Migrating the RWAP SuperBASIC Manual Documentation

Release 0.1.1

Norman Dunbar

Oct 07, 2016

Contents:

CHAPTER

ONE

INTRODUCTION

This book, for want of a better word, describes the trials and tribulations I went through to create the current (as of today, anyway!) online version of Rich Mellor's **Online SuperBASIC Manual** as seen originally at http://www.rwapsoftware.co.uk/SBASIC_reference_manual_online/Foreword.html and now, online at http://superbasic-manual.readthedocs.io/en/latest/ where the latest (English language) version can be found. Anyone feel like tranlsating?

1.1 Why Bother?

Many years ago, Rich wrote a proper book, using Text 87, I believe, and this was published by Roy Wood's company *Q Branch*. Although I never had a copy, I am informed that it was a very well received book, and regular updates were provided. The book itself was over 1,000 loose leaf pages - which helped in the updating.

Time goes by (that's its job after all!).

The internet arrives.

The QL is somewhat left behind, given a pretty basic lack of TCP/IP networking abilities.

A group of *old farts*[1] still hang on to the QL, after all, it is one of the better computer systems around, given its age, and the fact that it did have one of the better BASIC languages around, at the time. (1984!)

Even better, it had a *genuine* "32" bit processor, which was sort of correct, but it had an 8 bit data bus, so each of those 32 bits needed 4 fetches! Still, advertising.

1.2 The First HTML Version

Some work was done by authors unknown, to convert the original Text 87 documents into a first draft HTML document. This involved much messing around with printer drivers and then reading through the generated (and binary) file to replace certain characters with suitable HTML alternatives.

Note: Getting text out of a Text87 file is a nightmare as the internal file format is not documented, and the author doesn't appear to answer emails on the subject. A printer driver type thing is about all there is. Especially if you wish or need to retain some of the formatting.

The first version was a team effort - if you were involved, let me know and I'll update this book to include your name - and a set of very rough HTML files was the result. There were bugs, but the browsers of the

[1] *Old farts* is a genuine term of endearment, and not what you might be thinking!

time coped beautifully - by simply ignoring them! The section on the ED keyword, for example, had a table of key presses which never actually showed any of the arrow keys, for example.

There were the odd occasional spelling mistake - as would be expected in a work of this size. There probably still are!

There were tables that were no longer tables - HTML ignores multiple spaces and tabs, replacing them with a single space, in most cases. Not helpful to a nice tabular layout.

And there were program listings which were not program listings, they were paragraphs of italic text. All the lines ran together and any of the special HTML characters - '<', '>' or '&', for example, went awol as they were illegal characters in a non `<pre> .. </pre>` section. Sigh.

However, it was a good start.

1.3 My Own Involvement

Rich put a request out on the ql-users mailing list for help, and also on the Sinclair QL Forum. I believe the silence was deafening! And, being an author myself, pretty much as expected. So much for a community eh? [2]

I used to avoid having a lunch break in whatever location I was working at the time. This meant that I had to sit around for an hour or so, twiddling my thumbs or reading the Interwebs. Until one day I decided to have a look at tidying up the online manual - as requested by Rich in his call for help. I was bored!

I managed to *manually* convert all the forewords and appendices to something resembling half-decent HTML, but it was still pretty dire to be honest. Each file is many hundreds of lines long, and full of *stuff* generated by the various conversion programs. It wasn't easy to work with and each chapter took ages to convert from really crappy HTML into mildly crappy HTML.

By the time I left that contract, I had only got as far as converting the first three chapters of the keywords - A, B and C. That was it. At least the program listings looks like listings now, but it was an absolute nightmare converting each line from something like[3]:

```
... <span style="font-style: italic;">100 do_stuff(a, b, c)...</span> ...
↪<span style="font-style: italic;"><br>110 do_more_stuff: and_more_
↪stuff(d,e,f) ...</span>
```

into:

```
<pre>
100 do_stuff(a, b, c)...
110 do_more_stuff: and_more_stuff(d,e,f) ...
</pre>
```

Which is fine for a couple of lines, but a nightmare for some of the larger examples, even with a text editor that lets me record and save the odd occasional macro to do the stripping out of the various `<span>`s etc.

Then, I left that contract and spent almost 3 years working in a location that was completely tied down - no personal internet usage, no ability to install even useful tools that could have helped me do my job

[2] To be fair, this is a common occurence in many communities. Everyone wants stuff done, but only a few dedicated people do anything to do the work. Others, sadly, don't, can't or won't!

