USERS' GUIDE FOR ee9: AN ENGLISH ELECTRIC KDF9 EMULATOR

by **BILL FINDLAY**

0: INTRODUCTION

This document is a guide for users of **ee9**, a program emulating the EE KDF9 computer. Readers not yet familiar with the KDF9 should consult the companion document, *The English Electric KDF9*; you will will be referred to the 'Manual' for further details; this is the *KDF9 Programming Manual*, the nearest thing we have to a definitive reference work. **ee9** is intended to be portable to any system with a basic POSIX API. It is written in Ada 2012 using GNAT, the GNU Ada compiler and is available for a range of computer architectures and operating systems. For the characteristics of this version, see §6.

1: USING ee9 CONVENIENTLY

ee9 can be invoked directly, but it is very much easier to use the shell command files described in this section. The default setup runs them in the directory `Testing`, so `Testing` should be in your search path to make them visible to the shell. That is what the following descriptions assume. However, they can be used elsewhere so long as you establish a compatible execution environment: see Appendix 1 and §3.1, *inter alia*.

ee9 uses the Latin-1 character set and requires the terminal window background to be set to a colour other than black or red; white gives the best approximation to the appearance of KDF9's console Flexowriter printout.

Using these commands on Microsoft Windows requires a `bash`-compatible shell, although you might run **ee9** itself without one. See the '`m`' visibility option (§5) to suppress SGR codes in a Windows command line that lacks ANSI terminal support.

A list of **ee9** error messages and explanations is given in Appendix 8.

`nine` *prog* [*data* | **`-`**] [*mode* | **`-`**] [*miscellany* | **`-`**] [*tt* [*tt*]]

`nine` runs the KDF9 machine code program *prog*. The executable is taken from `Binary/`*prog*; the data for paper tape reader 1 (TR1), if specified, is taken from `Data/`*data*`.txt`. A parameter may be given as '**`-`**' if it is not needed but a later parameter must be specified. For a first go, all the optional parameters can be omitted, as they default wisely.

The allowable execution ***mode*** characters (see §2.1) are:

- **`f`**: for **f**ast mode, the default (if **`f`** is explicitly specified, the real CPU time for the run is displayed on termination)
- **`p`**: for **p**ause mode
- **`t`**: for **t**race mode
- **`x`:** for e**x**ternal trace mode

The allowable characters in the optional ***miscellany*** parameter are described in §5.2.

The parameters *tt* specify the character codes to be used by either one or two paper tape punches (the first *tt*), or paper tape readers (the second *tt*). *tt* is either one or two characters, each being either `k`/`K` or `l`/`L`, specifying KDF9 paper tape code or Latin-1 code respectively. If *tt* is one character, it refers to unit 0 of the type (i.e. `TP0` or `TR0`). If *tt* is two characters, the first refers to unit 0 and the second to unit 1 of the type. The default code is Latin-1. See also §3.4 and Appendix 2. A program run by **`nine`** (or **`nine_priv`**), with **ee9** emulating the Director API, is said to be running 'directly' in this Guide.

`tsdnine` *prog* [*data* | **`-`**] [*mode* | **`-`**] [*miscellany* | **`-`**] [*tt* [*tt*]]

`tsdnine` runs a program with data supplied as TR1, under the control of the Time Sharing Director (TSD). The parameters are as for **`nine`**., except that a *data* file name without a **`.txt`** suffix is taken to be in KDF9 code. Before booting, **`tsdnine`** ensures that the magnetic tape files have valid labels. See Appendix 9 for a primer on operating the TSD.

Binary programs for execution by **`nine`** or **`tsdnine`** can be created by compiling Usercode or Kidsgrove Algol sources.

`ucc` *prog*

`ucc` compiles a Usercode source program using the **`kal3`** assembler. The source code is taken from `Assembly/`*prog*`.k3` and the object program is placed in `Binary/`*prog*; a compilation listing is stored in `Assembly/`*prog*`-listing.txt`.

kalgol *prog*

kalgol has one parameter, the name of an Algol source program held in Kidsgrove/*prog*.a60. The *mode* and *miscellany* parameters for the *compiler* may be given in the environment variables KIDSMODE and KIDSMISC. By default KIDSMISC=w to suppress the compiler's voluminous Flexowriter output. The Usercode object program is left in Assembly and the executable binary in Binary. See Appendix 12 for more help with Algol.

kids *prog* [*data* | –] [*mode* | –] [*miscellany* | –] [*tt* [*tt*]]

kids both compiles and runs a Kidsgrove Algol program (using **kalgol** and **nine**). It is not necessary to recompile a Kidsgrove Algol program each time you run it, although it normally takes only a fraction of a second to do so. The executable object program left in the Binary directory can be run using the **nine** command in the usual way. To compile a Kidsgrove Algol program without running it, use the **kalgol** command.

nine_priv *prog* [*data* | –] [*mode* | –] [*miscellany* | –] [*tt* [*tt*]] }

nine_priv operates like **nine**, but the program in Binary/*prog* is executed in the **priv**ileged **test program** mode.

crnine *prog* [*data* | –] [*mode* | –] [*miscellany* | –]

crnine operates analogously to the **nine** command, except that the data, if specified, is made available via the punched card reader file, CR0.

whet *prog* [*mode* | –] [*miscellany* | –]

whet runs the Whetstone Algol system on the Algol program and its stream 20 data, given consecutively in the single file Whetstone/*prog*.a60. They are read respectively by the Translator (compiler) and the Controller (interpreter). See Appendix 12 for more help with Algol.

stone [1964 | thecrows]

stone runs the Whetstone Benchmark program using **whet.** If 1964 or thecrows is given it runs in authentic timing mode; in other words, as slowly as Whetstone Algol did in 1964 on a real KDF9, taking about 7 minutes of real time.

sttl *prog* [*store* [*time*]]

sttl uses the **sed** stream editor to amend the ST (store size) and TL (time limit) fields of the Usercode object program generated by **kalgol**. If the *time* parameter is omitted, a value of 3000 seconds is used. If the *store* parameter is omitted, a value of 7680 words is used.

strop *prog*

strop uses the **sed** stream editor to do most of the work of converting an Algol program in Whetstone format to Kidsgrove format. It takes the name of a program file in the Whetstone directory and writes the converted program to a file of the same name in the Kidsgrove directory. The latter may need a little tidying-up before it passes compilation by **kalgol**.

ports *prog*

ports (**strop** backwards) uses the **sed** stream editor to reverse the transformations made by **strop**. It takes the name of a program file in the Kidsgrove directory and writes the converted program to a file of the same name in the Whetstone directory. You may find that a more convenient way of typing in a new Whetstone program, because using the Flexowriter-style underlining for reserved words is both tedious and error-prone.

2: Emulation Modes

2.1: Diagnostic Modes

A KDF9 program is run, at option, in one of four **diagnostic modes**. These are:

- **fast** mode; in which **ee9** runs the program at maximum speed, with no execution tracing or interactive diagnostic facilities available
- **pause** mode, in which **ee9** single-shots the program, pausing to interact with the user after each instruction
- **trace** mode, in which **ee9** runs the program at speed with extensive retrospective tracing enabled
- **external** trace mode, in which **ee9** writes a summary of every traced instruction to an external file

More precisely, things work as follows.

In fast mode **ee9** interacts with the user only by providing informative messages, either because the KDF9 program has terminated, or to log significant events during the run (such as the allocation of an I/O device). All tracing overhead is avoided in fast mode.

In pause mode **ee9** uses console-window text I/O to interact with the user. After each instruction is executed a short summary of the machine state is displayed and a prompt asks the user how to continue. The user replies with an optional single letter (which may be given in upper case or lower case) followed by RETURN, selecting one of the following:

- **f**: execution proceeds in **f**ast mode
- **p**: execution proceeds in **p**ause mode
- **t**: execution proceeds in **t**race mode
- (nothing): execution proceeds in the current mode.

All retrospective tracing types described in §4 are available in pause mode, trace mode, and external trace mode; but the manner of execution depends on whether the current instruction execution lies within a set range of addresses, and within set instruction-count bounds. If so, instructions are added to their appropriate traces; and breakpoints and watchpoints are monitored. If not, execution proceeds as in fast mode (but at about a third of the speed).

2.2: Run states

The run state specifies how the emulated KDF9 is to run the program:

- In **boot** mode the KDF9 reads a 9-word bootstrap routine from `TR0`, then jumps to word 0, in **Director** state.
- In **problem program** mode **ee9** reads into core, from `TR0`, a binary program prepared by a compiler. Its execution starts at word 0, in **program** state. **ee9** itself implements any `OUT`s (system calls) requested by the program, so that it is not necessary to have a Director running.
- In **test program** mode **ee9** reads a binary program into core from `TR0`, just as in problem program mode. Its execution starts at word 0, but in **Director** state. The emulator implements any `OUT`s requested by the program.

2.3: Breakpoints, Fetchpoints , Storepoints, and Watchpoints

Certain addresses in core can be marked as breakpoints or as watchpoints, to force diagnostic interaction with the user. A **breakpoint** is set on an instruction word, and causes interaction after an instruction beginning in that word has been executed. A **fetchpoint** is set on a data word, and causes interaction after data has been fetched from that word. A **storepoint** is set on a data word, and causes interaction after data has been stored into that word. A **watchpoint** combines a fetchpoint and a storepoint on the same word.

2.4: Authentic Timing Mode

At option, **ee9** can be made to insert timed pauses into its execution so that the elapsed time of a program run by **ee9** approximates the elapsed time of a run on the real KDF9 hardware. This may be instructive for younger users, who have never seen characters being output by a computer, one at a time, and with noticeable delays! This mode can be set using the authenticity setting, see under '**A**' in §5.1, or by means of the command-line miscellany parameter **`t`**; see §5.2.

3: Inputs and Outputs

3.1: Emulated KDF9 I/O Devices

At the start of a run **ee9** looks in the current directory for files to represent the KDF9 peripherals. There is a fixed assignment for the console Flexowriter, which is associated with the user's interactive terminal window. Other devices are associated with files having names derived from the device type. Magnetic tape deck *d*, for example, is always associated with the file named MT*d*. The full list of these associations is as follows:

- a **c**ard **p**unch is CP*d*
- a **c**ard **r**eader is CR*d*
- a **dr**um store is DR*d*
- a **f**ixed **d**isc store is FD*d*
- a **g**raph **p**lotter is GP*d*
- a **l**ine **p**rinter is LP*d*
- a KDF9 type 1081 **m**agnetic **t**ape deck is MT*d*
- a **S**tandard **I**nterface device is SI*d*
- an IBM-compatible **s**even-**t**rack tape deck is ST*d*
- a paper **t**ape **p**unch is TP*d*
- a paper **t**ape **r**eader is TR*d*

Every device file actually used in a run must exist, and be appropriately accessible, when **ee9** is invoked. A magnetic tape file, if not completely empty, must have a valid label block. (But see **`RLT`** in Appendix 11.) The assignment of devices to buffers (KDF9 DMA channels) is set by default, but can be changed by the user: see the `K` option in §5.1.

3.2: The Flexowriter Console Typewriter

The terminal window is the means by which users, in their rôle as KDF9 operators, can mimic Flexowriter I/O. The Flexowriter is used to type-in responses to prompts output by problem programs or by Director. Repeatedly typing these responses quickly becomes tedious. If a file named FW0 exists, it is used as a source of "canned" responses. They are defined, with their identifying prompts, in FW0; and are picked up automatically by **ee9**. If a prompt spreads over more than one line, a KDF9 Line Shift can be represented in FW0 by a '®', and a KDF9 Page Change by a '©'.

When a prompt is issued, **ee9** scans `FW0`, down from the last match found. If it finds a new match, it injects the given response into the Flexowriter input stream; but if it reaches the end of the file without finding a match, it returns control of the Flexowriter to the user's terminal window, so that a manual response can be given. If a prompt matches a line in `FW0` that specifies a null response string (c.f. the second '`OUT;`' in the following example) then **ee9** terminates the run.

For example, the Whetstone Algol compiler prompts '`[q] OUT;`' to which a typical reply is '`N.|`'. If the Algol program compiles, it runs and prompts '`[q] STREAM;`' to which a typical reply is '`30.|`'; but if the compilation fails the compiler loops back to its '`[q] OUT;`' prompt, where the user will normally want to terminate the run so that the Algol source code can be amended. The following data in `FW0` will achieve this without user intervention:

```
[q] OUT;N.|
[q] STREAM;30.|
[q] OUT;
```

For a second example, as the Time Sharing Director bootstraps it issues a series of requests for basic configuration parameters. The following data in `FW0` supplies suitable responses without user intervention:

```
CORE MODULES;8.|
OUT 8 REEL NO;9.|
LEVELS;N.|
DATE  D/M/Y;4/5/67.|
TIME ON 24-HOUR CLOCK®HOURS/MINS;1/23.|
```

This facility had a real equivalent: the Flexowriter incorporated an 'edge-punched card' reader. It read data (in paper tape code) from the edge of a non-standard punched card. Cards prepared with replies to prompts could be inserted into the reader and read at the maximum rate, thus speeding input and avoiding any delay due to typing errors by the operator.

Note that ee9 requires every Flexowriter input string to be terminated by a RETURN, even when a read-to-End Message instruction is being obeyed. In reality, KDF9 would end the transfer immediately at the End Message (EM) character, or when the required number of characters had been read; but data is not transferred to **ee9**'s input buffer until a RETURN is typed. A merely terminating RETURN is discarded from the input by **ee9**, and is not passed to the KDF9 program.

In response to CTRL-C, **ee9** outputs a prompt of its own that lets the diagnostic mode be changed. Replying with a RETURN (only) causes a FLEX interrupt; when running Director in boot mode, this evokes a '`TINT;`' prompt.

Output to the Flexowriter was typed in red; input from the computer operators in black. This is simulated in **ee9** by using ANSI-standard SGR escape sequences to vary the displayed font style. Some Windows terminal utilities do not implement SGR escape sequences, so Flexowriter output styling can be disabled with a miscellany parameter or a settings file option.

3.3: READING MORE THAN ONE ROLL OF PAPER TAPE OR DECK OF CARDS

A means is provided to simulate the way in which KDF9's computer operators could satisfy a program's demand for data with several physically-separate rolls of paper tape or decks of cards, loaded into a reader in succession. If a program attempts to read from a tape reader, and the end of the associated file has been reached, **ee9** allows the user to specify a successor file to which the tape reader is re-attached. A prompt requesting more data is displayed.

- If the user responds '`=`' then the tape reader is connected to the user's terminal so that the data may be typed then and there.
- If the user responds '`/`' then the pathname of a file supplying the next "roll of tape" may be typed.
- If the user responds '`@`' then the name of a file within the directory specified by the `KDF9_DATA` environment variable may be typed; see Appendix 1. With both '`/`' and '`@`' a final "`.txt`" may be omitted from the name. If the named file is not found or not readable the prompt is repeated.
- If the user responds '`q`' or '`Q`', or signals end-of-data (e.g. by typing `^D`) then execution terminates.
- If the user presses RETURN (only) the buffer is set to the abnormal state. Attempting another read without clearing the error will cause a Lock-In Violation.
- The foregoing also applies, *mutatis mutandis*, to punched card readers.
- See the **N** option in §5, about disabling this feature in non-interactive mode.

Examples (user's responses in **green**):

1.
```
ee9: Type @ or / to name a file, = to type the data, ENTER key for EOF, Q or q to quit: =
ee9: Type the data for TR1:
3|
```
2.
```
ee9: Type @ or / to name a file, = to type the data, ENTER key for EOF, Q or q to quit: /
ee9: Give the pathname of the file: ../next.txt
The KDF9 operator must have made a mistake: ../next.txt cannot be read.
```
3.
```
ee9: Type @ or / to name a file, = to type the data, ENTER key for EOF, Q or q to quit: /
ee9: Give the pathname of the file: Data/next.txt
```
4.
```
ee9: Type @ or / to name a file, = to type the data, ENTER key for EOF, Q or q to quit: @
ee9: Give the name of a file in Data/: next
```
5.
```
ee9: Type @ or / to name a file, = to type the data, ENTER key for EOF, Q or q to quit: q
…
Final State: Impossible I/O operation: quit requested by the user on TR1.
```

3.4: REPRESENTING THE KDF9 CHARACTER SETS

The KDF9 had its own, somewhat idiosyncratic, character sets. They are tabulated in Appendix 2a. For the convenience of the user, external data read and written by the paper tape readers and punches may be represented in either KDF9 paper tape code, or in the ISO Latin-1 character set with automatic conversion between Latin-1 and KDF9 codes. Several characters in the KDF9 paper tape set are absent from Latin-1, so a simple transliteration is used to represent them externally.

