

CORE Serial Interface (CS - 232) Manual

revision 3.0 By R. Karr, D. Sokol & T.J. Schmidt

Copyright @ 1987,1988 by CL9

Table of Contents

Introduction.....	1
Baud Rate.....	1
Hardware Protocol.....	1
Getting Started.....	2
Table 1: CORE key equivalents.....	3
Special CSUI Commands.....	5
Table 2: the special commands.....	5
CORE Memory Usage.....	6
How to Read Key Definitions.....	7
Descriptions of the Special Commands.....	8
Figure 1: CORE Display.....	9
CORE Serial Port Schematic.....	16

Introduction

This manual describes the procedures for using the CL9 CS-232 serial interface cable to communicate between a computer or terminal and the CORE universal remote controller.

The CORE itself contains a set of routines (called the CORE Serial User Interface or CSUI) which enables it to communicate with a computer or terminal at a rate of 9600 or 19200 baud. This interface allows users to control all of the CORE's functions from a keyboard or software program as if they were physically pressing keys on the CORE's keyboard. Also, there are additional functions available with CSUI which transmit data between CORE and the host computer.

This manual is intended to be used by someone who is already familiar with the CORE. (See the CORE Reference Manual for any questions regarding programming the CORE.) It also assumes proficiency on the computer (or system) communicating with the CORE. It assumes that you are able to initialize the serial port on your computer, set the baud rate to 9600 or 19200, and send characters as desired. If you are going to use the special commands, you must also be able to send and receive blocks of binary data and store them in your computer's memory and/or on disk.

Baud Rate

The CORE is shipped with the baud rate set to 19200.
To use 9600 baud, hold the '9' key down while pressing reset.
To reset to 19200 baud, hold the '1' key down while pressing reset.
A 'c'/reset (complete Clear) will reset the baud rate to 19200.

Hardware Protocol

The CORE is configured to transmit (and receive) 8 bits with 1 stop bit and no parity. As shipped, transmit is on pin 3, receive is on pin 2. Pins 2 and 3 are reversible using an internal switch (see Schematic on p. 15.) Pin 20 (DTR) is connected to +5. Ground is on pin 7.

Note: in the descriptions which follow, numbers preceded by a dollar sign (i.e. \$20) are hexadecimal values.

Getting Started

To begin serial communication, CORE must be in its normal display state, with the time on the top line and the location displayed on the bottom line. For example, if you are in the middle of setting an event timer, learning a program or editing a definition manually, CORE will not respond to characters sent over the serial port. You must manually exit these modes first. (**Note:** If the display is timed out, CORE will always be able to accept serial input).

To enter serial mode, send any character over the serial port. CORE Serial User Interface then outputs a "~" (ASCII \$7E = 126) to the serial port. CSUI then waits for a character from the serial port. If a character is not received in an allotted period of time (default timeout = 3 seconds), the CORE_ will sound a reject tone and resume its normal operation. The CORE display returns to the PAGE and KEY that was in effect prior to activation of the serial interface.

The next character sent to CORE over the serial port will again echo a "~" and so on.

1. If you send one of the characters listed in Table 1, CORE will respond as it would when the corresponding key is pressed on the CORE itself. The CORE will echo the character sent **at the completion of the command**. You **must** wait for the echo before sending the next character. (If the CORE is sending an infrared signal, the interrupts are disabled and no characters can be received.)
2. If you send one of the special control characters in Table 2, you will initiate one of the special sequences described in the section on Special CSUI Commands.
3. A Carriage Return or ^M (ASCII \$0D = 13) will echo as a Carriage Return, followed by a LineFeed (ASCII \$0A = 10)
4. Any other character will be echoed as a BELL or ^G (ASCII 07).

TABLE 1: ASCII character CORE Key Key Value


ASCII character	CORE Key	Key Value
A		\$00
B		\$01
C		\$02
P		\$03
a		\$04
b		\$05
c		\$06
d		\$07
1		\$08
2		\$09
3		\$0A
4		\$0B
5		\$0C
6		\$0D
7		\$0E
8		\$0F
9		\$10
0		\$11
E		\$12
F		\$13
=		\$14
@		\$15
e		\$16
X		\$17
<		\$18
+		\$19
-		\$1A
>		\$1B
S		\$1C
K		\$1D
[\$1E
]		\$1F

Table 1 shows the ASCII characters which should be sent to CORE to simulate the CORE keys. CORE echoes the same character that is sent. The third column shows the key value as it appears in key definitions.

