KPgolem

Overview

Interface hardware and software to punch and read on IBM Keypunch machines, initially tested with model 029 but should work, at least punching, on model 026. It is an open source design, available on Github

There are three elements to the interface:

1. Modifications to the keypunch and wiring one or two cables that will connect to the interface box. The second cable is optional and supports reading and verifying of cards.
2. An interface box based on an Arduino microcontroller and relay boards that will sit inside the keypunch, hooked by cables to the keypunch mechanism, connected by a serial cable to the user via a terminal, programs, or the optional third element, PC based software.
3. A program that runs under Windows on a desktop, laptop or Surface Pro computer, communicating over a serial port to the keypunch interface program. This program allows text files on the PC to be punched onto cards by the keypunch, as well as the reading of existing cards through the keypunch into Windows text files.

The easiest way to use the system is to combine all three elements, providing a user-friendly modern graphical interface and straightforward process for punching decks from the computer. The first part of this documentation will cover the PC based program and its use.

The protocol for the serial link that connects to the keypunch interface box was designed to make it easy to use and to debug. All messages are in ASCII with a flexible simple set of command verbs and nouns. The responses are easy to interpret without any reference materials.

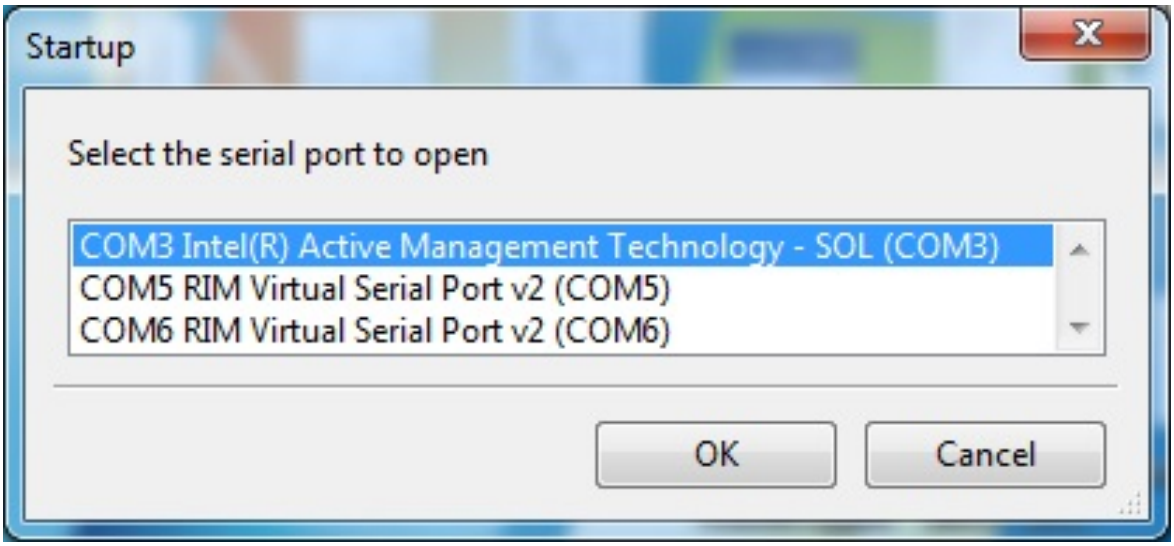
The keypunch can be completely controlled and used to its full potential through a simple terminal connection, without any programming. It is also amenable to connection and use from a variety of programs and systems.

The second part of the manual documents the protocol for the serial connection, for those who will control the interface without the included PC program. There are some additional capabilities of the keypunch interface which are not exploited by the PC based software.

PC based software to drive the keypunch

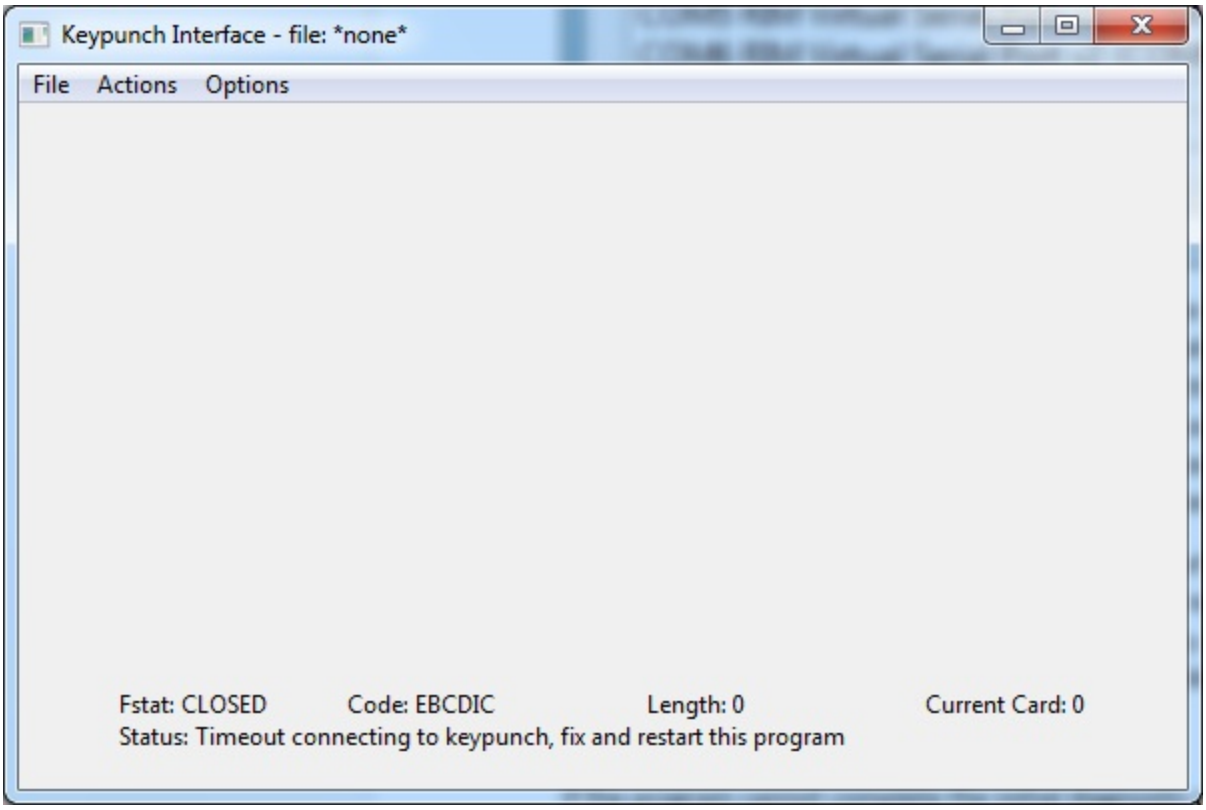
The KeyPunch Interface program is started by launching the executable file of the same name. It maintains a configuration file, creating it if necessary, to track the user's selection of serial port and encoding to relief the user from having to reselect these on each invocation.

If the configuration file (KeyPunchInterface.ini) does not have a remembered serial port, the program will begin by showing a dialog box with a list of all the available serial ports on the Windows system. The user highlights the one that is connected to the keypunch and confirms with the "Okay" button to begin using that port.



Once the port was selected, the choice is saved in the configuration file. Every subsequent start of the program will use the saved serial port and immediately connect to the keypunch, skipping the dialog box. There are two ways that the user can return to the dialog box to select a serial port – if the saved port is no longer available under Windows or if the user selects a menu option to forget the saved port. In those two cases, the configuration file entry is erased and the user is asked to close and restart the program to begin with the dialog box and choose a new serial port.

The program will try to connect to the keypunch interface, requesting a simple built in diagnostic (Diag 0) that verifies the connection of the punch and the reader cables from the keypunch mechanism to the interface box. The program has a ten second timer to catch the case where the interface box is not connected, not powered up, or there is another problem that blocks connectivity.

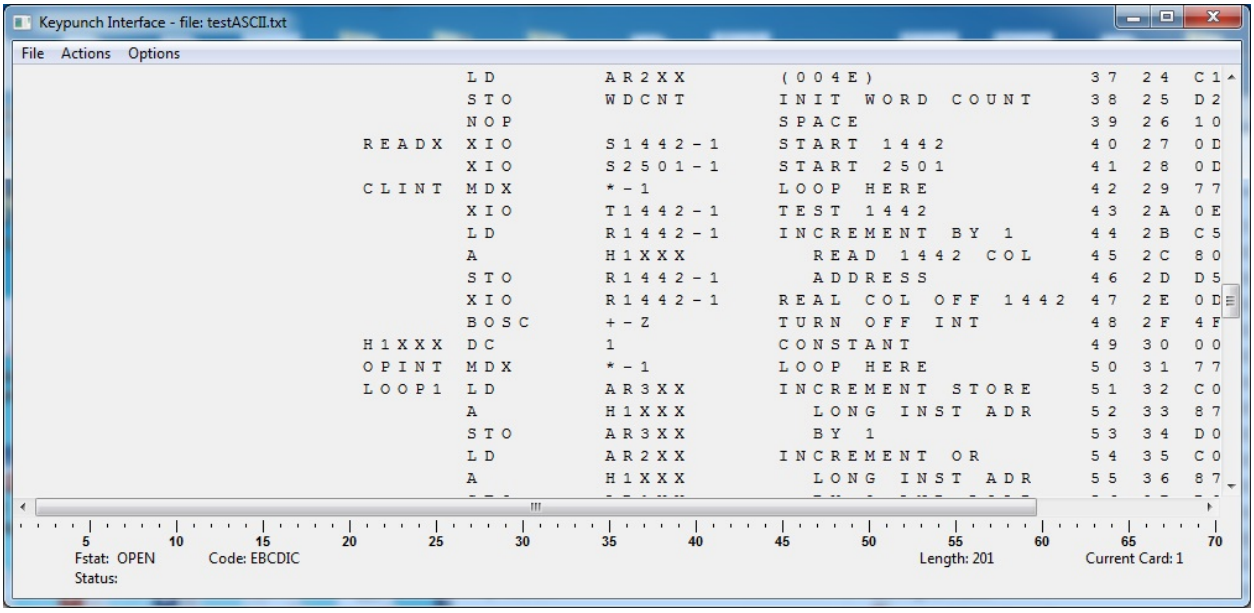


If the program cannot complete the initial diagnostic on the interface box, it informs the user of the error but will continue to run, although with reduced functionality. It makes no sense to attempt any function that needs the keypunch interface or mechanisms, so these menu choices are disabled (grayed out). The user can still open files and move around in them.

