HOWTO: HAVE FUN WITH ee9, VERSION 2.1

CONTENTS

ADMONITION

First of all, make sure you have read, understood and inwardly digested the contents of the `README` file for this release of **ee9**. Secondly, do the same for '`Users Guide for ee9.pdf`'. I know, I know! Reading documentation is **no** fun! But it **is** good for you. You'll get no pudding until you've eaten your greens!

If that has not put you off the notion entirely, here is some guidance as to how best to use **ee9** with Usercode programs, Whetstone Algol programs and the EE Time Sharing Director. To be honest, and despite the foregoing, you can get a much quicker start by reading 'Getting started' and 'Simple Examples' in the `README` file.

USING USERCODE

The EE KDF9 Usercode manual is the indispensable guide to KDF9 assembly-level programming. It is available in DjVu format at:

http://www.findlayw.plus.com/KDF9/Documents/Usercode%20Manual.djvu

In the olden days, Usercode programs were compiled to machine code using KDF9's native assembler. Unfortunately neither a binary nor a source text has survived (so far as I know—if you know differently, please get in touch). Instead, we have David Holdsworth's brand-new Usercode assembler **kal3**. It runs on your own computer, generating a KDF9 machine code file that **ee9** can load and run.

A small selection of programs can be found in the directory `Testing/Assembly`.

I provide some shell commands, in the `Testing` directory, to make it a little easier to compile and run Usercode. These take Usercode programs from text files named with the '`.k3`' suffix, make listings in files named with the '`.txt`' suffix, and generate KDF9 machine code programs in files named with the '`.kdf9`' suffix. The suffix is optional in a name given as a parameter to the **ucc**, **nine** and **lud** commands: if supplied, it must be correct; if omitted, it is automatically restored.

- The **ucc** ('**U**ser**c**ode **c**ompile') command runs **kal3**, saves a compilation listing, and names the KDF9 machine code file after the input file; e.g.:

```
./ucc my_prog
```

compiles the program in the file `Testing/Assembly/my_prog.k3`, leaving a listing in `Testing/Assembly/my_prog-listing.txt`, and a binary in `Testing/Binary/my_prog.kdf9`, conveniently placed for the **nine** command.

- The **nine** command runs a KDF9 machine code program in `Testing/Binary`; e.g.:

```
./nine my_prog my_data t
```

runs the binary `Testing/Binary/my_prog.kdf9` in **t**race mode (or **f**, **p**, **s**, or nothing at all), having attached TR0 to the data file `Testing/Assembly/my_data.txt`.
The tracing verbosity can specified by a fourth parameter, composed of the same characters as the `V` option in a settings file. For example:

```
./nine my_prog my_data t hps
```

runs `my_prog` in trace mode, the tracing verbosity being reduced by omitting the **h**istogram, **p**eripheral I/O trace, and **s**ignature. If either the TP0 or the LP0 file is found to be non-empty at the end of a run, it is displayed on the terminal. The data file and the diagnostic mode are both optional. To give a mode, but not an input file, specify the file parameter as a minus sign; e.g.:

```
./nine my_prog – f
```

- The **nine_test** command runs a KDF9 machine code program in **test program** mode, but in every other way it has the same specification as the **nine** command.

YOU PROBABLY DON'T NEED TO KNOW THIS:

• The **lud** ('**l**ink **U**sercode **d**ata') command is used by nine to copy a data file into TR0. You can invoke it directly to supply a data file in TR0 for a run of **ee9** that you call yourself. For example, the following is somewhat like the first `./nine` command, above:

```
./lud my_data
./ee9 -sp -dt <Binary/my_prog > TP0
```

WALKING WITH WALGOL (THE WHETSTONE ALGOL 60 SYSTEM)

With the best will in the world it is impossible to describe the execution of an Algol program, using Walgol, as a 'run'; even 'walking' suggests a celerity that was foreign to its nature. Perhaps 'crawl' conveys the best impression, but **ee9** runs code so much faster than the KDF9 hardware that you are unlikely to get bored!

The EE KDF9 Algol manual is available in PDF and DjVu formats at:
http://www.findlayw.plus.com/KDF9/Documents/EE%20KDF9%20Algol%20Manual.pdf
and
http://www.findlayw.plus.com/KDF9/Documents/EE%20KDF9%20Algol%20Manual.djvu

A selection of Algol programs can be found in the directory `Testing/Algol`.