Using the commands in Table 1 only, you can simulate the operation of the CORE keys from a computer or terminal. (Refer to the CORE Reference Manual for information on the syntax of CORE commands.)

There are a few differences between operating CORE using CSUI and in the normal manner using the CORE keyboard:

1. You must be aware that CSUI "times out" after a certain amount of time, normally 3 seconds. When CSUI times out, the next character sent will NOT be seen as a command but will echo as a "~" or wakeup character. If you are writing a software program to use CSUI, you must be careful to insure that CSUI is awake whenever you want to send a command. If desired, the length of the timeout can be increased (even made infinite) using the ^T command, described in the section on Special Commands.

2. You can go into learn mode with the  key (ASCII "[") in order to create new key definitions for CORE. However, you cannot capture infrared commands while in CSUI. Capturing can only be done when you are operating CORE directly from the CORE keyboard.

3. The cursor and other symbols will not flash or blink when in CSUI mode. For example, the bell symbol which normally flashes after a timed event has executed will simply appear solid.

4. CORE will also respond to physical keypresses while in CSUI. However, you should generally wait for CSUI to time out before pressing any keys on CORE (the results could be unpredictable in certain cases).

Special CSUI commands

These are additional commands usable only with CSUI, i.e. they are not CORE keyboard commands. The functions are:


<u>ASCII character</u>	<u>Function</u>
^C	Quit CSUI.
^D	Read the LCD display status
^K	Replace a single key definition
^L	Replace user memory.
^M	Carriage Return/Line Feed
^R	Send a program to CORE & run it.
^T	Set timeout.
^U	Read CORE user memory.
^W	Read a single key definition.

Most of the special commands involve transmission of long blocks of data to and from CORE. Therefore, they are intended to be used with a host computer and not with a terminal.

The protocol for error-free data transfer is as follows:

Data is transferred in blocks of 256 characters (or less, in some cases). When a block is sent or received, CORE sends an 8-bit checksum to the host computer. The checksum is calculated by summing all bytes received or transmitted. If an overflow (result greater than 256) occurs when adding, the carry is thrown away (i.e. sum wraps at zero).

The host should compare the checksum received from the CORE with its own calculated checksum. If the checksum matches, the host sends a "space" (ASCII \$20 = 32) to tell the CORE that the block is OK (CACK). If the checksum does not match, the host sends the CORE a "U" (ASCII \$55 = 85) to tell the CORE the block is incorrect (C-NAK), in which case that block is sent again, and the checksum re-calculated. This continues until all blocks are sent. The checksum is initialized to zero for each new or re-tried block. (\$55 was chosen as the C-NAK code because it is an alternating bit pattern [0,1,0,1,0,1,0,1, which tends to clear serial receivers.)

Note: the documentation of the Special Commands applies to Version 4.0 and higher of the CORE software. If you have version 3.E or lower, some of the commands may not apply, or there may be differences in application. You can call CL9 Customer Service for information on how to get a software upgrade. To find out what version you have, press the  key twice on CORE. You should see a 4-digit hexadecimal number, followed by a 2-digit hexadecimal number. If the 2-digit number is 40 or greater, you have the correct CSUI software.

CORE Memory Usage

When using the special CSUI commands, it can be useful to have some knowledge of CORE data structures and memory usage.

The CORE usable memory space is from locations \$4200 to \$7FFF.
\$4200 Event Queue
\$4280 - ? User Key Definitions

\$7900-\$7FFF contains variables and workspace used by CORE during normal operation.

Some important pointers (all pointers are two bytes, in low/high format)
7D00: Start Event Queue (normally points to \$4200)
7D02: Next Event in Queue
7D04: Last Byte of Event Queue
7D06: Start of Key Definitions (initialized to \$4280; can be higher)
7D08: End of Key Definitions

The key definitions, which normally begin at location \$4280, are structured in the following way:

Page: a one-byte binary value from \$00 to \$OF representing the Page Location of the program.

Key: a one-byte binary value from \$00 to \$OF representing the Key location of the program. A Key value of \$FF is used to represent a direct-access Page location, e.g. location a- is represented as Page \$OA, Key \$FF.

Length: a one-byte binary value from \$00 to \$FA representing the length of the program to follow. If the length is zero, there is no program, and the next byte in memory will be the start of the next program

Program: the program contains the number of bytes specified above in Length.