The main screen has a section across the bottom that displays key status information. It is divided into two lines across the bottom. The bottom line has a status message. Both successful actions and error conditions can post messages here. The upper of the two lines delivers the status of the program with four key metrics.

Fstat is the status of the file, either OPEN or CLOSED. Code informs the user of the encoding that is selected; this can be binary encoded in ASCII, BCD encoded in ASCII, EBCDIC encoded in ASCII or others. All allow the user to create and manage simple ASCII text files to represent punch card images. These files have a suffix of .txt allowing them to be easily edited by Notepad, Word or other text editor program.

The title bar of the main window displays the filename of the currently opened file, or *None* when files have been closed. The remainder of the window will display the card images with automatic scroll bars depending on the size of the file.



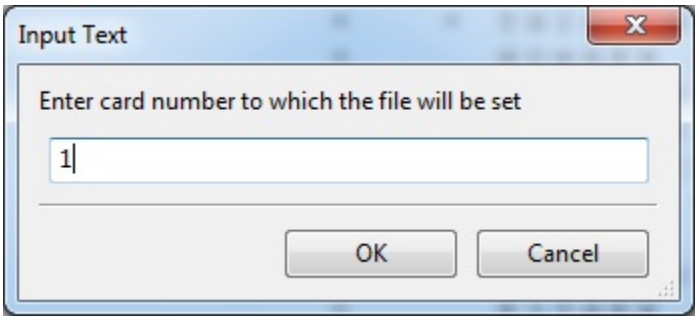
Here we can see a card deck containing 200 cards, encoded as EBCDIC in ASCII, with card number 100 as the currently active card. That means the keypunch will punch that card image onto a blank card and continue advancing through the file, sequentially from card 101 down to the last one. The window was resized to show all 80 card columns across – the right hand eight columns are card sequence numbers as was common with source program decks in the mainframe era.

There is a vertical scroll bar since all 200 card images are not visible in the window. If the window were resized with less horizontal space, such that not all 80 columns would be visible, then a horizontal scrollbar will appear.

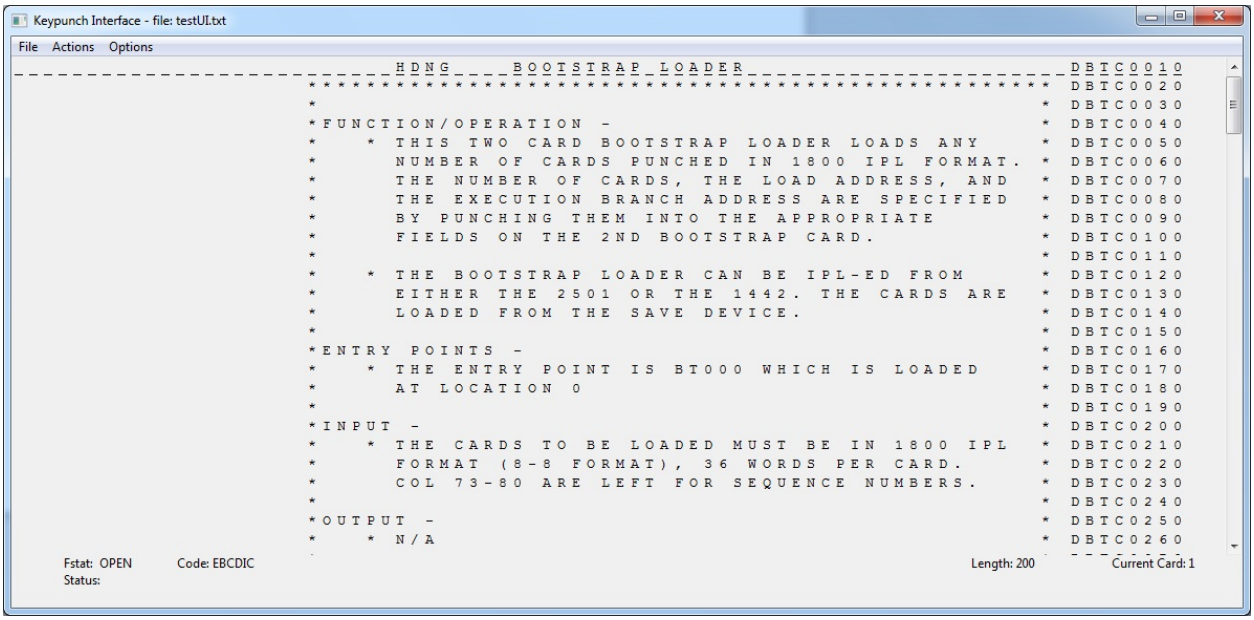
In addition, a ruler across the bottom of the window shows the card column numbers for easy reference.

Whenever the keypunch is paused, a menu choice will allow movement to a different card, which becomes the next one to be punched when punching is restarted. Files open up in paused mode, giving the user the opportunity to move away from card 1 to whatever card number they wish to begin punching.

The Go To item of the Action menu will put up a dialog box where the user enters the card number to which they wish to advance or rewind the current deck. The current card is marked with underlines, for ease of visual identification.

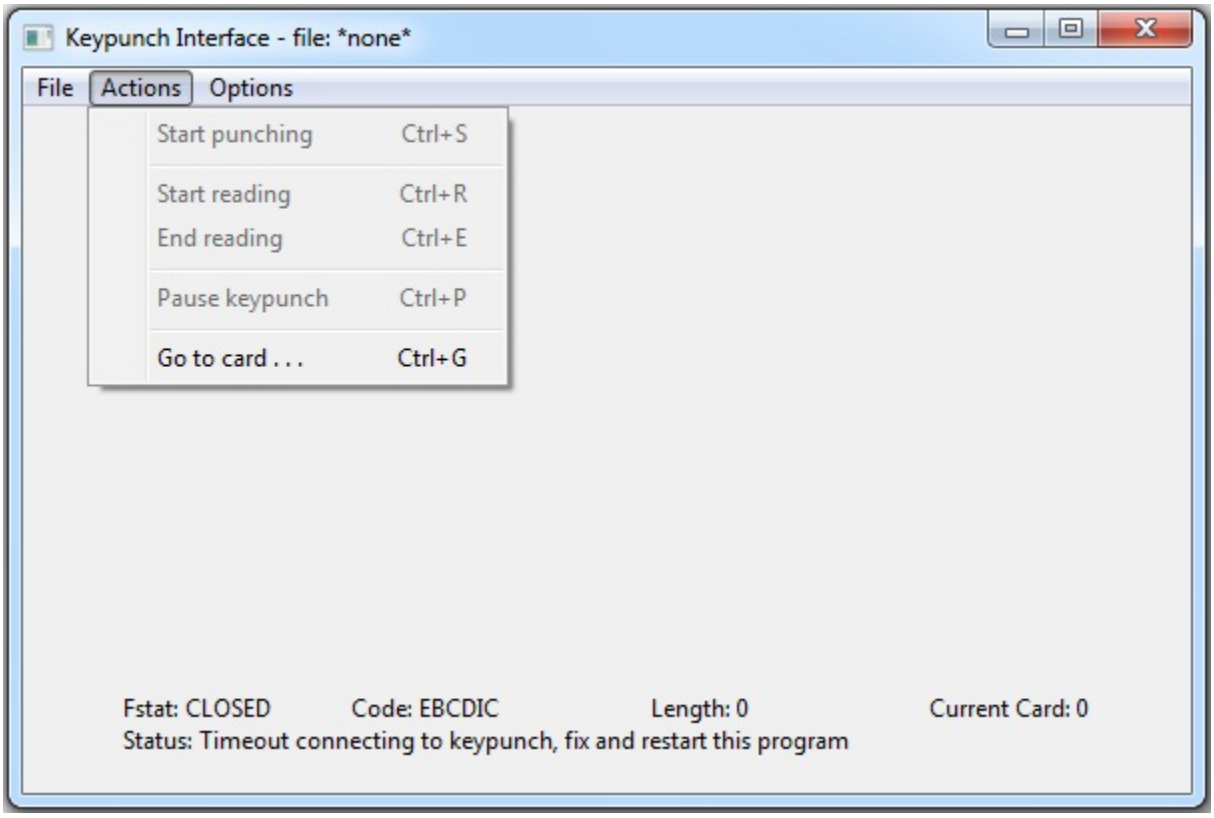


The current card number will jump back to card 1, with the card image highlighted by underlines as seen below. This is useful if a card is jammed or damaged, as the user can pause the punching, use Go To rewind to the damaged card and restart punching to get a new copy.



The program has three menus, File, Actions and Options, each of which offers a number of selectable items that control the program and keypunch. Almost every menu item has a shortcut, such as Control + Q to quit the program.

The Actions menu bar is how the user starts the keypunch punching the open file, pauses the keypunch, and changes the current card with Go To. In addition, if cards are being read on the keypunch to be stored in a newly created text file on the PC, the user would start reading instead of choosing start punch. In the screen shot, the entries are grayed out (disabled) because the program could not complete the Diagnostic 0 to verify a good connection to the keypunch interface box.



Ctrl-S starts the keypunch punching cards sequentially from the current card position, if a text file was opened with card images to punch out. If the keypunch is subsequently paused, it can be restarted by issuing Start Punching another time. This can only be issued if an existing file was opened.

