## ELLIOTT 900 SERIES SIMULATORS

Terry Froggatt first wrote an Ada simulator for the Elliott 900 series in 1985, partly to evaluate early compilers and partly as an exercise in writing portable code. An updated version was modified for Meridian Ada in 1992 and simulated the full 64K words of store. It was run from the DOS prompt. It did not have any software to run with it, although a subset of Load and Go SIR was provided to enable simple programs to be input.

Subsequently Don Hunter produced a modification of Terry's system that ran the Elliott ALGOL 60 system, together with a few demonstration programs. This simulator went through a number of iterations adding additional features such as the ability to handle libraries, graph plotting and bug fixes. It too was DOS based.

The Ada simulator can be found on the Computer Conservation Society web site:
ftp://ftp.cs.man.ac.uk/pub/CCS-Archive/simulators/Elliott903/

An extended version of the Ada simulator and further demonstration programs is available from Don Hunter's web site: http://www.gnhunter.demon.co.uk/903/

The Ada simulator was written for early 16/32 bit versions of Windows and does not run on more recent 32/64 bit versions. Eric Baiger ported it to gnat so that it could be run on Unix and Windows 2000 or later. His simulator is available at:
http://www.baigar.de/TornadoComputerUnit/index.html - EMULATOR

In 2010 as a programming exercise I transcribed Don's system to F# and enhanced it to include emulation of all the different 900 series architectures (920A, 920B, 920M and 920C) including so-called "undefined (or secondary) effects" in the documentation, using information provided by Terry Froggatt. Other improvements included more flexible handling of input-output, translation of modern Unicode files to their Elliott paper tape (or "telecode") equivalents and many additional aids for debugging programs running on the simulator.

In addition to the specific models, the simulator provides a generic "900" model which behaves identically to the Ada simulator – in particular, the 900 model has no secondary effects.

Using tapes from Terry’s archive, versions of Elliott ALGOL, FORTRAN, SIR, subroutine libraries, test programs and utility programs were added alongside Don's extended version of Elliott ALGOL and Terry's BASIC system.

In 2011 I acquired one of Don’s 903 computers and with it more tapes of Elliott’s software, enabling me to cross-check with Terry's archive and fill in a few gaps to further expand the range of software running on the simulator.

In 2012 I added further support for use of 900 series line printers, card readers and magnetic tape units. Additionally, I provided support for simulating a Benson-Lehner digital plotter as used on 903 computers as an alternative to the plotting facilities provided in Don's simulator.

## INSTALLING THE F# SIMULATOR SYSTEM.

The simulator is available as a GITHub Repository:

https://github.com/andrewjherbert/SIM900

To install the simulator, download it from GITHub.

The result of the download will be a folder containing:

1. A file README.MD which gives a precis of the information in this chapter.
2. A file "simulator.soln" – this is a "solution" file which you can use to load the system into Microsoft Visual Studio.
3. A sub-folder "simulator" which contains the source, object and binary files for the simulator.
4. A sub-folder "Demos" which contains further folders of file containing simulator scripts to run demonstrations of Elliott 900 software including language systems such as Algol 60 and FORTRAN.
5. A sub folder "Manual" which holds Microsoft Word and PDF documents comprising this manual. This folder has an "index.htm" file which if opened in a browser shows table of contents with hyperlinks to each chapter of the manual.

Sadly the simulator only works under Microsoft Windows - the GUI was not designed to be portable. (The simulator started as a project to learn F# rather than an exercise in writing a portable system.)

There are two ways to run simulations. Either running under Visual Studio (debugging) control or as a standalone binary.

If running under Visual Studio, load simulator.sln as a solution to make the simulator the current solution/project. Start the simulator via the Debug menu and as the first command type "DEMOS". This will locate and make the folder of demonstrations the working folder. Then use the simulator "LS" and "CD" commands to move around the demo folder and its sub-folders and the command "DO" to run an individual demo, as in "do demo1".

If running standalone, copy the folder simulator/bin/Release/net471 to a convenient place with a possibly more helpful name, e.g., SIM900. Note that you need

all of the contents of this folder to stay together as they comprise the F# run-time environment for the simulator.

Then also copy the folder Demos into this new folder. Running a command shell, make the new folder current e.g., "cd ...\SIM900", and then type SIM900 to run the simulator. As with Visual Studio, use the DEMOS command to make Demos folder the current folder.

If the Demos folder is not found because you have moved it somewhere outside the folder hierarchy containing the simulator use the "CD" command with the path to its location.

To begin writing sample programs, take one of the demo scripts (.DAT files) in the appropriate sub-directory, copy it to a new file, still with a .DAT extension) and the edit the copy to become your script. For example, to write a simple ALGOL program, copy DEMO1.DAT in the 903ALGOL sub-directory to MYDEMO.DAT and then edit the program in MYDEMO as required.

## THE COMMAND LINE INTERPRETER.

The simulator is controlled by a command interface accessed as a console window. Commands can be typed in interactively or read from a file with simple parameter substitution if required.

When reading from a file any error condition returns control to the top, interactive level.

