

Tiny Cobol Introduction

Jonathan Riddell, jr@jridell.org

A brief introduction to Tiny Cobol for those who are new to either Tiny Cobol or Cobol in general.

1. This Document

1.1. Goal

A brief introduction to Tiny Cobol for those who are new to either Tiny Cobol or Cobol in general, taking the reader through looking at Cobol, a first program and a more complicated program using make.

1.2. New Versions of This Document

The latest version of this document can be found on the Tiny Cobol website at <http://tiny-cobol.sourceforge.net>.

1.3. Copyrights

This document is Copyright Jonathan Riddell 2001.

You may copy and distribute it only under the terms of the GNU Free Documentation License, available at <http://www.gnu.org/copyleft/fdl.html>

1.4. Thanks

Thanks to the Tiny Cobol team and especially to Rildo Pragana (<http://www.pragana.net/>) our faithful leader.

Thanks to Richard Bland, my lecturer and teacher of all things Cobol.

Cheers also to the people from Debian for including a working sgmltools program. It's much appreciated.

1.5. Translations

There are no translations of this document yet, but any helpers are welcome.

And yes, this document is in British English. Brackets are like this (), and periods are something girls have every month. Think yourself lucky I didn't put it in Scots (<http://scots.jriddell.org>).

2. Introduction to Cobol and Tiny Cobol

2.1. About Cobol

Cobol is one of the oldest third generation languages. It was developed in 1959 (around the same time as Fortran) to replace processor specific assembly languages (second generation languages). The first ANSI standard was the 68 standard (although it had been commonly used since 1961). Later standards were the 74 standard and the 85 standard. Most Cobol programs used today conform to the 85 standard (and so does Tiny Cobol).

Cobol's main advantages over competing languages (such as C) was the fast input/output speed it could deliver, fixed-point arithmetic for accurate accounting, and English-like syntax for documentation and readability. This made it very useful for

large data-processing jobs such as keeping track of a million bank accounts each night or processing pay slips. The English-like syntax made programs easier to understand, so business programs could be easily modified over their lifetime as business rules changed.

Cobol is significantly different from block structured languages such as Pascal, C and C's descendants. It has no block structure and thus no way of hiding variables; there is more freedom in the program typing; numbers are closer to human arithmetic e.g. fixed point or decimal numbers are generally used instead of floating point; I/O is record orientated, not stream orientated; recursion is not allowed and the language itself is much larger because it doesn't use libraries.

More information on Cobol can be found at the Cobol FAQ at <http://www.cobolreport.com/faqs/cobolfaq.htm>

2.2. About Tiny Cobol

Traditionally the sort of people who use Free software are not the sort of people who use Cobol. This is what The Jargon File has to say on the subject:

COBOL /koh'bol/ n. [COmmon Business-Oriented Language] (Synonymous with evil.) A weak, verbose, and flabby language used by card wallopers to do boring mindless things on dinosaur mainframes. Hackers believe that all COBOL programmers are suits or code grinders, and no self-respecting hacker will ever admit to having learned the language. Its very name is seldom uttered without ritual expressions of disgust or horror. One popular one is Edsger W. Dijkstra's famous observation that "The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence."

Pretty harsh criticism indeed. But to counter this are good reasons why Cobol is relevant today.

- There are about 100 billion lines of Cobol in use today
- A great deal of development is still done in Cobol - there are thought to be 1 million developers world wide and 2 billion lines of Cobol written each year

- There is considerable demand for Cobol programmers to work on legacy systems
- Many university courses teach Cobol
- You might disagree with the author of The Jargon File

So if Cobol is still relevant today that means it will be relevant to a good many Free software users and that means it is both useful and important for there to be a Free Cobol compiler.

Which is where Tiny Cobol comes in. It's creation is headed by Rildo Pragana and was first made for the restrictive environment of DOS. Tiny Cobol is now made for the much more programmer friendly environment of Linux and has advanced a long way.

