README: ABOUT THE KDF9 EMULATOR **ee9**, VERSION 2.0r

CONTENTS

INTRODUCTION

A. GETTING ACQUAINTED

Version 2.0r of **ee9**, my KDF9 emulator, is the first to support the KDF9 graph plotter. **ee9** runs a variety of interesting and important KDF9 programs. It has proven very reliable, and has no known bugs, but of course you may discover some. Please report them to me, using the address `kdf9@findlayw.plus.com`, to help advance the debugging effort.

The download package for OS X includes binaries that run on 10.10 (Yosemite), 10.9 (Mavericks), 10.8 (Mountain Lion), 10.7 (Lion) and 10.6 (Snow Leopard). Additional download packages include binaries for other OS/hardware combinations. Mike Hore has provided a build of **ee9** and **kal3** for a PowerPC G5 Mac running OS X 10.5 (Leopard). Bill Gallagher has provided builds for Intel x86_32 and x86_64 under Linux, for Intel x86_32 under Microsoft Windows, and for the ARM11-based Raspberry Pi under Raspbian Linux. The source code of **ee9** is released under the terms of the GPL, version 3.

These distributions can be found at:

`http://www.findlayw.plus.com/KDF9/emulation/`

B. GETTING STARTED

A number of files in the `Testing` directory need to have execute permission set. If necessary, you can ensure this by running (in `Testing`) the command:

```
    sh ./set_permissions
```

before doing anything else.

That done, you can verify the correct operation of my binary, or a port to another platform, by running (in `Testing`) the command:

```
    ./ee9_self_test
```

To be honest, this checks that your **ee9** implementation gives the same results as mine, on a variety of Usercode and Whetstone Algol programs. It runs a set of test cases and allows you to compare expected and actual outputs and execution digital signatures (these are hashes of the register values at the end of each instruction-execution). It automatically compares its results file with a file containing the results from a run on my own computer, taken to be the standard of correctness. If your **ee9** passes these tests we can safely assume that it is working as well as mine.

THE EE9 EMULATOR SYSTEM, V2.0r: WHAT YOU GET

A. IN THE `Documents` DIRECTORY:

1. A guide to using **`ee9`**: '`Users Guide for ee9.pdf`': **read this document first**!
2. A '`HOWTO`' file that gives instruction on the operating procedures to be followed for compiling and executing both Usercode and Whetstone Algol (Walgol) programs.
3. Two papers by me, documenting the KDF9's hardware and software:
   '`The English Electric KDF9.pdf`' and
   '`The Hardware of the KDF9.pdf`'.
4. A list of all the error numbers generated by the Whetstone Translator and Controller:
   '`Walgol Error Numbers.pdf`'
5. A file, '`ee9 Release History.pdf`', that describes the amendment history of all **`ee9`** releases to date.

B. IN THE `Testing` DIRECTORY:

1. An `Algol` subdirectory, containing Whetstone Algol 60 source-program and data files
2. An `Assembly` subdirectory, containing Usercode assembly-language source program files and their data files
3. A `Binary` subdirectory, containing KDF9 machine code programs for:
   a. The Whetstone Translator (compiler): `KMW0201--UPU`
   b. The Whetstone Controller (interpreter/VM): `KMW0301--UPU`
   c. The EE Time Sharing Director (KDF9 OS): `KKT40E007UPU`
   d. A selection of programs compiled from the `Assembly` subdirectory.
4. A collection of files to represent the KDF9's I/O devices; for example: `LP0`, `TR1`, and so on
5. An executable binary of **`kal3`**, David Holdsworth's new Usercode compiler
6. UNIX shell scripts to facilitate running **`kal3`**, KDF9 object programs, the Whetstone Algol system, and the Time Sharing Director.
7. An executable binary of my KDF9 emulator: **`ee9`**
8. A small set of shell files to implement a basic self-checking process for **`ee9`**:
   `ee9_self_test`, **`ee9_test_case`**, and **`ee9_test_run`**
9. A text file containing the results of running **`ee9_self_test`** on my own computer, used as a comparator for its own results: `ee9_good_test_case_log.txt`

C. IN THE `Source` DIRECTORY:

Complete Ada 2005 source code for **`ee9`** is provided. It should compile without errors and without warning messages. For Windows only, there is also a short routine, written in C, to fetch a value defined in a C header file in Cygwin.

