Technical Information Manual

Revision n.6
3 December 2007

MOD. C111 C

ETHERNET CAMAC
CRATE CONTROLLER
MANUAL REV. 6

NPO:
00108/04:C111C.MUTx/06

CAEN will repair or replace any product within the guarantee period if the Guarantor declares that the product is defective due to workmanship or materials and has not been caused by mishandling, negligence on behalf of the User, accident or any abnormal conditions or operations.

CAEN declines all responsibility for damages or injuries caused by an improper use of the Modules due to negligence on behalf of the User. It is strongly recommended to read thoroughly the CAEN User's Manual before any kind of operation.

$$\epsilon\epsilon$$

CAEN reserves the right to change partially or entirely the contents of this Manual at any time and without giving any notice.

Disposal of the Product

The product must never be dumped in the Municipal Waste. Please check your local regulations for disposal of electronics products.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Document type:	**Title:**	**Revision date:**	**Revision:**
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

C.A.E.N.

1. General description

1.1. Overview

The CAEN Mod. C111C is a complete CAMAC controller that allows advanced interaction by means of standard Ethernet services, such as a local web server and TCP socket based communication protocol. The basic hardware architecture of C111 C is illustrated below.
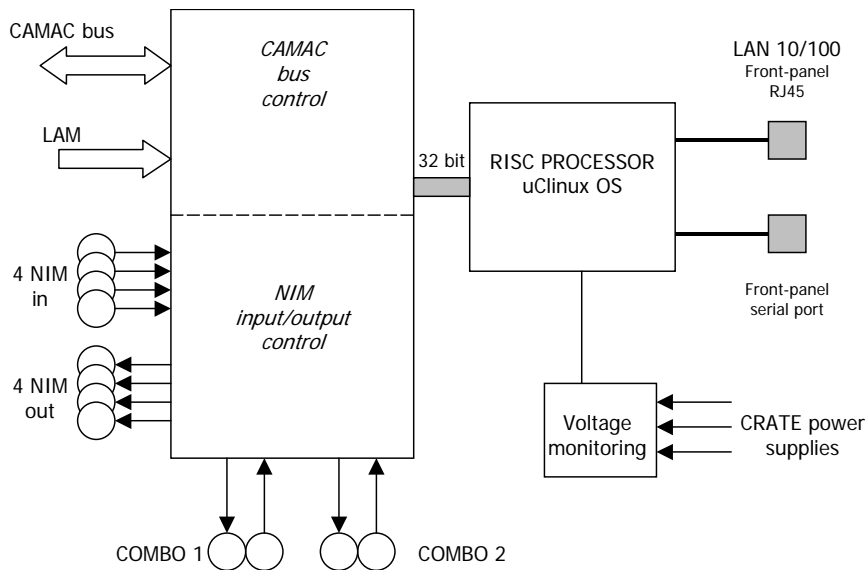
Fig. 1.1: Mod. C111C block diagram

A local processor runs a version of Linux optimized for low memory footprint; a CAMAC bus control subsection handles all bus access operations and interactions, and a separate NIM subsection manages I/O signals located on the front panel.

Table 1.1: Summary of features

CAMAC bus access	Full CAMAC bus control, including LAM detection Plugs into slots 24 and 25
Local NIM I/O Section	4 outputs, 4 inputs, event counters, 2 COMBO I/O (trigger/busy) modules programmable pulse generators input event counters NIM default settings can be reloaded with front panel button
Remote Control Library	ANSI C remote control library derived from the ESONE standard, with extensions to control local resources Remote control of all functions through TCP socket
Local Web Server	Dynamic local web server allows advanced monitoring and control without the need to install dedicated software (perfect for crate setup and maintenance) User page with results from script
Advanced Scripting Engine	Embedded script interpreter allows local execution of C-like code, with full control on CAMAC and NIM functions No need to install cross-compilation toolchains
Front Panel Indicators	X and Q signals on last access 4 user LEDs (controllable from script) Fault, connection status and NIM default indicators

C.A.E.N.