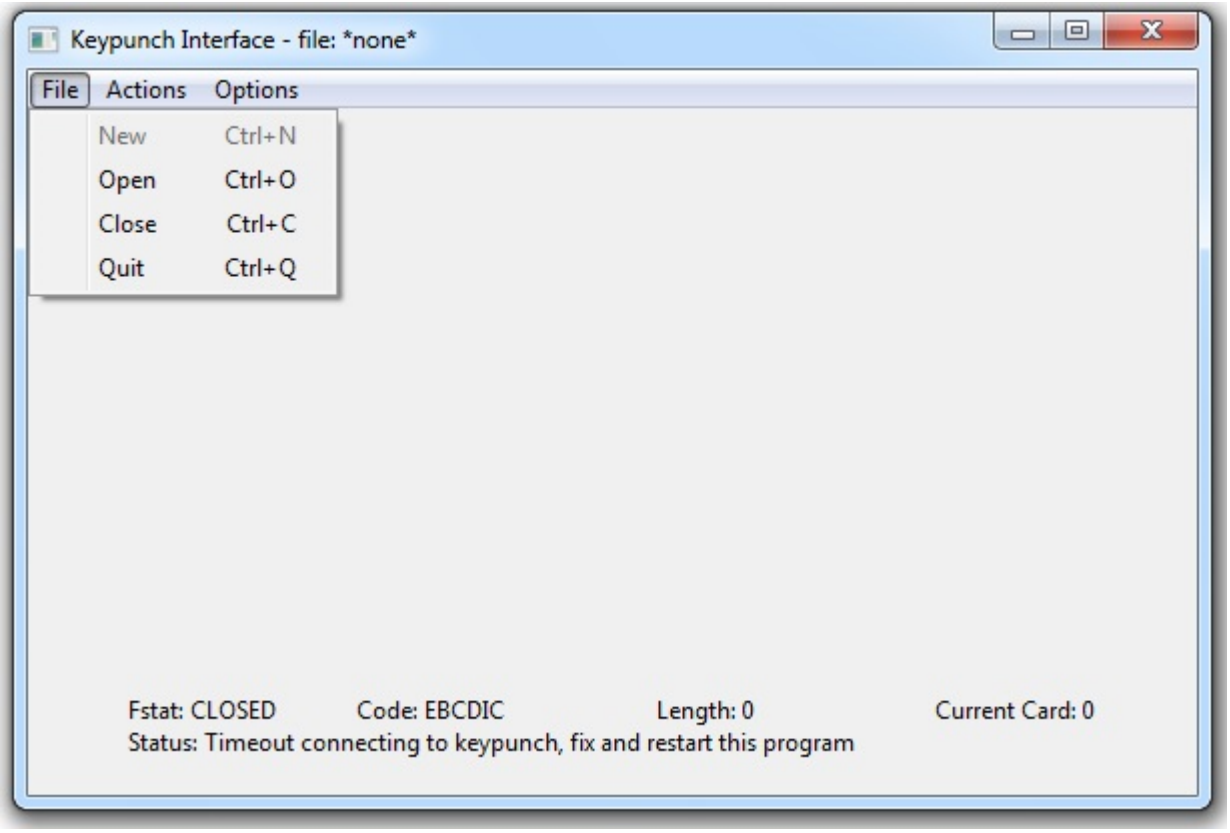
Ctrl-R, Start Reading, is used when a new empty file is created to hold card images read in from a physical card deck. The keypunch will continue to read cards, appending them to the growing file being created on the PC, until the user declares the last card has been read by selecting the End Reading menu item.

Ctrl-E, End Reading, will stop the process of reading in cards. It, like Start Reading, can only be issued if a new file was created for reading.

Ctrl-P, Pause Keypunch, will stop sending card images to the keypunch if a file is being punched, or stop reading additional cards into a file if we are reading a deck.

Ctrl-G will launch the Go To dialog box where a card number is entered to become the current card.

The File menu is used to open existing PC files for punching on cards, create new empty file to read cards into, close an open file, and quit the program.

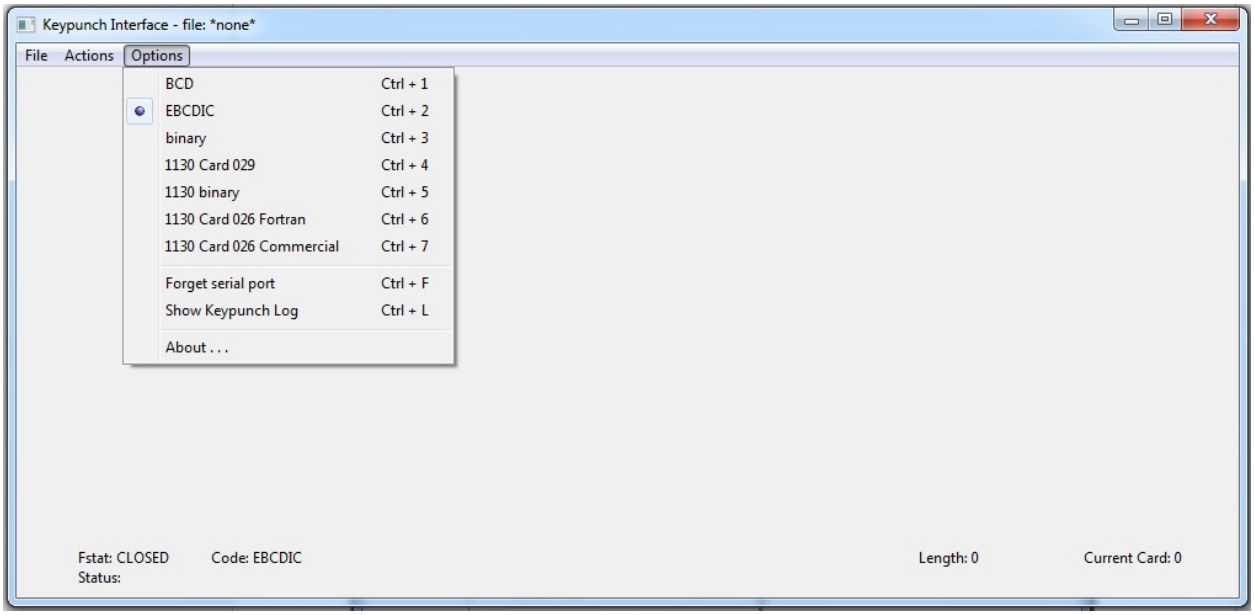


Ctrl-N creates a file that will be filled card image by card image as the keypunch reads in a card deck. It is an error if the filename already exists. This option is disabled if the program can't connect to the keypunch interface box or if the keypunch does not have the read cable connected to the interface.

Ctrl-O opens an existing text file, displaying the card images in preparation for punching via the keypunch. The program validates the card images, ensuring that no line has more than 80 columns of data and that there are no characters in the file that are not in the encoding. See the appendix for the character encoding – the ASCII characters and the card patterns and native characters they represent. For binary mode, the file much consist of groups of four hex digits, each group separated by a space.

Ctrl-C closes any open file, while Ctrl-Q will shut down the program.

The Options menu is used to select one of the encodings for the text files, forget the saved serial port from the configuration file, display some information about the program, and to display the initially hidden log window.



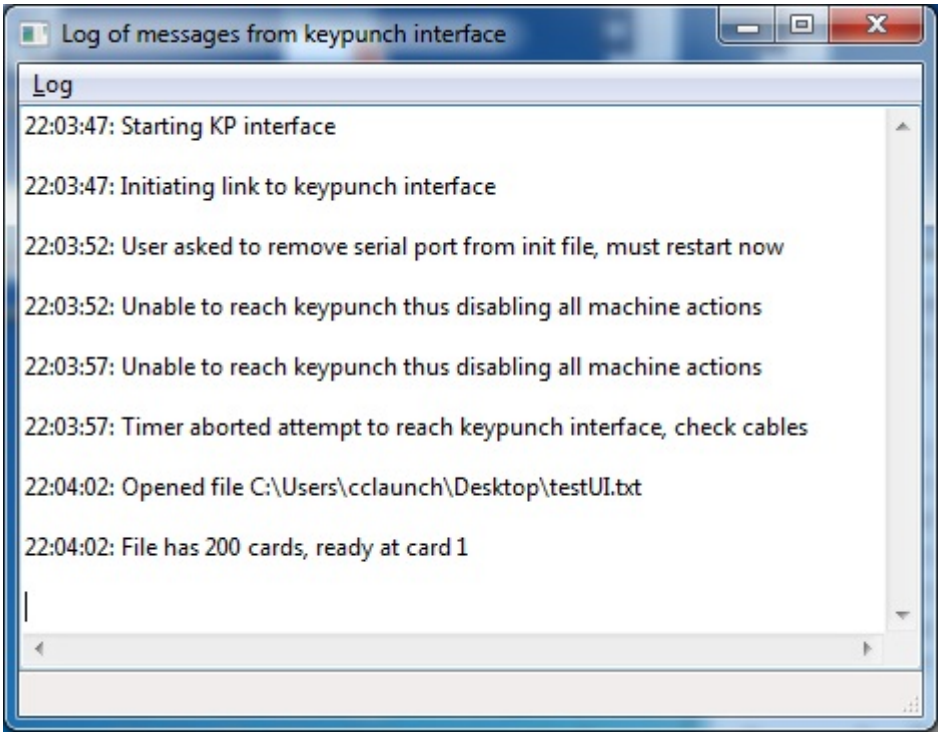
Currently, the program implements seven encodings, BCD, EBCDIC, binary, and four formats for use with the IBM 1130 Simulator). The program will issue the appropriate commands to the keypunch interface to switch it into the same mode and will verify that it did so successfully. The IBM 1130 Simulator formats use the EBCDIC and binary modes of KPgolem, accomplishing the translation or reformatting between the PC files and this program

Ctrl-1 through Ctrl-7 select the seven encodings. The appendix details the encodings.

Ctrl-F will remove the serial port entry from the configuration file, after which the program must be restarted to go through the serial port selection dialog.