The commands provided essentially represent the facilities of control panel for the physical machine and are generally named after the individual controls whose effect they implement. In addition there are some additional facilities for debugging purposes.

Commands can be typed either as their full (long) name or an abbreviation. This is shown in the table below by putting the characters of the abbreviated name in upper case.

Upper and lower case are not significant in commands.

Where a command takes a <telecode> parameter, it can be one of ACD, 900, 903 or 920. A <mode> parameter can be any of MODE1, MODE2 or MODE3. Telecode parameters control how textual input (in Windows Unicode) is mapped on the paper tape codes used on Elliott machines. Mode parameters determine how the individual rows of punched tape symbols are mapped into the accumulator on input – more details are given in the later section on character codes.

| COMMAND | EFFECT |
|---|---|
| ALGol ON | Turn on ALGOL intermediate code tracing. Requires ALGOL Interpreter to be in store. Assumes AJH version of ALGOL Interpreter. |
| ALGol OFF | Turn off ALGOL intermediate code tracing. |
| ARCHIVE | Select SoftwareArchive folder as current folder, if available |

ATtach MT n <file> [WP]
: Set up a magnetic tape file and associate with magnetic tape handler n (n=0,1,2,3). If the file exists it is opened, otherwise a new file is created. If WP is included, the tape is opened for reading and writing (as if a write permit ring had been fitted).

ATtach PLT
: Start a new plotter window.

ATtach PTP FILE <file>
: Direct further paper tape output to the specified file. The representation used is determined by the file extension (any one of .900, .903, .920, .RAW, .BIN, .RLB, .DAT or .TXT).

ATtach PTP FILE <file> <telecode>
: Direct further paper tape output to the specified file. The representation used is determined by the telecode parameter, i.e., one of 900, 903, 920 or ACD).

ATtach PTP FILE <file> LEGIBLE
: Direct further paper tape output to the file <name> translating to the "legible" format.

ATtach PTR FILE <file> [<mode>]
: Read named file and use the contents for future paper tape input. The assumed representation is implied by the file extension (any one of .900, .903, .920, .RAW, .BIN, .RLB, .DAT or .TXT). If specified the request reader mode is applied (MODE1, MODE2 or MODE3). MODE3 is the default.

ATtach PTR FILE <file> <telecode> [<mode>]
: Read named file and use the contents for future paper tape input. The assumed representation is implied by the telecode parameter, i.e., one of 900, 903, 920 or ACD). A mode (MODE1 or MODE3) can be specified for 920 telecode.

ATtach PTR INLINE [<telecode> [<mode>]]
: Read the lines following up until a terminating line consisting of <!!> (only) and use for future paper tape input. By the default the lines are interpreted as 900 telecode. A mode (MODE1 or MODE3) can be specified for 920 telecode.

Attach PTR INLINE BIN [mode]
: Read the lines following up until a terminating line consisting of <!!> (only) and interpret as binary character codes. A mode can be specified. MODE 3 is the default.

Attach TTY CONSOLE
: Take further teletype input from the teleprinter window. The input is interpreted according to the current telecode default.

ATtach TTY INLINE
: Read the lines following up to and including the first line ending with <!!> and use for future teletype input. The lines are interpreted according to the current telecode default. As characters are read, the simulator will echo them to the teletype output mimicking the effect of reading a paper tape through a teletype.

ATtach TTY INLINE BIN <mode>
: Read the lines following up to and including the first line ending with <!!> and use for future teletype input. The lines are interpreted as binary format. As characters are read, the simulator will echo them to the teletype output mimicking the effect of reading a paper tape through a teletype.

AUTO
: Simulates turning the computer on in "AUTO" mode (in which the JUMP and other control buttons is disabled. In AUTO mode the computer automatically jumps to location 8177 when switched on). (This command is just syntactic sugar for JUMP 8177).

Break OFF [<address>]
Disable breakpoint (if any) at specified address. If no address is specified all current breakpoints are disabled.

Break ON <address> {<address>}
Set breakpoints at specified addresses

Comment <text> Treat line as commentary.

ChangeDirectory <name>
Set current directory.

Clear [<address>] Clear the 8K module of store containing address: if no address specified clear all of store.

COMPare master copy Compare files to detect paper tape punching errors. Assumes copy file is result of reading back punching master file and reports single dropped characters or mispunched characters in the copy file. Other errors terminate the command. Both files must have .RAW, or .BIN, .RLB, .TXT, .DAT, .900, .903, .920 or .ACD as their file extension.

DEMOS Select DEMOS folder as current folder, if available.

DEtach CRD Close the file associated with the card reader and put the reader in the manual state.

DEtach LPR Finalize line printer output on current printer window and use a new window for future printing. Put the line printer into the manual state.

DEtach MT n Close the file associated with magnetic tape handler n (n=0,1,2,3) and put the handler in the manual state.

DEtach PLT Finalize plotting on current plotter window and use a new window for future plotting.

DEtach PTP Disconnect punch from output file, if any.

DEtach PTR — Disconnect reader from input file, if any.

Detach TTY — Close current teletype window, if any. (A new window will open for any subsequent teletype I/O request)

DIRectory — List contents of current directory.

Display — Show current value of registers.