[3] Yes, most of it was all on one, long, line!

etc. So, no work done at all on the manual I'm afraid - my wife wasn't too keen on me *wasting* my spare time doing *computer stuff*, so doing anything at home was a non-starter too. Apparently, I needed to *get a life* - whatever one of those is. [4]

1.4 This Version

This version came about as a result of something else I was having to do. Producing a web site from text files created in ReStructuredText. I was rather impressed and ran a little trial before creating a project on *GitHub* and on *ReadTheDocs* to cater for the need to have the source files in a version control system - so that when I screwed up an edit, I could always revert the changes! And ReadTheDocs gave me the ability to host the manual for free, well, for the inclusion of a couple of small adverts, occasionally.

Every time I commit a change and push it to the working branch, the working version of the manual gets rebuild in HTML, PDF and EPUB formats. This is the rough and ready versions and is not really suitable for general use.

The master branch is only ever updated, other than when I make a mistake and forget to checkout the working branch of course, when I finish a complete file, such as `KeywordsX.html` which will become `KeywordsX.clean.rst` and then built. The docs on the master branch are the good copies, ready for use.

Never, ever update the docs for the master branch, unless you have already proven them in working first.

[4] I think it means shopping for shoes and clothes, for her that is!

CHAPTER

TWO

SOFTWARE REQUIRED

I did all my stuff on Linux, well most of it, I did install everything on a Windows 7 box, just to see if it was any good. Surprisingly enough it was, but then again, that's the value of Open Source stuff. It *usually* works!

Overall, the biggie of the conversion process is a tool called Sphinx, or Sphinx-doc to avoid searching Google for statues of lion beasts from Ancient Egypt! ;-)

Head over to http://www.sphinx-doc.org/en/stable/install.html where you will find details of installing just about everything you need for Windows to run the Sphinx documentation builder.

2.1 Python

Most Windows users do not have Python, so we begin with the installation of Python itself. If you have already installed Python, please skip this section.

Go to https://www.python.org/, the main download site for Python. Look at the left sidebar and under "Quick Links", click "Windows Installer" to download.

Version 2.7 is suitable, but 3.x is probably better in the long run if you plan on doing more Python work on Windows.

Run the installer and install the package somewhere suitable. Make sure you update your path to suit, or nothing will work!

The Sphinx link above has all the details, and indeed, some of the text I copied above came from there!

2.2 Pip

Once Python is installed, install pip:

```
c:\> python get-pip.py
```

Simple stuff. pip is the Python Installer Program and is useful in installing Python utilities. We need it to install Sphinx.

2.3 Sphinx

Once Pip is installed, install Sphinx:

```
c:\> pip install sphinx
```

Once this is complete, make sure it worked by running the command:

```
c:\> sphinx-build --version
```

Which will show you that your PATH etc are set up well enough to build things. You should see something like:

```
Sphinx (sphinx-build) 1.4.6
```

2.4 Sphinx RTD Theme

Sphinx used to come with a Read The Docs theme installed automatically, but more recent versions don't include it for some unknown reason. Not to worry, if you try to build docs using that theme, you will be prompted to install it as follows:

```
c:\> pip install sphinx_rtd_theme
```

CHAPTER

THREE

CONVERSION SOFTWARE

The software above is all you need to be able to run the *building* of the docs, however, if like me, you are *converting* from the old HTML files, you need a few more bits and pieces.

3.1 Cmake

https://cmake.org/download/ is where you can find the Cmake downloader for Windows. You will need it to compile HTML Tidy.

3.2 HTML Tidy

HTML Tidy is used to clean up the rough HTML generated from the Text 87 sources and product something that has been validated for cleanliness and standards adherence. It is best to have a clean starting point before converting to other formats.

https://github.com/htacg/tidy-html5 is the URL to head on over to, and from there you can download the source code. It can be built from source using something called cmake, which you should also install.

3.3 Pandoc

Pandoc is used to convert the cleaned up HTML into ReStructuredText format. This can also convert numerous input formats into many different output formats, it is the *Swiss Army Knife* of document conversions.

Head on over to http://pandoc.org/installing.html where you will find instructions and downloads for installing Pandoc.

3.4 Git

Git is needed to allow the SuperBASIC-Manaul repository to be updated. The download details can be found at https://git-for-windows.github.io/.

