Building a *Free Pascal* Cross Compiler for the Sinclair QL

Norman Dunbar

5th April 2021

1 Introduction

I already had the *Free Pascal Compiler* (FPC) installed on my Linux laptop, however, I don't use it that often – I'm not very good at remembering how to write Pascal code – so I decided that for this experiment in getting the bare bones of the Sinclair QL version of the *Free Pascal Compiler* I would set up a brand spanking new VM running the same Linux version as my laptop. This is Linux Mint 20.1, the 64 bit version.

The process of setting up the VM will not be discussed here, normally I would have used *VirtualBox* but as I had just done my 10 yearly wipe and refresh of the laptop, I hadn't yet installed it and I wanted to try out the `libVirt` system using QEMU and the kernel's KVM *stuff*[1].

The upshot of all this is, I had made things a little more complicated than they needed to be, but that's how I wanted it. If I messed things up really badly, I could easily wipe the VM and start again with little or no problem, whereas if I tried to do everything on the laptop, and messed up, I might need to be sorting things out for a while to get back to where I was. Experiments don't always work out fine!

1.1 Code Conventions

1.1.1 Line Numbers and Continuations

In the listings which follow, lines will be numbered. Hopefully you will see that the numbers are outside of the code block in the document. You don't have to type in the numbers!

Also, keep an eye out for long lines which have wrapped around. They will not have line numbers on the continuation lines and those will be indented to show that they are continued from above. Like this:

```
1 echo "This is a short line."
2 echo "This is a long line of what should be code and it is
     all on a single line, but has wrapped around."
3 echo "This is another short line."
```

Lines of output from various commands will be shows thus:

```
Hello World!
```

[1]That's a technical term.

There will not be any line numbers, unless absolutely necessary, and long lines will be split and indented as before.

1.1.2 Privileges

Unless otherwise noted, all commands will be executed as my local user, *norman*, and not as *root*. I may need to obtain root privileges from time to time, and to do that, I'll prefix the appropriate commands with `sudo` rather than logging in as the root user. When using `sudo`, you are initially prompted for a password and that password will be cached for a short period of time. During this time, any other `sudo` commands will not prompt.

1.1.3 Free Pascal Compiler Versions

As I worked on this document, there were many changes made to the sources of the *Free Pascal Compiler*, this included the occasional version number change. I started at version 3.2.0 and at some point, it had risen to 3.3.1. To avoid confusion, I have avoided hard coding the version numbers into paths where it is included. Those file and directory names will be noted as "n.n.n" regardless of the version in question.

1.2 Building the Development VM

The plan of action, after installing the VM and Linux, is to:

- Install any development software required.
- Install the *Free Pascal Compiler* for Linux 64 bit.
- Install the FPC Source Code.
- Install and build the required assembler, *vasm*.
- Install and build the required linker, *vlink*.
- Build and install the QL version of FPC.

That would, hopefully, give me a working cross compiler so that I could try and write QL programs, compile them under Linux, copy them to my QL then test them. Easy stuff, no? My own QL system is Marcel's excellent *QPC* so I'm able to run the cross compiled programs without any problems. However, if you are using an actual QL, then you may need to compile the Pascal programs using the option to add an XTcc trailer record, and use my little `XTcc_bin` utility to convert the files to executable on the QL.

As I worked through the experiment, it became obvious that some work would be needed on the existing, brief, system Unit for the QL, so I will also need to be able to edit and recompile the Sinclair QL Runtime Library (RTL) for FPC.

2 Development Software

Once the VM is created and Linux Mint installed, we need to install the software tools and packages which will allow us the ability to build the cross compiler.

2.1 Installing the Software

The following software is the minimum required to build the cross compiler and build Pascal programs for the QL:

- *Subversion*; the version control system used by the FPC developers. We need this to be able to download the source code for the compiler, and to keep the software up to date. We could avoid this step and just download a zip file, if necessary. *Subversion* comes in handy when changing the RTL as it allows a patch kit to be created to update the source code for others to use.
- *Build-essential*; a package on Mint that installs the various compiler tools and libraries necessary to compile stuff[1].
- The *ssh server*; to allow me to connect a terminal session from my laptop to the VM to do development work, and to copy down compiled programs.
- *Git*; which is not strictly necessary, but I use it myself and I wanted it installed, just in case. Feel free to leave it off if you don't use it.

```
# Update the Mint software database.
sudo apt update
[sudo] password for norman:
... lots of output here.

# Patch the Mint system.
sudo apt upgrade
... lots of output here.

# Install development tools and packages.
sudo apt install build-essential subversion git
... lots more output here,
```

[1]Another technical term!