For the Flexowriter, and for Latin-1 I/O with tape punches and tape readers, Case Normal and Case Shift characters are generated on input, and interpreted on output. This means that when you are typing an input text, it is not necessary to type Case Normal and Case Shift characters, although there usually is no harm in doing so. When such a text is being read as the input stream for a two-shift device, an appropriate case-character is generated automatically by the emulator, if the Latin-1 character being read is not available in the input device's current shift state. Two-shift output works in the complementary manner. Two-shift devices always start out in the Case Normal condition.

For example, the external Latin-1 string '`Bill Findlay`' may be read into the KDF9 core store as the characters '`BßILL ñFßINDLAY`', with ß denoting the Case **S**hift character and ñ denoting the Case **N**ormal character. A KDF9 program that writes the characters '`BßILL ñFßINDLAY`' to a two-shift device in Latin-1 mode will generate the Latin-1 string '`Bill Findlay`' as its external representation.

Text-file input to **ee9** may use any of `CR`, `LF`, or a `CRLF` pair as the line terminator: **ee9** treats all three the same. Text-file output from **ee9** writes the line terminator most appropriate for the host's operating system.

Non-graphic KDF9 characters also have Latin-1 external representations, to enable 1-to-1 inter-conversion between the internal and external data formats. Apart from the format effectors (Horizontal Tab, New Line, Form Feed), users should never need to type these characters, as they could not be typed on a Flexowriter.

In KDF9 mode, characters are transmitted verbatim using the KDF9 paper tape format, in which devices use an 8-bit byte externally, supplementing the 6 internal data bits with two 'parity' bits (see also the Manual, §17.4 and Appendix 2b).

KDF9 object programs are never given in Latin-1, but always natively, in the KDF9 paper tape code.

Characters displayed in tracing output are shown using their Case Normal Latin-1 transliterations, except as follows:

- the KDF9 Horizontal Tab character is depicted as ¬
- the KDF9 Line Shift or LS character (Carriage **R**eturn) is depicted as ®
- the KDF9 Page **C**hange or PC character (Form Feed) is depicted as ©
- the KDF9 **F**iller character is depicted as ø

These depictions are also used by the card reader and card punch. Card images of less than 80 characters are padded with blanks to fill all 80 columns; any input line longer than a card is truncated to 80 characters. In 'direct' mode, card images may have up to 160 characters, notionally two per card column. The card punch writes files that are valid as input to the card reader and can be read to exactly reproduce the original data in core.

The representation of magnetic tapes is described in detail in Appendix 10.

Drums and disc drives are represented as plain POSIX files using the Latin-1 Case Normal transliteration. Sector boundaries are not explicitly represented: they are implicit in the data address within the POSIX file, occurring every 320 bytes for disc drives and every 1024 bytes for drums.

3.5: Graph Plotting

ee9 includes an emulation of the model 564 Calcomp graph plotter, as described in Appendix 6, §5, p.302 of the Manual (but see what I say about this in §6: Caveats, and in Appendix 3). There was provision on the KDF9 to switch a buffer manually between a tape punch and a graph plotter; in **ee9** this is done with **g** in the *miscellany* parameter, with a settings file option **G**, or with a settings file option **K**. When one or more of these is given, GP0 replaces TP1 on the shared buffer.

The KDF9 graph plotter takes commands that move the plotting position in steps of 0.005 inches; see Appendix 3 for the command codes. Steps are accumulated into vectors by **ee9**, and PostScript vector drawing commands are output to `GP0`, creating an Encapsulated PostScript (EPS) file. It is possible to fit the plotter with 'pens' having a variety of ink colour and ball-point tip size: see under **G** in §5.

4: Logging and Tracing

Messages that record the progress of the emulation, and details of any errors that were detected, are written to the interactive console window, along with interactive diagnostics and output produced for the KDF9 Flexowriter. A selection of these messages is also written to the file `KDF9_log.txt`. On completion of a run, the final machine state, any requested core store areas, and any retrospective traces may be written to the log file and to the console window.

It is possible to request the output of certain areas of the KDF9's core store, in a variety of suitable formats. These printouts can be taken either before the start of execution; or on termination; or at both times, to allow comparisons.

In the **interrupt** trace, which is produced only in boot mode, interrupt requests are listed with the privilege state and priority of the interrupting device; the elapsed time of occurrence (in μs); and the value of `ICR`, the Instruction Count Register, which is a count of the number of instructions executed so far. The interrupt trace also logs EXITD, return from interrupt, instructions.

The tracing of executed instructions is subject to instruction-count and address-range bounds. Instruction executions within those bounds are traced; those that fall outside the bounds are not.

In the peripheral I/O trace, the events shown are transfer initiations and terminations, busy-buffer and store-access lockouts, and I/O status test operations. Each is listed with the device name, Q-store parameter, privilege state (P for problem program state and D for Director state) and priority of the transfer; the elapsed time of occurrence of the event; and the current instruction count.

The C part of the parameter used in a disc seek operation is logged in the format D*d*P*pp*S*ss*, where *d*, *pp* and *ss* are, respectively, the drive number, platter number and *seek area* number being addressed. In non-seek disc operations the C part is shown as S*ss*, where *ss* is the starting *sector* number for the transfer.

The C part of the parameter used in a drum transfer is logged in the format D*d*T*tt*S*s* where *d*, *tt* and *s* are the drive number, track number and the starting sector number.

Transfer operations appear twice, once for the initiation (S) and once for the termination (E). Lockouts appear once, when they happen. A test operation gives the result of the test as a Boolean (Y or N) in the T field of a trace.

In the retrospective trace, instructions are listed in order, starting with the most recently executed. The trace includes the instruction itself, and its most relevant operand; 'ND' and 'SD', the Nest and SJNS Depths; 'V' and/or 'T' showing whether overflow and/or the test register is set; the CPU time of occurrence of the event; and the value the current instruction count. In the case of a store order, the traced operand is the value written to store. In the case of a fetch order, it is the value fetched. For a Q-store order, it is the content of the relevant Q register. For a conditional jump it is the determining value. For subroutine jump or exit, it is the relevant value in the SJNS. For a 1-syllable or 2-syllable ALU order, it is the value left in the top of the nest. And so on.

External trace mode is like retrospective mode, with additional output to the file `trace.txt`. This output has one line for each traced instruction. It contains: the instruction's address; the value of ICR; the CPU time; the nest depth; the SJNS depth; 'V' and/or 'T' if overflow and/or the test register is set; the value in N1, if the nest if non-empty; and the disassembled instruction.

When tracing, and if requested, **ee9** will tally the number of traced executions of each type of KDF9 instruction. On termination of a run a histogram of dynamic instruction-type frequencies may be logged. A similar plot may be produced showing the frequency with which each instruction-containing word is executed.

At option, all tracing modes can compute a digital signature of the execution: a 48-bit cumulative hash, displayed in octal, of the contents of all the relevant KDF9 registers (nest, SJNS and Q stores) at the end of each traced instruction. Known values for this hash can be used to verify the proper operation of an implementation of **ee9**. (When the signature is enabled, the starting time-of-day is forced to midnight, to ensure a repeatable hash value.)

For examples of the output that can be obtained using these facilities, see Appendices 7A-7G.

5: THE MODE SETTINGS FILES AND THE MISCELLANY PARAMETER

The emulator has default settings for all of its options, but they may be over-ridden by settings specified in files that the emulator attempts to read as part of its initialization, and/or by specifying a miscellany parameter on the command line.

5.1: SETTINGS FILES

The file `settings_1.txt` applies to a first or sole program to be run, and `settings_2.txt` applies to a second program overlaid by it (e.g. the Whetstone Controller, overlaid after a successful compilation by the Translator). A setting specified by the command line over-rides a similar option specified in the `settings_1.txt` file.

The settings files contain a line for each option to be set, beginning with a letter that specifies the option concerned. This may be followed by one or two parameters. Numbers may be given in octal—preceded by a hash sign ('#')—or in decimal; and this convention is also used systematically in output messages from the emulator. The options are presented here in upper case, but lower/mixed case is also accepted. The available options are as follows:

A MODERN_TIMES_MODE | AUTHENTIC_TIME_MODE

The **A** flag sets aspects of the AUTHENTICITY of execution. It takes one symbolic parameter. It is possible to set authentic elapsed timing (see §2.4), with the `AUTHENTIC_TIME_MODE` parameter; in `MODERN_TIMES_MODE` emulation proceeds at the full speed of the host computer.

B *start* **[** *end* **]**

This flag has either one or two parameters, which are instruction-word addresses. It sets a BREAKPOINT on every instruction word in the given range of addresses.

C *l h*

This flag is used to set two COUNT values, say *l* and *h*, that determine when tracing is done. No breakpoint or watchpoint fires, and no instruction is traced, unless $l \le i \le h$ is satisfied; where *i* is the current value of `ICR`. With suitable *l* and *h* values, tracing can confined to a set time during execution (for example, the last few instruction executions before a program fails). The values *l* and *h* are given as unsigned decimal integers.

D FAST_MODE | TRACE_MODE | PAUSE_MODE | EXTERNAL_MODE

This flag sets the DIAGNOSTIC mode, specifying the type of tracing and the kind of logging that may be generated.

F*i start end* and **I***i start end*

These flags have two parameters, which are word addresses. They request that the contents of that range of addresses be output in a specified interpretation, INITIALLY, or FINALLY (i.e. after the end of the run).
For both `I`*i* and `F`*i*, the interpretation is given by the string of letters *i*, each letter of which must be one of: **A**, for strings in ASCII/LATIN-1 code; **C**, for strings in card code; **L**, for strings in LINEPRINTER code; **N**, for strings in paper tape code, with case NORMAL shown; **S**, for strings in paper tape code, with case SHIFT shown; **T**, for strings in paper TAPE code, with shift characters actioned; **O**, for syllabic OCTAL/ ORDERS; **U**, for orders in pseudo-USERCODE format; and **W**, for data WORDS in octal, syllabic octal, line printer characters, Q store format, and signed decimal.

When **U** is specified, **D** can also be given to display machine code addresses in DECIMAL instead of octal. For an example of pseudo-Usercode format, see Appendix 6.

The `PIC`, `PID`, `POC` and `POD` instructions for cards and paper tape permit the processing of data in arbitrary character codes. The **A** format for core-store printing is provided to facilitate the debugging of modern KDF9 programs that process data in ASCII/Latin-1, the native character set of **ee9**.

G [*colour* **[** *tip size* **]]**

This flag allows one or two optional symbolic parameters. The first, if given, sets the GRAPH PLOTTING pen colour from the list: `BLACK` (the default), `BLUE`, `BROWN`, `CYAN`, `DARK_BLUE`, `DARK_CYAN`, `DARK_GREEN`, `DARK_GREY`, `DARK_MAGENTA`, `DARK_RED`, `GREEN`, `GREY`, `MAGENTA`, `RED`, `WHITE`, `YELLOW`. If a colour is given, a second parameter may be given to set the pen tip size from the list: `EXTRA_EXTRA_FINE` (the default, 1 plotter step wide), `EXTRA_FINE` (2 steps wide), `FINE` (4 steps), `MEDIUM` (6), `MEDIUM_BROAD` (8), `BROAD` (10), `EXTRA_BROAD` (12).

In any case, the shared buffer is switched from `TP1` to `GP0`.

H*h cutoff***%**

In tracing mode, **ee9** collects statistics on the execution of the KDF9 program. These may be displayed at the end of a run in the form of **h**istograms. Two plots are available. If *h* contains `P` or `p`, an execution-frequency **p**rofile is output. If *h* contains `T` or `t`, then an instruction **t**ype frequency histogram is output. Both may be requested. *cutoff* is a number in the range 0.0 to 100.0, taken to be a percentage; the **%** is required to follow the number immediately. Histogram bins that account for less than that fraction of all executed instructions are suppressed. Nothing is output if the option is suppressed by the miscellany parameter.

K { *buffer code* }

This flag sets the **K**DF9 configuration by specifying the device types connected to the buffers. The parameters consist of one or more pairs of items, the first a *buffer* number (in the range 0 to 15) and the second a two-letter *code*, being the first two letters of the device name (e.g. `CR` or `LP`). `GP` may be specified to switch TP1's buffer to GP0. At most 1 `DR` unit, 1 `FD` unit, 2 `TP`, `TR`, `CP`, `CR`, `LP`, or `SI` units, at most 14 tape decks of either kind, and only one of `DR` and `FD` can be configured. To leave a buffer with an absent device, specify the code `AD`; any attempt to use that buffer will fail. Every configuration must have `FW` on buffer 0 and `TR` on buffer 1. Any buffer not listed keeps its default device, which is unchanged by specifying:

```
K 0 FW 1 TR 2 TR 3 TP 4 TP 5 LP 6 CR 7 CP 8 MT 9 MT 10 MT 11 MT 12 MT 13 MT 14 FD 15 ST
```

The **K** flag is actioned only at the start of a run; any **K** options in the `settings_2.txt` file are ignored. N.B. The commands described in §1 may not work properly with a non-default configuration, especially if only one TR and no TP or LP is included.

L [*t* **]**

This flag gives a value, *t*, that sets an execution time LIMIT on how long the KDF9 program is allowed to execute before being terminated. The limit is specified in instruction executions rather than seconds, so the program is terminated if `ICR` > *t* at the end of any instruction execution. The value *t* is given as an unsigned decimal integer. If the **L** flag is given without a parameter the time limit is taken to be the default time limit for non-interactive mode (see §6).

N [*t*]

The **N** flag has one optional parameter, with the same meaning at the *t* parameter of the **L** flag. It makes **ee9** run in NON-INTERACTIVE mode, suitable for invocation from a command script. In this mode it is not possible to supply responses to prompts, whether from the KDF9 program or from **ee9** itself; so if an interactive input is requested in non-interactive mode, **ee9** terminates with a suitable diagnostic message. If the **N** flag is given without a parameter, or on the command line, the time limit is taken to be the default time limit for non-interactive mode (see §6).

O

The **O** flag **o**utputs the diagnostics (post-run dumps, histograms and traces) requested in `settings_1.txt` when a program terminates and nominates an **o**verlaid successor by means of `OUT 1` or `OUT 2`.

P *address value*

This flag is used to **P**OKE the *value* (given in octal or decimal) into the *address*. The *address* takes the form of an octal or decimal number, immediately followed by one of the letters **W**, **U**, **L**, **S** or **C**. If **W** is specified, the *value* overwrites the whole **w**ord at the *address*. If **U** or **L** is given, the *value* overwrites the specified halfword. If **S** or **C** is given, it must be followed by the position within the word of the **s**yllable or **c**haracter, respectively, to be overwritten. If a part-word is specified, the *value* must be within its storable range.

Q

The **Q** flag outputs any specified pre-run dumps and then **q**uits. It is useful for getting a dis-assembly of a program without running it.

R *a b*

This flag is used to set two addresses, say *a* and *b*, that delimit the **R**ANGE of instructions where tracing is done. No breakpoint or watchpoint fires, and no instruction is traced, unless $a \le i \le b$ is satisfied; where *i* is the address of the word containing the instruction to be executed. With suitable *a* and *b* values, instruction tracing can confined to the sequence of instructions that you are currently debugging.

S *start* [*end*]

This flag has either one or two parameters, which are data-word addresses. It sets a **S**TOREPOINT on every word in the given range of addresses.

T BOOT_MODE | PROGRAM_MODE | TEST_PROGRAM_MODE

This flag is used to set the **T**EST mode, specifying the kind of run.

V ABDEFHIKMPRSTWZ}

The **V** flag is used to set the **V**ISIBILITY of diagnostic output. It is immediately followed by a selection of the letters: **A** to suppress Director **A**PI messages, **E** to suppress confirmatory or warning messages, but not error messages, from **ee9**, **F** to suppress the **F**INAL STATE of the KDF9 at the end of a run, **H** for the **H**ISTOGRAMS, **I** for the **I**NTERRUPT trace, **P** for the **P**ERIPHERAL I/O trace, **R** for the **R**ETRO trace, **S** for the digital **S**IGNATURE, **W** for console Flexowriter output, and **Z** to combine the effects of all the output-suppression options.

A trace is output if it is provided by the requested diagnostic mode, **and** its output is not suppressed.

The default is that all traces provided by the diagnostic mode are to be output, i.e. **not** suppressed.

The option **B** installs the **B**SI interface, SI0, on buffer 15/#17.

The option **D** enables the output of any optional **D**EBUGGING output.

The option **K** installs the drum, DR0, on buffer 14/#16.

The option **M** **disables** the use of ANSI SGR escape codes for **M**icrosoft Windows.

The option **T** enables execution with authentic **T**IMING (see §2.4).

W *start* [*end*]

This flag has either one or two parameters, which are data-word addresses. It sets a WATCHPOINT (i.e., a **F**ETCHPOINT **and** a STOREPOINT) on every word in the given range of addresses. Note that it is not possible to set a **F**ETCHPOINT only.