2.3. The Competition

- Cobol for GCC (<http://cobolforgcc.sourceforge.net/>), is a project to develop a Cobol compiler which will integrate with the GNU Compiler Collection, GCC. It is not yet in a useful state.

3. Installation

3.1. Getting Tiny Cobol

Tiny Cobol only currently works with Linux or FreeBSD on an i386 machine. It has now succesfully been compiled on the win32 platform using the cygwin tools, try looking through the mailing list archives if you are interested in doing this (binaries might also be available). It will most likely not compile on anything else, although you're welcome to try.

You can download Tiny Cobol from it's website at Sourceforge:

<http://tiny-cobol.sourceforge.net> (<http://tiny-cobol.sourceforge.net/>). There are some RPM versions available, but most people will just want to download the GZipped versions. The size of the file is less than half a megabyte.

Copy this file to somewhere like `/usr/local/src` and unzip it (it will create it's own directory). Enter the directory and take a look at the README file and read completely the INSTALL file. The INSTALL file will tell you what libraries you need; if you don't have any of these libraries then install them from your distribution CD or download them from the suggested sites. It will also tell you that you need GCC installed, if this isn't installed you won't get far so do this now, again from your distribution's CD or ftp server or by downloading it from GNU.

3.2. Compiling and Installing

To prepare for compilation go to the Tiny Cobol directory and type `./configure` (there are several options to customise configure, see the file INSTALL for details). Then type `'make'` to compile. You don't need to be root to do this.

Now change directory, `'cd test_suite'` and type `'make tests'`. You'll see a stream of results as the newly compiled compiler goes to work on the test programs. Not all of these will pass, remember this is alpha software, but most should.

Now go to the original directory and type `'make install'`, which will copy a library, the compiler and the preprocessor and a few other files so they are ready for use. You will need to be root to do this. The binaries are put in `/usr/local/bin`, the libraries in `/usr/local/lib` and all other files in `/usr/local/share/htcobol/`, unless specified otherwise with the configure program.

3.3. Help, it didn't work!

If something went wrong and you don't know how to fix it, read through this document again. Make sure you have all the important stuff like GCC and the necessary libraries

installed (you should be able to get these off whatever Linux-based Distribution CD you use). Next read the Tiny Cobol webpage at <http://tiny-cobol.sourceforge.net>.

Still haven't sorted it out? Go and join the tiny-cobol-users mailing list and ask the friendly people there. You can sign up at <http://lists.sourceforge.net/mailman/listinfo/tiny-cobol-users> or by sending an e-mail to tiny-cobol-users-request@lists.sourceforge.net (<mailto:tiny-cobol-users-request@lists.sourceforge.net>) with subscribe in the message's body. The address to post to is tiny-cobol-users@lists.sourceforge.net (<mailto:tiny-cobol-users@lists.sourceforge.net>). You should also look through the archives incase your question has been asked before. They are at <http://lists.sourceforge.net/archives/tiny-cobol-users/>.

There is also a Brazilian language list at <http://listas.cipsga.org.br/cgi-bin/mailman/listinfo/cobol-br> and a Brazilian language webpage at <http://br.tinycobol.org>.

4. First Program

4.1. Editors

Like most programming languages, a Cobol program is just a text file, which you edit in your favourite text editor. Fortunately, to avoid any religious wars, both vim and Emacs work with Cobol.

4.1.1. Emacs

There is no Cobol mode which comes as standard for GNU Emacs. Instead download a copy of `cobol.el` from the Cobol for GCC project. You can download the file directly from their CVS at <http://cvs.sourceforge.net/cgi-bin/cvsweb.cgi/gcc/cobol/cobol.el?cvsroot=CobolForGCC>.