I provide some shell commands, in the `Testing` directory, to make it a little easier to compile and run using Walgol. These commands take Algol 60 programs from text files named with the '`.a60`' suffix. The suffix is optional in a name given as parameter to the **whet** and **lap** commands: if supplied, it must be correct; if omitted, it is automatically restored.

YOU DO NEED TO KNOW THE FOLLOWING:

You are likely to encounter Walgol compilation and execution errors. They have identifying numbers, tabulated separately for Translator and Controller, in the file '`Documents/Walgol Error Numbers.pdf`'.

A compilation listing is produced only if the response to the compiler's '`OUT;`' prompt is an OUT 8 stream number (reply '`10.|`' or '`30.|`'), rather than '`N.|`'. Stream 10 is directed to TP0; stream 30 goes to LP0. You may find that the listing disappoints: the source code is not included; instead the compiler outputs a table relating source line numbers to object code syllable addresses.

When the Algol object program starts, it prompts '`STREAM;`' for the number of the OUT 8 spooled output stream to be used; again, reply '`10.|`' or '`30.|`'. This should be the same as the stream number specified in the source program.

• The **whet** command compiles and executes a Whetstone Algol program; e.g.:

```
./whet my_prog
```

compiles and executes `Testing/Algol/my_prog.a60`, with the compiler in fast mode; alternative modes are as given, above, for the **nine** command. If you want to specify a mode for the execution of the object program by the Whetstone Controller, different from that used by the Whetstone Translator, then you need to set up an options file, `settings_2.txt`.
The tracing verbosity can specified by a fourth parameter, as for the **nine** command. For example:

```
./whet my_prog t hps
```

runs `my_prog` in trace mode, the tracing verbosity being reduced by omitting the **h**istogram, **p**eripheral I/O trace and **s**ignature.

A Walgol source program and its (stream 20) data are both read from TR0, so it is convenient to include the program and its data in the same text file (`my_prog.a60`, in the given example). Alternatively, only the program need be held in that file, with its data in files `TR0a`, `TR0b`, etc., using the facility for attaching TR0 at run-time to a succession of files.

The **whet** command sets up the FW0 file for the Whetstone system so that you do not need to type in any responses to Flexowriter prompts. If the given replies are unsuitable, change the data in `Testing/FW0_for_Whetstone`.
If either the TP0 or the LP0 file is found to be non-empty at the end of a run, it is displayed on the terminal.

BUT YOU PROBABLY DON'T NEED TO KNOW THIS:

• The **lap** ('**l**ink **A**lgol **p**rogram') command is used by **whet** to copy a data file into TR0. You can invoke it directly to supply a data file in TR0 for a run of **ee9** that you call yourself. For example, the following is somewhat like the latter example of the `./whet` command:

```
./lap my_prog
./dow t hps
```

AND YOU PROBABLY DON'T NEED TO KNOW THIS EITHER:

• The **dow** ('**do W**hetstone') command conveniently abbreviates a call on **ee9** to execute the Whetstone system. For example, the following is somewhat like the latter example of the `./dow` command:

```
./ee9 -sp -dt -mhps <Binary/KMW0201--UPU  >TP0
```

DIRECTING DIRECTOR (THE EE TIME SHARING OS)

EE implementation documentation describing the structure and functioning of Director is available at:

http://sw.ccs.bcs.org/KDF9/directorManuals/manuals.htm

• I provide a shell command, **tsd**, in the Testing directory, to make it a little easier to run the EE Time Sharing Director. The **tsd** command takes only the optional diagnostic mode flags, e.g.:

```
./tsd t hrs
```

runs the Time Sharing Director in trace mode, without **h**istogram, **r**etrotrace or **s**ignature.

The **tsd** command sets up the FW0 file for Director, so that you do not need to type in any responses to prompts during its initialization phase. If the given replies are unsuitable, change the data in `Testing/FW0_for_Director`. Before attempting to run Director, if you have not already run the self-test procedure, be sure to run the command:

```
./nine RLT Assembly/RLT_data.txt
```

to make sure all the magnetic tape decks have properly labelled tapes to mount.