X

The **X** flag over-rides other diagnostic options and runs the program in external tracing mode with only signature generation enabled.

/ | -

A line beginning with **/** or with **-** is taken to be comment and ignored.

5.2: THE MISCELLANY PARAMETER

The options permitted with the miscellany parameter on the command line are as follows:

```
abdefghikmnopqrstwxz.0123456789-
```

The miscellany parameter is scanned and put into effect from left to right; it is case-insensitive.

The letters `gnoqx` correspond to the option flags of the same name, with the defaults stated in §5.1 for their parameters.

The letters `abdefhikmprstwz` correspond to the same options given with the **V** flag, as defined in §5.1.

A digit *d* requests an execution time limit of (*d*+1)×100_000_000 instructions; '**.**' specifies a limit of 1_000_000.

A hyphen is ignored.

6: Implementation Characteristics and Caveats

Characteristics

The defaults for the settable options in the present implementation of **ee9** are as follows:

- the default run mode is `PROGRAM_MODE`
- the default diagnostic mode is `FAST_MODE`
- the default diagnostic visibility generates all traces, the digital signature, and the histogram
- the default is to run interactively; that is, with non-interactive mode disabled
- the default elapsed time mode is `MODERN_TIMES_MODE`
- the default time limit allows for effectively unlimited execution
- the default time limit in non-interactive mode is 100 million instruction executions
- the default count bounds, *l* and *h*, are 0 and the time limit, respectively
- the default range bounds, *a* and *b*, are #0 and #7777, respectively
- no breakpoint, watchpoint, storepoint, or core dump is pre-set
- the shared buffer is switched by default to TP1, not GP0
- the default graph plotter pen colour is `BLACK`
- the default graph plotter pen tip is `EXTRA_EXTRA_FINE` (1 plotter step wide)

Caveats

Do not be discouraged by the length of this list; **ee9** successfully runs all distributed KDF9 programs, whether surviving, resurrected or newly-written.

- TSD OUTs 15 and 46 are not yet implemented.
- There are some bogus OUTs, with numbers distinct from those of any historically valid OUT, that are implemented by **ee9** in program mode for various reasons of convenience. They may be withdrawn in future releases.
- KDF9's nest-depth checking caused a NOUV interrupt after the maximum or minimum depth had been transgressed. Presently, **ee9** checks for all of these violations before the offending instruction is executed. This makes little difference in practice. KDF9 had 'imprecise' interrupts, which made recovery from a NOUV error impossible: Director could do no more than terminate (or perhaps restart) the offending program. The behaviour of **ee9** makes debugging a bit easier.
- There is some doubt as to the semantics of the various division instructions, particularly with respect to rounding, and their results on overflow and on division by zero (other than setting the overflow bit).
- There is some doubt as to the semantics of the FLOAT and FLOATD instructions when the scale factor operand in N1 is outside the range $-128 \le N1 \le +127$ specified by the Manual. EE issued a library subroutine, L77, to deal with such cases, so it seems that the behaviour of the hardware with out of range operands was not entirely satisfactory. **ee9** takes a safety first approach and uses the least significant 8 bits of N1 as the effective scale factor, ignoring the rest of N1. This is compatible with the programming of the `sqrt` function in the Kidsgrove Algol system.
- The FRB and TOB orders deliver results that differ from the KDF9 hardware in unusual cases involving radix values greater than 31, which are stated to be illegal in the Manual, and to malfunction in the hardware documentation.
- The all-zero syllable is documented as an invalid order, but evidence has come to light that the hardware actually treated it as a "no-op", like DUMMY, so **ee9** now does the same, identifying it in traces as DUMMY0.
- The Standard Interface (SI) device is present only as a 'best guess' implementation, pending more information about it.
- The POE and POF orders, for TP and SI, produce NULs in lieu of paper tape runout; this is suppressed in Latin-1 mode.
- It is assumed that the POF order for TP and SI is the same as the POE order, but that the count is multiplied by 8.
- The POK and POL orders are fully implemented for the CP and FD devices.
- The PMK and PML orders are fully implemented for 7-track Ampex tape decks.
- There is considerable doubt as to the effects of the PMG, PMK, PML, POK, and POL orders for all other devices.
- The CT order lets an ongoing transfer terminate normally, as if it were MANUAL; the hardware forced an immediate end.
- PIC and PID transfers for FW, TR, TP and SI0 devices copy each input byte *verbatim* to the least significant 8 bits of a word; POC and POD for those devices copy the least significant 8 bits of each word *verbatim* to a byte of output.
- PID and POD transfers for FW, TR, TP and SI0 devices terminate after a word that has the 8-bit paper tape code for an EM, decimal 125, #175, in its least significant 8 bits.
- The POC and POD orders switch FW from writing to reading after a word that has the 8-bit paper tape code for a semicolon, decimal 60, #74, in its least significant 8 bits.
- **ee9** cannot properly implement the semantics of 5-hole paper tape. See Appendix 2C for more details.
- It is assumed that the graph plotter pen tip sizes and colours are the same as those of pens presently on sale.
- It is assumed that the plotter command codes listed in Appendix 5, §3, p.304 of the Manual are wrong, as there is an error whereby 11 plotting codes are claimed but only 9 are given. The codes used by **ee9** are given in Appendix 3 of this *Guide*. These have been confirmed by evidence from other computers of the era that had the same type of plotter.
- It is assumed that the POB and POD orders for the graph plotter have the same effect as POA and POC, respectively. The Manual says that the plotter responds to EM with a 'peculiar' pen movement, but does not end the transfer, and that

invalid codes have 'unpredictable' effects. **ee9** in these cases implements an arbitrary, but safe and potentially useful, operation: it moves the plotting position to the origin of the plotter co-ordinate system.

- It is assumed that the fixed-head area of a disc drive is platter 16, seek area 0. The disc storage supported by **ee9** includes 4 such drives.
- It is assumed that a drum has 40 tracks of 8 sectors. The drum storage supported by **ee9** includes 4 such drum units.
- Many other hypotheses have been made in emulating discs and drums; it remains to be seen whether they are justified.
- The emulation of OUT 8 by **ee9** has some differences from the authentic behaviour of TSD; see Appendix 5.
- When a program running directly uses OUT 8 to write a prompt to FW0 (a '**q**uery' in KDF9 parlance, including a '**;**' that turns the write into a read), **ee9** precedes the given data with '`[q] `'; but if the transfer is a simple output **m**essage **ee9** precedes it with '`[m] `'. This imitates the behaviour of the **non**-Time Sharing ('standard') Director. Also, such a query always terminates on reading EM, but Directors insist that the EM be preceded by a full stop.
- I/O lockouts in **ee9** remain in force for the whole duration of a transfer and are cleared *in toto* at the end; KDF9 cleared the lockout of each 32-word group as soon as the transfer crossed the boundary into the next group.
- There remain some issues related to the execution of problem programs under TSD control.

  (a) The CPU time reported by TSD may differ by about 1% from that reported by running directly under **ee9**.

  (b) TSD reports ludicrous execution times in the order of 800 minutes for some programs.

  (c) The coordination of a FW0 transfer initiated directly by a problem program, with TSD's typing-out of its own messages, is not entirely satisfactory. Note that programmers were strongly discouraged from direct output to FW0.
- Failures due to programming error can occur during a run. Some were required by the KDF9 architecture to cause a LIV interrupt in problem program state, and **ee9** implements that. LIV interrupts are inhibited in Director state, but **ee9** does not allow this, as it might make continued emulation fail unpredictably. Other failures were not detected by the KDF9 hardware, but are apparent to **ee9**. Impossible I/O operations in Director state end the run, because this indicates a serious malfunction. When running a program in boot mode they set the device abnormal and abandon the order; the onus is on the program to detect this and act accordingly—if it does not it may subsequently LIV. Failures when running a program directly end the run with a specific diagnostic. See Appendix 8 for a list of the various kinds of error message.

Acknowledgements

I am grateful to the group of supporters, all enthusiastic former KDF9 engineers, programmers, or satisfied users, for their encouragement during this project; and for their superb work in recreating a software ecosystem for **ee9**. I thank in particular David Hawley and Brian Randell, for their crucial caches of EE documents; David Holdsworth for his Usercode compilers and hardware insight; David Holdsworth, David Huxtable, Brian Wichmann, Graham Toal, and Roderick McLeod for resurrecting the Whetstone and Kidsgrove Algol systems, so that **ee9** has something really substantial to run; and David Holdsworth, Mike Hore, and Bill Gallagher for compiling and testing **ee9** ports. Others, whose names I have inexcusably forgotten, know who they are: to them also, my thanks. The plotter command file `GPT_data.txt` I code-converted to KDF9 plotting code from an ICL 1900 Series plotter test file made available by Bill Gallagher.

References and Further Reading

Included with distributions of **ee9**:

- *The English Electric KDF9*, *The Hardware of the KDF9*, *The Software of the KDF9*, *The KDF9 and Benchmarking*, *Buying a KDF9 in 1964* and *The KDF9: a Bibliography*
- *ee9 Users' Guide*, *ee9 Implementation Overview* and *ee9 Release History*
- *Kidsgrove Code Procedures*, *Kidsgrove Compilation Errors*, *Kidsgrove Compilation Limits*, *Kidsgrove Compiler Options*, *Kidsgrove IO Libraries*, *Kidsgrove Optional Output* and *Kidsgrove Runtime Errors*
- *README*
- *Whetstone Compilation Errors*, *Whetstone Controller-listing*, *Whetstone Runtime Errors* and *Whetstone Translator-listing*

Available at: www.findlayw.plus.com/KDF9:

- *KDF9 Programming Manual*, Publication 1002 mm ® 1001263; International Computers Ltd.; 'the Manual'
- *KDF9 ALGOL programming*, Publication 1003 mm ® 1000565; English Electric–Leo Computers Ltd.

See also

www.findlayw.plus.com/KDF9/#Emulator

which is updated occasionally with **ee9** news.

APPENDIX 1: ee9 COMMAND FUNDAMENTALS

The emulator is invoked from the command line, thus:

ee9 { **-s***s* | **-d***d* | **-m**[*m*] | **-TP***tt* | **-TR***tt* } **+***program_file_name*

where the parameters introduced by '**-**' are optional and can be given in any number or order; *m* is a short string that specifies a **m**iscellany of options; *d* specifies a **d**iagnostic execution mode; ***s*** is the initial CPU **s**tate for the KDF9 run; and *tt* specifies paper **t**ape device character codes. Note that *program_file_name* is a path name; unlike the *prog* parameter of the **nine** shell command it is not by default made relative to `Testing/Binary`.

The allowable state flag characters *s* are:

- **b**: for **b**ooting into Director state, which is how operating systems such as the TSD are loaded and run
- **p**: for **p**roblem program state, the default, allowing user programs to be run without a Director
- **t**: for **t**est program state, allowing programs to be run with OUTs serviced as in problem program state, but with the CPU actually in Director state; though inauthentic, this is useful for running 'hardware test' programs.

The allowable diagnostic flag characters *d* are:

- **f**: for **f**ast mode, the default
- **p**: for **p**ause mode
- **t**: for **t**race mode
- **x:** for e**x**ternal trace mode

The allowable characters in the string *m* (miscellany parameters) are described in §5.2.

The parameters *tt* specify the character code to be used by either one or two paper tape punches (with `-TP`), or paper tape readers (with `-TR`). *tt* is either one or two characters, each being either `k`/`K` or `l`/`L`, specifying KDF9 paper tape code or Latin-1 code respectively. If *tt* is one character, it refers to unit 0 of the type (i.e. `TP0` or `TR0`). If *tt* is two characters, the first refers to unit 0 and the second to unit 1 of the type. See also §3.4 and Appendix 2.

The current working directory in which **ee9** is invoked must contain files to represent the KDF9's I/O devices. See §3.

Invoking **ee9** is not trivial—particularly when running the TSD or the Algol systems, which require a significant amount of preparation—so there are shell commands to do the donkey work for you. These are described in §1.

Example

```
cp Whetstone/GPS.a60 TR1; ee9 –sp –df –mn +Binary/KMW0201–UPU    # run Whetstone Algol
whet GPS                                                        # do it the easy way
```

The **ee9** distribution contains a hierarchy of directories to locate its files systematically and conveniently for the casual user. `Adjuncts`, `Testing`, and the `Assembly`, `Binary`, `Data`, `Kidsgrove` and `Whetstone` directories within `Testing` contain materials for KDF9 programs to work with. The `Adjuncts` directory contains non-**ee9** components required to build and run the Kidsgrove Algol compiler. However, these locations are defaults only, and others can be substituted if that better meets the user's needs. To specify alternative places shell environment variables maybe used, as follows.

environment variable name	default path
ADJUNCTS	../Adjuncts
DIRECTOR	Binary/KKT40E007UPU
KDF9_USERCODE	Assembly
KDF9_BINARY	Binary
KDF9_DATA	Data
KIDSGROVE	Kidsgrove
WHETSTONE	Whetstone

If a KDF9 program overlays itself by means of OUT 1 (see Appendix 4), the executable is taken from the directory specified by `KDF9_BINARY`. If a program interactively requests additional data and the user gives the '`@`' response (see §3.3), the file is looked for in the directory specified by `KDF9_DATA`. The default path in each case is used if the environment variable is unset (either undefined or containing the empty string).

The `DIRECTOR` variable selects the version of TSD used by **tsdnine**.

The defaults for unset variables conform to the distributed structure when the current working directory is `Testing`, where executable binaries and shell files are normally held. Another directory can be used in place of `Testing`, but for full functionality it must contain all the same non-directory files as `Testing` and should be ahead of it in the user's search path.

Example

```
export KDF9_USERCODE=../../usercode
export KDF9_BINARY=../../bin
kids ZELLER
…
Running the KDF9 problem program ../../bin/KAB00DH--USU in fast mode.
…
The Usercode is in ../../usercode/ZELLER.k3 and the KDF9 executable is in ../../bin/ZELLER
…
Running the KDF9 problem program ../../bin/ZELLER in fast mode.
```

APPENDIX 2A: KDF9 INTERNAL CHARACTER CODES AND THEIR TRANSCRIPTIONS

Line printer	SP		LF	FF			%	'
Punched Cards	SP	ISO:″	ISO:®	ISO:©	ISO:¬	ISO:#	%	'
Normal Case	SP	ISO:″	LF	FF	HT	ISO:#	CS ISO:ß	CN ISO:ñ
Shift Case	SP	ISO:″	LF	FF	HT	ISO:#	CS ISO:ß	CN ISO:ñ
Octal code	00	01	02	03	04	05	06	07

Line printer	:	=	(	)	£	*	,	/
Punched Cards	:	=	(	)	£	*	,	/
Normal Case	ISO:&	ISO:?	ISO:!	ISO:%	ISO: '	ISO:$	ISO:~	/
Shift Case	ISO:&	ISO:?	ISO:!	ISO:%	ISO: '	ISO:$	ISO:~	:
Octal code	10	11	12	13	14	15	16	17

Line printer	0	1	2	3	4	5	6	7
Punched Cards	0	1	2	3	4	5	6	7
Normal Case	0	1	2	3	4	5	6	7
Shift Case	↑ ISO:^	[	]	<	>	=	×	÷
Octal code	20	21	22	23	24	25	26	27

Line printer	8	9		₁₀ ISO:¤	;	+	-	.
Punched Cards	8	9	_ ISO: _	₁₀ ISO:¤	;	+	-	.
Normal Case	8	9	_ ISO: _	₁₀ ISO:¤	;	+	-	.
Shift Case	(	)	_ ISO: _	£	;	≠ ISO:±	*	,
Octal code	30	31	32	33	34	35	36	37

Line printer		A	B	C	D	E	F	G
Punched Cards	ISO:@	A	B	C	D	E	F	G
Normal Case	ISO:@	A	B	C	D	E	F	G
Shift Case	ISO:@	a	b	c	d	e	f	g
Octal code	40	41	42	43	44	45	46	47

Line printer	H	I	J	K	L	M	N	O
Punched Cards	H	I	J	K	L	M	N	O
Normal Case	H	I	J	K	L	M	N	O
Shift Case	h	i	j	k	l	m	n	o
Octal code	50	51	52	53	54	55	56	57

Line printer	P	Q	R	S	T	U	V	W
Punched Cards	P	Q	R	S	T	U	V	W
Normal Case	P	Q	R	S	T	U	V	W
Shift Case	p	q	r	s	t	u	v	w
Octal code	60	61	62	63	64	65	66	67

Line printer	X	Y	Z					
Punched Cards	X	Y	Z	ISO:{	ISO:}	→ ISO:|	ISO:\	Ø
Normal Case	X	Y	Z	ISO:{	ISO:}	→ ISO:|	ISO:\	see notes
Shift Case	x	y	z	ISO:{	ISO:}	→ ISO:|	ISO:\	see notes
Octal code	70	71	72	73	74	75	76	77

NOTES

The transcription provides a Latin-1 representation for every KDF9 internal character code:

- Code 77 is represented by Ø on punched card devices and fast storage devices invariably, on two-shift devices for 'character mode' transfers only, and is otherwise suppressed on both input and output.
- An empty cell indicates a code that is completely suppressed by the line printer.
- SP is a blank space.
- LF is Line Feed, Line Shift (LS) in KDF9 terminology; it prints as ® in NEST displays, *etc*.
- FF is Form Feed, Page Change (PC) in KDF9 terminology; it prints as © in NEST displays, *etc*.
- HT is Horizontal Tab; it prints as ¬ in NEST displays, *etc*.
- CS is Case Shift.
- CN is Case Normal.
- 'y ISO:x' indicates that x is the ISO Latin-1 transcription of the non-Latin-1 KDF9 character y.
- 'ISO:x' indicates that x is the ISO Latin-1 external representation of a non-legible KDF9 character.
- Fast storage devices (e.g. magnetic tapes) always use the Normal Case representation.
- Except for 'character mode' output, ß and ñ are acted upon by the Flexowriter, not transferred literally, so that output is presented in the correct case.