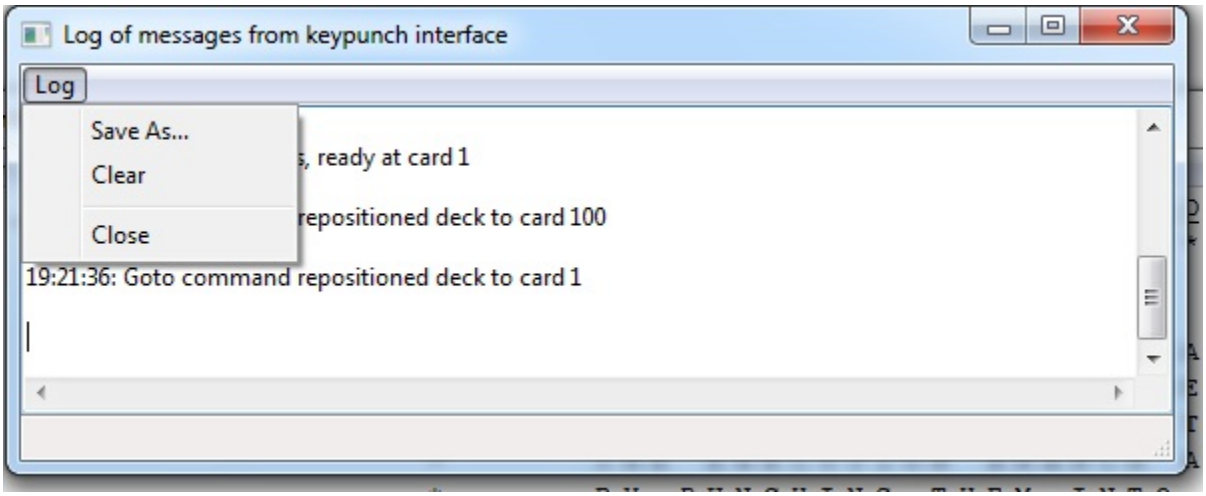
Ctrl-L, Show Keypunch Log, will make visible a Log window that is saving various status and error messages generated by the program. In addition, every line sent by the keypunch interface back to the program is recorded in the log.

Finally, the About entry will pop up a dialog with a short identification of the program.



The log window can be closed, but remains active collecting messages which will still be there when it is made visible again by Ctrl-L

This window has a Log menu that offers a few choices – saving a copy of the log, clearing the current entries, and making it hidden again.



Keypunch Interface box protocol over serial link

The serial cable link between the keypunch interface box and the external device that will drive the keypunch is flexible, allowing manually typed commands through a terminal as well as programmatic control. The program documented in the first section of this guide is just one example of software that can control the keypunch.

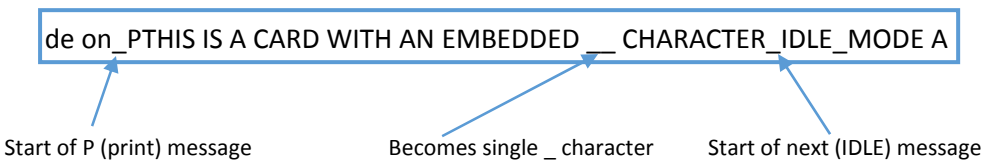
Features of the link protocol are:

- 7-bit ASCII data only over the link
- 9600 baud speed
- 8 bit serial
- One stop bit
- No parity
- XON/XOFF (software) flow control used
- No use of RTS/CTS, DTE or other control lines of RS232
- Each message spans from an initial delimiter up to the next delimiter
- Default delimiter for the KPgolem box is _ (underscore)
- The delimiter character can be put as a data character in a message using an escape sequence
- Messages (commands) sent into KPgolem consist of a verb followed by optional parameters
- Messages can be ended with new line but those line end characters are ignored
- KPgolem can be configured to accept either full verbs and parameters or single characters
- The default for KPgolem, which is used by the PC program from the first section, is full verbs
- All responses from the KPgolem box are human-readable messages
- All actions are requested by a single command message to KPgolem and the results are indicated by one or more messages back
- All actions by KPgolem are completed before the next command message is read

<u>Delimiting a message in the serial stream inbound to the KPgolem box</u>

The incoming stream of characters will be scanned by KPgolem until it comes across the delimiter character (_). If a second _ immediately follows, the pair of underscores are considered to be an ordinary '_' that is part of the current message, not as the start of a new command. This provides an escape sequence to ignore the special role of _ as a message delimiter.

Any even number of sequential _ characters are treated as ordinary text (e.g. six _ in the middle of a message will result in that message having three _ within it. The last _ of an odd number in a row, including a lone _, is evaluated as a delimiter, the start of a new message.



Nothing in an incoming message will be acted upon until the next start of message delimiter plus one additional character is read. Everything stored up until the delimiter is the message. The reason for reading one more character after the delimiter is to ensure that we don't have an escape sequence, a pair of contiguous delimiters, since that would mean it is not the start of a new message.

Command verbs

Every message begins with a command verb plus any required or optional parameters. In most cases, the verb is followed by one or more spaces before the parameters. See the detailed verb documentation for its rules pertaining to spacing.

The implemented verbs are:

- IDLE – does nothing but confirms it was received
- P – punches the remaining text on the message in a card
- R – reads the card positioned in the keypunch read (duplicate) station and displays contents
- MODE – switches between ASCII and BINARY modes
- CODE – switches ASCII interpretation to produce EBCDIC or BCD characters/hole patterns
- LOAD – will change one ASCII input character to a specific hole pattern, overriding current CODE
- DIAG – executes various built-in diagnostic routines
- VERIFY – can set the machine to verify that the prior card matches what was sent while it is punching the next card

Replies from KPgolem to the user

Responses to a command, including error conditions discovered, are sent back from KPgolem after it reads and attempts to execute that command. It may be one or multiple messages returned, but the last message will include the character string **OK** in the message for easy matching in programs sending the command.

No unsolicited messages are sent from KPgolem to the user – all transmissions are in response to a command message issued by the user and all response messages for a command follow it directly, as a transaction which is completed entirely before the next command is read and acted upon.

Encoding of data to punch on or read from cards

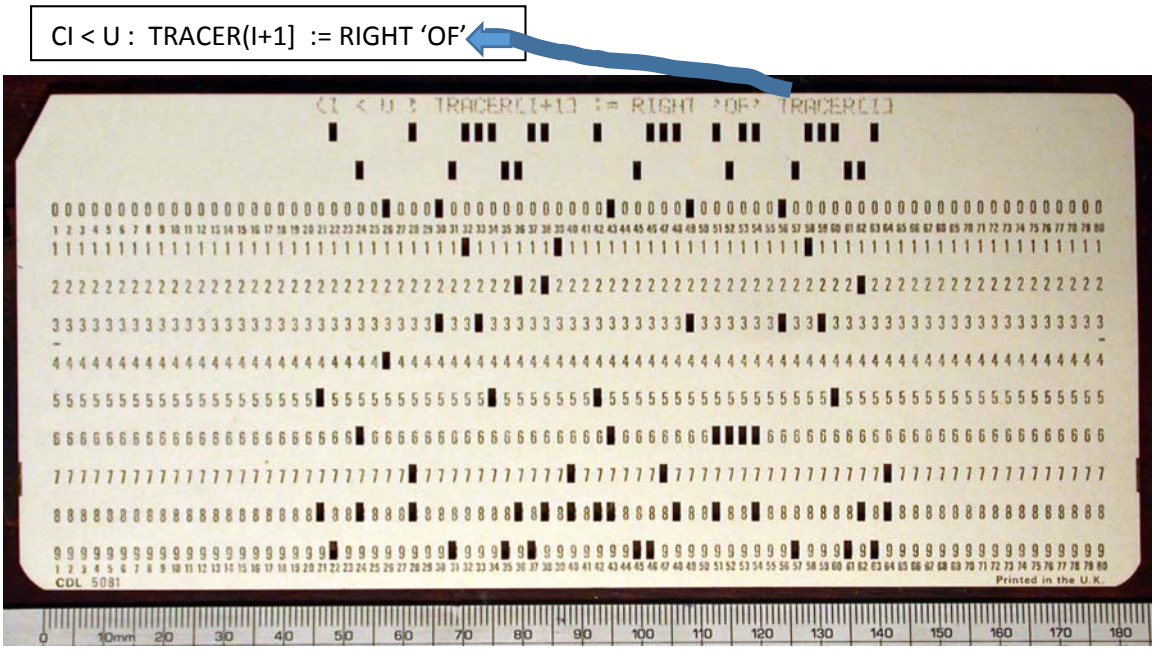
The data stream sent to KPgolem for a card image is always in ASCII, but it is in one of two modes. The first mode, ASCII, is a one for one translation of an ASCII character coming from the user to a set of holes to punch in a card column. The second mode, BINARY, uses groups of four hexadecimal digits for each card column, to individually control punching each of the twelve rows as well as punching a blank or releasing the card.

When in ASCII mode, the translation table that determines which holes are punched when a certain ASCII character arrives is called an encoding. The KPgolem box supports two standard encodings plus customized translations that can be set dynamically through commands. The standard encodings are EBCDIC (IBM S/360 era character set, representative of an 029 keypunch keyboard), and BCD (IBM 1401 era character set, representative of an 026 keypunch keyboard).

IBM 1130 Simulator files are converted to binary or EBCDIC before transmission to and from KPgolem so no unique codeset exists on that interface box to support these formats. The 1130 encodings exist only in the PC files, to keep them consistent with files used by that simulator.