3.5 A C++ Compiler

On Linux, just about everything uses the G++ compiler. That's definitely what I use, however, Windows is the proverbial nightmare. Visual Studio Express Edition is freely available for download and use, but it's no longer proper C++ as Microsoft went over to their "java-like" .net nonsense years ago. Even their C++ compiler cannot compile proper standard C++ code - does that sound familiar? Microsoft and standards not matching up?

The best ever compiler on Windows was always Borland C++ and many years ago, they gave away the 5.5 version to anyone who wanted it. I used it happily for years on Windows. Sadly, Borland sold out to Embarcadero, but the 5.5 version is still available.

Even better, Embarcadero recently started giving away version 10 of the compiler which is right up to date. You can get it at https://www.embarcadero.com/free-tools then follow the links to the C++ compiler.

You will need to create an account, but this only causes a few special offers in your inbox, some of them useful!

If anyone is interested, this is what Embarcadero have to say about their free tool:

> *This free download of the C++ Compiler for C++Builder includes C++11 language support, the Dinkumware STL (Standard Template Library) framework, and the complete Embarcadero C/C++ Runtime Library (RTL). In this free version, you'll also find a number of C/C++ command line tools—such as the high performance linker and resource compiler.*

3.6 All Done!

That's (about) it! Nothing too excessive now, was it? Too much software perhaps? Well, nobody said that it was going to be easy. But remember, once all this is installed, you have a very good and useful document production system.

CHAPTER

FOUR

THE TOOLS

This section describes the tools that you should find in the tools directory, what they are for, and brief details of what they do.

Note: The following code examples assume that the `tools` sub-directory has been added to your PATH.

The tools are unlikely to be of much use to anyone - they are designed for the initial conversion of the old, rough, HTML into something that Pandoc can convert to ReStructuredText.

4.1 buildTools.sh

This script calls the Linux C++ compiler, g++, to compile all the various c++ source files into a binary. It is executed as follows:

```
buildTools.sh
```

Windows users will need to convert this to something resembling a batch file - if one doesn't already exist, but given the plethora of different C++ compilers available for Windows, it's unlikely that one will exist. (Update: I lied, see *buildTools.cmd* and adapt it for your particular C++ compiler - it defaults to Borland C++ 10.1)

4.2 fixHeadings

This C++ file removes some random characters from each command's main section heading. These get created when the document is converted from HTML to RST - by dint of some weird and wonderful HTML in the original file!

This utility also converts numerous HTML headings into bold text as they did not need to be headings as such.

This utility gets automatically run by the HTML2rst.sh shell script.

It can, however, be executed on it's own as follows:

```
fixHeadings < KeywordsX.clean.rst > output_file.rst
```

4.3 fixLinks

This C++ utility converts existing links to other commands in the generated RST files so that they point to `KeywordsX.clean.html#xxx` rather than to `KeywordsX.html#xxx`.

The fix also includes the lower-casing of the command name and the replacing of some unusable characters with hyphens - to match what Sphinx generates.

Any '$' or '%' in the link names or anchors - the bit after the '#' - get dropped.

4.4 fixLinkComments

During testing, fixLinks output the original link text, which it was converting, as a comment line in the RST file. It was found that those commenst caused erros in other utilities further down the chain, so they had to be removed.

As there were numerous comments, this utility was quickly written to find and destroy them. It is executed manually, on demand, by the following:

```
fixLinkComments < KeywordsX.clean.rst > output_file.rst
```

It should no lonegr be required but lives on in source code form!

4.5 fixSyntax

This utility tries it's best to convert the Syntax: and Location: lines for each command, into a nice looking table. It gets it right pretty much *most* of the time, but occasionally, we still need to do some manual editing.

This utility gets automatically run by the HTML2rst.sh shell script.

4.6 preHTMLTidy

This one simply corrects any invalid '<br href="...">...' tags, which are simply and utterly wrong, into the correct form which is '...'. It can be executed by itself but is normally called from within the HTMLTidy.sh script:

```
preHTMLTidy < KeywordsX.html > output_file.html
```

4.7 Process.sh

This script simply calls the following two to do the work for a single html file. Once completed, and if no errors were reported, the various source files, error texts etc are moved to their appropriate directory.