YOU PROBABLY DON'T NEED TO KNOW THIS:

• The **tsd** command conveniently abbreviates a call on **ee9** to execute the Time Sharing Director. For example, the following is somewhat like the latter example of the `./tsd` command:

```
./ee9 -tb -dt -hrs <Binary/KKT40E007UPU  >TP0
```

BUILDING YOUR OWN VERSION OF `ee9`

The `Build` directory contains a command file called **`mk9`**, which is used to build **`ee9`** binaries. **`mk9`** takes optional parameters that determine the build flags to be used for the compilation. These flags can easily be amended if you find that they are unsuitable for your development environment: see the **`mk9`** file itself.

The first parameter of **`mk9`** should be one of the following:

`ee9`: make an optimised binary, with no optional compilation warnings, but with a high level of runtime checking enabled (default)

`max`: make an optimised binary, with no optional compilation warnings, and with no runtime checking (it runs ~6% faster than using **`ee9`**)

`warn`: make an unoptimised binary, with many optional compilation warnings

`unop`: make an unoptimised binary, with no optional compilation warnings

`verbose`: make an optimised binary, with (too) many optional compilation warnings

`kal3`: compile the sources in `Build/kal3/` and leave the **`kal3`** object program in `Testing/kal3`

`all`: call **`mk9`** successively with the `clean`, `kal3`, and `ee9` parameters

`clean`: remove all workfiles created by a compilation

`tidy`: establish a standard execution environment in the `Testing` directory

`zip`: create a compressed zip archive of `emulation`, named <*distribution*>`.zip`

`distro`: calls **`mk9`** successively with the `all`, `clean`, `tidy` and `zip` parameters; the binaries included in the distribution were created using the `distro` parameter.

Note that optimised builds of **`ee9`** run about 4 times faster than unoptimised builds!

The second, *distribution*, parameter is given only if a build type is the first parameter, and is one of the following:

`UNIX`, `Unix`, `unix`, `LINUX`, `Linux`, `linux`, `Mac`, `OSX*`, `Intel_OSX`, `PPC`, `RPi`, `Raspbian`, or `raspbian` (where `*` is any string):

Build a binary for OS X, Linux, or other Intel UNIX system that supports ANSI-terminal escape sequences and has `/dev/tty` as the interactive terminal. **This is the default**, so, e.g.:

```
./mk9 ee9
```

has the same effect as:

```
./mk9 ee9 OSX_Yosemite
```

and as:

```
./mk9 ee9 Linux
```

`PPC`:

Build a binary for a PowerPC G5 Macintosh under OS X 10.5 (PowerPC Leopard) or later, e.g.:

```
./mk9 ee9 PPC
```

`Raspberry_Pi`, `RPi`, `Raspbian`, or `raspbian`:

Build a binary for the Raspbian operating system on the Raspberry Pi, e.g.:

```
./mk9 ee9 RPi
```

`WINDOWS`, `Windows`, or `windows`:

Build a binary for Microsoft Windows, e.g.:

```
./mk9 ee9 Windows
```

The compilation listing is left in a file named `Build/komlog.ada`; it should be free of any compilation warning messages. It starts with a header that identifies the **`ee9`** version, **`mk9`** build type, and compilation options used.

If you have difficulty in compiling **`ee9`** warning-free, please let me know, as it may indicate a portability defect.

P.S. For MacOS X 10.5 (PowerPC Leopard) users only.

The Ada compiler presently available for this system is rather old, and generates a spurious error when an optimising compilation is attempted. For that reason, the distribution was created with the special **mk9** second parameter `PPC`, which implements the `all` option with the `unop` parameter, in place of the `ee9` parameter. The un-optimized binary of **ee9** that results is slower as a consequence, but it is still much faster than the KDF9 hardware.

P.S. For Raspberry Pi users only.

The Ada compiler distributed for use with Raspbian fails by running out of stack space when an optimising compilation of the `ioc-magtape.adb` file is attempted. For that reason, **mk9** separately compiles `ioc-magtape.adb` at optimisation level `-O0` (for the Raspberry Pi only). The un-optimized binary that results is then linked in with an optimised compilation of all of the rest of the code. Since the speed of `ioc-magtape.adb` is not important to the performance of **ee9**, this workaround has negligible overall effect.