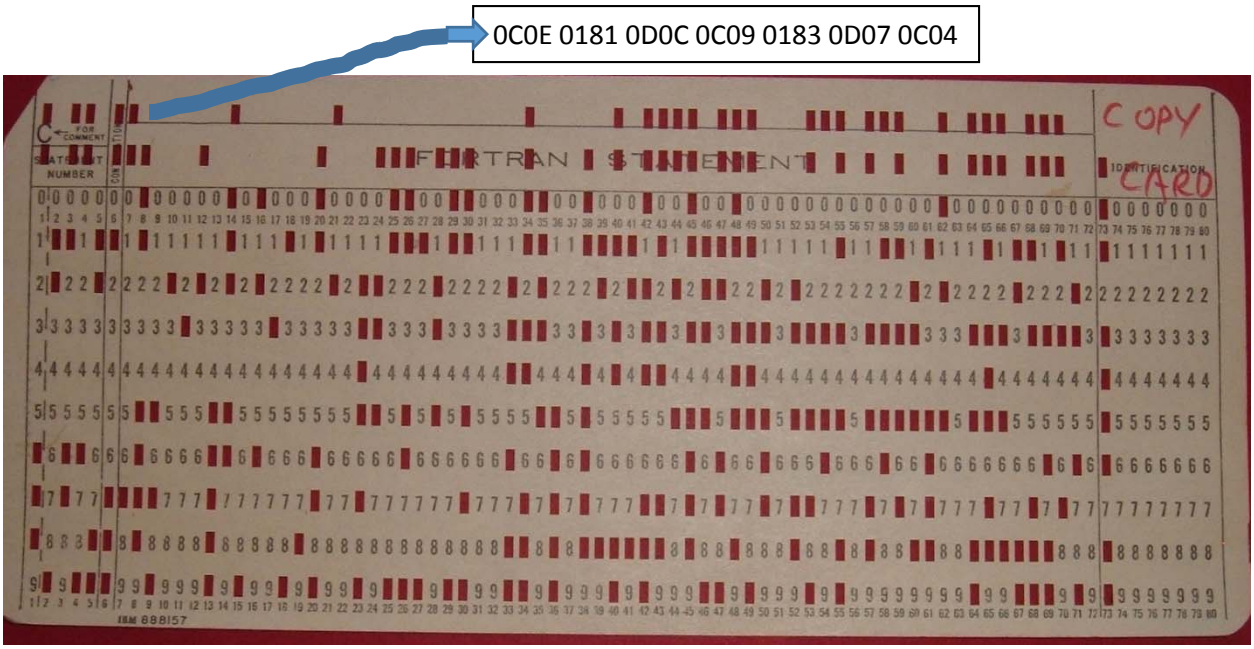
Issuing the MODE and CODE commands allows the user to switch the KPgolem unit between the 1:1 ASCII mode using EBCDIC or BCD or the 4 digits to 1 column BINARY mode. The LOAD command allows the user to specify custom translation of ASCII characters to hole patterns.

Reading a card returns the results in the mode and code set by the user. If the machine is in ASCII mode then a read command will return 80 ASCII characters which are the reverse translation of the BCD or EBCDIC encoding tables or user modified table. If the machine is in BINARY mode, reading a card returns 400 characters, four digits for each column with a space separating each of the 80 column groups.



Reading a card in ASCII mode

The card is read, producing a stream of ASCII characters starting at column 1 and ending at column 80, taking each character read from the card and looking it up in the encoding table to find the ASCII character that represents it.



Binary mode card being read

In binary mode, each column is encoded in the four hexadecimal digits (in ASCII) that reflects the holes punched. For the card shown above, column 1 has rows 12, 11, 6, 7 and 8 punched, which produces the string of characters 0C0E with a space padding it to the right before the next column. Similarly, column 2 has rows 1, 2 and 9 punched which encodes to the string 0181. Proceeding over to column 71, we see that rows 11, 0, 1, 3, 4, 5, 6, 8 and 9 are punched – this is encoded as 033B. The remaining columns are each encoded as 1000 which is the representation for a blank column.

Verification of punched cards

When the user issues _VERIFY ON to the interface box, it will remember the data it punches into a card, so that while it is punching the next card's image, it can simultaneously be reading the prior card through the read station and compare what is actually read to what was intended to be punched.

This slows down the machine because it must read through as many columns as either card has – the one being read for verification or the next card being punched. If the next card is shorter, it is punched with spaces out to the length of verified card.

The PC program documented in the first section of this guide does not make use of the verification function. However, users connecting via a terminal or other programs could issue the commands to put the box into verification mode.

Loading custom character translations into the encoding tables

The LOAD command exists to specify the decimal value of an ASCII character, then go to the entry in the current CODE table that is assigned to the chosen character. That table location will be updated to use the four hexadecimal digits on the LOAD command whenever punching and the chosen character is found.

Replacing the lookup is therefore changing the translation or encoding of that ASCII character. Since the table holds a four hex digit value, it can specify any combination of the 12 rows to be punched or request one of the control functions.

The special value of 8000 sets a control value that means invalid character. Thus, any ASCII character which is translated to 8000 will generate an error message from the KPgolem box while punching that card – specifying the column where an invalid character was found.

As an example, in the standard EBCDIC code table, when an ascii ^ us sent to punch, the table entry for ^ (decimal value 94) has 0406 which specifies that rows 11, 7 and 8 be punched. That hole pattern is the EBCDIC character ¬ allowing the user to type ^ when they want to punch ¬.

Using a _LOAD 94 0442 we are asking for ASCII character ^ to now punch rows 11, 3 and 8 which is the EBCDIC character $. After using the LOAD command, any cards punched having a ^ will punch a $ up until the next time we switch the MODE and/or CODE of KPgolem (or reload the entry for ^)

Issuing a MODE ASCII and a CODE of EBCDIC or BCD will give the user the initial translation values for those encodings, but LOAD commands can modify any that are wanted. In addition,, a MODE of USER can be requested, which gives a translation table with every ASCII character declared invalid (special code 8000). The user can then issue LOAD commands to build up all the valid translations of ASCII characters to their intended hole patterns.

Role of the IDLE command

While the IDLE command does not take any action, it does provide a start of message delimiter plus another character which is not the delimiter – this completes the prior message and causes it to be activated. Without this, one could issue a legal command such as _LOAD ASCII but it would not be executed until _ and anything else was transmitted. _IDLE serves to complete the prior message and cause it be executed immediately.

The DIAG command to run diagnostics

The KPgolem box has the ability to run some built in diagnostic or verification routines. In its current implementation, there are only two routines. DIAG 0 is used at startup and any time a user or program wants a quick check of the health of the link between the user and the box. DIAG 1 performs a more extensive test of various keypunch functions including punching, punching binary (using the multipunch key), spacing, releasing and reading data from cards.

DIAG 0 checks for the existence of properly wired cables between the KPgolem box and the keypunch. It indicates whether the punch cable is attached, whether the read cable is attached, and whether the keypunch believes a card is registered in the punch station. A number of messages are returned with the status of these elements, then it sends an OK message that the diagnostic is complete.

DIAG 1 is a more involved test, as it punches a specific data pattern on one card and then punches spaces into the next card so that it can read and verify that the card was punched correctly. It uses all 12 rows, spaces, and uses the multipunch key to combine some columns. If a punch is not working, the column that was punched with that row will prove to be blank as it is read. If the multipunch key doesn't work, the various holes that should be in the same column will be spread across several card columns. Similarly, failure of space will shift holes into a column that we expect to read as blank.

This diagnostic routine will emit messages of its intent and then the valid or incorrect state of each column is reported. Once it has completed, regardless of whether the outcome was good or not, it reports OK that diagnostic 1 is complete.

Eventually there can be other routines with their own diagnostic number, as specific tests are developed that may be useful to those using this box.

Startup message from the KPgolem box to the user

When the keypunch is powered on, the interface box also receives power and boots up. It emits a welcome message at startup which includes the version number of the code running inside the box. Startup is the only time unsolicited messages are sent from the box up to the user.

Command Syntax and Rules

Bold characters are fixed characters that must be entered exactly as given.

Underscore is the delimiter and must be entered where shown.

Braces '{' and '}' indicated optional parameters.

Brackets '[' and ']' represent sets of alternatives, separated by the vertical bar

A vertical bar indicates a choice among the elements on either side of the bar.

Italic characters stand for data issued by the user that is intended to be punched on a card or specified as a parameter value.

The asterisk '*' will be used when a single space must exist at this point in the command, while a plus '+' is used when one or more spaces are to be placed at this location in the command.

9 is used when one or more decimal digit must be entered at this point in the command

is used when a single hexadecimal digit must be entered in its place.

Print (P) command

When the mode is ASCII, the command is

_P*user data up to 80 columns of ASCII*

When the mode is BINARY, the command is zero to 80 groups of four hex digits with one space between them.

_P{####{*####}

Read (R) command

_R

Mode (MODE) command

_MODE+[**ASCII** | **BINARY**]

Code (CODE) command

_CODE+[**BCD** | **EBCDIC** | **USER**]

Load (LOAD) command

_LOAD+99+####

Verify (VERIFY) command

_VERIFY+[**ON** | **OFF**]

Diagnostic (DIAG) command

_DIAG+9

Idle (IDLE) command

_IDLE

REFERENCE MATERIAL

EBCDIC encoding in ASCII

EBCDIC glyph	ASCII glyph	Hollerith	EBCDIC code	ASCII code
A	**A**	12-1	x'C1'	x'41'
B	**B**	12-2	x'C2'	x'42'
C	**C**	12-3	x'C3'	x'43'
D	**D**	12-4	x'C4'	x'44'
E	**E**	12-5	x'C5'	x'45'
F	**F**	12-6	x'C6'	x'46'
G	**G**	12-7	x'C7'	x'47'
H	**H**	12-8	x'C8'	x'48'
I	**I**	12-9	x'C9'	x'49'
J	**J**	11-1	x'D1'	x'4A'
K	**K**	11-2	x'D2'	x'4B'
L	**L**	11-3	x'D3'	x'4C'
M	**M**	11-4	x'D4'	x'4D'
N	**N**	11-5	x'D5'	x'4E'
O	**O**	11-6	x'D6'	x'4F'
P	**P**	11-7	x'D7'	x'50'
Q	**Q**	11-8	x'D8'	x'51'
R	**R**	11-9	x'D9'	x'52'
S	**S**	0-2	x'E2'	x'53'
T	**T**	0-3	x'E3'	x'54'
U	**U**	0-4	x'E4'	x'55'
V	**V**	0-5	x'E5'	x'56'
W	**W**	0-6	x'E6'	x'57'
X	**X**	0-7	x'E7'	x'58'
Y	**Y**	0-8	x'E8'	x'59'
Z	**Z**	0-9	x'E9'	x'5A'
0	**0**	0	x'F0'	x'30'
1	**1**	1	x'F1'	x'31'
2	**2**	2	x'F2'	x'32'
3	**3**	3	x'F3'	x'33'
4	**4**	4	x'F4'	x'34'
5	**5**	5	x'F5'	x'35'
6	**6**	6	x'F6'	x'36'
7	**7**	7	x'F7'	x'37'
8	**8**	8	x'F8'	x'38'
9	**9**	9	x'F9'	x'39'