Display <address> [<address>]
Show the contents of memory at a specified address or between a range of addresses. (Note the alphanumeric representation of a word uses 900 telecode symbols.)

DO <file> <parameters>
Execute the specified command file. The parameters can consist of a sequence of words. Within the command file formal parameters are written as %n representing the n-th actual parameter, counting the first as %1. (%0 can be used to access the file name).

DumpAsSir <file> [<literals>]
Output the contents of memory to the specified file in a form suitable for assembly using SIR. Some attempt is made to reverse compile the binary using simple heuristics. If specified, literals should be an address marking the start of an area of store to be treated as if filled by SIR literal constants. This option only works with programs 8K words long or less.

DUmpimage <count> <name>
Write a file containing the contents of the specified number of words of memory starting from location 0 suitable for reloading with the LOADIMAGE command.

Enter <address>
Store the value of the word generator at the specified address.

Enter <register> <value>

Store the word generator to the specified value and store it in the specified register, A, Q, B or S. Value may be a natural number, a signed integer, a signed decimal fraction, an octal group or a quasi instruction in the SIR assembler syntax.

Enter <address> <value>
Set the word generator to the specified value and store it in the specified memory location. Value may be a natural number, a signed integer, a signed decimal fraction, an octal group or a quasi instruction in the SIR assembler syntax.

FAST — Run the simulation as fast as possible.

FIND <value> — Search the entire store reporting locations whose contents are equal to the specified value. Value may be a natural number, a signed integer, a signed decimal fraction, an octal group or a quasi instruction in the SIR assembler syntax.

FIXRLB <from> <to> [<mode1>]
Scan the from file, which should be .BIN, .RLB or .RAW and is assumed to be in RLB format and correct any checksum errors, putting result in the to file. If mode1 is included, the files are read and written in mode 1 rather than mode 3.

HELP [<command>] — Give information about requested command (all commands if none specified).

INTerrupt <level> — Simulate an interrupt on the specified level (in range 1-3).

Jump [<address>] — If an address is specified, set this address on the word generator keys. Jump to the address on the word generator keys, forcing execution at level 1 for all models except 920C. The address may be a natural number, a signed integer, a signed decimal fraction or an octal group.

Jump II
: Only defined for the model 920C – jumps to 8181 forcing level 1.

Keys <value>
: Set word generator keys to specified value. Value may be a signed integer, a signed decimal fraction, an octal group or a quasi instruction in the SIR assembler syntax.

LEGible <file> <text>
: Write text as a legible header at the front of the specified file.

ListDirectory
: List contents of current directory.

LOadimage <file>
: Load a memory image from a file in the format produced by DUMPIMAGE. (Note words 0-7 are NOT updated by the image).

LoadModule <n> <file>
: Load the specified module of memory with the contents of the specified file (which is expected to be in Don Hunter's module file format).

LOG OFF
: Switch off logging of simulator program functions (used for debugging the simulator).

LOG ON
: Start logging simulator program functions (used for debugging the simulator).

LOG <file>
: Save logging data to specified file (otherwise logging information is sent to the console window).

Monitor OFF
: Cancel all monitor points.

Monitor OFF <address>
: Cancel monitor point at specified address.

Monitor ON <address> <region> {<region>}
: Set monitor point at address. Whenever the specified location is executed as an instruction, the contents of the registers are displayed, followed by the specified memory regions. Each region can take one of the following forms:

| | | |
|---|---|---|
| | | <address>: display single location<br><address>/<address>: display address range. The address can take two forms: n or n1+n2+..+n1 where each n is an unsigned decimal number. In the second form, the specified address is calculated by reading the contents of location n1, adding n2 to the result to compute a new address, reading the contents of that address and so on up to nn. Minus (-) can be used in place of + in which case the increment is subtracted. (This command is based on the Elliott QMON / MONITOR utility program.) |
| NewLine | OFF | On teletype input only pass through the carriage return part of an end of line sequence. |
| NewLine | ON | On teletype input only pass through the line feed part of an end of line sequence. |
| NonStop | OFF | When running commands from a file, return to interactive input on any error or warning message from the simulator (default). |
| NonStop | ON | If running commands from a file read next command from file on any error or warning message from the simulator. |
| NonPrinting | OFF | Turn off visible rendering of non-printing characters in output (e.g., suppress displaying halt code as <! Halt !>). |
| NonPrinting | ON | Turn on visible rendering of non-printing characters in output (e.g., display halt code as <! Halt !>). |
| Obey | | Obey the bit pattern on the word generator as an instruction. |
| Obey value | | Set the word generator to the value and execute the value as an instruction. Value may be a natural number, a signed integer, a signed decimal fraction, an octal group or a quasi instruction in the SIR assembler syntax. |

OFF — Turn simulated computer off.

ON <architecture> [<memory> [<speed> [<reader>]]]

Turn simulated computer on. If no other options are specified match the behaviour of the Froggatt/Hunter generic simulator (with 903 instruction times).

<architecture> determines which version of the Elliott 900 series is emulated; it can be one of 900, 920A,920B, 920M, 920C, 903, 905, ARCH 9000, ARCH 9050.