As I mentioned, I need to install and enable the ssh server, so that it runs now, and at startup:

```
# Install SSH server.
sudo apt install openssh-server
... More output here.

# Start ssh server now.
sudo systemctl start ssh

# Start ssh server at system startup.
sudo systemctl enable ssh
```

I need the current IP address of the VM to allow me to use `scp` to copy the file to my laptop from the VM, and upload it to QPC:

```
# What's my VM's IP address?
hostname -I
```

```
192.168.2.225
```

The above list sets up the VM ready to compile code. In addition to the above, I also need:

- To install FPC for the Linux host; the compiler is used to compile a Sinclair QL cross compiler version of itself, so it is required to be present.
- To build the QL version, we also need to install the source code for the compiler.

3 Installing the Host Compiler

The "host" is my Linux VM. It requires that a version of FPC be installed to compile and run programs for Linux 64 bit systems. Once installed it will be used to build the QL specific version of FPC that will run on the host, but create executables for the "target", the QL. Cross compiling can get a little bit confusing at times.

3.1 Download FPC

Point your favourite browser at https://www.freepascal.org/download.html and on that page, scroll down to where you find your particular system. Mine is X86-64 Linux, so I clicked that link.

On the next page, select a mirror – I used Source Forge – and click the link. If you click on one of the other mirrors, you get different options. Source Forge is *amusing* in that you have to choose your required version, again. There are quite a few and some are pre-build cross compilers, so be careful in what you choose. I required `fpc-n.n.n-x86_64-linux.tar` and that just happens to be the version that Source Forge decided was ideal for me and selected it as the "Download Latest Version" option, right at the top in a big green box.

Let the download run, it's about 85 Mb, and when completed, make a note of where you saved it, then in a file explorer session, navigate to the downloaded file, and extract it. Once extracted, it's a simple case of making sure that the file `install.sh` is executable, and running it. The extraction process is as follows:

```
# Switch to downloads and extract the compiler.
cd ~/Downloads
tar -xvf fpc-n.n.n-x86-64-linux.tar
... Some filenames whizz by here
```

3.2 Installing FPC

```
# Install the compiler.
cd fpc-n.n.n-x86_64-linux
chmod ug+x install.sh

```

```
sudo ./install.sh
[sudo] password for norman:
```

The first prompt from the installer is for a location to install FPC and the RTL. This location should be on your path which invariably means that you will need root privileges, which is why we need use `sudo` to run the installer. I chose `/usr/local` as my install location, as follows:

```
This shell script will attempt to install the Free Pascal
   Compiler
version n.n.n with the items you select

Install prefix (/usr or /usr/local) [/usr] :  /usr/local

Installing compiler and RTL for x86_64-linux...
... Lots more here.

Running on linux
Write permission in /etc.
Writing sample configuration file to /etc/fpc.cfg
Writing sample configuration file to /usr/local/lib/fpc/n.n.n
   /ide/text/fp.cfg
Writing sample configuration file to /usr/local/lib/fpc/n.n.n
   /ide/text/fp.ini
Writing sample configuration file to /etc/fppkg.cfg
Writing sample configuration file to /etc/fppkg/default

End of installation.

Refer to the documentation for more information.
```

You may wish to make a note of those locations for the various configuration files, in case you need or want to change things. They should be fine for the host system – I didn't have to change mine.

After installing the compiler, we are then prompted to install the documentation and demonstration files. I chose not to, but if you wish to do so:

```
Install documentation (Y/n) ? Y
Installing documentation in /usr/local/share/doc/fpc-n.n.n
   ...
Done.

# For the demos location, just press ENTER.

Install demos (Y/n) ? Y
Install demos in [/usr/local/share/doc/fpc-n.n.n/examples] :
Installing demos in /usr/local/share/doc/fpc-n.n.n/examples
   ...
Done.

... Lots of extra text here.
```

3.3 Testing the Host Compiler

We now need to make sure that the compiler works. I created myself a `SourceCode` directory tree for all my source code, then changed into it ready for action:

```
mkdir -p ~/SourceCode/Pascal
cd ~/SourceCode/Pascal
```

Create the following `hello.pp` file:

```
program hello;
const
    helloText = 'Hello FPC World!';

begin
    writeln(helloText);
end.
```

Compile the test code:

```
fpc hello.pp
```

```
Free Pascal Compiler version n.n.n [2020/06/09] for X86-64
Copyright (c) 1993-2020 by Florian Klaempfl and others
Target OS: Linux for i386
Compiling hello.pp
Linking hello
8 lines compiled, 0.2 sec
```

Looks good, now test it:

```
./hello
```

```
Hello FPC World!
```

So far the host compiler is looking good and can at least compile a simple Pascal program. We are ready to use FPC to rebuild itself as a Sinclair QL cross compiler.