D. IN THE `Build` DIRECTORY:

1. A shell command file to build **`ee9`**, **`mk9`**
2. GNAT option files for various alternative builds of **`ee9`**: `adc-*.adc`;
   The currently distributed binary is a testing build, for obvious reasons.
   (A build without runtime testing is less than 6% faster on my MacBook Pro.)
3. The option file used to generate the distributed binary of **`ee9`**: `gnat.adc`
4. The compilation listing of that build: `komlog.ada`

SIMPLE EXAMPLES

These examples assume that the current working directory is `Testing`. It is convenient to have a terminal window open in the `Testing` directory, if you will be making multiple runs of **`ee9`**. If you further put `Testing` in your shell `PATH` variable, you will be able to run the commands supplied in `Testing` with no need for the '`./`' prefix shown in the examples below.

The shell commands **`tsd`**, **`whet`**, **`nine`** and **`nine_test`** conveniently abbreviate common usages of **`ee9`**. They invoke, in turn, simpler commands that you may prefer to use directly. For much more detailed help with this, see the `HOWTO` file in the `Documents` directory.

A. RUNNING THE WHETSTONE ALGOL SYSTEM:
To run the Whetstone system on the Algol 60 program `Algol/sieve.a60`:

```
        ./whet sieve
```

B. COMPILING AND RUNNING USERCODE PROGRAMS:
To compile `Usercode/HiGuys.k3`, and run it in trace mode (N.B.: `HiGuys` needs no data file, hence the '`-`'):

```
        ./ucc HiGuys
        ./nine HiGuys - t
```

C. RUNNING THE TIME SHARING DIRECTOR:
To execute a run of Director in trace mode:

```
        ./tsd t
```

MISCELLANEOUS OS ISSUES

A. POSIX ENVIRONMENT
You may benefit from ensuring that your system is set up to use the Latin-1 character set. You can find out by issuing the bash command:

```
        set | grep LANG
```

If the output does not contain '`ISO8859-1`' then you need to reset your locale and character set. The bash command:

```
        export LANG=en_GB.ISO8859-1
```

(or its equivalent using another shell) will achieve this for UK users. (This is not relevant to Windows.)

If you are using Windows and you want to run the shell command files that I provide to make life with **`ee9`** a little easier, or if you want to compile your own version of **`ee9`**, then you will need to install Cygwin, which provides a POSIX command set and API for Windows, including a `bash`-compatible shell named `dash`. A suitable version can be obtained from: `http://www.cygwin.com/install.html`

B. GETTING A GNAT ADA COMPILER
If you want to build **`ee9`** yourself, you will need a recent version of GNAT, the GNU Ada compiler, because **`ee9`** is written in Ada 2005. Current versions of GNAT for several systems, including OS X, x86_32 Linux, x86_64 Linux, and x86_32 Windows, can be obtained with a GPL licence from AdaCore at: `http://libre.adacore.com/libre`

A good option if you have a PowerPC (G3/G4/G5) Mac running OS X Leopard or Tiger is:

`http://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2009-tiger-ppc/`

(N.B. on OS X you will also need to install XCode, Apple's SDK and IDE, and its BSD command-line subsystem.)

GNAT is supported by Raspbian on the Raspberry Pi, so use the `apt-get` utility to fetch and install it, thus:

```
   apt-get update
   apt-get install gnat
```

There are compilers for many other hardware/software combinations at:

`http://sourceforge.net/projects/gnuada/files/`

If your system is not one of these, you may find pointers to a suitable port at the Ada Programming Wikibook. If none of these meets your needs, try asking in the `comp.lang.ada` USENET newsgroup. People there are very helpful and may already have just the port you need. As a last resort, you may be able to compile your own version of GNAT; the source code is available for download from the Free Software Foundation. (Please don't ask me for help with this; I have absolutely no idea how to go about it!)

C. GETTING THE USERCODE COMPILER, **`kal3`**
David Holdsworth's **`kal3`** (a new KDF9 Usercode compiler) and **`KDF9Flex`** (a program to facilitate the creation of Algol 60 source code in a variety of convenient representations) can be obtained in source code from links given here:

`http://sw.ccs.bcs.org/KDF9/walgol.htm`

If you are using OS X, Linux, Raspbian or Windows, then you will find a binary of **`kal3`** in the appropriate download package. If not you will have to compile **`kal3`** for yourself, following David Holdsworth's (simple) instructions. Current sources for **`kal3`** are also provided in `Build/kal3/`, and these were used to generate the **`kal3`** binaries included in the download packages. If you want to recompile **`kal3`**, it would be worth checking the website in case a newer version is available. It is trivial to compile **`kal3`**, using the `kal3` option of **`mk9`**; a couple of warning messages are produced on my computer, but these do not presage any problems in use.