<memory> is the system memory capacity: it can be one of 8K, 16K, 24K, 32K, 48K, 56K, 64K, 80K, 96K, 128K. (Sizes above 64K are only available with 920C/905/ARCH9050). The default is 64K.

<speed> is the cycle time of the memory: it can be any of 1, 2, 6 or 7 (microseconds) – 7 for 920A, 6 for 920B/903/ARCH9000, 5 or 2 for 920M, 2 or 1 for 920C/905/ARCH9050. 0 selects the fastest memory for the processor model.

<reader> is the paper tape reader speed in characters per second: it can be one of 250, 500, 1000. 1000 is the default.

ORIGIN x y — Set the plotter pen position to x, y relative to 0, 0 at the bottom left hand corner of the plotter window. For 900 plotting the default origin is 320, 160 and for all other models: 0, 0.(Starts a new plotter window).

Pause — Prompt the user to type a RETURN before going on to the next command.

Print <file> — Pretty print a translated version of the specified file based on its extension (.ACD, .900, .903, .920, .BIN, .RLB, .RAW, .DAT, .TXT). DAT and TXT are treated as 900. If the first decoded byte of a BIN or RAW file is a 900 or a 920 telecode newline, the file is translated to text.

Otherwise it is printed out in binary mode format. RLB (Mode 3 relocatable binary) files are printed out in a format which displays the loader directives in a SIR like representation.

QCheck from to format [from to format ...]
Output store in specified region as SIR code, encoding each word according to the format code: O for instruction (order), F for fraction, I for (signed) integer or B for octal. A patch directive is output before the dump, and between regions that are not consecutive. (This command is based on the 900 series QCHECK utility program.)

Quit — Exit the simulator.

RESET — Reset the computer.

Restart — Resume execution after a stop.

Restart addr — Resume execution, stopping when SCR equals the specified addr.

REWIND — Rewind the current paper tape input source, making the first character of the source the next one to be read.

RLBTOSIR source destination [MODE1]
Read RLB from source file and print as SIR equivalent in destination file. If MODE1 is present the source file is read in Mode 1 otherwise Mode 3. The source file must have a .BIN,.RAW or .RLB extension.

RUNOUT OFF — Stop adding leading and trailing runout to text input (file or inline).

RUNOUT ON — Add leading and trailing runout to text input (file or inline).

SCALE n — Scale plotter output by factor 1/n. (Starts a new plotter window).

SCALE a b — Scale plotter output by a/b. (Starts a new plotter window).

| | |
|---|---|
| SCBdecode | Decode the sum checked binary tape in the simulated paper tape reader. This decoder understands a number of standard Elliott loader formats and can be used to analyse the contents of a binary tape. |
| SELect IN <device> | Select one of PTR (reader), AUTO or TTY (teletype) for paper tape input. |
| SELect OUT <device> | Select one of PTP (punch), AUTO or TTY for paper tape output. |
| SHow Breakpoints | Display the currently active breakpoints |
| SHow Configuration | Display the configuration of the emulated machine (architecture, memory size and cycle time, reader speed). |
| SHow Monitors | Display the currently active monitor points. |
| SHow Times | Display simulated and actual execution times since last RESET of SHOW TIMES. |
| SKIP | Scan forward on the tape currently attached to the paper tape reader looking for a non-blank, then continue scanning until a contiguous block of 20 blanks is found and halt after it. (Generally used to skip over legible headers). |
| SLOW | Run the simulation at the real speed of the simulated machine. |
| STacK | Output a summary of the ALGOL Interpreter stack. Assumes AJH or HUNTER ALGOL Interpreter is loaded. |
| Step <count> | Set a limit on the number of instructions that can be executed following the next JUMP or RESTART. Execution halts earlier if there is a machine stop (e.g., due to reaching a loop stop, an I/O problem or illegal instruction etc., otherwise continues until <count> further instructions have been executed. If the |

simulation is stopped and no <count> is specified the next instruction in sequence is executed as a single step and the machine remains stopped.

STopped <address> Continue with next command in a command file if machine is in a loop stop at the specified address otherwise return to interactive input of commands.

SWAPXY Plot with Y as the horizontal axis rather than X. (Starts a new plotter window).

TeleCode <telecode> Set the default telecode to one of ACD, 900, 903, 920.

TIDYup Clear all break points and monitor points. Turn tracing off. Clear STEP count. Set NONPRINTING ON and NEWLINE ON.

TOBINary <file> Convert a .RAW, .ACD, .TXT, .DAT, .900, .903 or .920 file to a binary format file with same prefix (i.e., TEST.900 generates TEST.BIN). .TXT and .DAT are interpreted as 900 telecode.

TORAW <file> Convert a .BIN, .RLB, .ACD, .TXT, .DAT, .900, .903 or .920 file to a raw format file with the same prefix (i.e., TEST.900 generates TEST.RAW). .TXT and .DAT are interpreted as 900 telecode. Leading and trailing runout in the source file is removed. A fixed leader and trailer, both of 180 blanks, is written.

