The Shorte Language

Reference Manual

15 October 2013

Document Number 34567

Revision 1.0.58

Revision History

Revision History		
Revision	**Date**	**Description**
1.0.0	08 July, 2013	Initial draft of the Shorte Reference Manual
1.0.58	15 Oct, 2013	Updated the documentation to describe preliminary install instructions, the new @h and @xml tags and the procedure to assign wikiwords to headings.

Table of Contents

1: About the Shorte Language

The Shorte language is a text based programming language used to generate documentation in a format that is familar to writing source code. It supports:

- include files for modularizing a document
- conditional includes and conditional text
- easy revision control and diffing of documentation
- cross referencing of C source code

1.1: Why another Language?

I started this project about two years ago because I wasn't happy with other markups like reStructuredText. There are a lot of really good markup tools out there but I decided to try my hand and creating my own since I could easily extend it and make it do what I wanted.

I wanted a markup language similar to HTML but not as verbose where I could sections within a document and have optional modifiers or attributes on tags to have more control over the document.

I also wanted the tool to be able to automatically cross reference my source code and pull it in similar to Doxygen but I found doxygen hard to format quite like I wanted.

1.2: Document Structure

Shorte documents generally end with a .tpl extension and follow the format

```
0001 # Document heading here
0002 @doctitle My Title
0003 @docsubtitle My Subtitle
0004 ...
0005
0006 # The beginning of the body
0007 @body
0008 @h1 Some title here
0009 Some text here
0010 ...
```

1.3: Shorte Comments

Shorte currently only supports single line comments using the # character at the beginning of a line.

```
0001 # This is a single line comment
0002 # and a second line to the same single line comment
0003 This is not a comment
```

If you want to use the # character elsewhere in the document it should normally be escaped with a \ character. This is not necessary inside source code blocks such as @c, @java, @python, etc.

1.4: Conditional Text

The Shorte language supports two types of conditional text

- PHY style inline blocks
- conditionals using the if="xxx" attribute on tags

1.4.1: PHY Style Code Blocks

These blocks of code are similar to the inline PHY syntax. You use the <? ... ?> syntax to inline a block of Python code. Any output must get assigned to a variable called **result** which gets return in its expanded form. In this way you can conditionally generate text or use Python to create documentation.

Variables can be passed to the interpreter using the **-m** command line parameter.

```
0001 <?
0002 result = ''
0003 if(1):
0004     result += 'This is some *bold text* here'
0005 if(0):
0006     result += 'But this line is not included'
0007 ?>
```

When output you will see something like:

This is some **bold text** here

1.4.2: Conditional Attributes

Conditional test is also supported using the **if=** attribute on a tag. For example:

```
0001 # Include this table
0002 @table: if="1"
0003 - Col 1 | Col2
0004 - Data1 | Data 2
0005
0006 # But not this table
0007 @table: if="0"
0008 - Col 3  | col 4
0009 - Data 3 | Data 4
```

will expand to:

Col 1	Col2
Data1	Data 2

As will the inline code blocks you can specify variables to pass to the **if** text to evaluate using the **-m** command line paramter.

1.5: Include Files

Shorte supports include files. There are two tags, @include and @include_child which are used to include files.

They can be included anywhere in the body of the document.

```
0001 @body
0002 @include "chapters/chapter_one.tpl"
0003 @include "chapters/chapter_two.tpl"
0004
0005 @include: if="ALLOW_CHAPTER3 == True"
0006 chapters/chapter_three.tpl
0007
0008 # Here we'll use the @include_child tag since
0009 # the @include tag normally breaks the flow of
0010 # conditional statements. By using @include_child
0011 # this file will only be included if ALLOW_CHAPTER3 == True
0012 @include_child "chapters/child_of_chapter_three.tpl"
```

Note:

Shorte currently can't handle include paths. The include path has to be a sub-directory where the top level file is included. Eventually support for include paths will be added.

1.6: Inline Formatting

TBD: Add description of this section

1.7: Shorte Tags

Shorte uses the @ character as a simple markup character. Wherever possible it attempts to avoid having an end character to make the document more readable and simplify typing. The document is entered by the use of tags that have the syntax @tag.

The following table describes the tags currently supported by Shorte:

Shorte Supported Tags	
Tag	**Description**
Document Metadata (only in document header)	
@doctitle	The title associated with the document
@docsubtitle	The subtitle associated with the document
@docversion	The version associatied with the document
@docnumber	The number associated with the document
@docrevisions	The revision history associated with the document

Document Body	
Heading Tags	
@h1	A top level header similar to H1 in HTML
@h2	A header similar to H2 in HTML
@h3	A header similar to H3 in HTML
@h4	A header similar to H4 in HTML
@h5	A header similar to H5 in HTML
Text Entry Tags	
@text	A document text block
@p	A paragraph similar to the **P** tag from HTML
@pre	A block of unformatted test similar to the **PRE** tag from HTML
Includes	
@include	This tag is used to include another file (breaks conditional cascade)
@include_child	This tag is used to include a child file (supports conditional cascade)
Images and Image Maps	
@image	An inline image
@imagemap	Include an HTML image map
Lists and Tables	
@ul	An un-ordered list
@ol	An ordered list
@table	A table
Notes, TBDs and Questions	
@note	A note

@question	A question
@tbd	A To Be Determined block
@questions	A list of questions
Structures and Functions	
@define	A C style #define
@enum	An enumeration
@vector	Similar to @struct but generates a bitfield
@struct	A C style structure
@prototype	C function prototypes
@functionsummary	A function summary
@typesummary	A type summary
Source Code Tags	
@c	A block of C code
@d	A block of D code
@bash	A block of bash code
@python	A block of python code
@sql	A block of SQL code
@java	A block of Java code
@tcl	A block of TCL code
@vera	A block of Vera code
@perl	A block of Perl code
@code	A block of unknown source code
@shorte	A block of shorte code
@xml	A block of XML code

Other Tags	
@shell	TBD
@inkscape	Include an SVG created in Inkscape
@checklist	Generate a checklist
@acronyms	A list of acronyms
@embed	An embedded object (HTML only)
Sequence Diagrams	
@sequence	Generate a sequence diagram
Test Case Definitions	
@testcase	A test case description
@testcasesummary	A test case summary

2: Installation Instructions

2.1: Installing on Windows

To install on Windows the easiest thing to do is to download a pre-compiled build of Shorte. This uses Py2exe to generate an executable version of the tool rather than requiring Python be installed on the system. The latest windows build can be downloaded from:

http://home-ott/~belliott/projects/shorte/releases/win32

2.2: Installing on Linux

2.2.1: From a pre-built Binary

The latest 64 bit linux build can be downloaded from: from:

http://home-ott/~belliott/projects/shorte/releases/linux64

2.2.2: From Source

To install from sources several prerequisites are required. They are shown in the following table:

Installation prerequisites		
Tool	**Tool Version**	**Description**
GCC	3.4 or later	A compiler used to compile the cairo plugin for Python
Make	3.81 or later	In order to run the makefile associated with the cairo plugin
Cairo	1.11 or later	Cairo is required for generating images such as structure definitions or sequence diagrams.
SWIG	2.0 or later	SWIG is required in order to build the cairo plugin for Python.
Python	2.6 or 2.7	Shorte is built in Python. Version 2.6 or later is required but 3.x is currently not supported.
Py2exe	Version for Python	This tool is used to generate

2.3: Changing the Path to LibreOffice/OpenOffice

In order to generate PDF documents it is necessary to modify the shorte.cfg file to setup the path to OpenOffice/LibreOffice. This may be done the following field in the config file:

```
0001 # Paths to open office swriter
0002 path.oowriter.win32=C:/Program Files/LibreOffice 4/program/swriter.exe
0003 path.oowriter.linux=/home/belliott/libreoffice/opt/libreoffice4.1/program/swriter
0004 path.oowriter.osx=/Applications/LibreOffice.app/Contents/MacOS/soffice
```

2.4: Setting up LibreOffice/OpenOffice

Shorte currently uses LibreOffice or OpenOffice for generating PDF or ODT documents. To do this it runs a conversion script convert_to_pdf.odt. This script handles updating the table of contents and converting the document to a PDF.

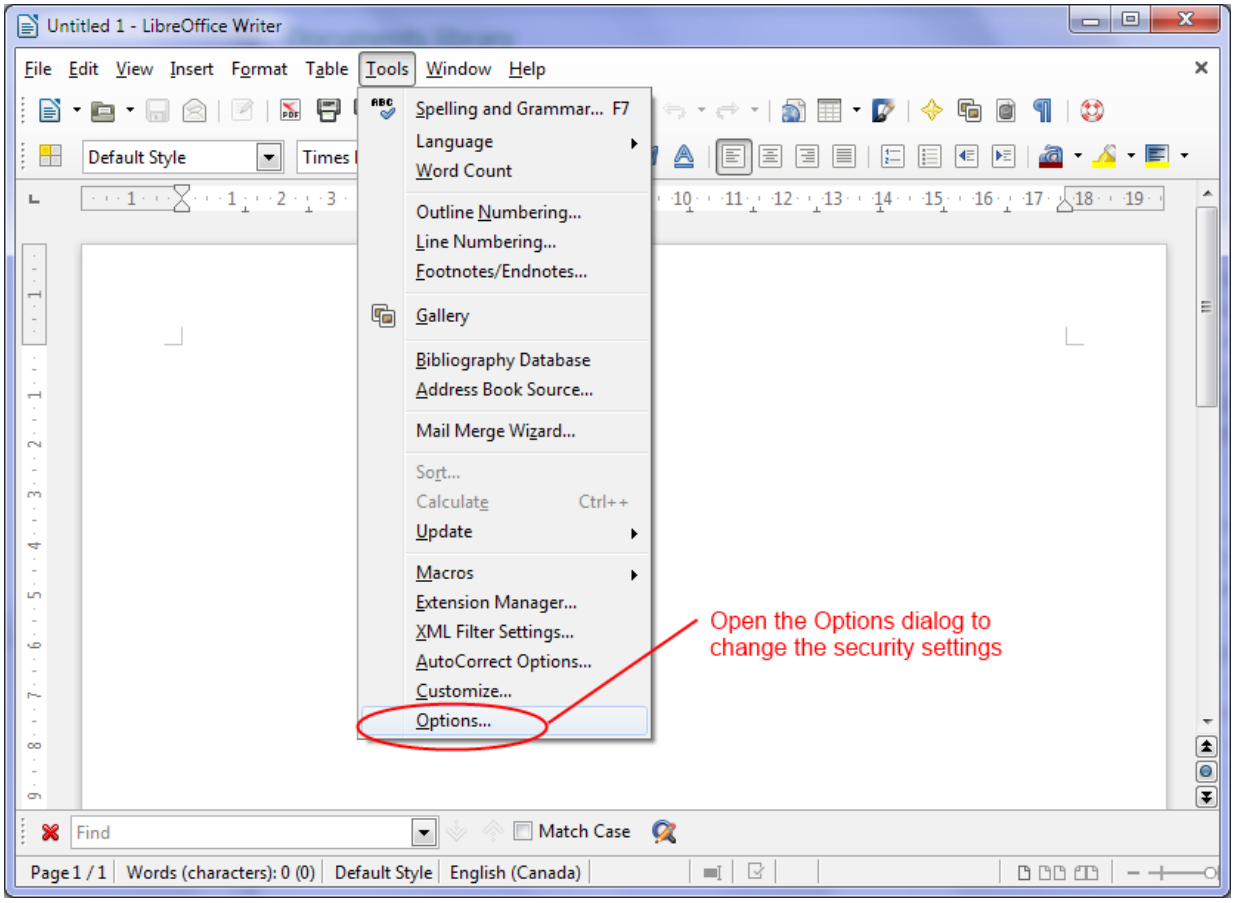
To run the script OpenOffice/LibreOffice must be setup to run scripts from the following directory:

```
${path_to_shorte}/templates/odt
```