Open up Emacs and compile the Lisp file with M-x byte-compile-file. Move the resulting Cobol.elc file to /usr/share/emacs/20.7/lisp/textmodes/ or the equivalent for your installation. Finally add this code to the .emacs file (or .gnu-emacs in some distributions) in your home directory:

```
(autoload 'cobol-mode "cobol" "COBOL Editing mode" t)
(setq auto-mode-alist
      (append '(("\\.cpp$" . c++-mode)
                ("\\.hpp$" . c++-mode)
                ("\\.lsp$" . lisp-mode)
                ("\\.scm$" . scheme-mode)
                ("\\.pl$" . perl-mode)
                ("\\.cbl$" . cobol-mode)
                ("\\.CBL$" . cobol-mode)
                ("\\.COB$" . cobol-mode)
                ("\\.cob$" . cobol-mode)
                ("\\.CPY$" . cobol-mode)
                ("\\.cpy$" . cobol-mode)
                ) auto-mode-alist))

;; Auto font lock mode
(defvar font-lock-auto-mode-list
  (list 'c-mode 'c++-mode 'c++-c-mode 'emacs-lisp-mode 'lisp-mode
        'perl-mode 'scheme-mode 'scribe-mode 'shell-script-mode 'cobol-mode
        'dired-mode)
  "List of modes to always start in font-lock-mode")

(defvar font-lock-mode-keyword-alist
  '((c++-c-mode . c-font-lock-keywords)
    (perl-mode . perl-font-lock-keywords)
    (cobol-mode . cobol-font-lock-keywords)
    (dired-mode . dired-font-lock-keywords))
  "Associations between modes and keywords")

(add-hook 'cobol-mode-hook
```

```
(lambda ()  
  (set (make-local-variable 'dabbrev-case-fold-  
search) t)  
  (set (make-local-variable 'dabbrev-case-  
replace) t))
```

Open up a Cobol file in Emacs and type 'M-x cobol-mode' and 'M-x font-lock-mode' to get pretty colours. Thanks to the Cobol for GCC team for doing this Emacs mode.

4.1.2. VIM

For a guide to doing Cobol in VIM go to <http://dimensional.com/~sitaram/cobol/>.

4.1.3. Others

There are plenty of other editors out there. An interesting one is THE, an editor which mimics mainframe editors such as Xedit. It's available from <http://www.lightlink.com/hessling/>

4.2. First Program

Here is a Hello World program in Cobol:

```
* Hello World Program  
* GPL Copyleft Jonathan Riddell 2001  
  IDENTIFICATION DIVISION.  
  PROGRAM-ID.    hello.  
  ENVIRONMENT DIVISION.  
  DATA DIVISION.  
  
  PROCEDURE DIVISION.  
    DISPLAY "hello ," WITH NO ADVANCING
```



```
DISPLAY "world!"  
STOP RUN.
```

The structure of the program is quite simple, four sections two of which have content. The PROGRAM-ID is a simple name for the program, you can optionally have other information in the IDENTIFICATION DIVISION. The ENVIRONMENT DIVISION can (optionally now) contain information about your configuration, such as the compiler you're using. The DATA DIVISION contains any variable declarations we might need. Finally the PROCEDURE DIVISION has the actual program instructions.

You might notice that there is a lot of capitals in the code. Using capitals for Cobol's reserved words keeps the program more backwards compatible with other compilers and ensures that the reserved words stand out against literals and variables. With Tiny Cobol you are perfectly free not to use capitals which will stop your Caps Lock wearing out so fast.

The other thing to notice is the spacing. Back in the days of punched cards Cobol kept a strict control on where parts of the code should be:

```
column 1-6    : line number.  
column 7      : indicator area,  
                asterisk for comment lines,  
                minus for continuation lines,  
                slash for page skip on compilation listing  
                otherwise space.  
column 8-11   : Margin A, here starts division, section,  
                paragraph identifiers and some level numbers.  
column 12-72  : Margin B, for everything that does not belong  
                in Margin A.  
column 73-80  : program identification area.
```

Tiny Cobol doesn't force you to keep to this regime but it might keep your programs more tidy if you do, it also stops any other compiler fussing about spacing and it keeps the Cobol mode in Emacs happy. Tiny Cobol does, however, insist on having the asterisk that denotes a comment in the first or second column.