TOTELEcode <file> <telecode>
Reads in a file and output a copy file in the specified telecode. The input file must have a .ACD, .BIN, .RAW, .DAT, .TXT, .900, .903 or .903 extension.

TRace OFF level Turn off instruction by instruction tracing for the specified interrupt level (1-4). If no level is specified all tracing is stopped.

TRace ON level Turn on instruction by instruction tracing of the simulated machine’s execution for

| | |
|---|---|
| | the specified interrupt level (1-4). If no level is specified, tracing is turned on for all levels. |
| TraceBack | Produces a list of the most recently executed instructions and the value of the accumulator following the instruction. |
| TraceINTerrupt <level> ON | Turn on trace interrupts for the specified level. |
| TraceINTerrupt <level> OFF | Turn off trace interrupts for the specified level. |
| TraceRegion <from> <to> | Limit tracing to instructions in specified region of store. |
| TraceRegion OFF | Suppress region tracing – i.e., trace instructions from any location in store. |
| TTYtelecode telecode | Set telecode for translating teletype output sent to console, can be on of 900, 903 or 920. |
| Watch OFF | Turn off memory watch. |
| Watch ON location | Break execution if specified word in store is accessed. |
| VerifyImage file | Compare image file against memory and report differences (halts if more than 50 found). |

Whenever the simulator stops execution, e.g., on arriving at a breakpoint, hitting a step count, or detecting an error condition control returns to the interactive console and execution can be resumed using a RESTART command if it possible to do so.

If the simulated program executes a loop stop this is detected and control returns to the next command if executing a command file or to the console otherwise. The STOPPED command can be used to make further execution of a file conditional on stopping at a specified address.

If a paper tape input instruction detects no data is available for input control returns to the interactive console. An input source can be attached using the ATTACH command and the program resumed by RESTART. Similarly if a paper tape punch output instruction finds no output destination is available, control returns to the interactive console. An output destination can be attached using the ATTACH command and the program resumed by RESTART.

## SIMULATED HARDWARE AND DEVICES

The following sections explain how the simulator replicates the 900 series hardware.

### Initial Instructions.

The Froggatt/Hunter simulator simplified the handling of initial instructions by simply copying them to memory at the start of a simulation run, and some of Don Hunter's programs rely on the initial instruction locations not being over written by use of the JUMP button. In the current simulator, the 900 model copies the initial instructions to store ONLY on a jump to 8181 to achieve the same effect. Note that some Elliott programs, such as 903 FORTRAN, won't run on the 900 model as they require the precise behaviour of the physical machine to work correctly.

### Peripherals.

The simulator supports the basic paper tape and teletype facilities, the multiplexor, digital graph plotter, card reader, line printer and magnetic tape devices.

## PAPER TAPE FACILITIES.

There were several different paper tape character codes used during the lifetime of the 900 series (called "telecodes" by Elliotts). The oldest, 920 telecode, was a seven bit code based on that used on Elliott 803 and 503 computers. This was followed by 903 telecode based on an early version of eight bit ASCII, later revised to become 900 telecode which remains close to modern ASCII. The simulator provides means to input representations of all these telecodes and also a hybrid code adopted by Elliott's Airborne Computing and Maritime Aircraft Systems Divisions). More details are given in the section on character codes below.

### Reader and punch.

The simulator allows Windows ASCII and Unicode text files to be connected to the simulated paper tape reader and punch using the "ATTACH PTR FILE ..." and "ATTACH PTP FILE ..." commands (and to be disconnected using the corresponding DETACH commands).

In addition to attaching files, there is an 'INLINE' variant of the ATTACH command (e.g., "ATTACH PTR INLINE") that allows input to be taken from a simulator command script.

If a file is associated with the paper tape reader it can be rewound, using the REWIND command which makes the first character in the file to be the next one read.

The default interpretation of an attached file is determined by its extension, with extensions of .ACD, .900, .903 and .920 indicating text files containing Unicode characters representing symbols in the corresponding telecode. The extensions .RAW, .BIN and .RLB denote a further set of file formats for holding binary representations of paper tape, for example, tapes to be loaded using initial instructions. These formats are explained in more detail later on.

In addition to the character code it is possible to specify the mode in which a tape reader is to be operated.

For INLINE input, the telecode defaults to 900 telecode unless this has been overridden by the TELECODE command, in which case that specified by the TELECODE command is used, unless itself overridden by a telecode option specified in the ATTACH command itself.

The extensions .DAT and .TXT are synonyms for .900. In the examples .DAT is generally used for simulator command scripts and .TXT for data files.

If the simulator detects the same character being repeatedly punched many times it halts with a warning message – this catches programs that use repeated punching of binary codes to signal error conditions (e.g., some of the test programs).

## Teleprinter.

Teleprinter input and output is via a teleprinter window that automatically pops up when the device is first used.

Teleprinter input can be taken from a file or from a command script if required using the ATTACH TTY FILE ... and ATTACH TTY INLINE commands.

Alternatively teletype input can be taken interactively from the teletype window by using the ATTACH TTY CONSOLE command. (And the TELECODE command should be used to determine how to map input characters to one of the Elliott telecodes).

The DETACH TTY command can be used to undo any of these associations (and an error will be signalled if further teleprinter input is requested before a new association is made).