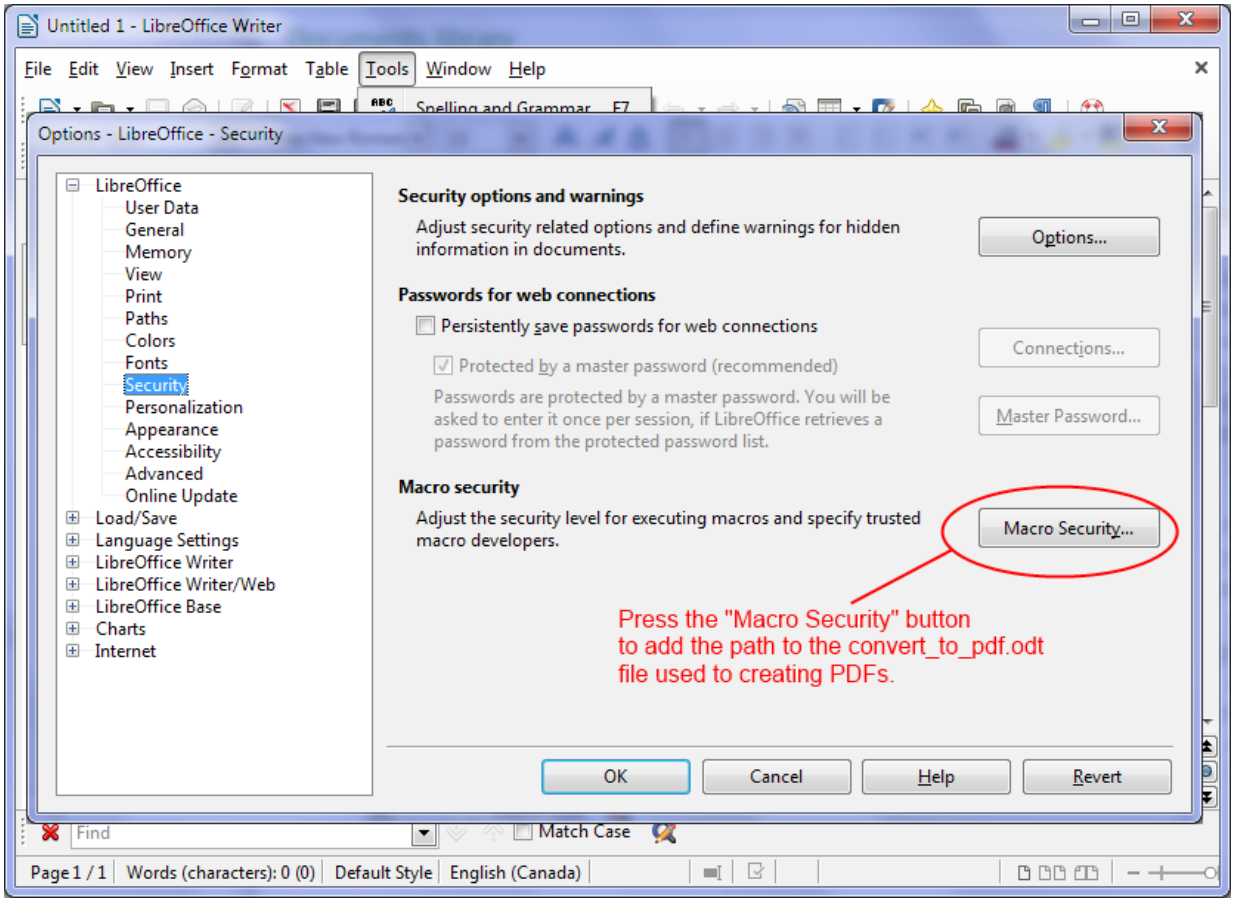
Instructions to do this are shown below.

Change the Macro Permissions for running convert_to_pdf.odt

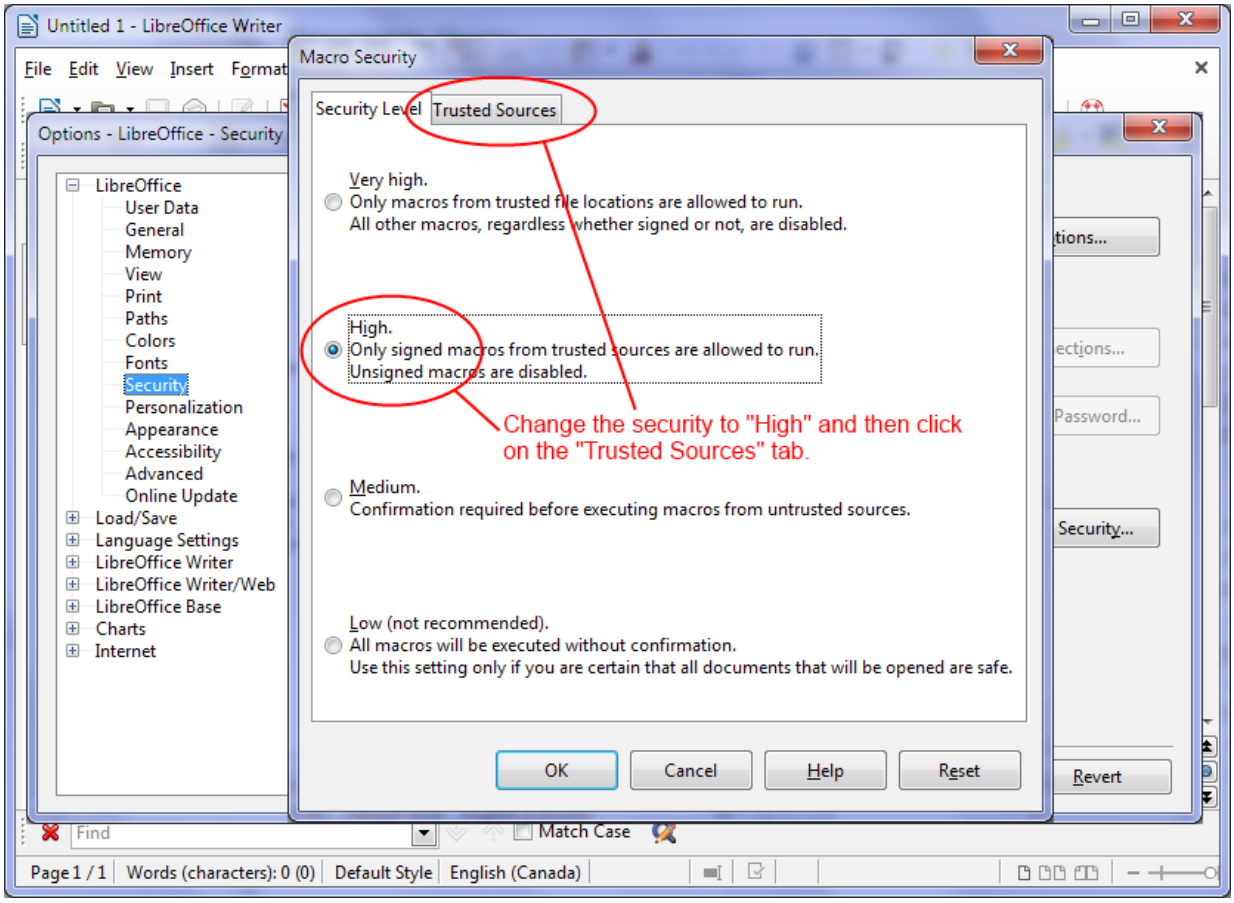
First open OpenOffice/LibreOffice writer and select **options" from the** Tools* menu:



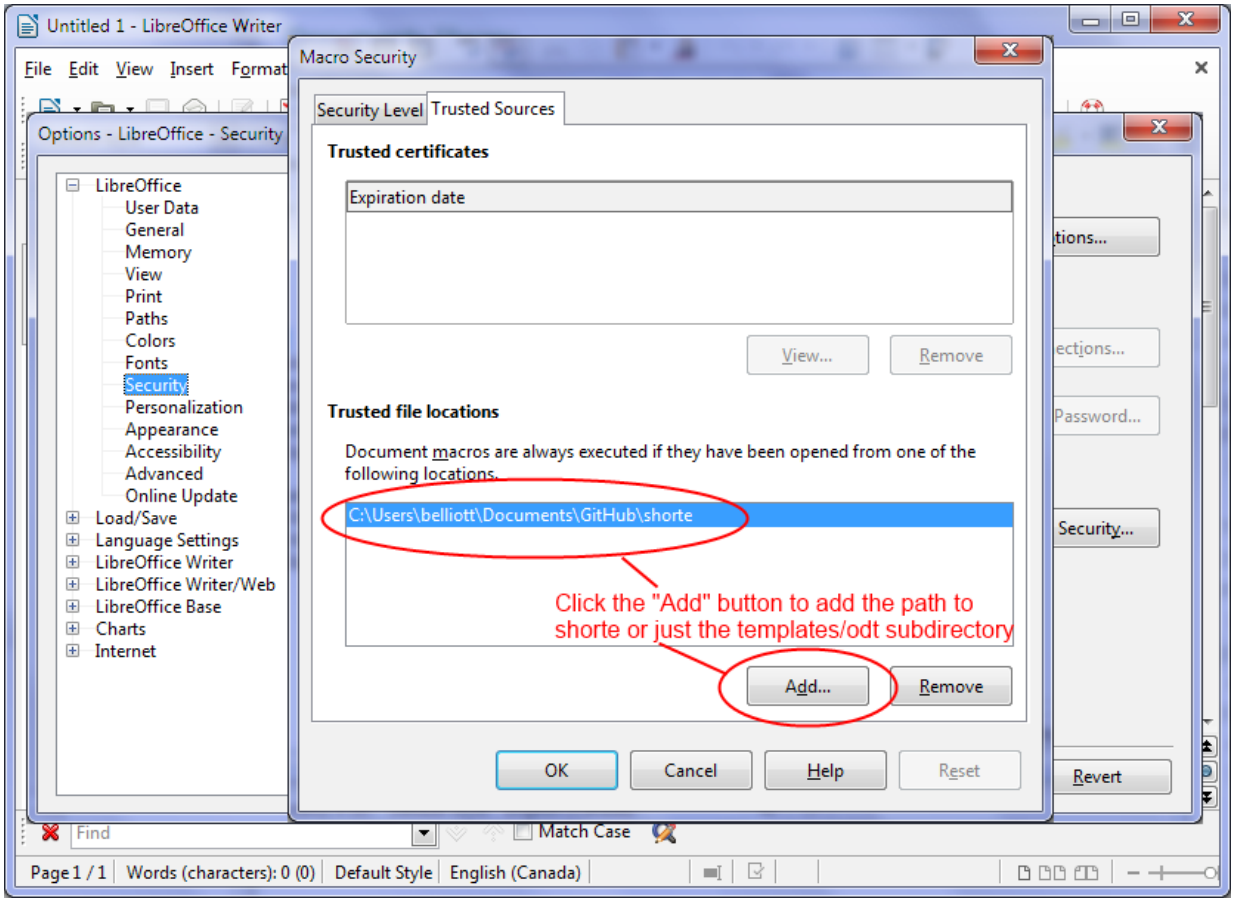
Under LibreOffice/OpenOffice select **Security** to bringup the **Macro Security** settings. Press the **Macro Security** button



Change the macro security to **High** and then click on the **Trusted Sources** tab to add shorte to the path.



In the **Macro Security** dialog under the **Trusted Sources** tab click **Add** to add Shorte to the path.


Untitled 1 - LibreOffice Writer
File Edit View Insert Format
Options - LibreOffice - Security
LibreOffice
User Data
General
Memory
View
Print
Paths
Colors
Fonts
Security
Personalization
Appearance
Accessibility
Advanced
Online Update
Load/Save
Language Settings
LibreOffice Writer
LibreOffice Writer/Web
LibreOffice Base
Charts
Internet
Macro Security
Security Level
Trusted Sources
Trusted certificates
Expiration date
View...
Remove
Trusted file locations
Document macros are always executed if they have been opened from one of the following locations.
C:\Users\belliott\Documents\GitHub\shorte
Click the "Add" button to add the path to shorte or just the templates/odt subdirectory
Add...
Remove
OK
Cancel
Help
Reset
Revert
Find
Match Case
Page 1 / 1
Words (characters): 0 (0)
Default Style
English (Canada)

3: The Command Line

```
$ shorte.py -h
Usage: shorte.py [options]

Options:
  -h, --help            show this help message and exit
  -f FILES, --files=FILES
                        The list of files to generate
  -l FILE_LIST, --list=FILE_LIST
                        The list of files to generate in an input file
  -o OUTPUT_DIR, --output=OUTPUT_DIR
                        The directory where output is generated
  -v VERSION, --version=VERSION
                        The version of the document
  -t THEME, --theme=THEME
                        The output theme
  -n NAME, --name=NAME  The document name or title
  -p PACKAGE, --package=PACKAGE
                        The output package. Supported types are html, odt,
                        word, and pdf
  -b OUTPUT_FORMAT, --output_format=OUTPUT_FORMAT
                        Set the output format in C generated code: bitfields,
                        byte_array, or defines
  -y, --diagnostic_code
                        Generate diagnostic code in generate code
  -c CONFIG, --config=CONFIG
                        The config file to load
  -s SETTINGS, --settings=SETTINGS
                        A list of settings to use that overrides the standard
                        config file
  -x PARSER, --parser=PARSER
                        The parser to use
  -a, --about           About this program
  -m MACROS, --macros=MACROS
                        Macro substitution
  -d DEFINE, --define=DEFINE
                        Macro substitution
  -r REPLACE, --search_and_replace=REPLACE
                        An input search and replace module that is loaded to
                        pre-process input files and replace any references
```

3.1: Some Command Line Examples

Parse a list of source files defined by source_file_list.py and generate Shorte modules describing each of the source files.

```
shorte.py -l source_file_list.py -x cpp -p shorte -r bin/cs4224_snr.py -m
'SKU=CS4343;VERSION=1.1'; -o build-output/srcs
```

- The **-r cs4224_snr.py** file allows a search and replace to be performed on the sources as they are generated.
- The **-m flag** passes a list of macros that can be used within the document for conditional inclusion or conditional text.
- The **-o build-output/srcs** parameter says to generate the files in the build-output/srcs directory.

- The **-x cpp** option switches the parser to the CPP parser instead of the default Shorte parser.
- The **-p shorte** parameter says to generate Shorte code from the C sources.