Now on with compiling the program. Save the code above as `hello.cob` and type the following command:

```
htcobol hello
```

Chances are that `htcobol` will warn that `$COBDIR` isn't set, as long as you installed into the default directory this shouldn't be a problem but if you changed the directory or if you get annoyed by extraneous warning messages then add `'export COBDIR=/usr/local/share/htcobol'` or similar to your `.bashrc` or `.profile` file. It will also say that it is 'Processing compiler parameters', this is a good thing.

`htcobol` will produce the files `hello.lis` and `hello.s`. `hello.lis` is a conversion of the Cobol code into something more compiler friendly. `hello.s` is in assembly code, the human readable equivalent of machine code.

Update: Tiny-Cobol no longer compiles to assembly but direct to an executable programme. The next two steps can therefore be left out.

If your shell can't find a program called `htcobol` firstly check that your `$PATH` variable points to `/usr/local/bin/htcobol` or wherever you installed it to (type `echo $PATH`). If it is there then double check that it configured, compiled and installed correctly.

If you mistyped the code above you'll have noticed that Tiny Cobol isn't yet very good at reporting errors. In fact all you'll get is a Segmentation Fault. Double check that you spelled everything correctly, didn't miss out any punctuation and the asterisks of the comments are in the first column.

The next step is to compile the assembly file `hello.s` into machine code. The command for this is:

```
as -o hello.o hello.s
```

`as` is part of the GNU Compiler Collection, GCC, and if you don't have it installed you have a very funny distribution indeed (given that you've already compiled Tiny Cobol today). Check your distribution documentation for how to install it.

hello.o is an object file, but not a runnable executable because it hasn't been linked with the necessary libraries. This is the final command:

```
gcc -o hello hello.o -lhtcobol -lm
```

If you've got this far you're unlikely to have had any problems with this, but if gcc hasn't found the libraries libhtcobol or libm then you might have to update the libraries database with the ldconfig command.

Finally there is a working program which can be run with:

```
./hello
```

And you should see the output:

```
Hello, world!
```

Which was quite predictable. If you look at the program code it should be obvious what DISPLAY does and what WITH NO ADVANCING does (prints without a line break).

5. Further Programming

5.1. Another Program

Here is a slightly more complicated program, enter it into a file and save it as addition.cob.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.      addition.  
ENVIRONMENT DIVISION.
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77  myvar PIC 999 .
77  a  PIC 99V999 VALUE 1.777 .
77  b  PIC 99V99  VALUE ZERO .
77  c  PIC 9V9    VALUE 5.5 .
77  d  PIC 9v9    VALUE 5.5 .