4.8 HTMLTidy.sh

This script takes care of all the HTML handling to get a clean (cleanish?) HTML file to convert to RST. It calls out to preHTMLTidy first, then runs the HTML Tidy utility to attempt to clean up the HTML.

If there are any errors reported, the utility will stop.

If there are any warnings, these are listed, but in this case, the utility carries on.

Sed is then called to strip out a list of headings that have line breaks in them, either before - "
<Hn>" or after "<Hn>
" as these are just plain wrong! The edit also strips out any heading that is immediately followed by a line feed - "<Hn>n".

The output from all this is a clean HTML file named `KeywordsX.clean.html`. It shouldn't need to be, but can be called on it's own, as follows:

```
HTMLtidy.sh KeywordsX.html
```

This will produce the following files in the current directory:

- KeywordsX.html.tmp - a temporary work file.
- KeywordsX.clean.html - a ReStructuredText source file ready for manual tidying up and future processing.
- KeywordsX.errors.txt - a file holding all the errors and warnings from the (most recent) run of the HTML Tidy utility.

4.9 HTML2rst.sh

This script takes the clean HTML file produced by HTMLTidy.sh, and converts it to an RST file named `KeywordsX.clean.rst`.

It does this by running sed to strip out Rich's table of contents which is present at the head of each and every file in the original online manual - this is no longer needed.

Pandoc is then called to convert this truncated HTML file to RST.

Once the RST file is created, the utilities fixSyntax, fixHeadings and fixLinks are executed to do the needful.

It shouldn't *need* to be, but it *can* be called on it's own, as follows:

```
HTML2rst.sh KeywordsX.clean.html
```

This will produce the following files in the current directory:

- KeywordsX.clean.rst.tmp - a temporary work file.
- KeywordsX.clean.rst - a ReStructuredText source file ready for manual tidying up and future processing.

CHAPTER

FIVE

STEP BY STEP GUIDE TO THE MIGRATION

5.1 Clone the git Repository

In a shell (DOS) session, yes, we use the command line a lot here, I cloned my repository where all the latest up to date (master) and working sources can be found.

```
cd SourceCode
git clone https://github.com/NormanDunbar/SuperBASIC-Manual.git
```

This command created a directory named SuperBASIC-Manual and within it, the following sub-directories were created:

- clean - this is where the cleaned up html files are saved. If you have fixed all the errors and warnings them the files here are genuine HTML5 compliant.
- errors - where the errors and warnings from HTMLTidy are sent.
- html - this is where the old, dirty, html files are saved. These may be used in future to run a diff to see what, if anything, has been changed and may need updating in the RST [1] (ReStructuredText) files.
- sphinx - where the make files live. These help you build the various documentation formats.
- sphinx/source - where the numerous RST files live. These are the files I edited. This will be where any corrections of updates will be done too.
- tools - where the various tools, and their source code live. See the tools page for details.

Some build specific directories are created on demand too, once the appropriate build command is called. These are, or will be:

- sphinx/build/buildtrees - nothing to see here, move along! A working area used by the sphinx software. Don't edit anything here please! It's basically a cache so that only changed files are recompiled into HTML or Latex etc.
- sphinx/build/html - where the HTML generated documentation is/will be found.
- sphinx/build/latex - where the LATEX and LATEXPDF generated documentation is/will be found.
- sphinx/build/linkcheck - where the linkcheck command puts its output.

These latter directories will not appear until after the first build (of the appropriate format).

[1] RST is what I'm calling the ReStructuredText files from now on. It saves typing, and I'm basically lazy!

5.2 Checkout the working branch

Currently I have only checked out the *master* branch. We *do not* work on this branch, unless we are merging changes from the *working* branch.

The master branch is only for the fully edited and corrected files. These are used by "Read the Docs" to build HTML, PDF and EPUB versions of the SuperBASIC Manual, every time there is a git push to the master branch.

All work is done on the working branch. The working branch also causes a rebuild of the html, pdf and epub versions, but only of the working docs. These are *not* considered suitable for general consumption - that's what the master branch is for - but it gives the author/convertor an idea of how it will look when completed.

Remember, **We do not break the build!** and **The master branch always builds!**

```
git checkout working
git branch
```

You should see:

```
 master
*working
```

The asterisk before the branch name shows you the current branch. We can see from the above that we are indeed on the correct, `working` branch.

5.3 Build the Tools

The various tools I've created don't have their binary versions included in the repository, for obvious reasons. The source code is either a Bash shell script (*.sh) or a C++ source file (*.cpp).

