SP Tools

Development tools for MMBasic running on the Colour Maximite 2.

Written in MMBasic 5.05 by Thomas Hugo Williams in 2020.

You can do what you like with this code subject to the LICENSE,
but if you use it then perhaps you would like to buy me a coffee? Donate

Contents

1. Installation

1. Download the latest release:
   - https://github.com/thwill1000/sptools/releases/download/r1b2/sptools-r1b2.zip
   - or clone/download the latest work in progress: https://github.com/thwill1000/sptools
2. Extract all the files to `/sptools/`
   - if you install in a different directory then you need to edit the value of `SPT_INSTALL_DIR$` in `/src/common/sptools.inc`.

2. `spflow` - Function/subroutine dependency generator

2.1. Features

`spflow` analyses an MMBasic ".bas" file and its ".inc" dependencies and prints a graph, charting control flow within the program; it tries to emulate the behaviour of GNU cflow.

e.g. Given these files:

foo.inc:

```
Sub foo()
  foo()
End Sub
```

example.bas:

```
#Include "foo.inc"

Sub bar()
  foo
End Sub

bar()
foo()
```

Then `spflow` outputs:

```
> RUN "/sptools/spflow.bas", "example.bas"
Generating MMBasic flowgraph from 'example.bas' ...

PASS 1
example.bas
  foo.inc

PASS 2
example.bas
  foo.inc

    1 *GLOBAL* <example.bas:1>
    2   bar() <example.bas:3>
    3     foo() <foo.inc:1>
    4       foo() <foo.inc:1> [recursive, see 3]
    5   foo() <foo.inc:1>
    6     foo() <foo.inc:1> [recursive, see 5]

Time taken = 0.3 s
```

2.2. Usage

```
RUN "/sptools/spflow.bas", [OPTION]... "input file" ["output file"]
```

2.3. Command-line options

- `-A, --all`
  - Produce graphs for all functions/subroutines, even those unreachable from the global scope.
- `-b, --brief`
  - Output the expanded subgraph for each subroutine only once, subsequent calls reference the output line containing the original expansion.
- `-h, --help`
  - Display basic help information, including a description of these options, and then exit.
- `--no-location`
  - Omit filenames and line numbers for each function/subroutine declaration from the output.
- `-v, --version`
  - Display version information and then exit.

2.4. Known issues

1. `spflow` cannot determine that an identifier refers to a function/subroutine (and thus include calls to it in its output) unless it finds a corresponding `Function id` or `Sub id` declaration. This is different to `cflow` where C functions calls can be recognised uniquely by their syntax, whereas MMBasic function calls are syntactically indistinguishable from array variable access.
   - this limitation makes running `spflow` on a ".inc" file of dubious utility.

3. `sptrans` - Transpiler and code-formatter

3.1. Features

- Flattens #Include hierarchies
  - useful for moving code from the CMM2 to other MMBasic flavours that currently do not support #Include.
  - supports multiple levels of #Include and does not require the files to have ".inc" file-extension
    - MMBasic 5.05 on the CMM2 only supports a single level of #Include, i.e. a ".bas" file can #Include one or more ".inc" files and that's it.
- Configurable code reformatting
  - automatic indentation.
  - automatic update of spacing between tokens.
  - remove comments.
  - remove empty-lines.

- Conditional commenting/uncommenting of code sections
  - useful for supporting multiple MMBasic flavours from a single source-tree.
- Configurable token replacement
  - useful for improving performance by inlining constants and shortening identifiers.
  - currently only supports a 1 → 1 mapping.