Document type:	Title:	Revision date:	Revision:
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

2. Starting up

- Please insert the controller **into slots 24 and 25 ONLY** of a standard CAMAC crate (rightmost when looking the crate from the front side)
- if default network settings are compatible with your setup, connect a LAN cable to the front-panel RJ45 socket
- Power up the CAMAC crate
- Wait about 20 seconds to allow completion of operating system boot (It might require a longer time, depending on your network configuration, especially DHCP).
- If default network settings are not suitable for your network environment, connect a terminal to the front panel serial port and make the necessary variations (see **Serial port control** section). After that, reboot (either by cycling the main power or by pressing the RESET front panel button) and wait about 20 seconds.
- Note: if you connect directly an PC with the module (i.e., a point-to-point connection) you MUST use a cross cable; this is a typical requirement for LAN devices.
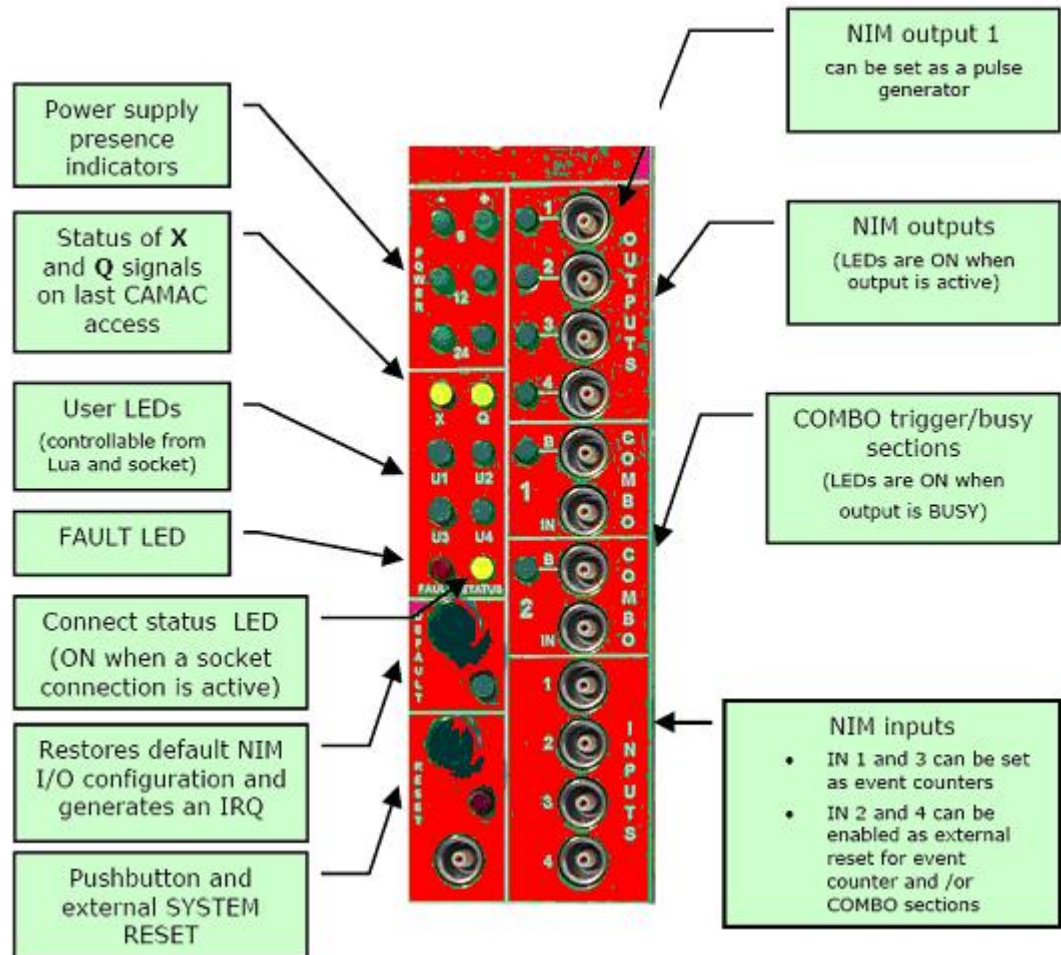