For Linux users there's a buildTools.sh script to compile everything. For Windows users, try buildTools.cmd but you will need to adjust the default compiler used to match your system. I'm on Linux, so I'm happy!

The script will compile the following:

- fixHeadings.cpp
- fixLinks.cpp
- fixLinkComments.cpp
- fixSyntax.cpp
- preHTMLTidy.cpp
- And any more I later add of course!

There are a number of shell scripts,. Linux format, to call the various tools above. These will most likely need converting to Windows batch files, or PowerShell scripts - if you are that masochistic! You are on your own here, especially as I can't find a decent replacement for the `sed` editor, which is used in the various scripts.

5.4 Fetch an HTML File

```
wget http://www.rwapsoftware.co.uk/SBASIC_reference_manual_online/
↪KeywordsX.html
```

This creates a file named KeywordsX.html in the current directory. Alternatively:

- Open the page in your Internet browser. The following assumes Chrome.
- Right-click
- View page source
- Right-click
- Save as ...
- Then save the source with the same filename that you originally opened - `KeywordsX.html`, for example.

Warning: Do not select the 'save as' option when you first right-clicked on the page in the browser. That option is entirely different and downloads any or all supporting css files, images etc.

5.5 Run Process.sh

For Linux users, ie me, this file does the lot. Well, unless HTML Tidy exits with an error of course, but if it's just warnings, it will continue.

This script does a lot of work:

- runs preHTMLTidy.
- runs HTMLTidy.sh to, hopefully, produce a clean html file.
- runs a sed script to do some basic tidying up to line breaks in wrong places.
- runs HTML2rst.sh to convert the clean html to ReStructuredText in KeywordsX.clean.rst.
- Processes the Syntax and Location sections to build a nice looking table format.
- Changes various headings that shouldn't be actual headings, into bold - warnings, notes etc.
- Moves the KeywordsX.html file to the html directory.
- Moves the KeywordsX.clean.html to the clean directory.
- Moves the errors & warnings file from HTMLTidy.sh, KeywordsX.errors.txt, to the errors directory.
- Moves the generated KeywordsX.clean.rst to the sphinx/source directory, ready for editing.

Windows users could try having a look at the following to get an idea of what they will need to do ...

5.6 Run PreTidy

Assuming we are working with KeywordsX.html, then:

```
copy KeywordsX.html to KeywordsX.html.tmp
tools/preTidy <KeywordsX.html.tmp >KeywordsX.html
```

This sorts out the faulty <br href=”some url”> tags into correct <a href=”some url”> tags prior to the proper run of html tidy.

5.7 Test HTMLTidy.sh

Run a test tidy of filename.html:

```
HTMLTidy.sh filename.html
```

This will display any errors first. If there are any, edit filename.html to correct them, them repeat the run of HTMLTidy.sh until it only shows warnings.

Warnings should be checked, but you can ignore stuff that says that there are unescaped ‘&’s or ‘<’ and ‘>’ etc. If you see warnings about “unknown entity dir1” or similar, the you are probably looking at something like:

```
dir1$=datad$&dir1$
```

where the ‘&’, which will be flagged as unescaped, is causing the variable dir1$ to be considered as an HTML entity, similar to " or < or & etc.

These can also be ignored. The problem(s) are caused by the original html file setting program listings as italic body text rather than wrapped in <pre> </pre> tags.

Note: A lot of the code examples don’t have spaces in the lines, unless absolutely necessary - it makes for interesting reading!

Things like ‘escaping linefeed in <a> tag” should be fixed. You will find something like:

```
<a href="some url">
anchor text here</a>
```

which should be all on one line, not on two:

```
<a href="some url">anchor text here</a>
```

Once happy that the warnings etc can be ignored, move on.

5.8 Edit the Source

This is where the fun begins. All - or as much as possible - of the automatic tidying has been done. The rest is up to me/you/us!

5.8.1 Paragraphs

Start typing at the left margin. Don't stop typing at the end of a line - but if you do, don't worry, the line feed will be ignored, and replaced with a space as the various lines making up the paragraph are merged.

Leave a blank line between paragraphs, otherwise it all runs into one.

5.8.2 Syntax/Location Table

Making sure that the synatx/location table is correctly formatted. The code generates a single line for each, but where there are "or" options in either Syntax or Location, these need to be on separate lines with a double pipe (||) to force a newline in the table.