The simulated teleprinter operates in half-duplex mode, so programs should not echo back input characters. On the real machine half versus full duplex was a switchable option on the teleprinter device.

## SELECT switches.

The function of the SELECT switches is provided in the simulator by use of the SELECT IN and SELECT OUT commands (e.g., SELECT IN TTY, SELECT OUT AUTO). Note the binding of a file to a device using the ATTACH command is completely independent of the setting of the SELECT command.

## Character Codes and simulator file formats.

The simulator allows paper tape (and teleprinter) input from and output to files in a number of formats: RAW and BIN are simple binary encodings; 920, 903, 900 and ACD are Unicode representations for the various Elliott telecodes. The simulator automatically maps the characters in the text to the binary code of the corresponding character in the Elliott telecode.

Each telecode is tabulated below. In some cases there are alternative Unicode symbols that stand for a single telecode symbol: this is intended to simplify input from a keyboard where some of the more obscure characters are not available.

The Unicode representations use the escape sequence <! Halt !> (or <! HALT !> or <! H !>)to represent a halt (stop) code. The sequence <! Bell !> is used to represents a bell character and <! R !> for blank tape (runout). During textual output codes that do not correspond to symbols in the selected telecode are output in the form <! ddd !> where ddd is the value of the code in decimal. Input of such sequences is not currently supported.

The escape sequence <! LEGIBLE .... !> is used to allow legible headers to be represented. When used, eight such sequences of equal length should appear as adjacent lines of text, each representing a row of holes in the header, and when read as columns giving a visible rendition of the legible header. On input these sequences are skipped over transparently, as if the operator had positioned the tape immediately after the header.

You should be wary of how some of the special characters are handled in Windows command windows.

To be fully sure of what is stored, save text files as Unicode rather than ANSI or UFT-8 (the simulator uses UTF-16 internally).

The RAW file format is simply a sequence of bytes, directly representing rows of paper tape.

The BIN (for 'binary') format encodes each character as a decimal integer in the range 0-255 with white space separating successive characters.

Binary files can include comments – that is arbitrary text bracketed by ( and ) which is skipped on input. Comments can include any character other than further closing brackets since these would taken as terminating the comment.

The form (! LEGIBLE ... !) is used to represent legible headers in binary files analogously to text files.

Binary files can also be given the extension .RLB: this is used to distinguish 'relocatable binary' tapes produced by compilers and assemblers from other kinds of binary tape. In particular the simulator PRINT command interprets and checks .RLB files rather than just printing the individual codes.

There is also the option to produce punch output in a "LEGIBLE" format that represents the pattern of holes punched on physical tape.

By default the simulator does not add any blank leading or trailing runout to text input (inline or from a file). However some programs, such as the 903 EDIT utility, regard leading and trailing runout as significant. In the case of binary or raw files blanks can be edited in manually. For text files the RUNOUT ON command can be used to automatically precede any text by 30 blanks (3" tape) and follow it by 60 blanks (6" tape).

Note on legible headers

The programs used to read in original Elliott paper tapes attempt to automatically locate legible headers at the start of tapes and put them in the <! LEGIBLE ... !> or (! LEGIBLE ... !) form as appropriate depending on whether the rest of the tape is detected to be in a recognisable telecode or not.

If programs are run on the simulator that produce legible headers these are not automatically detected and the legible characters will simply appear as characters in the output. The SKIP command scans any tape currently attached to the paper tape reader looking for the first non-blank and then scans forward until it has read a contiguous block of 20 blanks: this is generally sufficient to get past any legible header.

## 920 Telecode.

920 telecode is based on Elliott 503 telecode for compatibility with the older Elliott 503 and 803 computers.

The code values are those when reading 920 telecode characters in Mode 1.

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | nul | , | 4 | - | H | R | | | | º | @ | d | n | x |
| 1 | | # | £ | 5 | . | I | S | | | | < | e | o | y |
| 2 | lf | : | 6 | ; | J | T | \| | | | > | f | p | z |
| 3 | pt | & | 7 | A | K | U | | | | ↑ | ^ | g | q | |
| 4 | tab | * | 8 | B | L | V | sp | | | | ~ | h | r | |
| 5 | bsp | / | 9 | C | M | W | | | | | % | i | s | |
| 6 | | 0 | ⑩ | D | N | X | | hlt | | | ? | j | t | |
| 7 | | 1 | ⑪ | E | O | Y | | | | | a | k | u | |
| 8 | ( | 2 | = | F | P | Z | | | [ | | b | l | v | _ |
| 9 | ) | 3 | + | G | Q | | | | ] | | c | m | w | del |

nul = null / blank / runout, lf = line feed, pt = paper throw, tab = horizontal tab, bsp = backspace, sp = space, hlt = halt/stop code, del = delete / erase.

º represents the subscript-10 symbol which does not appear in Unicode. ⑩ is permitted as an alternative to °.

⑩ and ⑪ represent 10 and 11 as single characters that also do not appear in Unicode.

| and _ are non-escaping in 920 telecode (i.e., the next character is treated as an overstrike).