The source_file_list.py file will look something like:

```
0001 result = '''
0002 modules/high_level/cs4224.c
0003 modules/high_level/cs4224.h
0004
0005 # Only include FC-AN and KR-AN in the duplex guide
0006 if(SKU == 'CS4343'):
0007     result += '''
0008 modules/kran/cs4224_kran.c
0009 '''
```

3.2: Creating a Merge File

A merge file re-assembles a Shorte document into a single file

4: The Document Header

The first part of any Shorte document is the document header. It is structured like HTML but isn't a strict. It is basically anything in the document before the @body tag. An example document header looks like:

```
0001 # The beginning of the document is assumed to be the document
0002 # header. As a convention normally the top level file will
0003 # contain metadata about the document.
0004
0005 # The title of the document
0006 @doctitle The Shorte Language
0007
0008 # The subtitle of the document
0009 @docsubtitle Reference Manual
0010
0011 # A version number (can be overwritten from the command line)
0012 @docversion 1.0
0013
0014 # A number to assign to the document
0015 @docnumber 34567
0016
0017 @docrevisions:
0018 - Revision | Date          | Description
0019 - 1.0.0    | 08 July, 2013 | Initial draft of the Shorte Reference Manual
```

4.1.1: @doctitle

The @doctitle defines the title associated with the document. Only the first instance of this tag is used. If a second instance is encountered it will be ignored.

4.1.2: @docsubtitle

The @docsubtitle defines a subtitle for the document. Only the first instance of this tag is used. If a second instance is encountered it will be ignored.

4.1.3: @docversion

The @docversion tag defines a version number for the document. This can be overridden at the command line.

4.1.4: @docnumber

The @docnumber tag defines a number to associate with the document.

4.1.5: @docrevisions

The @docrevisions tag defines a revision history for the document.

5: The Document Body

5.1: Heading Tags

Headings use the @hN format where **N** currently ranges from 1-5.

5.1.1: @h1

The @h1 tag is the highest level header. It is similar in use to the H1 tag from HTML.

```
0001 # An example header
0002 @h1 This is an example header
0003 This is some text for the example header
```

5.1.2: @h2

The @h2 tag is a hierarchial header directly beneath the @h1 tag. It is similar to the H2 tag from HTML.

```
0001 @h1 This is an example header
0002
0003 # An example second level header
0004 @h2 This is a sub header
0005 This is some text related to the sub @h1 tag.
```

5.1.3: @h3

The @h3 tag is a hierarchial header directly beneath the @h2 tag. It is similar to the H3 tag from HTML.

```
0001 @h1 This is an example header
0002
0003 @h2 This is a sub header
0004
0005 @h3 This is a third level header
0006 Some text related to this header
```

5.1.4: @h4

The @h4 tag is a hierarchial header directly beneath the @h3 tag. It is similar to the H4 tag from HTML.

```
0001 @h1 This is an example header
0002
0003 @h2 This is a sub header
0004
0005 @h3 This is a third level header
0006
0007 @h4 This is a fourth level header
0008 Some example text here
```

5.1.5: @h5

The @h5 tag is a hierarchial header directly beneath the @h4 tag. It is similar to the H5 tag from HTML.

```
0001 @h1 This is an example header
0002
```

```
@h2 This is a sub header

@h3 This is a third level header

@h4 This is a fourth level header

@h5 This is a fifth level header
Some example text here
```

5.1.6: @h

The @h tag can be used to create a header that is un-numbered.

```
@h This is an un-numbered header
Some random text after the header
```

5.1.7: Assigning Wikiwords

Sometimes it is desirable to assign wikiwords to a heading. This allows multi-word headings to be automatically linked but also allows the user to prevent a short heading from being automatically linked

```
@h2: wikiword="MyHeading"
Test

This is some text associated with MyHeading. MyHeading will be expanded
to the word "Test" but Test won't get expanded.
```

5.2: Text Entry Tags

5.2.1: @text

The @text tag creates a text block that is automatically parsed for things like bullets, indentation, or blocks of code.

```
blah blah blah

- An multi-level list
  - A second level in the list
    - A third level in the list

Another paragraph with @{hl, some inlined styling} and

- A second list

{{
and a block of code
}}
```

When rendered we get something that looks like this:

blah blah blah

- An multi-level list
  - A second level in the list
    - A third level in the list

Another paragraph with some inlined styling and

- A second list

```
and a block of code
```

5.2.2: @p

The @p tag is used to create a paragraph. It is similar to the **P** tag in HTML. It does not attempt to parse the text block like the @text tag does in order to extract lists or indented code.

```
0001 @p This is a paragraph in my document
0002 @p This is another paragraph in my document
```

This creates a two paragraphs that looks like:

This is a paragraph in my document

This is another paragraph in my document

5.2.3: @pre

The @pre tag creates a block of unformatted text:

```
0001 @pre
0002 This is a test
0003   this is a test
0004     this is also a test
```

When rendered it will look like:

```
This is a test
 this is a test
 this is also a test
```

5.3: Include Files

Shorte supports include files using either of the following tags:

Include	Description
@include	A normal include - interrupts any conditional text flow
@include_child	A child include - obeys conditional text flow cascading rules

5.3.1: @include

The @include tag is used to include another file. This is to allow breaking a document up into multiple modules. The @include will break any cascading of conditional statements in the document hierarchy. To cascade conditional text in the document hierarcy use the @include_child tag instead.

```
0001 @include "chapters/my_chapter.tpl"
```

Includes also support conditionals in order to support generating multiple documents from the same source. The example below uses a command line conditional called **VARIABLE** to include or exclude the file.

```
0001 @include: if="VARIABLE == 'xyz'"
0002 chapters/my_chapter.tpl
0003 chapters/my_chapter2.tpl
```

5.3.2: @include_child

The @include_child tag is an alternative to the @child tag. It behaves slightly differently in that it does not break the cascase of conditional text but continues the current cascade.

```
0001 @h1 My Title
0002 This section will continue inside the my_chapter.tpl file.
0003
0004 @include_child: if="VARIABLE == 'xyz'"
0005 chapters/my_chapter.tpl
```

5.4: Images and Image Maps

5.4.1: @image

The @image tag is used to include an image. Recommended image formats currently included .jpg or .png.

5.4.2: @imagemap

This tag is used to generate an Image map. It currently only works in the HTML output template. Links are not currently supported.

```
0001 @imagemap: id="one"
0002 - shape  | coords          | Label       | Description
0003 - circle | 50,50,50        | A Circle    | This is a description of my circle
0004 - rect   | 72,144,215,216 | A rectangle | This is a description of my rectangle.
0005
0006 @image: map="one" src="chapters/images/imagemap.png"
```

Will generate the following imagemap:

5.5: Lists and Tables

5.5.1: @ul

The @ul tag is used to create an unordered list similar to the **ul** tag in HTML. Lists can currently be indented 5 levels deep.

```
0001 @ul
0002 - Item 1
0003   - Subitem a
0004     - Sub-subitem y
0005 - Item 2
0006   - Subitem b
```

This generates the following output

- Item 1
  - Subitem a
    - Sub-subitem y
- Item 2
  - Subitem b

5.5.2: @ol

The @ol tag is used to create an unordered list similar to the **ol** tag in HTML. Lists can currently be indented 5 levels deep.

```
0001 @ol
0002 - Item 1
0003   - Subitem a
0004     - Sub-subitem y
0005 - Item 2
0006   - Subitem b
```