The key definitions are sorted in ascending order by Page and Key. That is, location 0-0 is first, followed by 0-1 (if present), etc. up to F-F. Following the last definition are the bytes **\$OF-\$FF-\$00**.

How to read key definitions

Each key on the CORE keyboard has a value from 0 to 31 (\$1F). Table 1 shows the key values for each CORE key.

In addition, there are a few special values that may show up.

1. If the high bit is set, the character is displayed with a dash. For example, the program **03 88 09** (all hexadecimal) translates to **P 1-2** on the CORE display.

2. A program value of **\$22** represents the **h** symbol that can appear in programmed intervals.

3. A program value of **\$21** refers to an infrared (IR) code, and is represented by the **C** symbol on the display. A value of **\$23** refers to a "one-shot" type of IR code, and is represented by a backward **C** symbol. Both of these values will be followed by several bytes of data representing the IR code itself (these bytes, of course, do not show up in the display).

The structure of the IR data is as follows:

Byte 0: \$21 or \$23

Byte 1: length of 1st word of IR code (L). The length includes byte 1; therefore the first word goes from Byte 1 to Byte L.

Byte 2: if bit 1 is set, there is a 2nd word. If bit 1 = 0, there's no 2nd word

Bytes 3 - L: rest of 1st word.

If the 2nd word is present (as determined by Byte 2 above), there will be another length byte and a series of bytes as above. There are never more than 2 words to the IR definition.

Descriptions of the Special Commands

Command: Exit serial Interface mode (^C)

Response: CORE sends back a "C" (ASCII \$43 = 67)

Description: This command is used to quit CSUI. The CORE returns to normal operation. The display will revert to the PAGE-KEY location that was active before CSUI was entered.

Command: Read LCD display (^D).

Description: This command is used to read the CORE's display status. The condition of each LCD segment and the display status is transmitted.

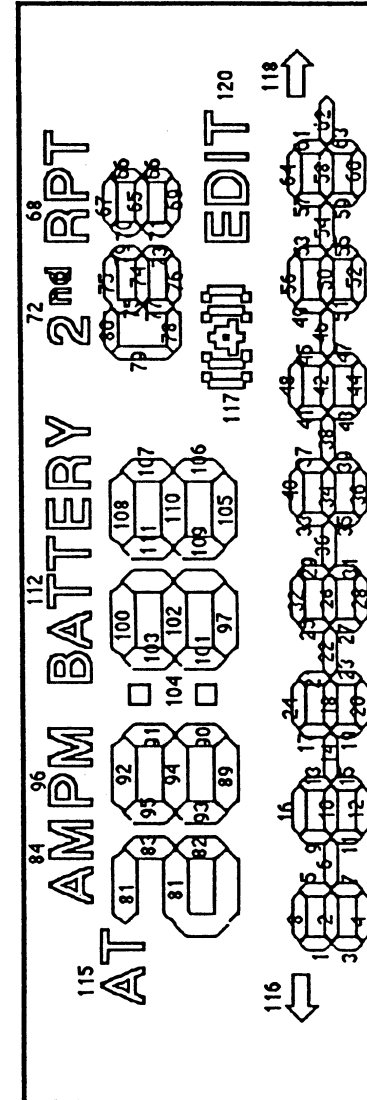
The host sends a ^D (ASCII 04). CORE echoes a D (ASCII \$44 = 68). CORE then sends the following data:

1. 32 bytes representing the current state of each segment on the CORE's LCD display. Only the lower 4 bits of each byte are relevant (the upper 4 contain junk and should be ignored).

The diagram on the next page shows the layout of the segments as they are numbered on the CORE display. They are numbered from 1 to 120. These segments correspond to the first 30 bytes received with the ^D command (the 31st and 32nd bytes are actually meaningless). For example, segments 1, 2, 3 and 4 correspond to bits 0, 1, 2 and 3 of the first byte (Byte 0). The general formula =

Byte # = Integer((Segment # -1) ÷ 4)
 Bit # = (Segment # - 1) Mod 4

Figure 1: CORE Display



Note: Segments 85, 86, 87, 88, 113, 114 and 119 are unused

2. 1 byte indicating where the cursor is on the display. (again, ignore the upper 4 bits)

- 0 = left arrow
- 1-8 = position on the edit line
- 9 = right arrow
- 11 = on the Key (top line, right of dash.)
- 13 = on the Page (top line, left of dash)

3. 1 byte indicating which segment(s) are flashing. A 1 indicates a particular segment is flashing.