Escape sequences

Other Unicode symbols are recognized and translated into 920 telecode escape sequences using the | symbol:

! -> |., # -> |=, $ -> |S, ‘ -> |<, ‘ -> |<, ` -> |>,
’ -> |>, ½ -> |2, @ -> |a, { -> |6, } -> |9

Only the |<, |> and |S escapes are used by Elliott software (since quote, unquote and dollar are all in the SIR internal code).

## 903 Telecode.

903 telecode is based on an early variant of ASCII. In comparison to modern ASCII and Unicode, acute quote, at symbol and the sterling symbol are in the wrong place and two unfamiliar symbols, half and subscript 10 are included in the code.

The code values shown in the table are those when reading 903 telecode in Mode 2, i.e., with parity removed.

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | nul | lf | | | ( | 2 | < | F | P | Z | d | n | x |
| 1 | | | | | ) | 3 | = | G | Q | [ | e | o | y |
| 2 | | | | sp | * | 4 | > | H | R | £ # | f | p | z |
| 3 | | | | ! | + | 5 | ₁₀ ? | I | S | ] | g | q | |
| 4 | | | | " | , | 6 | ’ ` | J | T | ↑ ^ | h | r | |
| 5 | | | | ½ | - | 7 | A | K | U | ← _ | I | s | |
| 6 | | | | $ | . | 8 | B | L | V | @ | j | t | |
| 7 | bel | | | % | / | 9 | C | M | W | a | k | u | del |
| 8 | | | | & | 0 | : | D | N | X | b | l | v | |
| 9 | tab | | | ‘ ’ | 1 | ; | E | O | Y | c | m | w | |

nul = null / blank / runout, bel = bell, tab = horizontal tab, lf = line feed, hlt = halt/stop code, cr = carriage return, sp = space, del = delete / erase.

₁₀ represents the subscript-10 symbol which does not appear in Unicode. ❿ is permitted as an alternative to °.

Where alternative characters are shown, the first is used on output; either can be accepted as input.

## 900 Telecode.

900 telecode is essentially modern ASCII.

The code values listed below are those when reading 900 telecode in Mode 2, i.e., with parity removed.

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | nul | lf | | | ( | 2 | < | F | P | Z | d | n | x |
| 1 | | pt | | | ) | 3 | = | G | Q | [ | e | o | y |
| 2 | | | | sp | * | 4 | > | H | R | \ | f | p | z |
| 3 | | | | ! | + | 5 | ? | I | S | ] | g | q | { |
| 4 | | | | " | , | 6 | @ | J | T | ^ ↑ | h | r | \| |
| 5 | | | ½ | # £ | - | 7 | A | K | U | _ ← | I | s | } |
| 6 | | | | $ | . | 8 | B | L | V | ` ’ | j | t | ~ |
| 7 | bel | | | % | / | 9 | C | M | W | a | k | u | del |
| 8 | | | | & | 0 | : | D | N | X | b | l | v | |
| 9 | tab | | | ' ‘ | 1 | ; | E | O | Y | c | m | w | |

nul = null / blank / runout, bel = bell, tab = horizontal tab, lf = line feed, hlt = halt/stop code, cr = carriage return, sp = space, del = delete / erase.

Where alternative characters are shown, the first is used on output; either can be accepted as input.

## ACD Telecode.

ACD telecode is an internal code used by ACD which merges 903 and 900 telecode, into a unified code. Some symbols are duplicated in the code to allow for 903 or 900 symbol input: on output the 900 code version is always used.

The code values are those when reading 900 telecode in Mode 2, i.e., with parity removed.

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | nul | lf | | | ( | 2 | < | F | P | Z | d | n | x |
| 1 | | | | | ) | 3 | = | G | Q | [ | e | o | y |
| 2 | | | | sp | * | 4 | > | H | R | £ | f | p | z |
| 3 | | | | ! | + | 5 | ° ? | I | S | ] | g | q | |
| 4 | | | | " | , | 6 | @ | J | T | ↑ ^ | h | r | |
| 5 | | | ½ | # £ | - | 7 | A | K | U | ← _ | I | s | |
| 6 | | | | $ | . | 8 | B | L | V | ` ¬ | j | t | |
| 7 | bel | | | % | / | 9 | C | M | W | a | k | u | del |
| 8 | | | | & | 0 | : | D | N | X | b | l | v | |
| 9 | tab | | | ' ‘ | 1 | ; | E | O | Y | c | m | w | |

nul = null / blank / runout, bel = bell, tab = horizontal tab, lf = line feed, hlt = halt/stop code, cr = carriage return, sp = space, del = delete / erase.

Where alternative characters are shown, the first is used on output; either can be accepted as input.

Note that £ and ` both have 2 positions in this code. Input programs should tolerate both values, and output programs should use values 35 and 96 respectively.

## Elliott internal character code.

Internally the Elliott languages, ALGOL, FORTRAN and SIR) use a six bit code, as tabulated below. The Elliott 903 telecode value is shown for each character in the code: by reference to the Elliott 920, 900 and ACD telecode tables above, the corresponding symbol in these telecodes can be readily identified. Note that 920 telecode does not include direct equivalents to $, ‘ (39),and ’ (96) so these are represented by escape sequences |s, |< and |> respectively.