PROCEDURE DIVISION.
    DISPLAY "Please enter a number"
    ACCEPT myvar
    DISPLAY "That number was ", myvar
    ADD a TO b ROUNDED
    DISPLAY "b is ", b
    ADD c TO d
    DISPLAY "d is ", d
    STOP RUN.
```

You can compile this with the same routine as you did with hello.cob, if you encounter any Segmentation Faults double check your program before complaining.

If you are new to Cobol, there is a bit to take in here. Firstly we have some variables declared in the **WORKING-STORAGE SECTION** of the **DATA DIVISION**. Cobol requires that all variables be declared in this way. The other **DATA DIVISION** sections deal with files and linkage.

The variable declarations here all start off with '77'. This means the variable is at level 77: it has no internal structure and no defined relationship to any other variable, it isn't part of a record (or what C calls structs). The name of the first variable is myvar; variable names can be in either upper or lower case, but with Tiny Cobol you must be consistent throughout your program.

The PIC clause specifies the type of variable. Using 9s means it is a decimal variable and using 3 9s makes a variable which can take the numbers 0 to 999. You can also have an 'S' before the 9s to make a signed variable. A 'V' in the PIC declaration means a variable with an assumed decimal point at the V position. For large variables you can write 9(15), which is a variable that has 15 places. Using As instead of 9s makes an

alphabetic variable, and using Xs makes an alphanumeric variable. Finally an initial VALUE can be assigned to the variable.

Inexperienced Cobol programmers may be wondering how an end of a statement is marked. The ends of blocks are marked with a full stop but statements aren't marked at all, instead the Cobol compiler recognises when a new statement begins by noticing one of Cobol's large number of verb words. The end of lines is not important in Cobol. Another thing to notice in the above code is the added commas. Commas are completely optional in Cobol, you can put them in anywhere you like or nowhere at all and it makes no difference. Use them sparingly and they will make your program neater.

When running the program it should ask you for a number, then print that number along with the two other calculations it just performed. The number given can't be longer than the 3 spaces allocated to it, else only the final three characters are taken. If fewer than three characters are given then it will print out with leading zeros. Non numeric characters will output some debugging code.

The adding of variables a and b should work fine, although you'll notice that the answer is rounded as we asked for. Adding c and d won't do anything since the answer is too big to fit in the variable d. Try changing c to, say, 1.5 and everything will work fine.

5.2. Make

Update: Tiny-cobol now compiles directly to an executable programme, rather than assembly code. Much of this is therefore obsolete (but the concepts should be useful).

By now you might be getting a little bored of the htcobol, as, gcc routine. Well there is a better way, make. Make takes in a file called a Makefile and uses the instructions in it to work out what need compiled in one easy step. It also checks that a file needs compiling before it does anything, if none of the program's dependancies have changed since the last compile, nothing has to be done.

Makefile rules are in the form:

```
target:dependencies
```

command

The target (which must begin in the first column) is the file you're trying to get. The dependencies are the files that it depends on. The command (which must be on a second line and preceded by a tab) is the command you would normally type at the shell to make your file. Here's a Makefile for addition:

```
#simple makefile for a one file cobol program

addition:addition.o
gcc -o addition addition.o -lhtcobol -lm

addition.o:addition.s
as -o addition addition.s

addition.s:addition.cob
htcobol addition
```

Save this as Makefile in the same directory as addition.cob and type make. It should compile, assemble and link for you.

By default make will compile the first rule in the file. This is to create the addition binary, but this depends on addition.o so that must be made first. addition.o depends on addition.s so this is the first command that make runs. Now it can create addition.o and finally addition.o is up to date so it can create addition. If you try typing make again it won't do anything because none of the files have changed so there's no need (make looks at the file's timestamp to determine this). Change addition.cob slightly and run make and the whole process runs again.

5.3. Advanced Make

But the previous Makefile has a problem: what if you want to adapt it to work with another program? There are quite a few addition's to change and with a more

complicated program that would take ages. Fortunately make has a good many tricks up it's sleeve. Here is a more advanced makefile:

```
#makefile for a one file cobol program

PROGRAM := addition
LIBS := -lhtcobol -lm
OBJECTS := $(PROGRAM).o

$(PROGRAM):$(OBJECTS)
gcc -o $(PROGRAM) $(OBJECTS) $(LIBS)

#rule for any file ending in .o
#depends on same prefix with .s
# $@ is the target
%.o:%.s
as -o $@ $<

%.s:%.cob
htcobol $(PROGRAM)
```

The Makefile starts off with some variables. PROGRAM is the name of the program we are making. You can see it used in the first rule (preceded by a dollar sign and delimited with brackets). LIBS is just the libraries that we need to compile the program. OBJECTS is any object files needed, in this case it's just the one and that has the same name as the program with a .o on the end, which is just how it's declared.

The second rule matches any target ending in .o (%.o) and it requires the same file with a .s suffix (%.s as a dependency). The automatic variable \$@ matches the name of the target and \$< matches the first dependancy. By now you should be also to work your way through the final target.

If you want to adapt this Makefile to another Cobol program, all you have to do is change the value of the \$PROGRAM variable.

6. The End

That's all for now. There should possibly be another chapter here with a program that uses more than one file and the preprocessor but that's beyond my ken for today. Maybe also some links to Cobol resources.

Jonathan Riddell, jr@jridell.org