This generates the following output

1. Item 1
    1.1. Subitem a
        a. Sub-subitem y
2. Item 2
    2.1. Subitem b

5.5.3: @table

The @table tag is used to create a table. The syntax is shown in the example below.

```
0001 @table
0002 - Header Col 1 | Header Col 2
0003 - Field 1      | Field 2
0004 - Field 3      | Field 4
0005 -& Section
0006 - Field 5      | Field 6
```

This generates the following output:

Header Col 1	Header Col 2
Field 1	Field 2
Field 3	Field 4
Section Header	
Field 5	Field 6

5.5.3.1: Spanning Columns

Spanning columns is accomplished by using one or more || after the column to span. Each additional | spans an extra column.

```
0001 @table
0002 - Column 1 | column 2 | Column 3 | Column 4 | Column 5
0003 - This column spans the whole table
0004 -& So does this header
0005 - || Blah blah || Blah blah
0006 -& This row has no spanning
0007 - one | two | three | four | five
```

This creates a table that looks like this:

Column 1	column 2	Column 3	Column 4	Column 5
This column spans the whole table				
So does this header				
		Blah blah		Blah blah
This row has no spanning				
one	two	three	four	five

5.5.3.2: Headings and Sub-headings

The first row in the table is generally treated as the header. You can mark any row as a header row by staring the line with -*

```
0001 - My Heading 1
0002 -* Also a heading
0003 -& This is a sub-heading
```

This creates a table that looks like:

My Heading 1
Also a heading
This is a sub-heading

5.5.3.3: Table Caption

To create a caption for a table you can do the following:

```
0001 @table: caption="This is a caption for my table"
0002 - My table
0003 - My data | some more data
```

This creates the following table:

My table	
My data	some more data

Caption:

This is a caption for my table

5.5.3.4: Table Title

To add a title to the table you can use the **title** attribute:

```
0001 @table: title="This is my table"
0002 - My table
0003 - My data | some more data
```

This creates the following table:

This is my table	
My table	
My data	some more data

5.6: Notes, TBD and Questions

5.6.1: @note

The @note tag is used to create notes within a section. Here is an example:

```
0001 @note
0002 This is a note here that I want to display
0003
0004 - It has a list
0005   - With some data
0006
0007 And another paragraph.
```

This renders to something like this:

Note:

This is a note here that I want to display

- It has a list
  - With some data

And another paragraph.

5.6.2: @tbd

The @tbd tag is used to highlight sections of a document that are still **To Be Determined**. They are similar in syntax to the @note tag

```
0001 @tbd
0002 This is a block of code that is to be determined. It
0003 works just like a textblock and supports
0004
0005 - lists
0006     - indented data
0007     - another item
0008 - second item in list
0009
0010 Another paragraph
0011
0012     some indented text here
0013     that wraps to a new line
0014
0015 A final paragraph
```

TBD:

This is a block of code that is to be determined. It works just like a textblock and supports

- lists
  - indented data
  - another item
- second item in list

Another paragraph

some indented text here that wraps to a new line

A final paragraph

5.6.3: @question

The @question tag is used to mark a question to the reader or mark anything that might still need to be answered

```
0001 @question
0002 This is a question
0003
0004 with another paragraph. It should eventually be switched
0005 to the same syntax as the @note and @tbd tag.
```

When rendered this looks like:

Question:

> This is a question with another paragraph. It should eventually be switched to the same syntax as the @note and @tbd tag.

5.6.4: @questions

The @questions tag is used to create a Q and A type section. For example,

```
0001 @questions
0002 Q: This is a question with some more info
0003 A: This is the answer to the question with a lot
0004    of detail that wraps across multiple lines and
0005    hopefully it will make the HTML look interesting
0006    but I'm not sure we'll just have to see what
0007    happens when it's rendered
0008
0009 Q: This is another question with some more information
0010 A: This is the answer to that question
```

Will render to:

> **Q:** This is a question with some more info

> **A:** This is the answer to the question with a lot of detail that wraps across multiple lines and hopefully it will make the HTML look interesting but I'm not sure we'll just have to see what happens when it's rendered

> **Q:** This is another question with some more information
> **A:** This is the answer to that question

5.7: Structures and Functions

5.7.1: @struct

The @struct tag defines a C style structure. It also supports generating a picture showing the layout of the structure. The **title** attribute should currently be a unique name since it is used to map any generated image to the structure itself as well as generate C code from the structure definition.

For example:

```
0001 @struct: title="struct1" caption="blah blah"
diagram="show:yes,align:128,bitorder:decrement"
0002 - Field | Name          | Description
0003 - 8x8   | serial_number | The serial number of the device
0004                           with some more description
0005 - 8x12  | part_number   | The part number of the device
0006 - 4     | some_number   | Some random 4 byte number
```

Will generate:

This is a caption. It is currently in the wrong place

struct1

Type	Field Name	Description
8x8	serial_number	The serial number of the device with some more description
8x12	part_number	The part number of the device
4	some_number	Some random 4 byte number

Another example:

```
0001 @struct: title="struct2" caption="blah blah"
0002 - Field | Name          | Description
0003 - 8x8   | serial_number | The serial number of the device
0004                           with some more description
```

```
0005 - 8x12   | part_number    | The part number of the device
0006 - 4      | some_number    | Some random 4 byte number
```

Will generate a structure without a picture:

This is a caption. It is currently in the wrong place

struct2		
Type	**Field Name**	**Description**
8x8	serial_number	The serial number of the device with some more description
8x12	part_number	The part number of the device
4	some_number	Some random 4 byte number

The bit order can also be reversed and the alignment can be changed:

```
0001 @struct: title="struct3" caption="This is a caption. It is currently in the wrong place"
diagram="show:yes,align:64,bitorder:increment"
0002 - Field | Name           | Description
0003 - 8x8   | serial_number  | The serial number of the device
0004                            with some more description
0005 - 8x12  | part_number    | The part number of the device
0006 - 4     | some_number    | Some random 4 byte number
```

This is a caption. It is currently in the wrong place

struct3		
Type	**Field Name**	**Description**
8x8	serial_number	The serial number of the device with some more description
8x12	part_number	The part number of the device
4	some_number	Some random 4 byte number

5.7.2: @vector

The @vector is similar to the @struct tag and creates a bitfield type containing multiple fields. Field sizes are generally outlined in bit ranges instead of bytes in the @struct tag.