4 Installing the Compiler Source Code

Once we have the host compiler working, we can get hold of the compiler's own source code and use the host to build a cross compiler for the Sinclair QL. The first step is to grab the source code and to do this we need to use *subversion*.

```
cd ~/SourceCode
svn checkout https://svn.freepascal.org/svn/fpc/trunk fpc
... lots and lots of stuff scrolling past here!
```

These commands will create a new directory, `SourceCode/fpc`, then checkout the main trunk of the FPC source code into the new directory.

5 Building the Cross Compiler

5.1 Build the Assembler and Linker

We need a certified assembler and a linker first. We must use the *vasm* assembler and the *vlink* linker, or things won't work. The two URLs where the source code for these utilities is to be found, are:

- http://sun.hasenbraten.de/vasm/index.php?view=relsrc
- http://sun.hasenbraten.de/vlink/index.php?view=relsrc

You need to get the latest sources and *vasm* version 1.8 or higher, and *vlink* version 0.16c are required.

In Linux, I was able to use the `wget` command, but it's a simple task to open the two URLs above, and click the link to download the source files. However you do it, it's probably wise to save the files into your `SourceCode` directory along with all the other source we are building.

```
cd ~/SourceCode
wget http://sun.hasenbraten.de/vasm/release/vasm.tar.gz
...
tar -xvzf vasm.tar.gz

wget http://sun.hasenbraten.de/vlink/release/vlink.tar.gz
...
tar -xvzf vlink.tar.gz
```

Now we need to extract and compile both utilities and copy the executable into a location on the path, first *vasm*:

```
tar -xzf vasm.tar.gz
cd vasm
make CPU=m68k SYNTAX=std
...
sudo cp vasmm68k_std /usr/local/bin/
cd ..
```

The build process will create an assembler binary named `vasmm68k_std`.

Next, the *vlink* linker:

```
tar -xzf vlink.tar.gz
cd vlink
make
...
sudo cp vlink /usr/local/bin/
cd ..
```

5.2 Rename the Assembler and Linker

After this, both files must be either renamed or sym-linked (on Linux) as follows. If you are on Windows then a rename should be sufficient.

- The assembler must be named `m68k-sinclairql-vasmm68k_std`.
- The linker must be named `m68k-sinclairql-vlink`.

```
cd /usr/local/bin
sudo ln -s vlink m68k-sinclairql-vlink
sudo ln -s vasmm68k_std m68k-sinclairql-vasmm68k_std

# Optional, just rename:
#sudo mv vlink m68k-sinclairql-vlink
#sudo mv vasmm68k_std m68k-sinclairql-vasmm68k_std
```

5.3 Build the Sinclair QL Cross Compiler

Now we can build and install the cross compiler. I'm creating a new directory, named `bin`, in my home directory, for the installation. This will need to be added to my `$PATH`[1] at some point:

```
# Create a new bin directory for the cross compiler.
mkdir -p ~/bin
export PATH=~/bin:$PATH

# Get back to the source.
cd ~/SourceCode/fpc

# Clean everything  out.
make clean OS_TARGET=sinclairql CPU_TARGET=m68k
... Huge piles of scrolling stuff here!

```

[1] On Linux Mint, and probably Ubuntu as well, when using the *bash* shell, if a user's `$HOME` directory includes a directory named `bin`, it is automatically added to `$PATH` at login time.

```
# Build the cross compiler.
make crossall OS_TARGET=sinclairql CPU_TARGET=m68k
... More huge piles of scrolling stuff here!


# Install the cross compiler.
make crossinstall OS_TARGET=sinclairql CPU_TARGET=m68k
   INSTALL_PREFIX="/home/norman/bin"
... Guess what happens here!
```

This will create a file named `/home/norman/bin/lib/fpc/n.n.n/ppcross68k`, but I prefer to call mine `fpc-ql`, so:

```
cd ~/bin
ln -s /lib/fpc/n.n.n/ppcross68k fpc-ql
```

I've got a sym-link set up but on Windows you can easily copy or rename it. Now we can test it.

```
fpc-ql
```

```
Free Pascal Compiler version n.n.n [2021/04/05] for m68k
...
CTRL-C
```

We can see that we have a compiler for the M68K CPU.