Fig. 2.1: Front Panel description

Please open your browser (on a host connected to the same LAN used by C111C) and point to its IP address; the default IP address is 192.168.0.98. From the web server pages, you already have control over the NIM I/O section and the possibility to perform individual commands on the CAMAC bus.
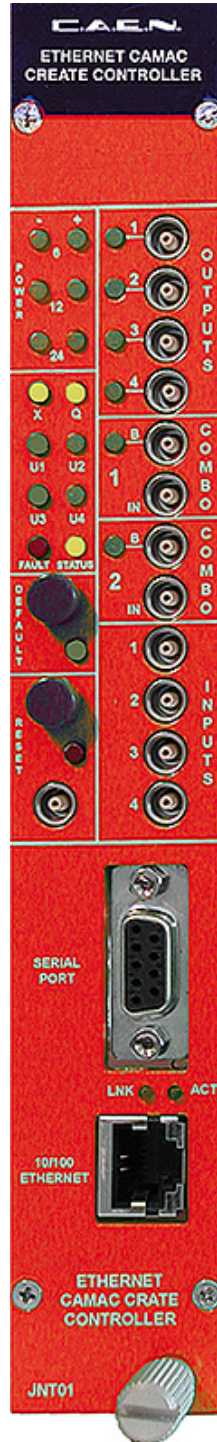
2.1. Front Panel

Fig. 2.2: Mod. C111C Front Panel

3. Serial port control

A serial port connector located on the front panel allows the user to modify system settings; this procedure is required if current network parameters are incompatible with the local network. It is also possible to modify startup options.

Default connection parameters are the following: 38400 baud, 8-N-1, no flow control. Please notice that the baud rate can be modified by a dedicated command. Echo is not enabled on the serial port, so please enable character echo on your serial terminal.

The following commands are available:

Table 3.1: Serial port available commands

help	Provides a quick list of commands on terminal
setip <new IP addr>	sets a new IP address, to be written in the format *aaa.bbb.ccc.ddd*
setmask <new mask>	sets a new IP mask
setgw <new gw addr>	sets a new gateway IP address
setdhcp <0|1>	if set to 1, enables the local DHCP client
getip getmask getgw getdhcp getmac	Allows retrieval of current network settings and of the internal MAC address
setrob <0|1>	if set to 1, enables the Lua Run-On-Boot option
setcscan <0|1>	if set to 1, enables the Crate Scan function (executed at startup only)
getrob getcscan	Allows retrieval of current startup settings (Lua Run-On-Boot and Crate Scan)
getserial	Allows retrieval of current board serial number
getcspeed setcspeed <baudarate>	Sets/gets current COM speed Allowed speeds: 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400; if the baudrate value is not allowed, speed defaults to 38400 baud.
getname setname <name string>	Sets/gets current name displayed on the web server index page (string name can be up to 16 characters)
listuser adduser <username:pwd> deluser <username:pwd>	Manage current authorized web users list (username and pwd are ascii string of any length)

Note:	*The front-panel connector requires a straight serial cable (pin 2 to pin 2, pin 3 to pin 3); only RX, TX and GND are required.*

The new firmware release (2.05 or greater) provides an additional feature that, when enabled, takes control over the serial port and allocates a TCP socket to serial converter. A TCP socket server is activated on port 2003; any terminal-like application can connect as a TCP client and interact remotely with the serial port. To enable the TCP socket server. To modify serial port settings when in TCP server mode, see within the System Parameters (ch. 10).

Please notice that when the TCP server is enabled, the above protocol is not implemented.

This feature is extremely useful if a serial-controlled device is located near the crate, i.e., a serial LCD display, a remote data acquisition or I/O expander, a local PLC.

4. Startup options

C111C startup options can be tuned in order to provide a fine control over performance and connectivity. There are some settings stored in nonvolatile memory that can be changed with the control socket or from the serial port, and some options that require placement or removal of jumpers inside the unit.

4.1. Startup flags

Two flags are available for startup fine-tuning:
Lua script Run-On-Boot flag
Crate Scan Enable flag
Both are stored in the local EEPROM and can be modified with dedicated commands on the serial port (**getrob**, **setrob**, **getcscan**, **setcscan**) or from the control socket.
The **Lua Run-On-Boot** flag, when enabled, tells the system to run the stored Lua script after starting up the application; its main usage is for automated crate initialization and for unattended control.
Crate Scan is a function available on C111C to allow automatic detection of cards inserted into the crate. It is a quick way of verifying the presence of cards that may be required by the acquisition code (either within the Lua script or on the host application).
Being a heuristic approach to card detection (there is no formal way of detecting a card when inserted into the crate) the Crate Scan function may interfere with specific CAMAC cards; it is thus possible to disable the Crate Scan function. Note that Crate Scan is executed <u>only at startup</u> (in order to avoid possible interaction with ongoing script or actions from host); if the feature is disabled, it will not be possible even from socket.