The following structure defines a 128 bit long bitfield with the bits shown in little endian order on a 64 bit boundary.

```
0001 @vector: title="vector1" caption="blah blah" diagram="show:yes,align:64,bitorder:increment"
0002 - Field   | Name        | Description
0003 - 0-8     | Blah        | Blah blah
0004 - 10      | *Reserved*  | Reserved for future use
0005 - 12      | My field    | da da da
0006
0007 - 32-63   | TBD         | Something here
0008 - 64-127  | field2      | This is a description
```

This renders to the following:

vector1

Type	Field Name	Description
0-8	Blah	Blah blah
9	Reserved	Automatically generated
11	Reserved	Automatically generated
12	My field	da da da
13 - 31	Reserved	Automatically generated
32-63	TBD	Something here
64-127	field2	This is a description

This is an example of the Ethernet Header shown in little endian format:

```
0001 @vector: title="Ethernet Header" caption="" diagram="show:yes,align:32,bitorder:decrement"
0002 - Field    | Name          | Description
0003 - 0-47     | Dest Addr     | The destination MAC address
0004 - 48-95    | Source Addr   | The source MAC address
0005 - 96-111   | Ethernet Type | The ethernet type
0006 - 112-159  | Data          | Variable length data field
```

Which renders to:

Ethernet Header		
Type	**Field Name**	**Description**
0-47	Dest Addr	The destination MAC address
48-95	Source Addr	The source MAC address
96-111	Ethernet Type	The ethernet type
112-159	Data	Variable length data field

5.7.3: @define

The @define is used to document a #define structure in C.

5.7.4: @enum

The @enum tag is used to define an enumeration.

This is a test enum

e_my_test	
Enum	**Description**
LEEDS_VLT_SUPPLY_1V_TX	1V supply TX
LEEDS_VLT_SUPPLY_1V_RX	1V supply RX
LEEDS_VLT_SUPPLY_1V_CRE	1V supply digital core
LEEDS_VLT_SUPPLY_1V_DIG_RX	1V supply digital RX
LEEDS_VLT_SUPPLY_1p8V_RX	1.8V supply RX
LEEDS_VLT_SUPPLY_1p8V_TX	1.8V supply TX
LEEDS_VLT_SUPPLY_2p5V	2.5V supply
LEEDS_VLT_SUPPLY_TP_P	Test point P

LEEDS_VLT_SUPPLY_TP_N	Test point N

5.7.5: @prototype

The @prototype is used to describe a function prototype. This might be used when architecting code or it can also be extracted from existing source code (Currently only C sources can be parsed).

```
@prototype: language="c"
- function: my_function
- description:
    This is a description of my function with some more text
    and blah blah blah. I'm sure if I put enough text here then
    it will likely wrap but I'm not absolutely sure. We'll see
    what it looks like when it is actually formatted. For kicks
    we'll link to the EPT acronym
- prototype:
    int my_function(int val1 [], int val2 [][5]);
- returns:
    TRUE on success, FALSE on failure
- params:
    -- val1 | I |
            1 = blah blah
                and more blah blah
                plus blah

            2 = blah blah blah

            0 = turn beacon on
    -- val2 | I |
            *1* = blah blah

            *2* = blah blah blah
- example:
    rc = my_function(val);

    if(rc != 0)
    {
        printf("Uh oh, something bad happened!n");
    }

- pseudocode:

    // Blah blah blah
    _call_sub_function()

    if(blah)
    {
        // Do something else
        _call_sub_function2()
    }

- see also:
    THis is a test
```

When rendered it will create output similar to the following with it's own header automatically added for wiki linking. This behavior may be controlled by the **prototype_add_header** field in the Shorte config file.

5.7.5.1: my_function

Function: my_function

This is a description of my function with some more text and blah blah blah. I'm sure if I put enough text here then it will likely wrap but I'm not absolutely sure. We'll see what it looks like when it is actually formatted. For kicks we'll link to the EPT acronym

Prototype:

```
int my_function(int val1 [], int val2 [][5]);
```

Parameters:

val1	[I]	1 = blah blah and more blah blah plus blah 2 = blah blah blah 0 = turn beacon on
val2	[I]	**1** = blah blah **2** = blah blah blah

Returns:

TRUE on success, FALSE on failure

Example:

The following example demonstrates the usage of this method:

```
0001 rc = my_function(val);
0002
0003 if(rc != 0)
0004 {
0005     printf("Uh oh, something bad happened!n");
0006 }
```

Pseudocode:

The following pseudocode describes the implementation of this method:

```
0001 // Blah blah blah
0002 _call_sub_function()
0003
0004 if(blah)
0005 {
0006     // Do something else
0007     _call_sub_function2()
0008 }
```

See Also:

THis is a test

5.7.6: @functionsummary

The @functionsummary tag creates a summary table of all prototypes cross referenced or defined within the document. In this document we've defined a single prototype my_function which should show up automatically when this tag is inserted into the document. Additionally if source code is included any prototypes would automatically get picked up in this table.

```
0001 @functionsummary
```

Generates:

user_guide

```
int my_function(int val1 [], int val2 [][5]);
```

This is a description of my function with some more text and blah blah blah. I'm sure if I put enough text here then it will likely wrap but I'm not absolutely sure. We'll see what it looks like when it is actually formatted. For kicks we'll link to the EPT acronym

5.7.7: @typesummary

The @typesummary tag creates a summary of all structures or enumerations defined within the document or in any parsed source code.

```
0001 @typesummary
```

Generates:

user_guide

struct struct1

This is a caption. It is currently in the wrong place

struct struct2

This is a caption. It is currently in the wrong place

struct struct3

This is a caption. It is currently in the wrong place

struct vector1

struct Ethernet Header

enum e_my_test

This is a test enum

5.8: Source Code Tags

Shorte was built with technical documentation in mind so it supports including a variety of source code snippets. These are described in the following section.

5.8.1: Executing Snippets

In many cases the code within these tags can be executed and the results captured within the document itself. This is useful for validating example code. Execution is done bu adding the following attribute:

exec="1"

to the tag. Remote execution is also possible if SSH keys are setup by adding the machine="xxx" and port="xxx" parameters.

5.8.2: @c

The @c tag is used to embed C code directly into the document and highlight it appropriately. For example, the following block of code inlines a C snippet. The code can also be run locally using g++ by passing the exec="1" attribute. See Executing Snippets for more information on setting up Shorte to execute code snippets.

```
@c: exec="0"
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    printf("hello world!n");
    return EXIT_SUCCESS;
}
```

This renders the following output:

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    printf("hello world!n");
    return EXIT_SUCCESS;
}
```

5.8.3: @python

This tag is used to embed Python code directly into the document and highlight it appropriately. If the code is a complete snippet it can also be executed on the local machine and the results returned. See Executing Snippets for more information on setting up Shorte to execute code snippets.

