd)
```

produces

- http://www.maplefish.com/todd is my home page.
- (http://www.maplefish.com/todd)

8.4 Targets (Deprecated)

A target begins with a **right** curly brace } followed by a visibility indicator, the target text and ends with a **left** curly brace {.

There are two visibility indicators:

1. Visible target indicator `+`.
2. Invisible target indicator `-`.

Visible target indicators cause the target text to appear in your text output, while invisible target indicators hide your target text.

```
}-Important Information-{  My dog may bite.
}+Very Important Information+{: My dog bites.
```

My dog may bite. Very Important Information: My dog bites.

You should choose target text that have no AFT mark up or special characters in them. Keep them short and safe.

8.5 References (Deprecated)

A reference begins with a **left** curly brace { followed by a visibility indicator, text, an optional target text and ends with a **right** curly brace }.

There are two visibility indicators:

1. Full Visible target indicator `+`.
2. Half visible target indicator `-`.

The first is used when the reference text **is** the target text. The second indicator is used along with a `@` character to indicate that target text will be supplied but shouldn't be visible. The form for this is `visible text@invisible target text`

```
{+Important Information+} is worth reading.
{-Information@Very Important Information-} is available.
```

Important Information (§ 8.4) is worth reading. Information (§ 8.4) is available.

8.6 URL Targets (Deprecated)

URL target references can be provided by using one of these several forms.

1. URL addresses. Use `http:`, `file:`, `ftp:` or `mailto:` in the target name.
2. Local files. Use `text@:filename` or `text@:filename#target`.

Here are few examples.

```
* {+http://www.maplefish.com/todd+}
* {-AFT Home Page@http://www.maplefish.com/todd-}
* {-AFT Reference Manual Source@:aft-refman.aft-}
* {-AFT HyperText Info@:aft-refman.html#URL Targets-}
```

produces

- http://www.maplefish.com/todd
- AFT Home Page http://www.maplefish.com/todd
- AFT Reference Manual Source aft-refman.aft
- AFT HyperText Info aft-refman.html#URLTargets

9 Miscellaneous

9.1 Indexing

This is an experimental new feature. It has not been determined if supporting indexing for AFT-sized documents is a useful thing.

However, if you are looking for the ability to provide more precise referencing than Sections (§ 5), then indexing may be it.

Indexing capability is provided by simply providing target style mark up near the word you wish to index. For example:

```
For example: (look in the index for this...)
        =[^contrived example]=
```

The above example is indexed under the term *example*. An index is just a target with a caret (`^`) preceding the term. However, unlike true targets, the term does not appear in the text.

If you have marked an index, then at the end of your document an Index section will be automatically generated. [4]

9.2 Footnotes (Endnotes)

AFT has a very simple *footnote* facility (for systems like HTML, where true page-footer notes are not available, they are treated as *end notes*).

Footnote syntax is a variation upon References (§ 8.2). You indicate a footnote by starting a reference with '`[Note:`'. All text following `Note:` up to a terminating brace `]` is taken as the footnote. For example:

```
Footnote syntax is a variation upon references.[Note: This \
is just a note about footnotes. You can go back \
to reading the manual now.] Okay?
```

produces

Footnote syntax is a variation upon references.[5] Okay?

In the above example we also showed how long footnotes (most will probably exceed a lines length) can be broken up using Line Continuations (§ 4.1).

9.3 Tables

A very simple table facility is provided with AFT. A table mode is introduced with a tab followed by a `!`. Any line not starting with this sequence terminates the table.

A table consists of the following parts, in the following order:

1. Caption.
2. Headers.

[4] This is currently only supported for HTML and LaTeX. For LaTeX, you must run `makeindex` to create the index.
[5] This is just a note about footnotes. You can go back to reading the manual now.

3. Data elements.

Header items and Data elements are separated by a single `!`. Any table Header or Data elements entry that consists of fewer than 2 items generate a warning and are ignored.

You can use !——-! to separate table rows (this improves the AFT source document readability).

For example:

```
 [tab]! _Very_ Important Dates!
 [tab]!----------------------!
 [tab]! Year ! Month ! Day    !
 [tab]!----------------------!
 [tab]! 1966 ! Oct   ! 9      !
 [tab]! 1999 ! Dec   ! 31     !
 [tab]! 2000 ! Jan   ! 1      !
```

produces

Very Important Dates

Year	Month	Day
1966	Oct	9
1999	Dec	31
2000	Jan	1

9.3.1 Row Spanning

Trying to fit long text onto a single line looks pretty ugly using the default Table mode. A small improvement in layout can be had by introducing the pragma `#--SET-CONTROL tableparser=new`. When this pragma is in effect, you **must** use line separators `---` between rows. This is what will tell AFT that you are done with a row.

So, now you can do something like:

```
 #---SET-CONTROL tableparser=new

! _Very_ Important Dates     !
!---------------------------!
! Year ! Month ! Day         !
!---------------------------!
! 1966 ! Oct   ! 9  (Todd's  !
!      !       ! Birthday.   !
!      !       ! Bring gifts)!
!---------------------------!
! 1999 ! Dec   ! 31          !
!---------------------------!
! 2000 ! Jan   ! 1           !
!---------------------------!
```

to produce:

Very Important Dates

Year	Month	Day
1966	Oct	9 (Todd's Birthday. Bring gifts)
1999	Dec	31
2000	Jan	1

9.4 Comments

If you want to type in text that will NOT be processed by AFT, then you simply preface each line, that you want to be ignored, with a `C` or `#` followed by 3 or more dashes `-`.

```
#------ This line is an AFT comment.
Maroc Ddot is
C------ Todd Coram! The secret is out!
unknown.
```

produces

Maroc Ddot is unknown.

9.5 Pragmas

Ah, pragmas. For those of you unsatisfied with the text manipulation of plain old AFT, you can dive into its dark corners with the following pragma features.

9.5.1 PASS

If AFT doesn't support a feature of the targeted output format, you can cheat and embed markup that is passed through directly. This is done by using the `#-PASS-xxx` command, where `xxx` is the `ID` of the output format. The `ID` can be found in the AFT *dat* file used to describe the output format.

For example, the following pass through will output raw HTML *only* when the targetted output format is HTML:

```
Is this
#---PASS-TROFF // Troff silliness not supported yet!
#---PASS-HTML <font color=red size="+1">
red
#---PASS-HTML </font>
?
```

Is this red ?

9.5.2 SET

`#--SET` is used to assign a chunk of text (including AFT markup) to a variable that can be freely referred to and expanded throughout your document. (If you are an AFT hacker and have peered into the *.dat files, you may see `#--SET` is use there too.)

The format of `SET` is:

```
#---SET variable=text
```

Throughout your document, `variable` can be referenced as %`variable`% and will be replaced by `text`.

Before you get too excited, note that `SET` only works for one line of text. Here is an example:

```
#---SET aftimage=aft.gif
*Image-center: %aftimage%
```

The advantage here is that you only have to modify `aftimage` if you wish to refer to a different image file and all expansions of %`aftimage`% will result int he different image file.

If you find that you are using `#--PASS` a lot, you may want to consider a variant on `SET` that (like `PASS`) uses the module name for conditional expansion. So, extending our previous example, if the image file is different depending on what output type you choose, you can do the following:

```
#---SET-HTML aftimage=aft.gif
#---SET-LaTeX aftimage=aft.eps
*Image-center: %aftimage%
```

There is a subtle but very important difference between `SET` and `SET-xxx`: the former is done during preprocessing and the latter is done post-processing. The effect is, you can include AFT markup in `SET` and output specific markup in `SET-xxx`.

10 End Credits

Thanks go out to Ward Cunningham, whose Wiki-Wiki Site editor inspired the initial work on AFT. Also, thanks to everyone who has used AFT over the years and have offered invaluable feedback.

Index