APPENDIX 2B: KDF9 PAPER TAPE CODES (WITH PARITY CHANNELS) AND THEIR TRANSCRIPTIONS

Line printer	SP		LF	FF			%	'
Punched Cards	SP	ISO:″	ISO:®	ISO:©	ISO:¬	ISO:#	%	'
Normal Case	SP	ISO:″	LF	FF	HT	ISO:#	CS ISO:ß	CN ISO:ñ
Shift Case	SP	ISO:″	LF	FF	HT	ISO:#	CS ISO:ß	CN ISO:ñ
Paper tape	220	021	022	003	024	005	006	027

Line printer	:	=	(	)	£	*	,	/
Punched Cards	:	=	(	)	£	*	,	/
Normal Case	ISO:&	ISO:?	ISO:!	ISO:%	ISO: '	ISO:$	ISO:~	/
Shift Case	ISO:&	ISO:?	ISO:!	ISO:%	ISO: '	ISO:$	ISO:~	:
Paper tape	030	011	012	033	014	035	036	017

Line printer	0	1	2	3	4	5	6	7
Punched Cards	0	1	2	3	4	5	6	7
Normal Case	0	1	2	3	4	5	6	7
Shift Case	↑ ISO:^	[	]	<	>	=	×	÷
Paper tape	060	041	042	063	044	065	066	047

Line printer	8	9		₁₀ ISO:¤	;	+	-	.
Punched Cards	8	9	_ ISO: _	₁₀ ISO:¤	;	+	-	.
Normal Case	8	9	_ ISO: _	₁₀ ISO:¤	;	+	-	.
Shift Case	(	)	_ ISO: _	£	;	≠ ISO:±	*	,
Paper tape	050	071	072	053	074	055	056	077

Line printer		A	B	C	D	E	F	G
Punched Cards	ISO:@	A	B	C	D	E	F	G
Normal Case	ISO:@	A	B	C	D	E	F	G
Shift Case	ISO:@	a	b	c	d	e	f	g
Paper tape	120	101	102	123	104	125	126	107

Line printer	H	I	J	K	L	M	N	O
Punched Cards	H	I	J	K	L	M	N	O
Normal Case	H	I	J	K	L	M	N	O
Shift Case	h	i	j	k	l	m	n	o
Paper tape	110	131	132	113	134	115	116	137

Line printer	P	Q	R	S	T	U	V	W
Punched Cards	P	Q	R	S	T	U	V	W
Normal Case	P	Q	R	S	T	U	V	W
Shift Case	p	q	r	s	t	u	v	w
Paper tape	140	161	162	143	164	145	146	167

Line printer	X	Y	Z					
Punched Cards	X	Y	Z	ISO:{	ISO:}	→ ISO: |	ISO:\	Ø
Normal Case	X	Y	Z	ISO:{	ISO:}	→ ISO: |	ISO:\	see note
Shift Case	x	y	z	ISO:{	ISO:}	→ ISO: |	ISO:\	see note
Paper tape	170	151	152	173	154	175	176	157

NOTE

- Internal code 77 is represented by Ø on punched card devices and fast storage devices invariably, on two-shift devices for 'character' mode transfers only, and is otherwise suppressed on both input and output.
- 'Character' mode transfers on two-shift devices use the full 8-bit paper tape code and transfers to End Message terminate after the 8-bit internal code #175, which is the Latin-1 code for '}', not '|'.

APPENDIX 2C: KDF9 5-HOLE (FERRANTI) PAPER TAPE CODES

Normal Case	ISO:@	A	B	C	D	E	F	G
KDF9 code	40	41	42	43	44	45	46	47
Letters Case	*fs*	A	B	C	D	E	F	G
Figures Case	*fs*	1	2	*	4	(	)	7
Ferranti code	00	01	02	03	04	05	06	07

Normal Case	H	I	J	K	L	M	N	O
KDF9 code	50	51	52	53	54	55	56	57
Letters Case	H	I	J	K	L	M	N	O
Figures Case	8	≠	=	-	*v*	*lf*	*sp*	,
Ferranti code	10	11	12	13	14	15	16	17

Normal Case	P	Q	R	S	T	U	V	W
KDF9 code	60	61	62	63	64	65	66	67
Letters Case	P	Q	R	S	T	U	V	W
Figures Case	0	>	≥	3	→	5	6	/
Ferranti code	20	21	22	23	24	25	26	27

Normal Case	X	Y	Z	ISO:{	ISO:}	→ ISO:|	ISO:\	see notes
KDF9 code	70	71	72	73	74	75	76	77
Letters Case	X	Y	Z	*ls*	.	?	£	*er*
Figures Case	×	9	+	*ls*	.	*n*	*cr*	*er*
Ferranti code	30	31	32	33	34	35	36	37

NOTES

The handling of 5-hole paper tape—in the teleprinter code of the Ferranti Pegasus, Sirius and Orion computers—is rather non-obvious. When a 5-hole tape was read by KDF9 the most significant bit of each input character was set to 1 by the hardware, adding #40 to the physical Ferranti code. On output to a 5-hole tape punch the hardware discarded that bit. This provided a 1-to-1 mapping and had the advantage that upper case letters got the same codes from 5-hole and 8-hole input tapes, namely the KDF9 native codes for those letters. **ee9** cannot implement this behaviour for direct transfers, because it does not always know whether a paper tape device is in 5-hole mode or 8-hole mode (for example, the type of the device attached to a buffer can be changed dynamically by the operators) and it has no way to find out. Consistency demands that transfers that are notionally from 5-hole tape readers and to 5-hole tape punches work in exactly the same way as for 8-hole devices in **ee9**.

Things are different when a program running under Director control writes to a 5-hole punch via OUT 8 streams 50-57. Most KDF9 characters with codes in the ranges #00-#37, and the letters A-Z, are output verbatim. Others are given special treatment. Director converts them in an attempt to preserve the appearance of a teleprinter transcription, inserting shift characters where necessary. Director also converts KDF9's LS character into a *fs cr lf* sequence. Clearly, this has the potential for muddle if the KDF9 data includes characters that are absent from the Ferranti teleprinter set.

The **mtp** and **a2b** utilities (see Appendices 10 and 11) have facilities for converting from Ferranti code to Latin-1. They assume that the Ferranti codes are those generated by OUT 8:

- Ferranti code 00, denoted *fs*, is the Figures Shift character. It switches printing to the Figures Case character set.
- Ferranti code 33, denoted *ls*, is the Letters Shift character. It switches printing to the Letters Case character set.
- Ferranti code 14 is the *v* character in Figures Case, but L in Letters Case. The *v* was treated by English Electric software as if it were the KDF9 $_{10}$ character. It is printed as '`v`' by **mtp** and **a2b**.
- Ferranti code 15, denoted *lf*, is the Line Feed character in Figures Case but M in Letters Case.
- Ferranti code 16, denoted *sp*, is the blank space character in Figures Case but N in Letters Case.
- Ferranti code 22 is the ≥ character in Figures Case, but R in Letters Case. It is printed as '`]`' by **mtp** and **a2b**.
- Ferranti code 24 is a right-arrow character in Figures Case, but T in Letters Case. It looks like, but is not, a KDF9 EM. It is printed as '»' by **mtp** and **a2b**.
- Ferranti code 35 is the *n* character in Figures Case, but ? in Letters Case. This Ferranti code works like EM for 5-track data, since it generated the KDF9 EM internal code (#75) when read by the hardware. In Figures Case it was treated by English Electric software as if it were an underlined EM. It is printed as '`n`' by **mtp** and **a2b**.
- The KDF9 LS character has the same CR/LF effect as the ASCII Newline, but in Ferranti code there are separate CR and LF characters, available only in Figures Case.
- Ferranti code 36, denoted *cr* above, is the Carriage Return character in Figures Case but £ in Letters Case.
- Ferranti code 37 was used to *er*ase paper tape characters by punching holes in all five positions of the tape frame. It is suppressed on output by **mtp** and **a2b**, as #77 is for devices working with KDF9 codes.

APPENDIX 3: KDF9 GRAPH PLOTTER CODES

Action	Decimal	Octal	Binary
None	0	#0	000000
Step paper back	1	#1	00000**1**
Step paper forward	2	#2	00001**0**
Step pen right	4	#4	00010**0**
Step pen right, paper back	5	#5	00010**1**
Step pen right, paper forward	6	#6	00011**0**
Step pen left	8	#10	00**1**000
Step pen left, paper back	9	#11	00**1**00**1**
Step pen left, paper forward	10	#12	00**1**01**0**
Lower pen	16	#20	0**1**0000
Raise pen	32	#40	**1**00000

NOTES

• The command list in Appendix 6, §5, p.302 of the Manual has an obvious error whereby 11 plotting codes are claimed but only 9 are given, the last of them being inconsistent with the others. The present list provides the two missing codes and corrects the incorrect code for a step right and backwards. The codes listed here have been confirmed by the evidence of other computers of the era that used the same plotter; the bit positions and their interpretations correspond directly to the control bits in the Calcomp plotter's hardware interface. All other 6-bit character codes represent invalid plotter commands.

APPENDIX 4: TIME SHARING DIRECTOR OUTS (SYSTEM CALLS)

Director provides an API that is accessed by the OUT instruction with parameters in the NEST. See Table A.4.1.

TABLE A4.1: DIRECTOR OUT PARAMETERS

N1	N2	N3	Action
0†			Terminate this run normally.
1†	Program ...	… name	Terminate this run and overlay the program whose name is in N2-3.
2†	Time limit in seconds		Restart this program. Allow it the new time limit given in N2.
3†			Return the CPU time used in N1. The result is given as seconds to 23 integral places.
4‡	Tape label		Allocate the tape deck with the reel having the 1-word label in N2. Return its buffer number in N1.
5‡	Device type code		Allocate an I/O device of the type given in N2; see Table A4.2. Return its buffer number in N1.
6‡	Buffer number		Deallocate the I/O device with buffer number in N1. If it is a tape deck, unload any mounted tape reel. (Tape decks are treated the the same by OUT 6 and OUT 7 in **ee9**.)
7‡	Buffer number		Deallocate an allocated tape deck with buffer number in N1. Do not unload any mounted tape reel.
8	Control word		Spool output or write to FW0. See Appendix 5.
9,			Return the time of day in N1. The result is given as seconds since midnight to 23 integral places.
10 ‡	Tape ...	… label	Allocate the tape deck with the reel having the 2-word label in N2-3. Return its buffer number in N1 and the 'Tape Serial Number' (TSN), as read from the label block, in N2.
11 *	Control word		Write core to allocated sectors of drum storage.
12 *	Control word		Read allocated sectors of drum storage into core.
13 *	Number of sectors		Allocate a number of drum sectors for exclusive use by this program run. Can be done once only per run.
14 *			Return the number of drum sectors available for allocation in N1, with the sign bit set if OUT 13 has already been executed.
16 **	Control word		Write to FW0, as if by OUT 8, but with a prefix of '`N␣ `'. See Appendix 5.
17 ‡			Return the CPU Time in N1 and the Notional Elapsed Time in N2. The results are given as seconds to 23 integral places.
41 ††	Control word		Write to sector(s) in the currently selected set of disc platters.
42 ††	Control word		Read from sector(s) in the currently selected set of disc platters.
43 ††	0 or -1		Select the first (0) or the second (-1) set of disc platters for transfers.
44 ††	Number of platters		Allocate a number ≤ 8 of disc platters. Can be done only twice per run.
45 ††	0 or -1		Deallocate all platters in the first (0) or the second (-1) set.
47 ††			Check the last FD transfer: set TR if a parity error was flagged.

NOTES

† See the Manual, §26.2. 1.

- Obeying OUT with an empty NEST has the same effect as OUT 0.
- OUT 1 overlays the current program with one read from external media. Its name consists of 12 Case Normal characters, e.g. "`KMW0301--UPU`".
- OUT 2 resumes the execution of the program with a new time limit, but without its previously allocated devices.

‡ See the Manual, §17.3.

- The label for OUT 4 consists of 8 Case Normal characters. e.g. "`PRINTEND`".
- The label for OUT 10 consists of a plus sign followed by 15 Case Normal characters. e.g. "`+KOUTPUT/0023004`".

* See the Manual, Appendix 6, §4.

- The control word for the drum I/O OUTs is of the form Q *starting sector* / *low core address* / *high core address*.
- The sectors allocated by OUT 13 are numbered from 0.

** OUT 16 writes to the console Flexowriter in a slightly different format from OUT 8.
The control word for OUT 16 takes the same form as for OUT 8, but only the FW0 stream is supported.

†† See the Manual, Appendix 6, §6.

- The control word for the disc I/O OUTs is of the form Q *starting sector* / *low core address* / *high core address*.
- The sectors allocated by OUT 44 are numbered from 0.

TABLE A4.2: DEVICE TYPE CODES

Type	**FW**	**TP**	**TR**	**LP**	**CR**	**TP** §	**CP**	**GP**	**SI**	**PDP-8** *	**MT** †
Decimal	0	1	2	3	4	5	7	16	17	53	55
Octal	#00	#01	#02	#03	#04	#05	#07	#20	#21	#65	#67

NOTES

• Add 8 (#10) to codes 1-7 to reclaim a previously-relinquished device.

§ Type 5 is for a 5-channel tape punch, using Ferranti paper tape code

* The PDP-8 front end for Eldon 2 and COTAN was attached to a magnetic tape buffer

† A magnetic tape deck given type #67 is treated as having an unlabelled tape, which is not to be read

APPENDIX 5: OUT 8

Director implements an output spooling system for paper tape punches and line printers. It also administers access to the console Flexowriter. Up to 32 virtual output streams are provided to each running problem program; see Table A5.1.

Stream output is written first to a designated magnetic tape. When that becomes full, or at the behest of the computer operator, it is rewound and the accumulated data for each stream is transferred from the tape to its associated device. A second magnetic tape may be designated to take output while this is happening, and the two tapes may switch rôles as often as necessary to transfer all the outputs to their intended destinations.

OUT 8 takes a parameter word in N2 that controls the spooling action. It is in Q store format. If the C-, I-, and M-parts are equal, and contain a stream number, that stream is closed. Otherwise the I-part is taken to be the starting address, *a*, of an output area and the M-part as its ending address, *b*.

If $b = a + 1$ and the area is laid out as follows:

- word *a*+0: *stream number*
- word *a*+1: control word in Q store format: 4095 / -1 / *n*

then a device-dependent 'gap' is written to the stream; for a tape punch stream this takes the form of runout (unpunched) tape, of length *n* characters, for *n* / 10 inches; applied to a line printer stream it produces a PC character.

Otherwise, and more usually, the words in address range *a* to *b* inclusive are written to the OUT 8 tape by a transfer-to-End-Message operation, the word at *a* having been overwritten by 'red tape' identifying the stream.

TABLE A5.1: OUT 8 STREAMS IMPLEMENTED BY DIRECTOR

stream number	Output Device
#00	Flexowriter
#10 .. 17	An 8-hole paper tape punch selected by the operator
#20 .. 27	Not used
#30 .. 37	The line printer
#40 .. 47	Not used
#50 .. 57	A 5-hole paper tape punch selected by the operator
#60 .. 67	Not used
#70 .. 77	The line printer **or** a paper tape punch, at the operator's discretion

Special provision is made for queries, prompted responses on the Flexowriter console, which are implemented immediately, not deferred. In this case the parameter word in N2 must have a C-part of #100000. The output area must be laid out with a prompt (ending with a '**;**' character) and be followed by an input area for the response. Various restrictions are imposed to preserve a tidy layout of the console's typed log. In particular, word *a* is overwritten by Director with suitable format effectors. When running directly, an OUT 8 query always terminates on reading EM, but Director also requires the EM to be preceded by a period, thus :'**.** | '.