5.4 Create the Configuration File

We need to create a configuration file for the cross compiler.

```
cd ~/bin/lib/fpc
mkdir etc

cd etc
```

You may have to create the `etc` directory, I did. A file named `fpc.cfg` is required with the following content:

```
#IFDEF CPUM68K
-Fu<path/to/install>/lib/fpc/$fpcversion/units/$fpccpu-$fpcos
-Fu<path/to/install>/lib/fpc/$fpcversion/units/$fpccpu-$fpcos
   /*
#IFDEF SINCLAIRQL
-FD</path/to/vasm-and-vlink>
-XPm68k-sinclairql-
```

```
#ENDIF
#ENDIF
```

You will note the use of place markers for the installation directories. As my installation directory was the `bin` directory in my home folder, my file looks like this:

```
#IFDEF CPUM68K
-Fu/home/norman/bin/lib/fpc/$fpcversion/units/$fpccpu-$fpcos
-Fu/home/norman/bin/lib/fpc/$fpcversion/units/$fpccpu-$fpcos
   /*
#IFDEF SINCLAIRQL
-FD/usr/local/bin
-XPm68k-sinclairql-
#ENDIF
#ENDIF
```

If you are only interested in building Pascal programs for an *actual* QL, and have no intention of developing anything for Windows or Linux, then this config file is an excellent option:

```
#IFDEF CPUM68K
-Tsinclairql
-Fu/home/norman/bin/lib/fpc/$fpcversion/units/$fpccpu-$fpcos
-Fu/home/norman/bin/lib/fpc/$fpcversion/units/$fpccpu-$fpcos
   /*
#IFDEF SINCLAIRQL
-FD/usr/local/bin
-XPm68k-sinclairql-
-WQxtcc
#ENDIF
#ENDIF
```

You may have noticed that I added the `-WQxtcc` option? This is the option that tells the cross compiler to add an XTcc trailer record to the end of the compiled binary, this holds the data space required by the program. You can use this on a normal QL to convert the file into an executable[2]. The default is to write a special header at the start of the file and most/all the QL emulators understand this and can execute the file directly. If you are running on an emulator, like me, then omit the `-WQxtcc` option.

5.5 Building Your First QL Program

Now, test the build. In the following, I've used the `-Tsinclairql` option to tell the compiler to compile for a QL. If you added the option to the config file, there's no need to use it here.

[2]Using my own excellent(!) utility, Xtcc_bin. Issue_6 of my eMagazine has all the details.

```
cd ~/SourceCode/Pascal
fpc-ql -Tsinclairql hello.pp
```

```
Warning 22: Attributes of section .text were changed from r-x
   - in Linker Script <link15795.res> to rwx- in hello.o.
```

You can safely ignore the warning. On Linux it's easy to determine if the correct file has been created provided the option to use an XTcc trailer was specified in the config file, or at compile time with the `-WQxtcc` option:

```
file hello.exe
```

```
hello.exe: QDOS executable 'FPC'
```

The executable is `hello.exe`, it's a QDOS executable and since FPC version 3.3.1, the job name is "Program" unless you give the code a "Program xxxx" statement in which case the job name will be "xxxx"[3]. All we have to do now is get it over to a QL and try it out.

[3] Ok, confusion! In FPC you do not have to give a program a name unlike standard Pascal, so the "Program" line can be left out. This gives the QL executables a default program name of "Program". If you do include the "Program" line, the QL executable takes the supplied program name as the job name. However, you can change the job name on the fly, if you wish, using the QL specific functions SetQLJobName(), and retrieve the job name with GetQLJobName() which returns a string or GetQLJobNamePtr() which returns a pointer.

See https://wiki.freepascal.org/Sinclair_QL#Job_Name for details.

6 Compiling QL Pascal programs

When writing your source code, standard Pascal requires that the first "executable" line in the file be the `Program ProgramName( ... )`, however, FPC doesn't require you to have this line at all. On the QL, the executable's job name will be determined from the `Program` line. If one is present, the job name will be as per the given program name, if that line is not present, the default job name will be "Program". There are some special QL functions that can be used to set or retrieve the job name. This may be useful if you are writing a file processing task of some kind, and you wish to add the current filename to the job's name. You are limited to 48 characters maximum though, so don't go too overboard!

The functions are:

- `Function SetQLJobName(const s: string): longint;` This function sets the job's name from a Pascal string and returns the number of characters successfully set as the Job name, or -1 if there was an error.
- `Function GetQLJobName: string;` This function returns the current job name as a Pascal string, or empty string if there was an error.
- `Function GetQLJobNamePtr: pointer;` This function returns a pointer to the Job name stored as a QL string (2 byte length + series of characters), or nil if there was an error.