A description in pseudo-code of the Crate Scan function follows:

```
for (slot=1; slot < 23; slot++)    {
  end_slot = 0
  for (fun=0; fun < sizeof(SCAN_FUNCTION); fun ++)
    for (addr = 0; addr < 32; addr ++)  {
      X = CSSA (slot, SCAN_FUNCTION[fun],addr, 0)
      if (X == 1)  {
            slot_status[slot] = 1
            end_slot = 1
            }
    }
  if (!end_slot)  slot_status[slot] = 0
  }
```

where SCAN_FUNCTION is the following array:
0, 1, 2, 3, 8, 9, 10, 11, 24, 25, 26, 27, 16, 17, 18, 19

4.2. Jumper settings

It is possible to force some startup options by placing or removing internal jumpers.
Only experienced personnel should perform this operation.
The operation is described in steps, as follows:
– switch off power from the crate
– remove the controller from the crate
– remove the lateral cover (left side when looking on the front panel) of the controller; you should see the internal boards with components facing your side

CAEN

Document type:	Title:	Revision date:	Revision:
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

– locate the jumper block, right behind the serial port connector
– note that if all jumpers are <u>removed</u> (default condition), then the unit will perform in the standard mode; insert jumpers only if you want to modify the standard setup, according to the table:

Table 4.1: Available jumpers

Insert jumper on	to disable	Notes
JMP10	TELNET	Disable telnet to avoid system access to local resources; used only for performing a firmware upgrade or for debug.
JMP9	WEB SERVER	Disable web server if HTTP access not required in order to obtain maximum performance; note that it cannot run if application is also disabled
JMP8	APPLICATION	Disable application if the system hangs after powerup (i.e., after failing a firmware upgrade)
JMP7	COM SERVER	Disable serial port application if the system keeps hanging after powerup; it may also be used to prevent local access to critical parameters (i.e., network settings)

Please notice that, on C111C, TELNET is an insecure method to control the unit; no password and no encryption are provided. Therefore, if the application requires it, it may be safer to disable telnet access.

5. Remote control

The unit has been designed to allow full control from a remote location, taking advantage of the available high-speed Ethernet interface.
A summary of the implemented TCP/IP services is detailed.

Table 5.1: Remote control available commands

Service	Port	Notes
HTTP server	80	Dynamic Web Server; it serves up to 5 client browsers at the same time.
TCP server	2000	TCP control socket for ASCII commands; up to 2 different clients are allowed at the same time (see **TCP ASCII control socket** section).
TCP server	2001	TCP control socket for binary commands only (see **TCP binary control socket** section).
TCP server	2002	TCP socket server for interrupt management (see **Interrupt handling** section).
TCP server	2003	TCP socket server for socket to serial link (see **Jumper settings** section)
Telnet server	23	System telnet server; it is used mainly for firmware updates and may be disabled by the user (see **Startup options** section). Please note that the telnet connection is unsafe (no password-protected access, no encryption).

The **local web server** allows an easy and quick access to CAMAC commands, test and monitoring functions. Simple CAMAC operations can be easily performed by means of a user-friendly web interface, with no need of programming or learning manuals. This very useful especially when performing quick lab tests on CAMAC modules. See section **Local Web Server** for details.

The **socket connection** is the main control method for general applications: the host computer opens a TCP connection to the C111C IP address at port 2000 and then starts sending commands. A command is a simple ASCII string. Command can be sent by host computers through a specific DAQ application, or manually, using a terminal program like **telnet** (for Unix/Linux) or **HyperTerminal** (for Windows). For example, a socket connection can be manually opened from a Linux host by typing: *telnet <jenet IP addr> 2000*. As soon as the connection is established, C111C is ready to accept commands from the host keyboard. See section **TCP ASCII control socket** for details, and section **ASCII commands reference** for a complete list of the ASCII commands.

The command set of C111C is composed of simple ASCII strings. Moreover, a command subset is also available in "**binary format**" to improve speed performances. Port 2001 is dedicated to this function. See section **TCP binary control socket** for details.

A **telnet server** is also available on port 23. Typing *telnet <jenet IP addr>* the user can access the C111C internal filesystem. It is recommended for expert users only.