- Bit 0 = bell is flashing.
- Bit 1 = location is flashing (Edit mode)
- Bit 2 = Alarm symbol is flashing.
- Bit 3 = EDIT is flashing.
- Bit 4 = 2ND is flashing.
- Bit 5 = single step is flashing
- Bit 6 = cursor is flashing
- Bit 7 = is unused.

Command: Replace a key definition (^K)

Description: This command is used to load a single key definition to a Page-Key location in the CORE. Any previous key definition stored at the location is erased.

The sequence goes as follows:

1. Host: ^K (ASCII \$0B= 11)
2. CORE: K (ASCII \$4B = 75)
3. Host: Page (binary number between \$00 and \$0F)
4. Host: Key (binary number between \$00 and \$0F, or \$FF for direct-access keys)
5. Host: Count byte (binary number of bytes of data to follow = \$00 to \$FA)
6. Host: Data block (0 to \$FA bytes)
7. CORE 1-byte checksum of bytes received
8. Host: If checksum is OK, sends C-ACK (ASCII \$20 = 32). *

* **Note on checksum:** It's a little tricky to deal with the case where there is an error in the checksum and you send the C-NAK (ASCII \$55 = 85)

First of all, on a retry, CORE expects data only (step 6). Then, on step 7, CORE forgets to clear the checksum so that what it sends following the retry will = (1st (incorrect) checksum + 2nd checksum (data only)] mod 256. It's a little dangerous to try to recover from this, because if the Page, Key or Count were received incorrectly by CORE, some garbage could get stored in your CORE's memory.

Recommended procedure: if you get an incorrect checksum from CORE:

1. send any character other than C-ACK or C-NAK. This causes CORE to abort the current command and wait about 3 seconds for more input (any input during this time is ignored).
2. Wait 3 seconds, then send the ^K command in its entirety again. (Instead of waiting 3 seconds, you can continue to send the "K until you receive the correct echo.)

Command: Replace user memory (^L)

Description: This command is used to replace the CORE user memory. This is the area of memory (locations \$4100 through \$7FFF of CORE memory) used to save the key definitions and event queue, as well as other variables.

Normally this command will be used to load a user file that has previously been saved on disk using the ^U command

The sequence goes as follows:

1. Host: ^L (ASCII \$0C = 12)
2. CORE: L (ASCII \$4C = 76)
3. Host: 1st 256-byte block
4. CORE: 1-byte checksum
5. Host: If checksum is OK, sends C-ACK (ASCII \$20 = 32) & next block.
If not OK, sends C-NAK (ASCII \$55 = 85) & repeats block

Steps 4-5 repeated for 63 blocks.

Note: Using this command destroys all key definitions previously stored in CORE's memory.

Command: Carriage Return (^M)

Character sent: ^M (ASCII \$0D = 13). CORE echoes a carriage return (ASCII \$0D) AND line feed (ASCII \$0A = 10)

Command: Download a program to RAM and run It (^R)

Description: This command is used to load and execute a machine language program to CORE's RAM and run it. If no address is specified the program will be loaded at \$6000. The program will execute after loading. The CORE uses a Mitsubishi M50743 processor, which runs a superset of 6502 machine language (i.e. 6502 code will run on the CORE).

The sequence goes as follows:

1. Host: ^R (ASCII \$12 = 18)
2. CORE: R (ASCII \$52 = 82)
3. Host: optional address to load and run program: this is a 2-byte binary number in low/high format. (if this value is not present, the program will automatically download and run at location \$6000). Low byte must be \$00; high byte should be a value between \$41 and \$7F.
4. Host: number of blocks (binary number between 1 and \$FF)..
5. Host: 1st 256-byte block
6. CORE: 1-byte checksum
7. Host: If checksum is OK, sends C-ACK (ASCII \$20 = 32) & next block.

If not OK, sends C-NAK (ASCII \$55 = 85) & repeats block

Steps 6-7 repeated for # of blocks specified in step 4.

Note: if you are going to download a program to CORE memory, you will overwrite whatever is in memory. You should generally put a program as high in memory as possible, to avoid overwriting your key definitions, but the end of your program should not go past \$7900.

Command: Set length of CSUI timeout (^T)

Description: This command allows the user to alter the CSUI timeout, which is the amount of time that CSUI will run before exiting to normal CORE operation. The timeout can be from 1 to 51 seconds, or infinite.