3.2. Usage

Use the program to transpile itself:

```
RUN "/sptools/sptrans.bas", "/sptools/src/sptrans/main.bas" "out.bas"
```

Or transpile Z-MIM:

```
RUN "/sptools/sptrans.bas", "/zmim/src/zmim_cm2.mbt" "/zmim_new.bas"
```

Or just reformat a file:

```
RUN "/sptools/sptrans.bas", -f --indent=2 --spacing=generous "in.bas" "out.bas"
```

3.3. Command-line options

- `-C, --colour`
  - Use VT100 control codes to syntax highlight the output.
  - This should only be used when accessing the CMM2 via a VT100 compatible terminal, otherwise you see the control codes verbatim.
  - It has no effect when an output file is specified.
- `-e, --empty-lines=off|single`
  - Control inclusion of empty lines in the transpiled output:
    - `off` (or 0) - do not include empty lines.
    - `single` (or 1) - include a single empty line before each Function/Sub, otherwise do not include any empty lines.
  - The default is to include all existing empty lines.
- `-f, --format-only`
  - Only format the output, do not follow #Includes or process directives.
  - Useful if you just want to reformat the whitespace in a single file.
- `-h, --help`
  - Display basic help information, including a description of these options, and then exit.
- `-i, --indent=<number>`
  - Automatically indent by `<number>` spaces per level, may be 0.
  - The default is to use the existing indentation.

- `-n, --no-comments`
  - Do not include comments in the transpiled output.
- `-s, --spacing=minimal|compact|generous`
  - Control output of spacing between tokens, see the description of the `'!spacing` directive for details:
    - `minimal` (or 0)
    - `compact` (or 1)
    - `generous` (or 2)
  - The default is to use the existing spacing.
- `-v, --version`
  - Display version information and then exit.

3.4. Transpiler directives

Directives can be added to the MMBasic code to control the behaviour of the transpiler:

- They all begin `'!` with the leading single-quote meaning that the MMBasic interpreter will ignore them if the untranspiled code is `RUN`.
- They must be the first token on a line.
- By convention if a file just contains directives, comments and #Include I give it the ".mbt" file-extension.
  - This is not enforced and the transpiler does not care what file-extension its input or output file has.

3.4.1. Directives that control formatting of transpiled code

Where present these directives override any formatting specified by the command-line options.

'!comments {on | off}

Controls the inclusion of comments in the transpiled output, e.g.

```
'!comments off
' This comment will not be included in the transpiled output,
' and nor will this one,
'!comments on
' but this one will be.
```

The default setting is `on` unless the `--no-comments` command-line option is used.

'!empty-lines {on | off | single}

Controls the inclusion of empty lines in the transpiled output:

- `off` - do not include empty lines.

- `on` - include existing empty lines.
- `single` - include a single empty line before each Function/Sub, otherwise do not include any empty lines.

e.g. `'!empty-lines single`

The default setting is `on` unless the `--empty-lines` command-line option is used.

'!indent {on | <number>}

Controls the code indentation of the transpiled output:

- `on` - use existing indentation.
- `<number>` - indent by `<number>` spaces per level, use 0 for no indentation.

e.g. `'!indent 2`

The default setting is `on` unless the `--indent` command-line option is used.

'!spacing {on | minimal | compact | generous}

Controls the spacing between tokens in the the transpiled output:

- `on` - use existing spacing.
- `minimal` - use the minimal spacing required for the code to be valid, e.g.

```
If Left$(s$,1)="B"Then
  st=-1:br=de_branch()' comment
EndIf
```

- `compact` - additional spacing, e.g.

```
If Left$(s$,1)="B" Then
  st=-1 : br=de_branch() ' comment
EndIf
```

- `generous` - even more spacing, e.g.

```
If Left$(s$, 1) = "B" Then
  st = -1 : br = de_branch() 'comment
EndIf
```

e.g. `'!spacing compact`

The default setting is `on` unless the `--spacing` command-line option is used.