EBCDIC glyph	ASCII glyph	Hollerith	EBCDIC code	ASCII code
a	**a**	12-0-1	x'81'	x'61'
b	**b**	12-0-2	x'82'	x'62'
c	**c**	12-0-3	x'83'	x'63'
d	**d**	12-0-4	x'84'	x'64'
e	**e**	12-0-5	x'85'	x'65'
f	**f**	12-0-6	x'86'	x'66'
g	**g**	12-0-7	x'87'	x'67'
h	**h**	12-0-8	x'88'	x'68'
i	**i**	12-0-9	x'89'	x'69'
j	**j**	12-11-1	x'91'	x'6A'
k	**k**	12-11-2	x'92'	x'6B'
l	**l**	12-11-3	x'93'	x'6C'
m	**m**	12-11-4	x'94'	x'6D'
n	**n**	12-11-5	x'95'	x'6E'
o	**o**	12-11-6	x'96'	x'6F'
p	**p**	12-11-7	x'97'	x'70'
q	**q**	12-11-8	x'98'	x'71'
r	**r**	12-11-0	x'99'	x'72'
s	**s**	11-0-2	x'A2'	x'73'
t	**t**	11-0-3	x'A3'	x'74'
u	**u**	11-0-4	x'A4'	x'75'
v	**v**	11-0-5	x'A5'	x'76'
w	**w**	11-0-6	x'A6'	x'77'
x	**x**	11-0-7	x'A7'	x'78'
y	**y**	11-0-8	x'A8'	x'79'
z	**Z**	11-0-9	x'A9'	x'7a'

Lower case EBCDIC characters

EBCDIC glyph	ASCII glyph	Hollerith	EBCDIC code	ASCII code
space	space	blank	x'40'	x'20'
.	.	12-3-8	x'4B'	x'2E'
<	<	12-4-8	x'4C'	x'3C'
(	(	12-5-8	x'4D'	x'28'
+	+	12-6-8	x'4E'	x'2B'
|	|	12-7-8	x'4F'	x'7C'
&	&	12	x'50'	x'26'
!	!	11-2-8	x'5A'	x'21'
$	$	11-3-8	x'5B'	x'24'
*	*	11-4-8	x'5C'	x'2A'
)	)	11-5-8	x'5D'	x'29'
;	;	11-6-8	x'5E'	x'3B'
-	-	11	x'60'	x'2D'
/	/	0-2-8	x'61'	x'2F'
,	,	0-3-8	x'6B'	x'2C'
%	%	0-4-8	x'6C'	x'25'
_	_	0-5-8	x'6D'	x'5F'
>	>	0-6-8	x'6E'	x'3E'
?	?	0-7-8	x'6F'	x'3F'
:	:	2-8	x'7A'	x'3A'
#	#	3-8	x'7B'	x'23'
@	@	4-8	x'7C'	x'40'
'	'	5-8	x'7D'	x'27'
=	=	6-8	x'7E'	x'3D'
"	"	7-8	x'7F'	x'22'
¢	[	12-2-8	x'4A'	x'5B'
¦	]	11-7-8	x'5F'	x'5D'
}	}	12-0	x'C0'	x'7B
{	{	11-0	x'D0'	x'7D'
~	~	11-0-1	x'A1'	x'7E
¬	^	11-7-8	x'5F'	x'5E'
`	`	1-8	x'79'	x'60'

Special characters in EBCDIC

Look up the desired character in EBCDIC in the left column (EBCDIC glyph), select the ASCII character in the next column over and enter that in the text file. The middle column shows the holes that will be punched in a card for that desired EBCDIC character. Multiple rows of holes are listed with hyphen between the row numbers, e.g. 11-7-8 means a hole in rows 11, 7 and 8.

1402 Reader Compatible BCD encoding in ASCII

BCD glyph	ASCII glyph	Hollerith	BCD code	ASCII code
space	space	blank	x'40'	x'20'
! (Minus Zero)	!	11-0	x'2A'	x'21'
≠				
grp mark	"	12-7-8	x7F'	x'22'
#	#	3-8	x'0B'	x'23'
$	$	11-3-8	x'6B'	x'24'
%	%	0-4-8	x'1C'	x'25'
&	&	12	x'70'	x'26'
≠				
rec mark	'	0-2-8	x'1A'	x'27'
%	(	0-4-8	x'1C'	x'28'
□				
	)	12-4-8	x'7C'	x'29'
*	*	11-4-8	x'2C'	x'2A'
&	+	12	x'70'	x'2B'
,	,	0-3-8	x'5B'	x'2C'
-	-	11	x'20'	x'2D'
.	.	12-3-8	x'3B'	x'2E'
/	/	0-1	x'31'	x'2F'
0	0	0	x'4A'	x'30'
1	1	1	x'01'	x'31'
2	2	2	x'02'	x'32'
3	3	3	x'43'	x'33'
4	4	4	x'04'	x'34'
5	5	5	x'45'	x'35'
6	6	6	x'46'	x'36'
7	7	7	x'47'	x'37'
8	8	8	x'08'	x'38'
9	9	9	x'49'	x'39'
:	:	5-8	x'0D'	x'3A'
<	<	12-6-8	x'3E'	x'3C'
#	=	3-8	x'0B'	x'3D'
>	>	6-8	x'0E'	x'3E'
? (Plus Zero)	?	12-0	x'7A'	x'3F'
@	@	4-8	x'4C'	x'40'

BCD glyph	ASCII glyph	Hollerith	BCD code	ASCII code
A	**A**	12-1	x'31'	x'41'
B	**B**	12-2	x'32'	x'42'
C	**C**	12-3	x'73'	x'43'
D	**D**	12-4	x'34'	x'44'
E	**E**	12-5	x'75'	x'45'
F	**F**	12-6	x'76'	x'46'
G	**G**	12-7	x'37'	x'47'
H	**H**	12-8	x'38'	x'48'
I	**I**	12-9	x'79'	x'49'
J	**J**	11-1	x'61'	x'4A'
K	**K**	11-2	x'62'	x'4B'
L	**L**	11-3	x'23'	x'4C'
M	**M**	11-4	x'64'	x'4D'
N	**N**	11-5	x'25'	x'4E'
O	**O**	11-6	x'26'	x'4F'
P	**P**	11-7	x'67'	x'50'
Q	**Q**	11-8	x'68'	x'51'
R	**R**	11-9	x'29'	x'52'
S	**S**	0-2	x'52'	x'53'
T	**T**	0-3	x'13'	x'54'
U	**U**	0-4	x'54'	x'55'
V	**V**	0-5	x'15'	x'56'
W	**W**	0-6	x'16'	x'57'
X	**X**	0-7	x'57'	x'58'
Y	**Y**	0-8	x'58'	x'59'
Z	**Z**	0-9	x'19'	x'5A'

BCD glyph	ASCII glyph	Hollerith	BCD code	ASCII code
A	**a**	12-1	x'31'	x'61'
B	**b**	12-2	x'32'	x'62'
C	**c**	12-3	x'73'	x'63'
D	**d**	12-4	x'34'	x'64'
E	**e**	12-5	x'75'	x'65'
F	**f**	12-6	x'76'	x'66'
G	**g**	12-7	x'37'	x'67'
H	**h**	12-8	x'38'	x'68'
I	**i**	12-9	x'79'	x'69'
J	**j**	11-1	x'61'	x'6A'
K	**k**	11-2	x'62'	x'6B'
L	**l**	11-3	x'23'	x'6C'
M	**m**	11-4	x'64'	x'6D'
N	**n**	11-5	x'25'	x'6E'
O	**o**	11-6	x'26'	x'6F'
P	**p**	11-7	x'67'	x'70'
Q	**q**	11-8	x'68'	x'71'
R	**r**	11-9	x'29'	x'72'
S	**s**	0-2	x'52'	x'73'
T	**t**	0-3	x'13'	x'74'
U	**u**	0-4	x'54'	x'75'
V	**v**	0-5	x'15'	x'76'
W	**w**	0-6	x'16'	x'77'
X	**x**	0-7	x'57'	x'78'
Y	**y**	0-8	x'58'	x'79'
Z	**z**	0-9	x'19'	x'7a'

Binary encoding in ASCII

Each column in a card is encoded as four hexadecimal digits and each group of four digits is separated from the prior column by a single space.

```
0840 0408 0820 0810 1000 0040 0020 2000
```

The line above will punch the characters CODE 43 in columns 1 to 6 then release the card to skip the remaining columns.

The first digit implements some control bits. From left to right, the bits specify:

Invalid Character	not used	Release Card	Space Once

The second digit controls punching in the top four rows of the card:

12	11	0	1

The third digit controls punching holes in the middle four rows of the card:

2	3	4	5

The fourth digit controls punching in the bottom four rows of a card:

6	7	8	9

Thus, coding 0900 specifies that a hole be punched in card rows 12 and 1.

The coding 1000 is a blank space, skipping to the next column of the card.

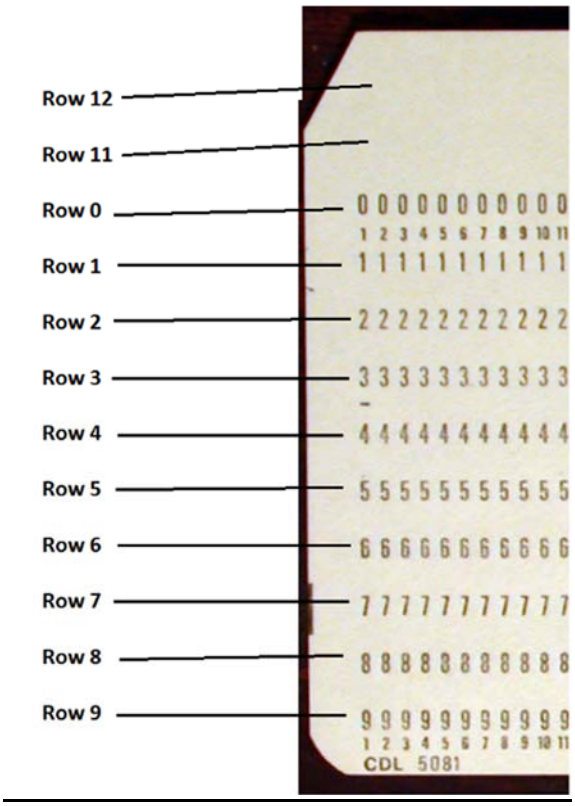
Ending a line with 2000 will cause the keypunch to do a high speed ejection of the card, faster than if the rest of the card was punched.