If a command only has a single form, then a table like this will suffice:

```
+----------+-------------------------------------+
| Syntax   | Command(a,b)                        |
+----------+-------------------------------------+
| Location | QL ROM, TOOKIT II, SOMEWHERE ELSE   |
+----------+-------------------------------------+
```

The above will render into the following:

Syntax	Command(a,b)
Location	QL ROM, TOOKIT II, SOMEWHERE ELSE

However, if there are alternate forms, the table needs to look like this:

```
+----------+-------------------------------------+
| Syntax   || Command(a,b) or                    |
|          || Alternative(a,b,c)                 |
+----------+-------------------------------------+
| Location || QL ROM, TOOKIT II, SOMEWHERE ELSE  |
+----------+-------------------------------------+
```

which becomes the following after a build:

Syntax	Command(a,b) or Alternative(a,b,c)
Location	QL ROM, TOOKIT II, SOMEWHERE ELSE

Which gives a better layout in the generated HTML. You should note the doubling up of the pipe characters in both the syntax and location sections of the table.

5.8.3 Tables

We use grid tables, where we design the table according to how we want it to look:

```
+--------+--------+--------+
| Cell 1 | Cell 2 | Cell 3 |
+--------+--------+--------+
| Cell 4 | Cell 5 | Cell 6 |
+--------+--------+--------+
```

Cell 1	Cell 2	Cell 3
Cell 4	Cell 5	Cell 6

We can add headings too:

```
+--------+--------+--------+
| Head 1 | Head 2 | Head 3 |
+========+========+========+
| Cell 1 | Cell 2 | Cell 3 |
+--------+--------+--------+
| Cell 4 | Cell 5 | Cell 6 |
+--------+--------+--------+
```

Head 1	Head 2	Head 3
Cell 1	Cell 2	Cell 3
Cell 4	Cell 5	Cell 6

Cells can span along the row:

```
+--------+--------+--------+
| Head 1 | Head 2 | Head 3 |
+========+========+========+
| Cell 1 | Cell 2 and 3    |
+--------+--------+--------+
| Cell 4 | Cell 5 | Cell 6 |
+--------+--------+--------+
```

Head 1	Head 2	Head 3
Cell 1	Cell 2 and 3	
Cell 4	Cell 5	Cell 6

And also, down the columns:

```
+--------+--------+--------+
| Head 1 | Head 2 | Head 3 |
+========+========+========+
| Cell 1 | Cell 2 | Cell 3 |
|        +--------+--------+
| Cell 4 | Cell 5 | Cell 6 |
+--------+--------+--------+
```

Head 1	Head 2	Head 3
Cell 1	Cell 2	Cell 3
	Cell 5	Cell 6
Cell 4		

Getting text on separate lines is interesting, and a bit of a hack. You've already seen it above for the Syntax/Location tables, but here we are again:

```
+--------+-----------------+
| Head 1 | Head 2          |
+========+=================+
| Cell 1 | Cell 2 - line 1 |
```

```
|          | Cell 2 - line 2 |
+----------+-----------------+
| Cell 4   | Cell 5          |
+----------+-----------------+
```

Head 1	Head 2
Cell 1	Cell 2 - line 1 Cell 2 - line 2
Cell 4	Cell 5

Looks good eh? No? Oh well, this works:

```
+----------+------------------+
| Head 1   || Head 2          |
+==========+==================+
| Cell 1   || Cell 2 - line 1 |
|          || Cell 2 - line 2 |
+----------+------------------+
| Cell 4   || Cell 5          |
+----------+------------------+
```

Head 1	Head 2
Cell 1	Cell 2 - line 1 Cell 2 - line 2
Cell 4	Cell 5

We have to "double pipe" all entries down the column(s) we want to have text on separate lines, unfortunately, plus the headings or we get strange indents.

5.8.4 Bullet Point Lists

These start with a hyphen and a space. Then the text of the list item follows on the same line. Do not press enter until you wish to start a new paragraph, and if intended to be part of the list item, it should be indented to line up under the first character after the hyphen space.

```
- This is a single line item in a list.
- This is another.

  However, this is not a third, but a second paragraph of the second list␣
↪item.

- And here we go back again. Item 3.
```

- This is a single line item in a list.
- This is another.

  However, this is not a third, but a second paragraph of the second list item.

- And here we go back again. Item 3.