In its problem program and test program modes, **ee9** emulates OUT 8 not by spooling, but by directly transferring the output to the intended destination device. Note that outputs to multiple streams intended for the same device are therefore interleaved by **ee9**, but are not interleaved by Director. In doing so **ee9** is somewhat more specific than Director in its provision of output streams, as indicated in Table A5.2. Streams 11, 13, 15 and 17 are directed to TP1 instead of TP0, as a small mitigation of the interleaving issue.

The 5-hole mode is not implemented by **ee9**; all paper tape output is in 8-hole mode.

Director requires a line of output to be written in a single OUT 8 call, the line ending in a LS, PC or EM character. **ee9** allows a line to be built up incrementally, in a series of calls to the emulated OUT 8. This behaviour was necessary to support an earlier version of the reconstructed I/O library in the Kidsgrove Algol run time environment.

So far these differences between **ee9** and Director have not been problematic.

TABLE A5.2: OUT 8 STREAMS IMPLEMENTED BY ee9

stream number	Output Device
#00	Flexowriter
#10, 12, 14, 16	TP0
#11, 13, 15, 17	TP1
#30 .. 37	LP0
#50 .. 57	TP1
#70 .. 77	LP0
others	Invalid

The KDF9 Algol I/O procedures work on notional I/O streams, identified by decimal numbers that are made to look like OUT 8 octal stream numbers. For example, output to Algol stream 30 goes to OUT 8 stream #30. Algol input uses the 20 range of stream numbers which OUT 8 does not use.

APPENDIX 6: DISASSEMBLED MACHINE CODE

The **U** format core printing routine analyses the program's control flow, and uses simple heuristics, to determine whether a given word represents data or instructions. These do a good job, but cannot always be correct. Words thought to be data are output in a variety of formats. Instructions are shown with octal or decimal operand addresses, at option. If the operand of a SET instruction is greater than 7, its octal representation is complemented by its decimal representation as a comment; and *vice versa*.

An instruction that is the target of a jump starts a new line labelled by its address. An address is also given for the instruction sequentially following a subroutine jump (JS…), as that is the link value stored in the SJNS, and may be useful for following the course of execution.

Here is an example for which the heuristics work well. Lines of the form `V`*n*`=`*value*`;` are local static data declarations; and the executable code begins with '`V14; =V13;`':

```
P51V15;
         V0=F0.019042127887;
         V1=F0.019042129240;
         V2=F0.038082414120;
         V3=F0.076666493927;
         V4=F0.121226383896;
         V5=F0.725940450930;
         V11=Q6/1/0;
         V12=Q4/1/0;
         V14=F1.0;
         V15=F0.5;
         V14; =V13;
         DUP; DUP; ×F; V14; +F; JSP40; =V6;
         V12; =Q13;
2;       V13; V6M13; +F; V15; ×F; =V13;
         V13; V6M13Q; ×F; JSP40; =V6M13; J2C13NZ;
         V11; =Q13;
         V0M13Q; ZERO; REV; FIX; FLOATD;
1;       V0M13; V5M13Q; ×+F; J1C13NZ;
         ROUNDF; ÷F; EXIT1;
```

And here is its **UD**-format output:

```
Core store interpreted as instructions.
1393/0:    #1742337743622052 =   68337217250346 = 1.90421278869E-2
1394/0:    #1742337743651250 =   68337217262248 = 1.90421292400E-2
1395/0:    #1744677611614626 =   68504697379222 = 3.80824141200E-2
...
1409/0:
        E1407; =E1406;
        DUP; DUP; ×F; E1407; +F;
        JSE1263/0;
1411/4:
        =E1399;
        E1405; =Q13;
1413/0:
        E1406; E1399M13; +F; E1408; ×F; =E1406;
        E1406; E1399M13Q; ×F;
        JSE1263/0;
1417/0:
        =E1399M13;
        JE1413/0C13NZ;
        E1404; =Q13; E1393M13Q; ZERO; REV; FIX; FLOATD;
1420/0:
        E1393M13; E1398M13Q; ×+F;
        JE1420/0C13NZ;
        ROUNDF; DIVF;
        EXIT 1;
```

APPENDIX 7: TRACING

These examples give an indication of the types of diagnostic output that can be obtained.

APPENDIX 7A: THE RETROSPECTIVE TRACE

Under ND is given the NEST depth and under SD the depth of the SJNS. Under VTD are the states of the **o**verflow and **t**est registers and whether running in **D**irector mode. For example:

```
Retrospective trace of all instructions.
                                                                ND SD VTD   CPU TIME     ICR
Ended #23/4:    OUT                 0                            3  0 V    1674616069   306451954
After #23/3:    ZERO               #0000000000000000             4  0 V    1674616056   306451953
After #23/0:    OUT                 4                            3  0 V    1674616054   306451952
After #22/3:    SET6               #0000000000000006             5  0 V    1674616041   306451951
After #22/1:    C13                #0000000000000004             4  0 V    1674616029   306451950
After #21/5:    POEQ13             Q #000004/ #000000/ #000454   3  0 V    1674616024   306451949
After #136/5:   EXIT2;(1)          #20/5                         3  0 V    1674616005   306451948
After #136/2:   JE#137/2NE         #0000000000000035             3  1 V    1674615986   306451947
After #135/5:   SET29              #0000000000000035             4  1 V    1674615981   306451946
After #135/2:   JE#133/5LTZ        #0000000000000035             3  1 V    1674615977   306451945
After #135/1:   DUP                #0000000000000035             4  1 V    1674615973   306451944
After #135/0:   -                  #0000000000000035             3  1 V    1674615971   306451943
After #134/3:   SET32              #0000000000000040             4  1 V    1674615970   306451942
After #134/1:   SHLD+6             #0000000000000075             3  1 V    1674615959   306451941
After #134/0:   ZERO               #0000000000000000             3  1 V    1674615956   306451940
After #133/5:   ERASE              #7500000000000000             2  1 V    1674615954   306451939
After #133/4:   DUP                #7500000000000000             3  1 V    1674615953   306451938
After #133/1:   E25472M7Q;(#61600) #7500000000000000             2  1 V    1674615951   306451937
After #132/5:   POBQ14             Q #000004/ #061600/ #061775   1  1 V    1674615944   306451935
After #132/3:   PIBQ15             Q #000002/ #061600/ #061775   1  1 V    1674615912   306451933
After #132/1:   IM15 TO Q14        Q #000004/ #061600/ #061775   1  1 V    1674615880   306451931
After #131/5:   =M15               Q #000002/ #061600/ #061775   1  1 V    1674615876   306451930
After #131/4:   +                  #0000000000061775             2  1 V    1674615866   306451929
After #131/2:   I15                #0000000000061600             3  1 V    1674615865   306451928
After #130/5:   SET125             #0000000000000175             2  1 V    1674615859   306451927
After #130/3:   =RM7               Q #000000/ #000001/ #000000   1  1 V    1674615855   306451926
After #130/2:   ZERO               #0000000000000000             2  1 V    1674615852   306451925
After #130/0:   =RM6               Q #000000/ #000001/ #057165   1  1 V    1674615850   306451924
After #20/5:    JSE#130/0          #20/5                         2  1 V    1674615847   306451923
After #20/2:    SET24181           #0000000000057165             2  0 V    1674615836   306451922
After #52/3:    JE#20/2EQZ         #0000000000000000             1  0 V    1674615832   306451921
After #52/1:    SHL+24             #0000000000000000             2  0 V    1674615821   306451920
After #51/4:    E0;(#0)            #4013001200000000             2  0 V    1674615817   306451919
After #51/3:    ERASE              #0000000000000004             1  0 V    1674615803   306451918
After #51/2:    ERASE              #0000000000000000             2  0 V    1674615802   306451917
After #51/0:    =RM2               Q #000000/ #000001/ #000000   3  0 V    1674615801   306451916
After #50/5:    DUP                #0000000000000000             4  0 V    1674615798   306451915
After #50/4:    DUP                #0000000000000000             3  0 V    1674615796   306451914
After #50/3:    ZERO               #0000000000000000             2  0 V    1674615794   306451913
After #160/1:   EXIT1;(0)          #50/0                         1  0 V    1674615792   306451912
After #157/5:   POAQ14             Q #000004/ #061600/ #061600   1  1 V    1674615779   306451911
```

(etc)

```
After earlier instructions, whose tracing is now lost.
```

APPENDIX 7B: THE EXTERNAL TRACE OF ALL INSTRUCTION TYPES

```
LOCATION  ICR       CPU TIME          ND SD VT  [N1]                |INSTRUCTION
#00000/0  1         8                  0  0                       |J#00012/0
#00012/0  2         12                 1  0    #0000000000000002  |SETB2
#00012/3  3         19                 2  0    #0000000000000005  |SETB5
#00013/0  4         32                 1  0    #0000000000000002  |OUT
#00013/3  5         35                 0  0                       |=C15
...
#00236/2  3439084   19706693           1  2 V  #0000000000000004  |JS#00063/2
#00063/2  3439085   19706697           2  2 V  #0000000000001243  |M2
#00063/4  3439086   19706699           3  2 V  #0000000000001243  |DUP
#00063/5  3439087   19706703           2  2 V  #0000000000002506  |+
#00064/0  3439088   19706705           1  2 V  #0000000000000004  |=M3
#00064/2  3439089   19706709           2  2 V  #0000000000000001  |M1
#00064/4  3439090   19706715           1  3 V  #0000000000000004  |=LINK
#00065/0  3439091   19706721           2  3 V  #0000000000000000  |E#253M3
```

(etc)

APPENDIX 7C: THE SINGLE-STEP TRACE

In pause mode **ee9** executes instructions one-by-one, interacting with the user at each step and presenting the values in the registers relevant to the instruction just obeyed. Here it is, stepping through `P40`:

```
At #01313/4 (715/4); ICR = 182908; EL = 9111955; the instruction was
#200:324:357, i.e. JSE#02357/0
 JB: #01313/4; SJNS depth:   2
NEST:
 N1:
#1766314631463146 =   69709037200998 = 1.999999999998E-1 =  0.495312499999997
     = Q #037546/ #063146/ #063146   = Q  16230/  26214/  26214
     = #077 #146 #146 #146 #146 #146 = "/V9F9F9F"
```

```
Breakpoint: at #02357/0 (d:ebug | f:ast | t:race | p:ause or q:uit)? p

At #02357/0 (1263/0); ICR = 182909; EL = 9111957; the instruction was
#042, i.e. DUP
NEST:
 N1:
#1766314631463146 =   69709037200998 = 1.999999999998E-1 =  0.495312499999997
     = Q #037546/ #063146/ #063146   = Q  16230/  26214/  26214
     = #077 #146 #146 #146 #146 #146 = "/V9F9F9F"
 N2:
#1766314631463146 =   69709037200998 = 1.999999999998E-1 =  0.495312499999997
     = Q #037546/ #063146/ #063146   = Q  16230/  26214/  26214
     = #077 #146 #146 #146 #146 #146 = "/V9F9F9F"
```

```
Breakpoint: at #02357/1 (d:ebug | f:ast | t:race | p:ause or q:uit)?

At #02357/1 (1263/1); ICR = 182910; EL = 9111961; the instruction was
#203:104:366, i.e. JE#02366/3LEZ
NEST:
 N1:
#1766314631463146 =   69709037200998 = 1.999999999998E-1 =  0.495312499999997
     = Q #037546/ #063146/ #063146   = Q  16230/  26214/  26214
     = #077 #146 #146 #146 #146 #146 = "/V9F9F9F"
```

```
Breakpoint: at #02357/4 (d:ebug | f:ast | t:race | p:ause or q:uit)?

At #02357/4 (1263/4); ICR = 182911; EL = 9111967; the instruction was
#045, i.e. FIX
NEST:
 N1:
#7777777777777776 =               -2 = -6.189700196427E+26 = -0.000000000000014
     = Q #177777/ #177777/ #177776   = Q     -1/     -1/     -2
     = #377 #377 #377 #377 #377 #376 = "ØØØØØØ\"
 N2:
#3146314631463000 =  112589990684160 = 6.044629096666E+22 =  0.799999999999272
     = Q #063146/ #063146/ #063000   = Q  26214/  26214/  26112
     = #146 #146 #146 #146 #146 #000 = "9F9F9F8 "
```

```
Breakpoint: at #02357/5 (f:ast | t:race | p:ause or q:uit)? q
```

APPENDIX 7D: THE PERIPHERAL I/O EVENT TRACE

Under CPL is given the TSD context (P, Q, R or S) if the transfer was initiated by a program, or D if initiated by Director; and the current priority level. Under the heading T the value of the T register is shown as Y (for 1) or N (for 0) in the case of an order, such as BUSY, that tests device status. S and E in the next column show the start and end of a transfer.

```
Retrospective trace of peripheral I/O events.
                                                                CPL T   EL. TIME     ICR
After #4310/3:  CLOQ7                   Q #072500/ #004640/ #077337  D 0 N   2052470495   366874014
After #2520/4:  TWEQ7               FW0 Q #000000/ #000472/ #000472  D 0   E 2052465840   366871442
After #2472/0:  BUSYQ0              FW0 Q #000000/ #000000/ #000000  D 0 Y   2051676802   366873352
After #1062/4:  CTQ6                TR1 Q #000002/ #177777/ #000002  D 0 N   2051672349   366872593
After #2520/4:  TWEQ7               FW0 Q #000000/ #000472/ #000472  D 0   S 2051665840   366871441
After #2472/0:  BUSYQ0              FW0 Q #000000/ #000000/ #000000  D 0 N   2051665650   366871408
After #21/5:    PGAPQ13             TP0 Q #000004/ #000000/ #000454  P 0   E 2051660310   366870057
After #1534/4:  MANUALQ6            TP0 Q #000004/ #000001/ #000004  D 0 Y   2051660310   366870512
After #1255/2:  PARQ7               TP0 Q #000004/ #000000/ #000000  D 0 N   2051660106   366870474
After #1255/2:  PARQ7               TP0 buffer lockout               P 0     2051660106   366870473
After #623/5:   BUSYQ7              TP0 Q #000004/ #000000/ #000000  D 0 Y   2051654372   366870467
After #21/5:    PGAPQ13             TP0 Q #000004/ #000000/ #000454  P 0   S 2051652220   366870056
After #132/5:   PWEQ14              TP0 Q #000004/ #066440/ #066635  P 0   E 2051660025   366869806
After #132/3:   PREQ15              TR1 Q #000002/ #066440/ #066635  P 0   E 2051651903   366869805
After #132/5:   PWEQ14              TP0 Q #000004/ #066440/ #066635  P 0   S 2051650935   366869805
After #157/5:   PWQ14               TP0 Q #000004/ #066440/ #066440  P 0   E 2051723445   366869784
After #132/3:   PREQ15              TR1 Q #000002/ #066440/ #066635  P 0   S 2051650903   366869804
After #157/5:   PWQ14               TP0 Q #000004/ #066440/ #066440  P 0   S 2051650725   366869783
...
After #132/3:   PREQ15              TR1 Q #000002/ #066440/ #066635  P 0   E 373054471    59581364
After #132/5:   PWEQ14              TP0 Q #000004/ #066440/ #066635  P 0   E 373010462    59578367
After #132/3:   PREQ15              TR1 Q #000002/ #066440/ #066635  P 0   S 372953471    59581363
After #132/3:   PREQ15              TR1 Q #000002/ #066440/ #066635  P 0   E 372945668    59578366
After #132/5:   PWEQ14              TP0 Q #000004/ #066440/ #066635  P 0   S 372937742    59578366
After #132/5:   PWEQ14              TP0 Q #000004/ #066440/ #066635  P 0   E 373001334    59576600
After #132/3:   PREQ15              TR1 Q #000002/ #066440/ #066635  P 0   S 372937668    59578365
After #132/3:   PREQ15              TR1 Q #000002/ #066440/ #066635  P 0   E 372936540    59576599
After #132/5:   PWEQ14              TP0 Q #000004/ #066440/ #066635  P 0   S 372928614    59576599
After #20/0:    PGAPQ13             TP0 Q #000004/ #000000/ #000454  P 0   E 375655473    59576588
After #132/3:   PREQ15              TR1 Q #000002/ #066440/ #066635  P 0   S 372928540    59576598
After #20/0:    PGAPQ13             TP0 Q #000004/ #000000/ #000454  P 0   S 372928473    59576587
(etc)
After earlier interrupts, whose tracing is now lost.
Total time waiting for unoverlapped I/O to finish = 6635ms.
```