```
@python: exec="1"
print "Hello world!"
```

This will execute the code on the local machine and return the output:

```
0001 print "Hello world!"
```

Result:

```
Hello world!
```

5.8.4: @bash

This tag is used to embed bash code directly into the document and highlight it appropriately.

5.8.5: @perl

This tag is used to embed Perl code directly into the document and highlight it appropriately.

5.8.6: @shorte

This tag is used to embed Shorte code directly into the document and highlight it appropriately.

5.8.7: @d

This tag is used to embed D code directly into the document and highlight it appropriately.

5.8.8: @sql

This tag is used to embed SQL code directly into the document and highlight it appropriately.

5.8.9: @java

This tag is used to embed Java code directly into the document and highlight it appropriately.

5.8.10: @tcl

This tag is used to embed TCL code directly into the document and highlight it appropriately.

5.8.11: @vera

This tag is used to embed Vera code directly into the document and highlight it appropriately.

5.8.12: @code

If the language is not supported by Shorte the @code tag can be used to at least mark it as a block of code even if it can't properly support syntax highlighting.

```
0001 @code
0002 This is a test of a language that isn't supported by
0003 Shorte.
0001 This is a test of a language that isn't supported by
0002 Shorte.
```

5.8.13: @shell

TBD: add description of this tag.

5.8.14: @xml

This tag can be used to insert a block of XML tag in a document and highlight it appropriately. Note that you currently have to escape the <? sequence to prevent it from being expanded by shorte.

```
0001 @xml
0002 <?xml version="1.0"?>
0003 <methodCall>
0004   <methodName>dev.dev_reg_read</methodName>
0005   <params>
0006     <param>
0007       <!-- The die being accessed (in decimal) -->
0008       <value><i4>1</i4></value>
0009     </param>
0010     <param>
0011       <!-- The address of the register being accessed (in decimal) -->
0012       <value><i4>0</i4></value>
0013     </param>
0014   </params>
0015 </methodCall>
0001 <?xml version="1.0"?>
0002 <methodCall>
0003   <methodName>dev.dev_reg_read</methodName>
0004   <params>
0005     <param>
0006       <!-- The die being accessed (in decimal) -->
0007       <value><i4>1</i4></value>
0008     </param>
0009     <param>
0010       <!-- The address of the register being accessed (in decimal) -->
0011       <value><i4>0</i4></value>
0012     </param>
0013   </params>
0014 </methodCall>
```

5.9: Other Tags

The following section describes some of the other more obscure tags that Shorte supports.

5.9.1: @inkscape

This allows including SVG files from Inkscape direction in the document. It requires Inkscape to be installed and the path properly configured. SVG files are automatically converted to .png files for inclusion since SVG files aren't widely supported.

5.9.2: @checklist

The @checklist tag creates a non-interactive checklist

- one
- two
- three
- four

5.9.3: @acronyms

Acronyms	
Acronym	**Definition**
EPT	Egress Parser Table
EPC	Egress Parser CAM

5.9.4: @embed

TBD - Add description of this tag

5.10: Sequence Diagrams

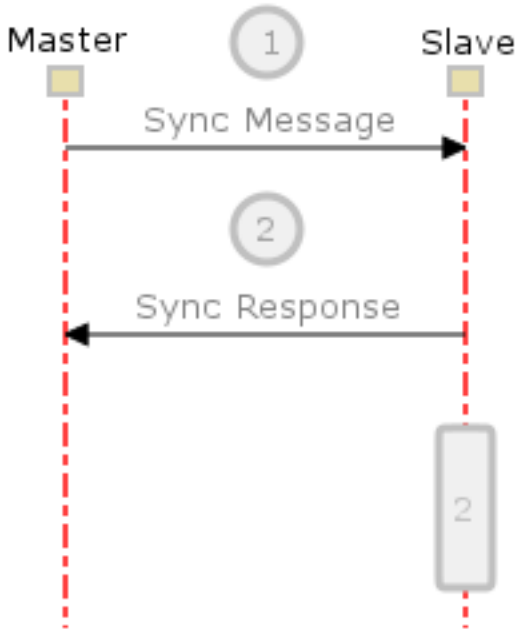
Shorte allows for automatic generation of sequence diagrams using the @sequence tag. The syntax is similar to creating a table. This section describes some examples of creating sequence diagrams.

5.10.1: @sequence

This tag is used to generate a sequence diagram.

```
0001 @sequence: title="Sequence Diagram Title" caption="Sequence diagram caption"
0002 - Type    | Source   | Sink         | Name                   | Description
0003
0004 - message | Master   | Slave        | Sync Message           | A sync message sent from
master to slave.
0005 - message | Slave    | Master       | Sync Response          | A response message from the
slave.
0006 - action  | Slave    |              | Random Event           | A random event on the slave.
```

The above code generates a sequence diagram that looks something like this:



Sequence Diagram Title				
Event	**Source**	**Sink**	**Name**	**Description**
1	Master	Slave	Sync Message	A sync message sent from master to slave.
2	Slave	Master	Sync Response	A response message from the slave.
2	Slave		Random Event	A random event on the slave.

Caption:

Sequence diagram caption

6: Test Cases

Shorte allows the creation of test reports that include a summary section which links to the results of a particular test case defined by the @testcase tag.

6.1.1: @testcasesummary

The @testcasesummary creates a summary of all the test cases defined in the document.

Adding this block of code:

```
0001 @testcasesummary
```

Will expand to:

user_guide

Version	PASSED	0.100000 sec	Status: Version
Register test	PASSED	10.480000 sec	Status: Register test
Register dump test	PASSED	11.760000 sec	Status: Register dump test

6.1.2: @testcase

The @testcase tag is used to define information about a testcase. It looks something like this:

```
0001 @testcase
0002 :name: Version
0003 :desc:
0004 This is a diagnostic method that is used to retrieve the
0005 API version information. It reads the version string from the API
0006 and does a very simple check to validate the sanity of the
0007 version string.
0008
0009 :status: PASSED
0010 :duration: 0.100000 sec
0011
0012 @testcase
0013 :name: Register test
0014 :desc:
0015 This test validates basic register access
0016 by reading the ASIC IDs and verifying the match the
0017 expected value.
0018
0019 :status: PASSED
0020 :duration: 10.480000 sec
```

```
0021
0022
0023 @testcase
0024 :name: Register dump test
0025 :desc:
0026 This test validates the register dump
0027 from the API. Due to restrictions in SWIG it is impossible
0028 to test all the methods so it uses a high-level print
0029 method to display the register dump.
0030
0031 :status: PASSED
0032 :duration: 11.760000 sec
```

When rendered these examples look like:

6.1.2.1: Version

Test Case: Version

Status: PASSED

Desc:

This is a diagnostic method that is used to retrieve the API version information. It reads the version string from the API and does a very simple check to validate the sanity of the version string.

6.1.2.2: Register test

Test Case: Register test

Status: PASSED

Desc:

This test validates basic register access by reading the ASIC IDs and verifying the match the expected value.

6.1.2.3: Register dump test

Test Case: Register dump test	
Status:	PASSED
Desc:	This test validates the register dump from the API. Due to restrictions in SWIG it is impossible to test all the methods so it uses a high-level print method to display the register dump.

For Additional Product and Ordering Information:

www.cortina-systems.com