You can nest lists:

```
- A top level item.

  - a nested item.
  - And another.

- Another top  level item.
```

- A top level item.
  - a nested item.
  - And another.
- Another top level item.

And so on.

5.8.5 Enumerated Lists

These are similar to the above, but start with a digit, or a hash dot (#.):

```
#. Item 1.
#. Item 2.
#. Item 3.

  #. Nested Item 1
  #. Nested Item 2

4. Item 4.
```

1. Item 1.
2. Item 2.
3. Item 3.
   1. Nested Item 1
   2. Nested Item 2
4. Item 4.

In the above, we had to start item 4 with the digit 4. There might be a way to get t his to work automagically, I'm still looking.

We can use Letters too:

```
a. Item a.
#. Item b.
#. Item c.
```

1. Item a.
2. Item b.
3. Item c.

Or Roman Numbers, but how? This should, but doesn't work:

```
I. Item I
#. Item II.
#. Item III.
#. Item IV.
#. Item V.
```

1. Item I
2. Item II.
3. Item III.
4. Item IV.
5. Item V.

As you can see the above doesn't work! It should allow i,ii,iii etc and I,II,III and so on, but this one doesn't seems to work.

5.8.6 Definition Lists

This is how we do definition lists:

```
A Term
    A definition goes here.

B Term
    B definition goes here. You can have many paragraphs too.

    Like this one you are currently reading.
```

A Term A definition goes here.

B Term B definition goes here. You can have many paragraphs too.

Like this one you are currently reading.

5.8.7 Program Listings

In-line

This is an example of an inline section of code, `1000 A$ = 'This is Fun!'`. We use the following to make it so:

```
This is an example of an inline section of code, ``1000 A$ = 'This is Fun!
↪'``.
```

Single line

A single line of code is defined thus:

```
::

    100 PRINT "Hello World!"
```

Which renders to the following:

```
100 PRINT "Hello World!"
```

The double colon can go at the end of the preceeding line, followed by a blank, followed by the indented code, or, can be on a line of its own, followed by a blank then the indented code.

Multiple lines

Multiple lines of code are defined thus:

```
::

    100 REPeat Silly
    110   PRINT "Hello World!"
    120 END REPeat Silly
```

Which renders to the following:

```
100 REPeat Silly
110   PRINT "Hello World!"
120 END REPeat Silly
```

The double colon can go at the end of the preceeding line, followed by a blank, followed by the indented code, or, can be on a line of its own, followed by a blank then the indented code.

5.9 Character Formatting

Bold is a pair of asterisks wrapped around the text yo uwant to be bold. No spaces between the leading asterisks and the text, or the end of the text and the trailing asterisks.

```
This will be **bold** text.
```

Which gives the following:

This will be **bold** text.

Italic is a single asterisk:

```
This will be *italic* text.
```

Which gives the following:

This will be **italic** text.

You want both? Tough, they cannot be nested.

5.10 Escaping Special Characters

The asterisk (*), the invisible space, the underscore (_), the backslash (\) (and more?) are special and have to be escaped using a back slash character (\):

```
Astersik = \*
Underscore = \_
Backslash = \\
Invisible\ space (did you see it?)
```

Whic gives the followig:

Astersik = *

Underscore = _

Backslash = \

Invisiblespace (did you see it?)

I think that's enough to be going on with! Check out this link for better information, but the above is fine for me.

5.11 Links

Oh yes, links, I forgot! Links are wrapped in backticks (‘) and double underscores (__) as per the following:

```
`link text here <URL Here>`__
```

- **External links** For example:

  ```
  This is a link to my `GitHub repository <https://github.com/
  ↪NormanDunbar/SuperBASIC-Manual.git>`__. Try it.
  ```

  This is a link to my GitHub repository. Try it.

- **Chapter & Section Headings** Each chapter and section heading becomes it's own link. Replace spaces, underscores etc with a hyphen in the link's URL to get to that section and lower case the remaining letters. For example:

  ```
  This is a link to the section above on `Escaping stuff <#escaping-
  ↪special-characters>`__.
  ```

  This is a link to the section above on *Escaping stuff*.

  However, if the section in in another document, then this is required:

  ```
  This is a link to the `Python <software.html#python>`__ section␣
  ↪in the Software chapter.
  ```

  This is a link to the Python section in the Software chapter.

CHAPTER

SIX

INDICES AND TABLES

- genindex