APPENDIX 7E: THE INTERRUPT TRACE

Under CPL is the context (P, Q, R or S) and priority level of the interrupted program. In addition, the trace shows control returning from Director to a problem program by means of the EXITD instruction, with the BA, NOL, and instruction address values for the resumed program.

```
Retrospective trace of interrupt requests.
                                                                CPL     EL. TIME     ICR
Ended #575/4:   EDT                 MT0 Q 8/#004335/#004336          P 0     435959729    69288606
After #76/4:    EDT                 FW0 Q 0/#005246/#005250          P 0     435943366    69285714
After #416/3:   OUT                 0                                P 0     433847319    69079602
After #116/3:   EXITD               BA #004640 NOL 8191 @ #416/2     P 0     433847299    69079600
After #415/5:   OUT                 8                                P 0     433827181    69076031
...
After #116/3:   EXITD               BA #004640 NOL 8191 @ #4430/0    P 0     429984577    68459560
After #4427/3:  CLOCK               1048580 KDF9 us                  P 0     429982227    68459073
After #116/3:   EXITD               BA #004640 NOL 8191 @ #4430/0    P 0     428934534    68285222
After #4427/3:  CLOCK               1048584 KDF9 us                  P 0     428925791    68283615
After #116/3:   EXITD               BA #004640 NOL 8191 @ #3153/4    P 0     427877727    68109705
After #60/5:    EDT                 MT0 Q 8/#011657/#011702          P 0     427877343    68109622
After #3153/4:  LOV                 at #011656 (E2574)               P 0     427877149    68109587
After #116/3:   EXITD               BA #004640 NOL 8191 @ #2056/4    P 0     427877113    68109584
After #2056/1:  OUT                 8                                P 0     427858473    68106231
...
After #116/3:   EXITD               BA #004640 NOL 8191 @ #3270/0    P 0     412965422    65759551
After #3267/3:  EDT                 FW0 Q 0/#000472/#000472          P 0     412956544    65757930
After #116/3:   EXITD               BA #004640 NOL 8191 @ #3153/4    P 0     412161464    65636232
After #60/5:    EDT                 MT0 Q 8/#011657/#011702          P 0     412161080    65636149
After #3153/4:  LOV                 at #011656 (E2574)               P 0     412160886    65636114
After #116/3:   EXITD               BA #004640 NOL 8191 @ #2056/4    P 0     412160850    65636111
After #2472/0:  EDT                 FW0 Q 0/#000472/#000472          P 0     412148327    65633868
After #2056/1:  OUT                 8                                P 0     411606886    65632757
...
After #116/3:   EXITD               BA #004640 NOL 8191 @ #2056/4    P 0     411352623    65591525
After #715/0:   EDT                 MT0 Q 8/#004335/#004336          P 0     411340926    65589427
After #2472/0:  EDT                 FW0 Q 0/#000472/#000472          P 0     411340010    65589283
After #712/5:   EDT                 MT0 Q 8/#004335/#004336          P 0     410541042    65587978
After #1027/5:  LOV                 in #066440/#066635 for TR1       P 0     410532559    65586511
After #2472/0:  EDT                 FW0 Q 0/#000472/#000472          P 0     410531063    65586264
After #457/4:   EDT                 MT0 Q 8/#004335/#004336          P 0     410035534    65585510
After #713/5:   LOV                 MT2 is busy                      P 0     410026286    65583901
After #2472/0:  EDT                 FW0 Q 0/#000472/#000472          P 0     410021744    65583150
After #1045/3:  EDT                 MT0 Q 8/#003021/#003023          P 0     409224283    65581808
After #2472/0:  EDT                 FW0 Q 0/#000472/#000472          P 0     409212381    65579669
After #2056/1:  OUT                 8                                P 0     409007105    65578559
After #116/3:   EXITD               BA #004640 NOL 8191 @ #3157/0    P 0     408914735    65562404
After #2472/0:  EDT                 FW0 Q 0/#000472/#000472          P 0     408905390    65560684
After #3156/3:  OUT                 17                               P 0     408408682    65559504
After #116/3:   EXITD               BA #004640 NOL 8191 @ #0/0       P 0     408407665    65559338
After #2565/0:  EDT                 TR1 Q 2/#004650/#011447          P 0     408332663    65546631
(etc)
After earlier interrupts, whose tracing is now lost.
```

APPENDIX 7F: THE INSTRUCTION-TYPE AND INSTRUCTION-LOCUS FREQUENCY HISTOGRAMS

Either, both or neither of these plots may be obtained, at option. The examples show two different programs.

```
Histogram of the opcodes of 3438705 executed instructions with frequency >= 0.20%
004: ×F                          54250      1.58% :||
012: PERM                        68668      2.00% :||
024: FLOAT                       10291      0.30% :
027: NEG                         54524      1.59% :||
033: NOT                         14503      0.42% :
034: ×D                          37146      1.08% :|
036: -                           66571      1.94% :||
037: SIGN                        33410      0.97% :|
041: ZERO                        41389      1.20% :|
042: DUP                        136649      3.97% :||||
043: DUPD                        33299      0.97% :|
045: FIX                         14447      0.42% :
050: CONT                        37951      1.10% :|
051: REVD                        48162      1.40% :|
052: ERASE                       52997      1.54% :||
056: +                          141379      4.11% :||||
060: /                           17040      0.50% :
062: /F                          12042      0.35% :
065: REV                        119555      3.48% :|||
066: CAB                         17098      0.50% :
074: +F                          45208      1.31% :|
075: -F                          49204      1.43% :|
102: MkMqQ                       20663      0.60% :|
140: M+Iq                        43817      1.27% :|
141: M-Iq                        69939      2.03% :||
151: MqTOQk                      15744      0.46% :
161: SHA                         51219      1.49% :|
164: SHL                         15755      0.46% :
166: SHLD                         7628      0.22% :
170: =[R]{Q|C|I|M}q             250329      7.28% :|||||||
171: {Q|C|I|M}q                 188054      5.47% :|||||
172: =+{Q|C|I|M}q                23666      0.69% :|
173: LINK                        40057      1.16% :|
174: =LINK                       64503      1.88% :||
206: JrNEZ                        9476      0.28% :
213: Jr                          62570      1.82% :||
215: JSr                         74752      2.17% :||
217: EXIT                        99197      2.88% :|||
221: JrEQ                         7125      0.21% :
222: JrLTZ                       12845      0.37% :
224: JrGTZ                       12795      0.37% :
226: JrEQZ                       44682      1.30% :|
260: JrCqNZ                      32457      0.94% :|
300: EeMq                       587012     17.07% :|||||||||||||||||
301: =EeMq                      214536      6.24% :||||||
303: =EeMqQ                     112303      3.27% :|||
304: SET                        201893      5.87% :||||||

Executions accounted for in the profile:  76.5%
```

```
Histogram of the loci of 306451954 executed instructions with frequency >= 0.50%
#63:                            21524493    7.0% :|||||||
#64:                            21524493    7.0% :|||||||
#65:                            14349662    4.6% :|||||
#66:                            7174831     2.3% :||
#70:                            2544372     0.8% :|
#71:                            1980726     0.6% :|
#72:                            2592966     0.8% :|
#73:                            3791307     1.2% :|
#74:                            2939526     0.9% :|
#75:                            28793967    9.4% :|||||||||
#76:                            7174831     2.3% :||
#77:                            11635855    3.8% :||||
#100:                           12165488    3.9% :||||
#101:                           12165488    3.9% :||||
#102:                           3041372     0.9% :|
#110:                           3274426     1.0% :|
#111:                           2971022     0.9% :|
#112:                           4667583     1.5% :||
#113:                           3073182     1.0% :|
#114:                           6069284     1.9% :||
#115:                           6069284     1.9% :||
#124:                           1630169     0.5% :|
#125:                           2220330     0.7% :|
#126:                           2331633     0.7% :|
#127:                           1554422     0.5% :|
#167:                           2697092     0.8% :|
#171:                           1992671     0.6% :|
#172:                           1977597     0.6% :|
#173:                           1977597     0.6% :|
#174:                           2022819     0.6% :|
#204:                           2697092     0.8% :|
#206:                           2692318     0.8% :|
#207:                           6742730     2.2% :||
#210:                           9439822     3.0% :|||
#211:                           13485460    4.4% :||||
#212:                           6573495     2.1% :||
#213:                           4552883     1.4% :|
#214:                           3034642     0.9% :|
#215:                           3034642     0.9% :|
#216:                           4552423     1.4% :|
#217:                           3022530     0.9% :|
#234:                           3009498     0.9% :|
#235:                           3022530     0.9% :|
#236:                           4539949     1.4% :|
#240:                           3022168     0.9% :|
#241:                           3035562     0.9% :|
#242:                           3371365     1.1% :|
#243:                           2022819     0.6% :|
#244:                           2058873     0.6% :|

Executions accounted for in the profile:  92.0%
```

APPENDIX 8: ERROR MESSAGES FROM ee9

The following headings introduce classes of error message. Each such message includes more detailed diagnostics.

LIV INTERRUPT

NOUV INTERRUPT

RESET INTERRUPT

These are invalid operations that were detected by the KDF9 hardware. A RESET interrupt always connotes a failure, as do NOUV and LIV interrupts in problem programs. NOUV cannot happen in Director. Although LIV could be signalled in Director, it did *not* interrupt. However, it is always treated by **ee9** as failing, because it signals states that do not permit emulation to continue successfully.

IMPOSSIBLE I/O OPERATION

IMPOSSIBLE I/O OPERATION IN DIRECTOR

INVALID OPERATION ATTEMPTED IN DIRECTOR

These failures are caused by actions that cannot be correctly interpreted. They include OUT orders with invalid parameters, that would have been rejected by Director, and miscellaneous other errors that **ee9** detects and fails.

FEATURE NOT YET IMPLEMENTED

Presently, only unimplemented OUTs and unimplemented device types give rise to this error.

THE KDF9 OPERATOR HAS MADE A MISTAKE

These failures are caused by errors in the configuration of the emulated KDF9 that result from mistakes made in the invocation of **ee9**; e.g. by running it in the wrong working directory, so that some required file is absent.

INVALID PAPER TAPE FILE SUPPLIED

These failures probably result from the user not having set up an object program correctly in KDF9 paper tape code.

APOLOGIES FOR THIS DISMAL FAILURE

These are failures of internal consistency checks made by **ee9**. They should never happen, but are handled defensively, with a view to debugging **ee9** itself. Messages from the `host_IO` package accompany an otherwise unhandled exception named `stream_IO_error`, and indicate that the execution environment is not compatible with running **ee9**. Each message includes a detailed diagnostic of the I/O stream concerned. Messages from the `POSIX` package accompany an otherwise unhandled exception named `POSIX_IO_error`, and indicate that the execution environment is not compatible with running **ee9**.

OTHERS

The following is detected during diagnostic output at the end of a run. Should never happen. 8-)

`Failure in finalize_ee9` …

APPENDIX 9: OPERATING A KDF9 WITH THE TIME SHARING DIRECTOR

The following material draws heavily on the EE document 'A1020 – Time Sharing Director – Mark I', by A. Doust and M.R. Wetherfield, the authors of the Time Sharing Director program.

Commands from the computer operators to Director are known as 'TINTs', short for **Typewriter Int**errupts. They are initiated by pressing the interrupt button on the console Flexowriter. Director responds by typing `TINT;` and waiting for a command to be input. Each such command is terminated by typing '**.→**'. With **ee9**, interrupt by typing CTRL-C and answer the prompt with RETURN. For '**→**', shown below, type '**|**'.

In the following, ***dd*** indicates a two-digit device (buffer) number; ***tt*** indicates a device type (see Table A4.2); ***p*** indicates a 12-character program name (also known as the PRN, or Program Reference Number); ***n*** is an octal number; ***l*** is a magnetic tape label; and ***s*** is a timesharing slot name (one of 'P', 'Q', 'R' or 'S').

A *s*.→
Terminate program in slot *s*, or terminate its loading.
A *s*+.→
Terminate program in slot *s* and type a basic diagnostic, showing the top of the SJNS and the NEST..

B *s n*.→
Read the octal integer *n* (up to 8 digits) and store its value into the less significant half of E0 of *s*. This can be used to convey options to the program at run time, separately from any tape or card data.

C *dd*.→
Load the magnetic tape on unit *dd*. Director reads the tape label, notes it, and positions the tape at the block immediately following, ready for allocation to a program that requests it by name, using OUT 4 or OUT 10.
D *dd*.→
Unload the magnetic tape on unit *dd*. The tape is rewound and disengaged from the deck, ready for removal.

E *dd***A.→** and **E** *dd***B.→**
Nominate the magnetic tape loaded on unit *dd* for the input of A/B programs.

F .→
Dummy (used if interrupt key pressed in error).

G.→
Type the peripheral unit list.

H.→
Type a list of 'wanted' tape labels. These are tapes that programs have requested, but have not yet been loaded.

I *s***0.→**
Even restart for program *s*, i.e. make it execute the jump in the first halfword of word 0. This, and the odd restart, are for use when a program fails in a manner that has been anticipated, and can recover from.
I *s***1.→**
O*dd* restart for program *s* , i.e. make it execute the jump in the second halfword of word 0.

L *dd*/*tt*.→
Change the peripheral device unit *dd* to type *tt*. If *tt* = 0, the unit is deleted from the free list.

M *x*{*s*?}*y*.→
Output a store print in syllabic octal. *x* and *y* are both octal integers. *y* words are output, starting at address *x*: if *y* is omitted, 1 is understood. If {*s*?} is P, Q, R or S then the base address of *s* is added to *x*. Any other separator (e.g. '**/**') gives the absolute (Director) address *x*.

R *s*.→
Resume *s* (after suspension).
S *s*.→
Suspend *s*.

T *n*/*w*.→
Load a B-program and give it priority level *n*, with a store limit of *w* words. Absence of *w* implies that the program may want the whole store. If the **/** is replaced by **P** the paper tape reader from which its call tape is read is pre-allocated to the program.

A-programs are loaded on the initiative of Director, not the human operators. When the resources that are needed become available, Director says:

n tt `P` *dd s*

n `APIU` *dd s*

The operator must have presented the program's call tape on the **A**-program **i**nput **u**nit *dd*. Director reads it in and continues:

`*` *date time*

n s `P` *p* **or** *n s* `M` *p*

depending on whether the program itself is to be read from paper tape or magnetic tape.

U.→
For each program in the machine, type its slot letter (P, Q, R or S), 12 character identifier, priority, base address, and running and elapsed times so far.

V ***n/n'.→***
Put program in priority level *n* into priority level *n'*, and vice versa.

Z ***dd.→***
Relabel the magnetic tape on unit *dd*. Director issues the query `TN/ID;` to which the operator replies `N/`*l*`.→`, e.g.:

```
TN/ID;N/+DIRECTORBINWORK.→
```

Director confirms the change, e.g.:

```
10L12 /Iden<+DIRECTORBINWORK>,TSN -00-0552
```

WHEN THINGS GO WRONG
If something goes wrong while reading in a program, Director types:

n `CRNP F`*x*

where: `CRNP` means 'Can't Read New Program' and *x* is:

0: No paper tape reader is available for reading the A-block; this applies to B-programs only.

1: Parity failure in the A-block.

2: the A-block not in correct form.

3: C-block sum check failure.

4: Parity failure in the B-block.

5: E2 and E3 of the B-block are not the same as the first two words of the A-block.

6: Program's store requirement too large.

7: Incorrect fillers for a C-block.

8: Parity failure in a C-block.

9: the unit specified by the A-block is not available for reading B-blocks and C-blocks.

Program input can be terminated by giving a bogus A-block, thus inducing `CRNP F2`. B-program input can be terminated by the operator: if no reader is available, `TINT A` *s*.→ causes `CRNP F0`, but after the A and B blocks have been read, and the program is waiting for core store, it causes `CRNP F6`.