The code 0242 punches holes in rows 0, 3 and 8. Any number of rows can be punched in one column, as the keypunch will use the multipunch key to punch no more than three holes at a time, iterating through the desired holes until they are all punched in that column.

Each hexadecimal digit represents the sixteen combinations of setting four bits on or off.

Hex Digit	Bit A	Bit B	Bit C	Bit D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

Conversion of four independent bits to a hexadecimal digit



Row numbering in each card column

IBM 1130 Simulator Binary Cards

Each card image consists of 160 bytes, two bytes per card column. The first byte specifies the presence or absence of a hole in rows 12, 11, 0, 1, 2, 3, 4 and 5, from the leftmost bit to the rightmost. The second byte specifies rows 6, 7, 8 and 9 in the left four bits while the right four bits are not used.

It is a simple matter to convert between this code and the KPgolem binary format, interpreting each group of four bits as a hexadecimal digit and then storing the character for that digit. This will expand the card image to 400 ASCII characters for transmission from this program to KPgolem.

IBM 1130 Simulator 029 Format Cards

EBCDIC glyph	Unicode glyph	Hollerith	EBCDIC code	Unicode
space	space	blank	x'40'	x'20'
¢	¢	12-2-8	x'4A'	U'00A2
.	.	12-3-8	x'4B'	x'2E'
<	<	12-4-8	x'4C'	x'3C'
(	(	12-5-8	x'4D'	x'28'
+	+	12-6-8	x'4E'	x'2B'
|	|	12-7-8	x'4F'	x'7C'
&	&	12	x'50'	x'26'
!	!	11-2-8	x'5A'	x'21'
$	$	11-3-8	x'5B'	x'24'
*	*	11-4-8	x'5C'	x'2A'
)	)	11-5-8	x'5D'	x'29'
;	;	11-6-8	x'5E'	x'3B'
¬	¬	11-7-8	x'5F'	U'00AC'
-	-	11	x'60'	x'2D'
,	,	0-3-8	x'6B'	x'2C'
%	%	0-4-8	x'6C'	x'25'
_	_	0-5-8	x'6D'	x'5F'
>	>	0-6-8	x'6E'	x'3E'
:	:	2-8	x'7A'	x'3A'
#	#	3-8	x'7B'	x'23'
@	@	4-8	x'7C'	x'40'
'	'	5-8	x'7D'	x'27'
=	=	6-8	x'7E'	x'3D'
"	"	7-8	x'7F'	x'22'
/	/	0-1	x'61'	x'2F'

EBCDIC glyph	Unicode glyph	Hollerith	EBCDIC code	Unicode
A	**A**	12-1	x'C1'	x'41'
B	**B**	12-2	x'C2'	x'42'
C	**C**	12-3	x'C3'	x'43'
D	**D**	12-4	x'C4'	x'44'
E	**E**	12-5	x'C5'	x'45'
F	**F**	12-6	x'C6'	x'46'
G	**G**	12-7	x'C7'	x'47'
H	**H**	12-8	x'C8'	x'48'
I	**I**	12-9	x'C9'	x'49'
J	**J**	11-1	x'D1'	x'4A'
K	**K**	11-2	x'D2'	x'4B'
L	**L**	11-3	x'D3'	x'4C'
M	**M**	11-4	x'D4'	x'4D'
N	**N**	11-5	x'D5'	x'4E'
O	**O**	11-6	x'D6'	x'4F'
P	**P**	11-7	x'D7'	x'50'
Q	**Q**	11-8	x'D8'	x'51'
R	**R**	11-9	x'D9'	x'52'
S	**S**	0-2	x'E2'	x'53'
T	**T**	0-3	x'E3'	x'54'
U	**U**	0-4	x'E4'	x'55'
V	**V**	0-5	x'E5'	x'56'
W	**W**	0-6	x'E6'	x'57'
X	**X**	0-7	x'E7'	x'58'
Y	**Y**	0-8	x'E8'	x'59'
Z	**Z**	0-9	x'E9'	x'5A'
0	**0**	0	x'F0'	x'30'
1	**1**	1	x'F1'	x'31'
2	**2**	2	x'F2'	x'32'
3	**3**	3	x'F3'	x'33'
4	**4**	4	x'F4'	x'34'
5	**5**	5	x'F5'	x'35'
6	**6**	6	x'F6'	x'36'
7	**7**	7	x'F7'	x'37'
8	**8**	8	x'F8'	x'38'
9	**9**	9	x'F9'	x'39'

EBCDIC glyph	Unicode glyph	Hollerith	EBCDIC code	Unicode
a	**a**	12-0-1	x'81'	x'61'
b	**b**	12-0-2	x'82'	x'62'
c	**c**	12-0-3	x'83'	x'63'
d	**d**	12-0-4	x'84'	x'64'
e	**e**	12-0-5	x'85'	x'65'
f	**f**	12-0-6	x'86'	x'66'
g	**g**	12-0-7	x'87'	x'67'
h	**h**	12-0-8	x'88'	x'68'
i	**i**	12-0-9	x'89'	x'69'
j	**j**	12-11-1	x'91'	x'6A'
k	**k**	12-11-2	x'92'	x'6B'
l	**l**	12-11-3	x'93'	x'6C'
m	**m**	12-11-4	x'94'	x'6D'
n	**n**	12-11-5	x'95'	x'6E'
o	**o**	12-11-6	x'96'	x'6F'
p	**p**	12-11-7	x'97'	x'70'
q	**q**	12-11-8	x'98'	x'71'
r	**r**	12-11-0	x'99'	x'72'
s	**s**	11-0-2	x'A2'	x'73'
t	**t**	11-0-3	x'A3'	x'74'
u	**u**	11-0-4	x'A4'	x'75'
v	**v**	11-0-5	x'A5'	x'76'
w	**w**	11-0-6	x'A6'	x'77'
x	**x**	11-0-7	x'A7'	x'78'
y	**y**	11-0-8	x'A8'	x'79'
z	**z**	11-0-9	x'A9'	x'7a'

Files used with the IBM 1130 Simulator, created by Brian Knittel, store card images in Unicode rather than ASCII, in order to accommodate the special characters **¢** and **¬** that are not part of the standard ASCII character set.

These are converted to the standard EBCDIC encoding for transmission to and from the KPgolem interface box.

IBM 1130 Simulator 026 Fortran Keycaps Card Format

EBCDIC glyph	Unicode glyph	Hollerith	EBCDIC code	Unicode
space	**space**	blank	x'40'	x'20'
.	**.**	12-3-8	x'4B'	x'2E'
(	**(**	0-5-8	x'4D'	x'28'
&	**&**	12	x'50'	x'26'
-	**-**	11	x'60'	x'2D'
,	**,**	0-3-8	x'6B'	x'2C'
$	**$**	11-3-8	x'5B'	x'24'
*****	*****	11-4-8	x'5C'	x'2A'
)	**)**	12-4-8	x'5D'	x'29'
'	**'**	4-8	x'7D'	x'27'
=	**=**	3-8	x'7E'	x'3D'
/	**/**	0-1	x'61'	x'2F'
0	**0**	0	x'F0'	x'30'
1	**1**	1	x'F1'	x'31'
2	**2**	2	x'F2'	x'32'
3	**3**	3	x'F3'	x'33'
4	**4**	4	x'F4'	x'34'
5	**5**	5	x'F5'	x'35'
6	**6**	6	x'F6'	x'36'
7	**7**	7	x'F7'	x'37'
8	**8**	8	x'F8'	x'38'
9	**9**	9	x'F9'	x'39'

EBCDIC glyph	Unicode glyph	Hollerith	EBCDIC code	Unicode
A	**A**	12-1	x'C1'	x'41'
B	**B**	12-2	x'C2'	x'42'
C	**C**	12-3	x'C3'	x'43'
D	**D**	12-4	x'C4'	x'44'
E	**E**	12-5	x'C5'	x'45'
F	**F**	12-6	x'C6'	x'46'
G	**G**	12-7	x'C7'	x'47'
H	**H**	12-8	x'C8'	x'48'
I	**I**	12-9	x'C9'	x'49'
J	**J**	11-1	x'D1'	x'4A'
K	**K**	11-2	x'D2'	x'4B'
L	**L**	11-3	x'D3'	x'4C'
M	**M**	11-4	x'D4'	x'4D'
N	**N**	11-5	x'D5'	x'4E'
O	**O**	11-6	x'D6'	x'4F'
P	**P**	11-7	x'D7'	x'50'
Q	**Q**	11-8	x'D8'	x'51'
R	**R**	11-9	x'D9'	x'52'
S	**S**	0-2	x'E2'	x'53'
T	**T**	0-3	x'E3'	x'54'
U	**U**	0-4	x'E4'	x'55'
V	**V**	0-5	x'E5'	x'56'
W	**W**	0-6	x'E6'	x'57'
X	**X**	0-7	x'E7'	x'58'
Y	**Y**	0-8	x'E8'	x'59'
Z	**Z**	0-9	x'E9'	x'5A'

Files in 026 Fortran character set keypunch images for the IBM 1130 Simulator are stored in Unicode rather than ASCII, but are converted to standard EBCDIC for transmission to and from the KPgolem box.