The host sends a ^T (ASCII \$14 = 20). CORE echoes a "T" (ASCII \$54 = 84).

The host then sends a **binary** number between 0 and 51. If zero is sent CSUI will **never** time out. A number greater than 51 will default to 51

Examples: ^T 1 sets the timeout to 49 seconds ("1" = ASCII 49).
^T ^A sets the timeout to 1 second (^A = ASCII 1).

NOTE: The default timeout is about 3 seconds. Whenever a c/reset is done, the timeout is set to the default.

Command: Read data from CORE's memory (^U)

Description: This command is used to read the CORE's memory, from locations \$4100 to \$7FFF. The number of blocks transmitted is 63.

The sequence goes as follows:

1. Host: ^U (ASCII \$15 = 21)
2. CORE: U (ASCII \$55 = 85)
3. CORE: 1st 256-byte block
4. CORE: 1-byte checksum
5. Host: if checksum is OK, sends C-ACK (ASCII \$20 = 32) & CORE sends next 256-byte block
if not OK, sends C-NAK (ASCII \$55 = 85) & CORE sends same 256-byte block again.

Steps 4-5 repeated for 63 blocks.

Command: Read a single key definition (^W).

Description: This command is used to read an individual key definition from the CORE's memory.

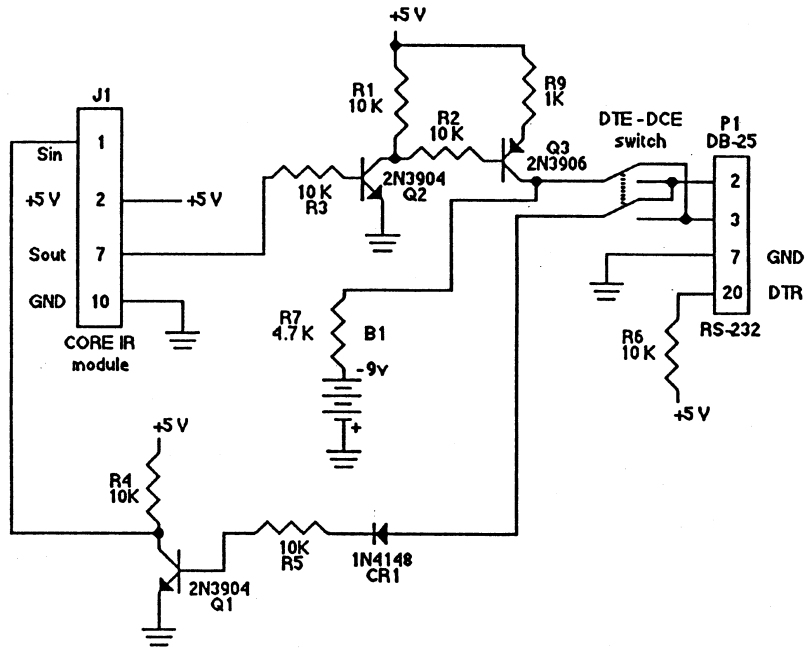
The sequence goes as follows:

1. Host: ^W (ASCII \$17 = 23)
2. CORE: W (ASCII \$57 = 87)
3. Host: Page (binary number between 0 and \$F)
4. Host: Key (binary number between 0 and \$F, or \$FF for direct-access keys)
5. CORE: echoes Page, Key. Then sends length of program to follow (\$00 to \$FA). A length of \$00 means key is not programmed.
6. CORE: Data block (0 to \$FA bytes)
7. CORE: 1-byte checksum of bytes sent
8. Host: If checksum is OK, sends C-ACK (ASCII \$20 = 32). *

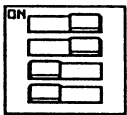
* **Note on checksum:** It's a little tricky to deal with the case where the checksum is wrong. If there is an error in the checksum and you send the C-NAK (ASCII \$55 = 85), CORE will re-send the data **ONLY** (no Page, Key or Length byte). In addition, CORE forgets to clear the checksum, so that what it sends following the retry will = [1st (incorrect) checksum + 2nd checksum (data only)] mod 256.

If CORE sends the wrong checksum back, the easiest way to deal with this is to send a C-ACK, so CORE will not repeat the data, and then send the ^W command **again**.

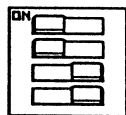
CORE SERIAL PORT SCHEMATIC - D. Sokol



CARD EDGE



CORE TRANSMITS ON
PIN 3



CORE TRANSMITS ON
PIN 2