In the event of a problem program failing, Director types:

n s `FAILS` *indicator* [*SJNS, N1, N2 and NEST depth, where needed*]

`REACT`;

This gives the operator the opportunity to abandon, or to restart the run, using **A** ***s.→***, **I** ***s0.→***, or **I** ***s1.→***. The *indicator* is taken from the following list of two or three digit codes:

`S`	A **S**purious interrupt, indicating a hardware (emulation) failure
`00L`	Lock-in violation
`00N`	Nest or SJNS over- or under-flow
`00T`	Time limit exceeded
`01`	Incorrect OUT number (it is still in N1)
`02`	Failure in OUT 5
`03`	Failure in OUT 6 or OUT 7
`04`	OUT 4 or OUT 10 requests an unavailable tape
`05`	OUT 1 obeyed with less than 3 items in nest
`06`	OUT 1 obeyed with incorrect program name
`07`	OUT 4 specifies incorrect identifier
`11`	OUT 10 specifies incorrect identifier
`71`	Closing OUT 8 stream without having written anything to it
`72`	Wrong terminator to output block in OUT 8
`73`	Incorrect character in OUT 8 block for the Flexowriter
`74`	Invalid stream number in OUT 8
`75`	No magnetic tape available for OUT 8
`76`	Parity failure in OUT 8 output
`77`	Invalid addresses for an OUT 8 block

The following failures do not give the option of a restart, i.e. `REACT;` is not typed:

`10` CRNP failure in OUT 1

`20` Incorrect program name in OUT 2

`21` Time limit for OUT 2 absent or incorrect

`22` OUT 2 obeyed with incorrect store limit in E1

NORMAL TERMINATION

When a program finishes without detected failure, the following message is typed out:

n p `/ / /` *e* `/` *run time* `/` *n.e.t.* `/` *elapsed time*

where: *run time* is the CPU time used; *n.e.t.* is the 'notional elapsed time' which gives an estimate of the elapsed time would have been experienced had there been no delays due to multiprogramming; *elapsed time* is the 'wall clock' time since the program started; and *e* is the 'ending number', which gives the reason for termination, namely:

0: OUT 0

2: OUT 2

4: TINT A

5: CRNP in OUT 1

6: any failure in OUT 2

7: result of any response to `REACT;`

In the case of successful overlay by OUT 1, Director types the priority, slot and name of the new program:

n s `OUT 1`

n `M` *p*

'DESCRIPTORS'

Messages of the form:

tt S dd s

indicate that unit *dd* is of type *tt* and has current status *S*. For units other than magnetic tape units, when the unit is unallocated *S* is the letter `U`. When the unit is pre-allocated to a program or allocated to a program, *S* is `P` or `A` respectively. *s* indicates the program to which the unit is (pre-)allocated. When a unit is allocated to Director (e.g. for TINT M), *S* is the letter `D`.

For magnetic tape units, these messages are only typed when the unit becomes loaded (`L`) or unloaded (`U`), except when the whole unit list is typed by TINT G). They then also include the Tape Serial Number (TSN) of the tape, e.g.:

```
10L12/Iden<PRINTEND>, TSN -20-0403
10L13/Ident<ZERO>, TSN -20-0284
```

When a list is typed by TINT G, the identifiers of tapes on loaded units only are typed, and the TSN is omitted, e.g.

```
10L12 PRINTEND
10L13 ZERO
```

When a unit is allocated or deallocated by a program, the descriptor is typed in the relevant column for P, Q, R or S. The only exceptions to this are the pre-allocation messages preceding A-program input, and messages concerning magnetic tape units. These are always typed in the Director column.

WAITING FOR ...

Messages of the form:

s `WAITING FOR TYPE` *tt*

mean that program *s* (which must in this case be a B-program) has attempted to allocate to itself a unit of type *tt*, but none is available. The operators have to decide whether or not to terminate program *s* (or another B-program), in order to free a device of type *tt*, or let it wait for one to be freed voluntarily.

APPENDIX 10: MAGNETIC TAPES IN ee9

Magnetic tapes are represented by Ada direct access files, using the `Ada.Direct_IO` library package, to allow selective overwriting of blocks. This is necessitated by the MWIPE and MGAP operations of the 1081-type tape deck. Tapes must be initialized (labelled) before any attempt is made to access them by a KDF9 problem program. (See **RLT** in Appendix 11.)

The "write permit ring", which was a metal band inserted into the tape reel concentrically with the hub, depressed a switch to enable the deck's write heads. It is modelled by the file's access permission. Making the file read-only simulates an absent ring, allowing operations that do not change the contents of the tape, and failing those that do.

Each tape block, and each length of erased tape, is represented by one or more "slices", a slice being a record in the direct access file. A slice has two components: a string and per-slice metadata. The string contains the Case Normal Latin-1 transliteration of all or part of a KDF9 data block. In the case of an erasure slice the string reserves space for possible future over-writing by data, but its contents are of no significance.

An emulated tape block is represented by a number of consecutive slices such that the total length of their contained strings is sufficient to encompass the data block or tape erasure. The maximum string size has been set so that blocks of 256 words (as used by POST), and short blocks containing OUT 8 print-line images, both have a space efficiency of better than 95%. The maximum block size (not slice size) is 32KW, or 256K characters, that being the maximum size of the core store.

No attempt is made to model either interblock gaps or the erased tape extending from the Beginning of Tape Window (BTW) to the start of the first block of data. However interblock gaps are taken into account when estimating the elapsed time of magnetic tape transfers.

Ampex 7-track tape files are recorded in exactly the same way as KDF9-native tapes, with the addition of single-slice blocks representing the two kinds of tape mark. They read into core as #170000000000 in the single input word. Tape marks do not appear in KDF9-native tapes.

The metadata in each record is as follows.

- Byte 0—a code indicating whether the slice represents a **d**ata block (code 'D'), a length of tape erased by the M**W**IPE operation (code 'W'), a length erased by the MGAP operation (code 'G'), an **e**ven parity tape mark (code 'e'), or an **o**dd parity tape mark (code 'o').
- Byte 1—a code made up as follows:

  (a) if the block is 'LBM' marked; i.e. if it was written by a MLWQq or MLWEQq instruction instead of the normal MWQq and MWEQq instructions, and so responds positively to the MLBQq order: +64;

  (b) if this is the last slice of a multi-slice block: +8; and

  (c) if this is the first slice of a multi-slice block: +1.

  Byte 1 therefore has the following possible values (decimal = octal = Latin-1):

  ```
  00 = 000 = NUL ⇒ no flags
  01 = 001 = SOH ⇒ first slice of block
  08 = 010 = BS  ⇒ last slice of block
  09 = 011 = HT  ⇒ only slice of block (first and last)
  64 = 100 = @   ⇒ LBM flag
  65 = 101 = A   ⇒ first slice of block with LBM flag
  72 = 110 = H   ⇒ last slice of block with LBM flag
  73 = 111 = I   ⇒ only (first and last) slice of block with LBM flag
  ```

- Byte 2—the length of the string in this slice.

This encoding makes the contents of a magnetic tape somewhat legible using the POSIX **od** command, but **mtp** usually offers an a more useful view.

THE `mtp` UTILITY

Convenient views of a tape file are obtainable using the **mtp** program. The command takes the form:

```
mtp {MT|ST}{01234567}[T|{PQRS}{1357}{01234567}]]
```

The nominated magnetic tape file is read and an interpretation of its content is written to the standard output. Any error messages are written to the standard error output.

- To despool a stream from an OUT 8 tape written by TSD, append the slot letter and stream number, e.g. `mtp MT0P30`:

```
Despooling OUT 8 stream 30 for slot P, written on 23/09/69 by WHETSTONEUPU

N=   0 J=   0 K=   0 X1=+1.00000000 X2=-1.00000000 X3=-1.00000000 X4=-1.00000000
N= 120 J= 140 K= 120 X1=-0.06834220 X2=-0.46263766 X3=-0.72971839 X4=-1.12397907
N= 140 J= 120 K= 120 X1=-0.05533645 X2=-0.44743656 X3=-0.71097339 X4=-1.10309806
N=3450 J=   1 K=   1 X1=+1.00000000 X2=-1.00000000 X3=-1.00000000 X4=-1.00000000
N=2100 J=   1 K=   2 X1=+6.00000000 X2=+6.00000000 X3=-0.71097339 X4=-1.10309806
N= 320 J=   1 K=   2 X1=+0.49040732 X2=+0.49040732 X3=+0.49039250 X4=+0.49039250
N=8990 J=   1 K=   2 X1=+1.00000000 X2=+1.00000000 X3=+0.99993750 X4=+0.99993750
N=6160 J=   1 K=   2 X1=+3.00000000 X2=+2.00000000 X3=+3.00000000 X4=-1.10309806
N=   0 J=   2 K=   3 X1=+1.00000000 X2=-1.00000000 X3=-1.00000000 X4=-1.00000000
N= 930 J=   2 K=   3 X1=+0.83466552 X2=+0.83466552 X3=+0.83466552 X4=+0.83466552

THAT TOOK  23.8 SECONDS, FOR  42.0 KWIPS
FINISHED ENDS 0
```

- If an OUT 8 stream is numbered in the 50-57 range it is taken to be written in Ferranti 5-hole paper tape code. In that case **mtp** attempts to emulate an original teleprinter transcription of the data. For example, this KDF9 code:

  ABCDEFGHIJKLMNOPQRSTUVWXYZ}|\Ø

  is transcribed thus in Figures Shift:

  12*4()78±=-v*lf* ,0>»3|56/×9+.n*cr*

  and thus in Letters Shift:

  ABCDEFGHIJKLMNOPQRSTUVWXYZ.?£

  Note that the Ø is suppressed.

- If a magnetic tape contains only legible **t**ext in paper **t**ape code, e.g. a Usercode object program as written out by the Kidsgrove Algol compiler, append a `T` to the tape deck name, e.g. `mtp MT3T`, to get its **t**ranscription into plain Latin-1.

- Invoked with just the deck name, e.g. `mtp MT0`, it gives a slice-by-slice analysis the tape file. Each character in a data block is displayed using its Case Normal Latin-1 transliteration. Each slice is displayed with its record number in the direct access file, the KDF9 data is enclosed in « » quotes, an asterisk flags a block written with a 'last block' marker, and complete blocks, possibly consisting of several slices, are separated by blank lines. Here is the analysis of the above OUT8 tape:

```
TAPE LABEL TSN '-00-1478', IDENTIFIER '+KOUTPUT/0023001'

  2       * «0001ØØP"P000001©»

  3       * «0001ØA&"©ØØØØP© »

  4         «0003ØØ&"WHETSTONEUPU   23/09/69®»

  5         «0001ØØ&"  STR30®»

  6         «0001ØØ&"©ØØØØP© »

  7         «0023ØØ&"N?ØØØØØØ   0ØØØØ J?ØØØØØ   0ØØØØ K?ØØØØØ   0ØØØØ X1?ØØØØ+1.000000000ØØØØØ »
  8         «X2?ØØØØ-1.000000000ØØØØØ X3?ØØØØ-1.000000000ØØØØØ X4?ØØØØ-1.000000000ØØØØØØØØØØØ®»

  9         «0023ØØ&"N?ØØØØØØ 120ØØØØ J?ØØØØØ 140ØØØØ K?ØØØØØ 120ØØØØ X1?ØØØØ-0.068342200ØØØØØ »
 10         «X2?ØØØØ-0.462637660ØØØØØ X3?ØØØØ-0.729718390ØØØØØ X4?ØØØØ-1.123979070ØØØØØØØØØØØ®»

 11         «0023ØØ&"N?ØØØØØØ 140ØØØØ J?ØØØØØ 120ØØØØ K?ØØØØØ 120ØØØØ X1?ØØØØ-0.055336450ØØØØØ »
 12         «X2?ØØØØ-0.447436560ØØØØØ X3?ØØØØ-0.710973390ØØØØØ X4?ØØØØ-1.103098060ØØØØØØØØØØØ®»

 13         «0023ØØ&"N?ØØØØØØ34500ØØØØ J?ØØØØØ   10ØØØ K?ØØØØØ   10ØØØ X1?ØØØØ+1.000000000ØØØØØ »
 14         «X2?ØØØØ-1.000000000ØØØØØ X3?ØØØØ-1.000000000ØØØØØ X4?ØØØØ-1.000000000ØØØØØØØØØØØ®»

 15         «0023ØØ&"N?ØØØØØØ21000ØØØØ J?ØØØØØ   10ØØØ K?ØØØØØ   20ØØØ X1?ØØØØ+6.000000000ØØØØØ »
 16         «X2?ØØØØ+6.000000000ØØØØØ X3?ØØØØ-0.710973390ØØØØØ X4?ØØØØ-1.103098060ØØØØØØØØØØØ®»

 17         «0023ØØ&"N?ØØØØØØ 320ØØØØ J?ØØØØØ   10ØØØ K?ØØØØØ   20ØØØ X1?ØØØØ+0.490407320ØØØØØ »
 18         «X2?ØØØØ+0.490407320ØØØØØ X3?ØØØØ+0.490392500ØØØØØ X4?ØØØØ+0.490392500ØØØØØØØØØØØ®»

 19         «0023ØØ&"N?ØØØØØØ89900ØØØØ J?ØØØØØ   10ØØØ K?ØØØØØ   20ØØØ X1?ØØØØ+1.000000000ØØØØØ »
 20         «X2?ØØØØ+1.000000000ØØØØØ X3?ØØØØ+0.999937500ØØØØØ X4?ØØØØ+0.999937500ØØØØØØØØØØØ®»

 21         «0023ØØ&"N?ØØØØØØ61600ØØØØ J?ØØØØØ   10ØØØ K?ØØØØØ   20ØØØ X1?ØØØØ+3.000000000ØØØØØ »
 22         «X2?ØØØØ+2.000000000ØØØØØ X3?ØØØØ+3.000000000ØØØØØ X4?ØØØØ-1.103098060ØØØØØØØØØØØ®»

 23         «0023ØØ&"N?ØØØØØØ   0ØØØØ J?ØØØØØ   20ØØØ K?ØØØØØ   30ØØØ X1?ØØØØ+1.000000000ØØØØØ »
 24         «X2?ØØØØ-1.000000000ØØØØØ X3?ØØØØ-1.000000000ØØØØØ X4?ØØØØ-1.000000000ØØØØØØØØØØØ®»

 25         «0023ØØ&"N?ØØØØØØ 930ØØØØ J?ØØØØØ   20ØØØ K?ØØØØØ   30ØØØ X1?ØØØØ+0.834665520ØØØØØ »
 26         «X2?ØØØØ+0.834665520ØØØØØ X3?ØØØØ+0.834665520ØØØØØ X4?ØØØØ+0.834665520ØØØØØØØØØØØ®»

 27         «0001ØØ&"ØØØØØØØ®»

 28         «0007ØØ&"THAT TOOK ØØØØØØ 23.8 ØØSECONDS~ FOR ØØØ 42.0 ØØKWIPSØØ®»

 29         «0002ØØ&"FINISHED ENDS 0®»

 30       * «0001ØZ&"©ØØØØP© »

 31       * «}ØØØØØØØ»

 32         WIPE of 1906 erased characters

EOF
```

The final '`EOF`' is not from the tape; it is a confirmation by **mtp** that all of the data on the tape has been shown.

APPENDIX 11: ANCILLARY PROGRAMS FOR USE WITH ee9

This section gives specifications for the utility programs—some shell command files, some in KDF9 Usercode, and others in Ada 2012—that are provided with distributions of **ee9** to facilitate its use.

THE a2b UTILITY

a2b reads data from standard input in a stated code and writes it to standard output in another form. Conversions are available between raw bytes and paper tape code, between paper tape code and Latin-1, and from paper tape code to a legible display. The command takes the form:

```
a2b  { -r2p | -L2p | -p2c [ F | D | M | P ] | -p2L | -p2t | -p2o | -p2r | -F2L }
```

Exactly one flag parameter must be given; it is actually case-insensitive. The effect is as follows:

`-r2p`	Take the input to be '**r**aw' bytes, perhaps representing KDF9 syllables; read groups of 6 to form 48-bit words; output each such word as 8 KDF9 characters in **p**aper tape code.
`-L2p`	Take the input to be 8-bit characters in **L**atin-1 code; transcribe them individually to KDF9 characters in **p**aper tape code.
`-p2L`	Take the input to be KDF9 characters in **p**aper tape code; transcribe them individually to 8-bit characters in **L**atin-1 code; ignore NULs (runout); report parity errors.
`-p2c`	Take the input to be a KDF9 object program in **p**aper tape code and output a valid **c**all tape for it, using the name extracted from words 2 and 3 of the program's A block; report parity errors. If one of the parameters D, M, or P is given a call type of that kind is created. The parameter F (for **f**ile) makes a call tape for the medium specified in the A block of the program file, and this is the default.
`-p2t`	Like `-p2L`, but effect case shifting and ignore redundant or non-printing characters, so as to give a clean output **t**ext; report parity errors.
`-p2o`	Take the input to be KDF9 characters in **p**aper tape code; read groups of 8 to form 48-bit words; output each such word in octal half/word, Q-store, syllable and character formats; report parity errors and incomplete words.
`-p2r`	Take the input to be KDF9 characters in **p**aper tape code; read groups of 8 to form 48-bit words; output each such word as 6 '**r**aw' bytes; report parity errors and incomplete words.
`-F2L`	Take the input to be characters in **F**erranti paper tape code; transcribe them to 8-bit characters in **L**atin-1 code.