IBM 1130 Simulator 026 Commercial Keycaps Card Format

EBCDIC glyph	Unicode glyph	Hollerith	EBCDIC code	Unicode
A	**A**	12-1	x'C1'	x'41'
B	**B**	12-2	x'C2'	x'42'
C	**C**	12-3	x'C3'	x'43'
D	**D**	12-4	x'C4'	x'44'
E	**E**	12-5	x'C5'	x'45'
F	**F**	12-6	x'C6'	x'46'
G	**G**	12-7	x'C7'	x'47'
H	**H**	12-8	x'C8'	x'48'
I	**I**	12-9	x'C9'	x'49'
J	**J**	11-1	x'D1'	x'4A'
K	**K**	11-2	x'D2'	x'4B'
L	**L**	11-3	x'D3'	x'4C'
M	**M**	11-4	x'D4'	x'4D'
N	**N**	11-5	x'D5'	x'4E'
O	**O**	11-6	x'D6'	x'4F'
P	**P**	11-7	x'D7'	x'50'
Q	**Q**	11-8	x'D8'	x'51'
R	**R**	11-9	x'D9'	x'52'
S	**S**	0-2	x'E2'	x'53'
T	**T**	0-3	x'E3'	x'54'
U	**U**	0-4	x'E4'	x'55'
V	**V**	0-5	x'E5'	x'56'
W	**W**	0-6	x'E6'	x'57'
X	**X**	0-7	x'E7'	x'58'
Y	**Y**	0-8	x'E8'	x'59'
Z	**Z**	0-9	x'E9'	x'5A'

EBCDIC glyph	Unicode glyph	Hollerith	EBCDIC code	Unicode
space	**space**	blank	x'40'	x'20'
.	**.**	12-3-8	x'4B'	x'2E'
(	**(**	0-4-8	x'4D'	x'28'
&	**&**	12	x'50'	x'26'
-	**-**	11	x'60'	x'2D'
,	**,**	0-3-8	x'6B'	x'2C'
$	**$**	11-3-8	x'5B'	x'24'
*****	*****	11-4-8	x'5C'	x'2A'
)	**)**	12-4-8	x'5D'	x'29'
'	**'**	4-8	x'7D'	x'27'
=	**=**	3-8	x'7E'	x'3D'
/	**/**	0-1	x'61'	x'2F'
0	**0**	0	x'F0'	x'30'
1	**1**	1	x'F1'	x'31'
2	**2**	2	x'F2'	x'32'
3	**3**	3	x'F3'	x'33'
4	**4**	4	x'F4'	x'34'
5	**5**	5	x'F5'	x'35'
6	**6**	6	x'F6'	x'36'
7	**7**	7	x'F7'	x'37'
8	**8**	8	x'F8'	x'38'
9	**9**	9	x'F9'	x'39'

h

Files using the IBM 1130 Simulator 026 keypunch with commercial keycaps encoding are stored in Unicode format but converted to EBCDIC for transmission to and from the KPgolem interface box.

Full BCD encoding in ASCII

BCD glyph	ASCII glyph	Hollerith	BCD code	ASCII code
space	space	blank	x'40'	x'20'
! (Minus Zero)	!	11-0	x'2A'	x'21'
‡				
grp mark	"	12-7-8	x7F'	x'22'
#	#	3-8	x'0B'	x'23'
$	$	11-3-8	x'6B'	x'24'
%	%	0-4-8	x'1C'	x'25'
	&			
&	&	12	x'70'	x'26'
‡				
rec mark	'	0-2-8	x'1A'	x'27'
√	(	7-8	x'4F'	x'28'
□				
	)	12-4-8	x'7C'	x'29'
*	*	11-4-8	x'2C'	x'2A'
” Tape Seg	+	0-7-8	x'1F'	x'2B'
,	,	0-3-8	x'5B'	x'2C'
-	-	11	x'20'	x'2D'
.	.	12-3-8	x'3B'	x'2E'
/	/	0-1	x'31'	x'2F'
0	0	0	x'4A'	x'30'
1	1	1	x'01'	x'31'
2	2	2	x'02'	x'32'
3	3	3	x'43'	x'33'
4	4	4	x'04'	x'34'
5	5	5	x'45'	x'35'
6	6	6	x'46'	x'36'
7	7	7	x'47'	x'37'
8	8	8	x'08'	x'38'
9	9	9	x'49'	x'39'
:	:	5-8	x'0D'	x'3A'
;	;	11-6-8	x'6E'	x'3B'
<	<	12-6-8	x'3E'	x'3C'
= Word sep	=	0-5-8	x'5D'	x'3D'
>	>	6-8	x'0E'	x'3E'
? (Plus Zero)	?	12-0	x'7A'	x'3F'
@	@	4-8	x'4C'	x'40'

BCD glyph	ASCII glyph	Hollerith	BCD code	ASCII code
A	**A**	12-1	x'31'	x'41'
B	**B**	12-2	x'32'	x'42'
C	**C**	12-3	x'73'	x'43'
D	**D**	12-4	x'34'	x'44'
E	**E**	12-5	x'75'	x'45'
F	**F**	12-6	x'76'	x'46'
G	**G**	12-7	x'37'	x'47'
H	**H**	12-8	x'38'	x'48'
I	**I**	12-9	x'79'	x'49'
J	**J**	11-1	x'61'	x'4A'
K	**K**	11-2	x'62'	x'4B'
L	**L**	11-3	x'23'	x'4C'
M	**M**	11-4	x'64'	x'4D'
N	**N**	11-5	x'25'	x'4E'
O	**O**	11-6	x'26'	x'4F'
P	**P**	11-7	x'67'	x'50'
Q	**Q**	11-8	x'68'	x'51'
R	**R**	11-9	x'29'	x'52'
S	**S**	0-2	x'52'	x'53'
T	**T**	0-3	x'13'	x'54'
U	**U**	0-4	x'54'	x'55'
V	**V**	0-5	x'15'	x'56'
W	**W**	0-6	x'16'	x'57'
X	**X**	0-7	x'57'	x'58'
Y	**Y**	0-8	x'58'	x'59'
Z	**Z**	0-9	x'19'	x'5A'
(	**[**	12-5-8	x'3D'	x'5B'
'	****	0-6-8	x'5E'	x'5C'
)	**]**	11-5-8	x'6D	x'5D'
¢	**^**	2-8	x'10'	x'5E'
Δ (Mode Chg)	**_**	11-7-8	x'2F'	x'5F'
n/a	**`**	n/a	n/a	x'60

BCD glyph	ASCII glyph	Hollerith	BCD code	ASCII code
A	**a**	12-1	x'31'	x'61'
B	**b**	12-2	x'32'	x'62'
C	**c**	12-3	x'73'	x'63'
D	**d**	12-4	x'34'	x'64'
E	**e**	12-5	x'75'	x'65'
F	**f**	12-6	x'76'	x'66'
G	**g**	12-7	x'37'	x'67'
H	**h**	12-8	x'38'	x'68'
I	**i**	12-9	x'79'	x'69'
J	**j**	11-1	x'61'	x'6A'
K	**k**	11-2	x'62'	x'6B'
L	**l**	11-3	x'23'	x'6C'
M	**m**	11-4	x'64'	x'6D'
N	**n**	11-5	x'25'	x'6E'
O	**o**	11-6	x'26'	x'6F'
P	**p**	11-7	x'67'	x'70'
Q	**q**	11-8	x'68'	x'71'
R	**r**	11-9	x'29'	x'72'
S	**s**	0-2	x'52'	x'73'
T	**t**	0-3	x'13'	x'74'
U	**u**	0-4	x'54'	x'75'
V	**v**	0-5	x'15'	x'76'
W	**w**	0-6	x'16'	x'77'
X	**x**	0-7	x'57'	x'78'
Y	**y**	0-8	x'58'	x'79'
Z	**z**	0-9	x'19'	x'7a'
n/a	**{**	n/a	n/a	x'7C'
n/a	**|**	n/a	n/a	x'7C'
n/a	**}**	n/a	n/a	x'7D'
n/a	**~**	n/a	n/a	x'7E'
n/a	**DEL**	n/a	n/a	x'7F'

The full BCD encoding is not implemented at this time, but is cited here for consistency with the 1401 Simulator under Simh and the ROPE development environment.

How the encodings are chosen

The BCD format was set to match the encoding of the original Computer History Museum keypunch interface built by Stan Paddock, to ensure compatibility with card decks, programs and tradition.

The IBM 1130 formats are set to duplicate the assignments used by the IBM 1130 Simulator, created by Brian Knittel and available for download (IBM1130.org website) to permit files used with the simulator to be punched on cards with the correct Hollerith hole patterns, and to allow reading of real card decks into files for use on the Simulator.

A file in US-ASCII, such as a text file on a Microsoft Windows system, can produce the most widely used characters from the IBM EBCDIC character set, using the hole patterns for those characters (Hollerith codes) to punch the character.

EBCDIC is an 8-bit encoding, while we are using only 7 bit ASCII for our protocol link to the interface box inside the keypunch. The 256 possible character values in EBCDIC can't be fully covered by the 128 possible values of ASCII characters. Some of the ASCII characters have control uses, such as ctrl-C, new line, carrier return etc. so we don't have all 128 possible ASCII characters to choose from.

Fortunately, the number of printable characters (glyphs) that are assigned in EBCDIC is quite a bit smaller than 256. It is further limited for the purposes of keypunching because there is a very limited set of keycaps, even with different glyphs when shifted rather than when unshifted. All the common keycap glyphs on an 029 are represented by ASCII characters, although the glyph for the ASCII character is a different printed symbol than the glyph we intend to punch on the card.

There is a complicated relationship between printed images (glyphs), patterns of holes in a column, the value of that character as a number inside the computer that will read it, the value of the ASCII character which is used to request a character in the native code, and the windows or mac keyboard character that is pushed to add an ASCII character into a PC file.

Examples of complications –

- A given character in a native code like BCD may print differently depending on the encoding of the printer train/drum.
- Two IBM keypunches may use different character sets, where the same key on each machine will punch different hole patterns and print a different image on the top edge of the card.
- We are all familiar with localized keyboards that implement different letters used in different regions and alphabets, but more subtle differences can exist in the code set implemented in a PC.
- IBM 026 keypunches were commonly sold with two character sets – Fortran and Commercial
- The editor program that writes and changes a text file on the PC may store the pressed key in different ways depending on the character set chosen.
- This keypunch program imposes meaning on the character in the file to generate a specific pattern of holes in the card
- Codesets in the computing system that reads the cards can transform one such pattern into different digital values
- Conversions in the software using the card input can further change the meaning

There are many possible ways to ‘encode’ the card patterns but the methods chosen for this program should produce the intended hole pattern when using standard US keyboards on windows, mac or linux set to the US codes, stored in a US-ASCII format file. In spite of how complex this sounds, in most cases proper results will ensue without any special attention paid by the user of the system.

Version 1.1