To compile a QL program, all you have to do is:

```
cd ~/SourceCode/Pascal
fpc-ql -Tsinclairql hello.pp
```

```
Warning 22: Attributes of section .text were changed from r-x
   - in Linker Script <link15795.res> to rwx- in hello.o.
```

As mentioned, if you added the `-Tsinclairql` option to the config file, you don't have to specify it here. I need to compile code for both Linux and the QL. I test my dodgy Pascal code on Linux first to be sure it compiles and runs correctly, then I repeat the operation for the QL. At least then I know that if it worked on Linux but doesn't on the QL, then it's the QL RTL that's most likely to be the cause.

You can safely ignore the warning about the attributes at the end of the compilation output.

The executable file for the QL will be created as "hello.exe" and will have a built in file header with details of the data space required unless you specified the `-WQxtcc` option either on the command line or in the config file. There are two options which control how the executable file for the QL gets it's data space information:

- `-WQqhdr` Set metadata to QDOS File Header style. A header record will be added to the file and QPC or other emulators can use this to execute the file directly, even from a `dos_` device. This option is the default and doesn't need to be specified.
- `-WQxtcc` Set metadata to XTcc style. This will be needed on an actual QL and you will need a utility to convert the file to an executable.

To compile the example program, there are a number of options:

- `fpc-ql hello.pp` this assumes that the `-Tsinclairql` option is to be found in the config file, if not, it will fail to compile. The executable will be created with header details embedded into the executable file ready for use on various QL Emulators.
- `fpc-ql -WQqhdr hello.pp` This is exactly the same as the variant above, however, it will override any `-WQ` option in the config file and force an embedded header to be used.
- `fpc-ql -Tsinclairql hello.pp` this variant explicitly specifies that the file must be compiled for the Sinclair QL and will overwrite any existing -T option in the config file. The executable will be created with header details embedded into the executable file ready for use on various QL Emulators.
- `fpc-ql -Tsinclairql -WQqhdr hello.pp` This is exactly the same as the variant above, however, it will override any `-WQ` option in the config file and force an embedded header to be used.
- `fpc-ql -WQxtcc hello.pp` The executable will be created with an XTCC trailer record holding dataspace details. This variant is suitable for use on an actual QL after the executable has been processed by an XTcc utility.
- `fpc-ql -Tsinclairql -WQxtcc hello.pp` This is exactly the same as the variant above, however, it will override any `-WQ` option in the config file and force an XTCC trailer record to be used.

7 Running Compiled Programs on the QL

7.1 Programs Compiled with -WQqhdr

These programs are best suited to the various QL emulators which know about the embedded header with the dataspace details. I use QPC and it copes happily with me executing these programs directly from the `dos_` drive. I don't have to make any changes, or somehow tell QPC that although the file is executing from a non-QL device, it still works perfectly.

```
ex dos1_hello.exe
```

And that's all there is to it! A 512 by 256 window will open and display the following:

```
Hello FPC World!
Press any key to exit.
```

7.2 Programs Compiled with -WQxtcc

The executable file which the cross compiler created can be copied over to a QL (or QPC or other emulator) and converted into a *proper* QDOS executable. In my case I can use my XTCC utility from my somewhat irregular Assembly Language eMagazine, Issue 6 – available from My GitHub[1], to read the details and write out an executable file.

```
ex win1_source_xtcc_xtcc_bin,ram1_hello.exe
```

The file, `hello.exe` is now ready to be executed.

```
ex ram1_hello.exe
```

A 512 by 256 window will open and display the following:

```
Hello FPC World!
Press any key to exit.
```

Success!

[1]https://github.com/NormanDunbar/QLAssemblyLanguageMagazine/releases/tag/Issue_6

8 Amending the Run Time Library

The system unit lives in the run time library and is responsible for all the various startup needs of a compiled Pascal program. It opens the standard files, sets the program name and so forth. Sometimes it may be necessary to edit and recompile the RTL.

8.1 Editing Source Code

The source for the RTL for the QL, lives in the `rtl/sinclairql` directory which you will find beneath the `fpc` directory created when you installed the source code previously. In my case, `SourceCode/fpc/rtl/sinclairql`.

Files of interest here are:

- `System.pp`
- `sysfile.inc`
- `sysdir.inc`
- `qdosfuncs.inc`
- `qdos.inc`

8.2 Compiling the RTL

8.3 Installing the RTL

8.4 Creating a Patch File

8.5 Applying a Patch File

9 Building the Run Time Library

10 Amending QLUnit

11 Building QLUnit