THE RLT UTILITY

RLT is a Usercode program that reads a set of tape labels from TR1 and writes them in succession to the magnetic tape files MT0-MT5 and ST0, having erased a gap in which to write the new label on each tape. The data given in TR1 consists of one or more lines of text. The first 8 characters of the line give the 'Tape Serial Number' (TSN). The next 8 or 16 characters give the mnemonic 'identifier' of the tape. If the tape is intended to be a scratch work tape, its identifier must consist of 8 blank spaces (all zeros in KDF9 character code). A 16-character identifier must start with the character '+'. There must be an EM character ('|' in **ee9**) after the last character of the identifier. A line starting with an EM character terminates the data.

Here is an example, showing the sort of data that is used to label the tapes by **ee9**'s regression test suite:

```
-00-1234WHETLIST|
00000000        |
-00-0552EFPBEAAG|
77777777MS-DUMP.|
-00-2339+KOUTPUT/0023004|
0-00-929PRINTEND|
0-00-777AMPEXTM4|
|
```

For ease of running, the command file **rlt** is provided to set up the environment and the **ee9** mode parameters for **RLT**. **rlt** truncates the former contents of all of the tape files and then runs **RLT** in test program mode, so that it can access the physical tape buffers without having had them allocated by Director (which requires the tapes to be already labelled.)

rlt takes the new labels from the file `RLT_data.txt` by default. For a different set of labels, supply the name of an alternative file as a parameter.

THE PLT UTILITY (NOT YET AVAILABLE)

PLT is a Usercode program that generates POST-format program library tapes that are used by the Time Sharing Director to hold programs that can be loaded into store at the command of an M-type call tape. (See **a2b** and **call_tape** above.)

If given no data **PLT** claims a ZERO (scratch) work tape, relabels it `+PROGRAM/LIBRARY`, and writes the sentinel blocks that structure it as an empty library.

If given a KDF9 object program in TR1, it appends that program to the library.

For ease of running, the command file **plt** is provided to set up the environment and the **ee9** input mode parameters correctly for **PLT**. Among other things, **plt** runs the **RLT** command to label the tape files suitably prior to running **PLT**.

See the EE System Routine Library Manual, §22.1, 'Library Tape and Program Structure', 1965.

to_9_from_1934
to_9_from_1934 reads the file `Data/wabbit_data_1900`, which must contain commands for the ICT 1934 plotter, converts them to KDF9 plotter commands, and writes them to the file `Data/wabbit_data_kdf9` in KDF9 8-track paper tape code. See ICT Technical publication 4003/C, 'System Manual Volume X: Visual Input/Output Equipment', 1966.

kidopt
kidopt takes in a list of symbolic Kidsgrove Algol compilation options and converts them to the bit patterns used to transfer options to the compiler. See Appendix 12.

ee9_regr_tests
ee9_regr_tests runs a comprehensive regression test of **ee9**. It does not include any tests of boot mode, because running a Director forces user interaction. This is considered a *regression* test because it compares its logged output with what is taken to be the correct output, and differences (if any) are reported.

cr_regr_tests
cr_regr_tests runs a comprehensive regression test of card reader and card punch emulation.

dr_regr_tests
dr_tests runs a set of regression tests of drum store emulation.

fd_regr_tests
fd_tests run a set of regression tests of fixed disc emulation.

kids_tests
kids_tests runs a selected set of tests of the Kidsgrove Algol compiler and runtime system.

liv_tests
liv_tests runs a comprehensive set of tests of basic CPU error-detection functionality.

mt_regr_tests
mt_regr_tests runs a comprehensive regression test of magnetic tape emulation.

nouv_tests
nouv_tests runs a comprehensive set of tests of basic CPU nesting store functionality.

si_tests
si_tests runs a set of tests for standard interface buffer emulation.

whet_tests
whet_tests runs a selected set of tests of the Whetstone Algol compiler and runtime system.

APPENDIX 12: WORKING WITH THE WHETSTONE AND KIDSGROVE ALGOL 60 COMPILERS

The Algol 60 Report defines a 'reference language' that ignores the practicalities of implementation on the graphically impoverished computers of the 1960s. Indeed it uses symbols that are not trivially available even today, such as `10`, ×, ÷, ⊃, and ≡. It further specifies that an implementation may work in a 'hardware language' that represents the reference language using a computer's native character set. The Whetstone ('WAlgol') and Kidsgrove ('KAlgol') hardware languages are based on the Friden Flexowriter character set listed in Appendix 2A. Symbols not present there are synthesized, primarily by means of the non-escaping underline character, so that **begin**, for example, is prepared for input to a compiler as `begin` and ≡ as `eqv`. **ee9** uses the Latin-1 break character as its transliteration of the Flexowriter underline, so that `begin` appears thus: `_b_e_g_i_n`. Another scheme, used on computers with even less flexible input than the KDF9, is the 'stropped' format: each reserved word is decorated with flag characters and *ad hoc* arrangements are made for other symbols. For example, many card oriented Algol systems, including EGDON Algol on the KDF9, represent **begin** as `'begin'`, with the word enclosed in apostrophes. The Eldon 2 Teletype-based interactive system for the KDF9 flags reserved words with an initial exclamation mark, thus: `!begin`, and this is the form I find easiest to use.

The 2020 input routines for the resurrected Kidsgrove system allow greater flexibility and convenience, with well-considered alternative lexical forms that are much nicer to type. The following table lists character transliterations and other hardware representations used with **ee9** for both the Whetstone and Kidsgrove compilers. The original Whetstone representations are accepted by the Kidsgrove compiler, but not vice versa; the Whetstone representations are the only ones accepted by the 1964-vintage Whetstone compiler. You may find it easiest to write a program in stropped format and convert it to original format, even if you only want to use the Whetstone compiler. See the **`strop`** and **`ports`** commands described in §1; they provide a quick and easy way of converting programs from one format to the other.

Algol 60	Flexowriter	**ee9** and `whet`	additional `kalgol` forms
↑	↑	^	`!up 'up'`
×	×	×	`*`
÷	÷	÷	`% !div 'div'`
10	10	º	`~ \\`
≠	≠	±	`# != !ne 'ne'`
≥	≥	_>	`>= !ge 'ge'`
≤	≤	_<	`<= !le 'le'`
	→	|	`****`
begin *etc*	`begin` *etc*	`_b_e_g_i_n` *etc*	`!begin 'begin'` *etc*
'string'	`[string]`	`_[string_]`	`{string}` (string in string quotes)
'_'	`[*]`	`_[*_]`	`{_} {$}` (string blank space in string quotes)
¬	`not`	`_n_o_t`	`!not 'not'`
∧	`and`	`_a_n_d`	`!and 'and'`
∨	`or`	`_o_r`	`!or 'or'`
⊃	`imp`	`_i_m_p`	`!imp 'imp'`
≡	`eqv`	`_e_q_v`	`!eqv 'eqv'`

WHETSTONE ALGOL (WALGOL)

A selection of Algol source programs can be found in the directory `Testing/Whetstone`. I provide a shell command, **`whet`**, in the `Testing` directory, that makes it easy to compile and run Algol programs using WAlgol. It expects programs to be in text files named with the '`.a60`' suffix. The suffix is optional in a name given as parameter to the **`whet`** command; if supplied, it must be correct; if omitted, it is automatically restored; so that, e.g.:

```
whet FLUID
```

compiles and runs `Whetstone/FLUID.a60`. There is some flexibility in the location of WAlgol source programs: see Appendix 1. A source program and its stream 20 data may both be kept in the same file. Alternatively, only the program need be held in that file, with its data being provided by the interactive input feature of **ee9**. (See §3.3.)

Additional *mode* and *miscellany* parameters are optional and as given in §1 for the **`nine`** command. More comprehensive options, as described in §5, may be specified in the `settings_1.txt` file. If you want to specify different options for the program run, as opposed to its compilation, put them in the second options file, `settings_2.txt`.

If either `TP0` or `LP0` is found to be non-empty at the end of a run, its content is displayed on the terminal.

The **`whet`** command sets up the `FW0` file for the Whetstone system so that you do not need to type in any responses to its Flexowriter prompts. If the given replies are unsuitable, change the data in `FW0_for_Whetstone`.

A compilation listing is produced only if the response to the compiler's `OUT;` prompt is an OUT 8 stream number: either `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 and object code addresses, which may later be quoted in runtime error messages. The line numbers are somewhat deceptive, as they seem to ignore blank lines and lines before the first **begin**.

When the WAlgol object program starts, it prompts `STREAM;` for the number of the output stream to be used; again, reply `10.|` or `30.|`. This should be the same as the stream number nominated for output in the source program. All of this is automated with the provided `FW0_for_Whetstone` file, which suppresses the listing and sends output to stream 30.

You are likely to encounter both compilation errors and execution errors. They are explained in the documents *Whetstone Compilation Errors* and *Whetstone Runtime Errors*. Runtime errors may be accompanied by a 'retrospective trace' which lists the procedures called but not yet exited, the value of any failing operand, and a source code line number.

KIDSGROVE ALGOL (KALGOL)

The KAlgol system has been resurrected by David Holdsworth, David Huxtable (one of its original authors) and Brian Wichmann. It is something of a chimaera. Several of its original components are lost and have been replaced with reverse engineered substitutes. The object program is generated in Usercode, which needs to be assembled. For these reasons running the Kidsgrove system is complex, and a little flaky. Nevertheless it is well worth the effort to see what a pioneering and very clever compiler was able to make of an intractable language like Algol 60 six decades ago.

A shell command, **`kalgol`**, in the `Testing` directory, hides this complexity and makes it easy to compile KAlgol programs. It has one parameter, the name of the program, taken from the directory `Testing/Kidsgrove`. Execution *mode* and *miscellany* parameters for the run of the compiler may be given in the environment variables `KIDSMODE` and `KIDSMISC` respectively. It is particularly useful to set `KIDSMISC=w` to suppress the compiler's voluminous Flexowriter output, and this is the default. For example:

```
kalgol FLUID
```

compiles `Kidsgrove/FLUID.a60`.

A shell command, **`kids`**, in the `Testing` directory, combines a call of **`kalgol`** to compile a program and a call of **`nine`** to run it. It has one mandatory parameter: the name of the program. Additional parameters are taken to be *data*, *mode* and *miscellany* options for the run of the object program by **`nine`**. Mode and miscellany parameters for the run of the compiler may be given as for **`kalgol`**. For example:

```
export KIDSMODE=t; export KIDSMISC=rsw
kids FLUID FLUID_data f
```

After a successful compilation with **`kids`** or **`kalgol`** the Usercode object program is left in the `Assembly` directory. The KDF9 executable is left in the `Binary` directory and can be run with the **`nine`** command, e.g.:

```
nine FLUID FLUID_data f
```

There is actually considerable flexibility in the location of KAlgol source programs, object programs, their data files, and other components of the Kidsgrove compilation system provided with **ee9**: see Appendix 1.

Error messages are explained in the documents *Kidsgrove Compilation Errors* and *Kidsgrove Runtime Errors*.

KAlgol procedure bodies can be written in a version of Usercode that allows access to formal parameters. This is how the standard functions and the I/O libraries are implemented. See the document *Kidsgrove Code Procedures* and, for a simple example, the `PLOT` program in `Kidsgrove`.

The standard functions of Algol 60, and the KDF9 Algol I/O procedures, are incorporated into a program by means of the **library** (`!library`, `'library'`, `_l_i_b_r_a_r_y`) directive, which is placed immediately after the first **begin** of a program and lists the (source code) libraries to be included. The document *Kidsgrove IO Libraries* names the libraries that were originally provided. The table below indicates the libraries (A0 through A15) that are available at the moment, and the procedures they contain. This is subject to occasional enhancements.

`A0`	the standard functions of Algol 60: `sqrt`, `ln`, `exp`, `sin`, `cos`, `arctan`, `entier`, `abs`, `sign`
`A1`	`open` and `close`; all input must come from stream 20, and all output must go to stream 30
`A4`	`read` and `read boolean` (but `read boolean` is unlikely to work with A1 as yet)
`A5`	`write`, `output`, `format`, `write boolean`
`A6`	the union of A1, A4, A5, and A15
`A12`	`newline`, `space`, `tab`, `gap`
`A13`	character I/O and Algol Basic Symbol I/O
`A14`	`copy text`
`A15`	`write text`

It is an error to import duplicates, so that A6 should not be specified in combination with any of A1, A4, A5, and A15.

Kidsgrove compilation options, affecting the object code, e.g. by suppressing optimization, may be given on the command line, thus:

```
kalgol FLUID with NO_OPT DEBUG
```

or by setting an environment variable, thus:

```
export KIDSOPTS="NO_OPT DEBUG"
kalgol FLUID
```

(The "`with` …" syntax is a nod to the `TRANSLATE` command of the KDF9's POST software development system.)

To communicate compilation options to **`kalgol`** when using the **`kids`** command, which has an ample sufficiency of optional parameters, use `KIDSOPTS`, thus:

```
export KIDSOPTS="NO_OPT DEBUG"
kids FLUID FLUID_data
```

The presently available compilation options are as follows:

option	code	meaning
NONE	#0	apply default options
DEBUG	#2	enable assignment tracing (strangely invoked by 'WITHOUT TEST' within POST)
NO_OPT	#4	suppress global optimisations
LABELS	#10	enable label tracing (strangely invoked by 'WITH TEXT' within POST)

• NONE specifies the compiler's default behaviour: an optimised, overflow-checking, assignment-tracing compilation with none of the other compile-time or run-time diagnostic outputs.

• DEBUG enables the generation of assignment tracing code. The compiler includes a call to a procedure that can print the value being assigned (in octal). The output from that procedure can be switched off and on dynamically using a procedure called TRACE (nothing to do with any TRACE option). If a compilation omits the DEBUG option all code related to assignment tracing is omitted from the object program.

• NO_OPT specifies the suppression of the compiler's global optimisations. The resulting program will probably run quite a bit slower. The optimiser is not completely reliable and may produce incorrect object code in difficult cases, but it is instructive to compare the Usercode produced with and without optimisation. Simple **for** loops, in particular, benefit enormously from optimisation.

• LABELS requests the inclusion of code to provide a control-flow trace that prints out the Usercode labels of instructions encountered as the object program executes. This option is useful only in conjunction with object program's Usercode assembly listing, which **kalgol** leaves in the Assembly directory on completion of a successful compilation.

Options are given to the compiler under **ee9** by means of the P, 'poke', flag in a settings file. The **kidopt** program takes in the list of symbolic options and outputs the corresponding P flag setting. **kalgol** copies settings_for_kalgol.txt into settings_1.txt and then appends the output from **kidopt**. This enables the user to set some persistent options for **ee9**'s runs of the compiler and allows them to be augmented with Kidsgrove-specific compilation options, per-run, by means of **kidopt**'s output. For more information about how **kalgol** works, see §7 of *ee9 Implementation Overview*.

N.B. This correspondence between option bits and object code effects is implemented in the early version of the Kidsgrove compiler that has been restored to working order and is now used with **ee9**. It is not entirely consistent with some surviving English Electric documentation, which is thought to support a later version of the compiler.

See Also

The Algol 60 language, and its usage on the KDF9 computer, are described in the following English Electric publications:

Report On The Algorithmic Language ALGOL 60
Ed. P. Naur; 1960. Available in: *Egdon System Reference Manual, Chapter 7, 'EGDON ALGOL'*.
Publication 103 550566; English Electric–Leo–Marconi Computers Limited; 1968.

KDF9 ALGOL Programming
Publication 1002 mm (R) 1000565, English Electric–Leo–Marconi Computers Limited; 1969.

The following papers describing the Kidsgrove and Whetstone systems were written by their original designers and implementers and constitute an invaluable historical resource:

Kidsgrove

Implementation of ALGOL 60 for the English Electric KDF9
F. G. Duncan
Computer Journal, Vol.5, No.2, pp. 130–132; 1962.

Input and output for ALGOL 60 on KDF9
F. G. Duncan
Computer Journal, Vol.5, No.4, pp. 341–344; 1963.

A multi-pass translation scheme for ALGOL 60
E.N. Hawkins and D.H.R. Huxtable
Annual Review in Automatic Programming, Vol.3, pp. 163-205, Pergamon Press; 1963.

On writing an optimizing translator for ALGOL 60
D.H.R. Huxtable
Introduction to System Programming, APIC Studies in Data Processing, No.4, pp. 137-155, Academic Press; 1964.

Whetstone

The Whetstone KDF9 ALGOL Translator
B. Randell
Introduction to System Programming, APIC Studies in Data Processing, No.4, pp. 122-136, Academic Press; 1964.

ALGOL 60 Implementation
B. Randell and L.J. Russell
Academic Press; 1964.

These documents are available online, at time of writing, at www.findlayw.plus.com/KDF9.