3.4.2. Directives that control conditional (un)commenting of code

!set <flag>

Sets <flag> for use with the `'!comment_if` and `'!uncomment_if` directives.

e.g. `'!set foo`

!clear <flag>

Clears <flag>.

e.g. `'!clear foo`

!comment_if <flag>

If <flag> is set then the transpiler will comment out all the following lines until the next `'!end_if`, e.g.

```
'!set foo
'!comment_if foo
Print "This line and those that follow it will be commented out,"
Print "including this one,"
'!endif
Print "but not this one."
```

!uncomment_if <flag>

If <flag> is set then the transpiler will remove **one** comment character from all the following lines until the next `'!end_if`, e.g.

```
'!set foo
'!uncomment_if foo
'Print "This line and those that follow it will be uncommented,"
'Print "including this one,"
''Print "but this one will still have a single comment character,"
'!endif
'Print "and this one will not be affected."
```

'!endif

Ends a `'!comment_if` or `'!uncomment_if` block.

e.g. `'!endif`

3.4.3. Directives that control replacement of tokens

'!replace <to> <from>

Tells the transpiler to replace **one** token with another, e.g.

When transpiled this:

```
'!replace apple pear
'!replace banana 30
'!replace "Hello, world!" "Goodbye, world!"
```

```
Dim apple = banana
Print "Hello, world!"
```

Will become this:

```
Dim pear = 30
Print "Goodbye, world!"
```

3.5. Known issues

1. Does not recognise `REM` statements as being comments.
2. Automatic indenting does not handle multiple statement lines correctly.
    - to be honest the auto-indent code is a "hive of scum and villainy" that I need to put under unit-test and rewrite.
3. Automatic spacing is a bit ropey for `compact` and ```spacious`` options.

4. `sptest` - Unit-test framework

4.1. Features

Implements rudimentary xUnit style unit-testing for MMBasic.

See the contents of `/sptools/src/sptest/common/tests` to see it being used in practice.

4.2. Usage

Run all the `tst_*.bas` files in the `tests/` sub-directory of the current working-directory:

```
RUN "/sptools/sptest.bas"
```

Notes:

1. If `tests/` does not exist then this will run all the `tst*.bas` files in the current working-directory.
2. Relies on each `tst_*.bas` file calling `run_tests()` as this is what chains the file execution together.

4.3. Command-line options

The `sptest` program current has no command-line options or arguments.

4.4. Known issues

1. The routines in `unittest.inc` and the other code it depends on in `src\common` all currently require the code being tested to use `Option Base 0` for arrays.

5. FAQ

5.1 General

5.1.1 Will you be supporting the original Colour Maximite / Mono Maximite / Pi-cromite / MMBasic for DOS ?

I do not intend to support the original Colour Maximite or Mono Maximite as the MMBasic 4.5 that these run is missing a number of important features that the code relies on. It is also questionable whether they have sufficient memory to run `spflow` which uses significant in-memory data-structures.

Pi-cromite and MMBasic for DOS ports are in theory possible, especially if their respective MMBasic implementations get some of the CMM2 updates, however I do not currently have the time to work on them, please feel free to contribute your own changes ;-)

5.1.2 What is the Colour Maximite 2 ?

The Colour Maximite 2 is a small self contained "Boot to BASIC" computer inspired by the home computers of the early 80's such as the Tandy TRS-80, Commodore 64 and Apple II.

While the concept of the Colour Maximite 2 is borrowed from the computers of the 80's the technology used is very much up to date. Its CPU is an ARM Cortex-M7 32-bit RISC processor running at 480MHz and it generates a VGA output at resolutions up to 800x600 pixels with up 65,536 colours.

The power of the ARM processor means it is capable of running BASIC at speeds comparable to running native machine-code on an 8-bit home computer with the additional advantage of vastly more memory and superior graphics and audio capabilities.

More information can be found on the official Colour Maximite 2 website at http://geoffg.net/maximite.html

5.1.3 How do I contact the author ?

The author can be contacted via:

- https://github.com as user "thwill1000"
- https://www.thebackshed.com/forum/index.php as user "thwill"

5.2. sptrans

5.2.1 Why didn't you just copy the design of the C preprocessor like FreeBASIC does ?

1. The current design was chosen so that a file annotated with !directives is still a valid MMBasic file for at least one flavour of MMBasic (currently MMBasic 5.05 on the CMM2) that can be RUN without first running the transpiler over it.

2. Because that would be a lot more work.

5.2.2. When is it getting C preprocessor style macro support ?

Not yet ;-)