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 |
|---|---|---|---|---|---|---|---|
| 0 | sp | * | 4 | > | H | R | £ |
| 1 | lf | + | 5 | ø | I | S | ] |
| 2 | " | , | 6 | ’ | J | T | ↑ |
| 3 | £ | - | 7 | A | K | U | ← |
| 4 | $ | . | 8 | B | L | V | |
| 5 | % | / | 9 | C | M | W | |
| 6 | & | 0 | : | D | N | X | |
| 7 | ‘ | 1 | ; | E | O | Y | |
| 8 | ( | 2 | < | F | P | Z | |
| 9 | ) | 3 | = | G | Q | [ | |

## 903 MULTIPLEXOR.

The simulator 903 machine model provides a configuration of multiplexor, 4100 interface matching unit, with card reader and line printer, digital graph plotter and a magnetic tape controller with four tape handlers. All the devices as associated with multiplexor group A.

## 4100 PERIPHERALS.

The use of 4100 devices is limited to the 920B model.

### Card Reader.

When using the simulator, if an attempt is made to read a card with no file attached, the simulator signals an attention with the device status 'in manual'. The reader can be put back in manual using the DETACH CRD command. If an attempt is made to transfer data when no card has been loaded into the reader, a missed transfer attention is signalled; similarly if the program falls behind in reading successive columns. (The precise behaviour of the card reader is not described in any of the available Elliott documents, but this behaviour is consistent with the coding of QCARDIN and its error reports).

The simulated system runs at the document rate of 300 cards/min, but the timing of interrupts within a card is a best guess as no specification for these has been found.

### Line Printer.

When using the simulator, the line printer will appear to be in manual until the ATTACH LPR command is given. Line printer output is directed to a new window that pops up when the device is first used. The line printer can be taken back to manual using the DETACH LPR command.

Note the simulator reports an error if an attempt is made to initiate a transfer when the printer is busy – the effect on a real 903 + 4100 line printer is not described in any of the available documentation.

## GRAPH PLOTTING.

he simulator uses single pixels to represent plotter steps.

The simulator has two plotting models, one associated with the generic 900 machine type, the other for all other machine types, including 920A, B, C and M. Both models plot to a graph plotter window that opens when the device is first used. A new window can be forced to open by using the DETACH PLT command. (An ATTACH PLT command is provided by has no effect).

The 900 model reproduces the behaviour required to run Don Hunter's ALGOL plotting demonstrations using a modified ALGOL plotting library (in the LIBRARY.RLB file from the HUNTER directory). Don's simulator produced plotter output on a 640 by 480 pixel VGA screen using MSDOS graphics.

For all other machine types a direct emulation of the plotter hardware is provided to enable the use of the standard Elliott libraries and test programs (see e.g., DEMOX52 in the directory TEST). The plotting window measure 1800 * 1000 pixels corresponding to a physical plot of 9 ins by 5 ins.

For either model, a new plotter window is opened on the first occurrence of a plotting instruction in a simulation run. All subsequent plotting takes place in that window. If the plotter is detached (using DETACH PLT) a new window is opened on the next plotter command and all subsequent plotting is directed to the new window.

The simulator ORIGIN command can be used to set the initial pen position on a new window. For 900 plotting, the default is X=320, y=160; for all other plotting X=0, Y=0 where X is the horizontal axis and Y the vertical. The SWAPXY command can be used to swap the X and Y axes, i.e., make the Y axis be horizontal rather than vertical. The SCALE command can be used to reduce a large plot by a specified scale factor to fit the available window.

## MAGNETIC TAPE.

The simulator uses binary files to simulate magnetic tapes. Blocks are written as a header followed by data bytes. The header contains a sentinel byte (127), parity type byte, block type byte (0=data block, 1=erase, 2=tapemark), position of start of previous block in the file (3 bytes, 0 if none) and length (3 bytes). If tape files are given a .BIN or .RAW file extension they can be readily viewed using the simulator PRINT command. Magnetic tape files can be kept between runs of programs.

A tape is mounted using the ATTACH command and removed using the DETACH command. A handler with no file attached appears as if set to manual. The only 'hardware' error that can arise is if a block is written using one parity and read with another, giving rise to a parity error indication.

The Elliott documentation gives some timings for magnetic tape operations and the specification of test program XMT71 gives further information, although the two are neither complete nor consistent. The simulator adopts the following timings:

| | |
|---|---|
| Time to read/write a word | 333 usecs |
| Prepare to Read (do nothing) | 9000 usecs |
| Prepare to Read | 9333 usecs |
| Prepare to Write (do nothing) | 14000 usecs |
| Prepare to Write | 14333 usecs |
| Read/Write (per word) | 333 usecs |
| Close Block (after reading) | 11800 usecs |
| Close Block (after writing) + | 20000 usecs |
| Erase | 111000 usecs |
| Back Space ++ | 20800 usecs |

\+ 25 ms if final write was a Write Word operation (required to make XMT71 succeed).
++ plus time taken to read characters of previous block, if any.