C111C can also notify the host computer that some asynchronous external events (LAM, COMBO trigger and DEFAULT button pressure) have occurred. The TCP port 2002 is dedicated to this function. See section **Interrupt handling** for details.

The internal software architecture is designed to allow control of multiple crates. It must be considered that, when using an Ethernet-based CAMAC controller, the distinction between crates is implicit as every crate is identified by a different IP address. Therefore the crate number is typically not a parameter in many command definitions.

5.1. TCP ASCII control socket

Remote control is organized as follows:
the local firmware opens a TCP socket server on port 2000 and waits for connection from a remote client;
when a client connection is detected, front panel LED status lights up; it will remain on until there are active connection to the socket server;
the server accepts ASCII commands from the remote client (see section **Commands Reference**); return string is always in the following format:

Table 5.2: control socket format

-1	Command exists, but parameter format or number is wrong
-2	Command does not exist
0 <return value>	Command returns correctly the return value (it may also be null)

note that more than one remote client can connect to the socket server; it is up to the programmer to avoid conflicts when accessing the same resources, as there is no built-in protection for access conflicts (in other words: stick to one remote client only unless you really know what your are doing). Multi-client usage is useful especially during development and debug.
The socket server NEVER generates data autonomously; in the special case when C111C needs to communicate to the host that a specific event has occurred (analogously to an interrupt request), it works through a separate communication channel (IRQ port 2002).

A complete reference of ASCII commands is available in section 10.

Note:	*On Windows 2000/XP, it is possible to perform a quick test with the Hyperterminal application, by specifying connection with TCP/IP and port 2000; on linux hosts, you can use the standard telnet client in "raw" mode , by typing* `telnet <Jenet IP address> 2000`

5.2. TCP binary control socket

A "binary command subset" is also available to increase speed and data transfer rate. A TCP server for binary commands is available on port 2001: binary commands must be sent through that port only.
A dedicated C/C++ library has been written to use these commands in a straightforward and transparent way, with no need to know all the implementation details described here. **Users writing their host DAQ applications in C or C++, can skip this chapter.**
See section **C library** for details.
In general, the binary command has the following format:

```
byte(0) = STX;
byte(1) = CMD_CODE;
byte(2) = databyte(0)
byte(3) = databyte(1)
....
byte(n) = databyte(k)
byte(n+1) = REQ_RESPONSE;
byte(n+2) = ETX;
where:
```

STX is the hexdecimal value 0x02
ETX is the hexdecimal value 0x04
CMD_CODE may be one of the followings value:
BIN_CFSA_CMD = 0x20 (equivalent to the ascii command cfsa)
BIN_CSSA_CMD = 0x21 (equivalent to the ascii command cssa)
BIN_CCCZ_CMD = 0x22 (equivalent to the ascii command cccc)
BIN_CCCC_CMD = 0x23 (equivalent to the ascii command ccci)
BIN_CCCI_CMD = 0x24 (equivalent to the ascii command ctci)
BIN_CTCI_CMD = 0x25 (equivalent to the ascii command ctci)
BIN_CTLM_CMD = 0x26 (equivalent to the ascii command ctlm)
BIN_CCLWT_CMD = 0x27 (equivalent to the ascii command cclwt)
BIN_LACK_CMD = 0x28 (equivalent to the ascii command lack)
BIN_CTSTAT_CMD = 0x29 (equivalent to the ascii command ctstat)
BIN_CLMR_CMD = 0x2A (equivalent to the ascii command clmr)
BIN_CSCAN_CMD = 0x2B (equivalent to the ascii command cscan)
BIN_NIM_SETOUTS_CMD = 0x30 (equivalent to the ascii command nim_setouts)
databyte(0) ..databyte(k) is of variable length according to the command code
REQ_RESPONSE may be:
NO_BIN_RESPONSE = 0xa0 (no response requested)
Any other value (response requested)

If one of the databyte(0)..databyte(k) contains 0x2, 0x4 e 0x10, then the databyte must be converted in two bytes accordingly to the following rule:
if databyte(n) = 0x2   => converted in => databyte(n) = 0x10; databyte(n+1) = 0x80 + 0x02;
if databyte(n) = 0x4   => converted in => databyte(n) = 0x10; databyte(n+1) = 0x80 + 0x04;
if databyte(n) = 0x10 => converted in => databyte(n) = 0x10; databyte(n+1) = 0x80 + 0x10;

A complete reference of binary commands is available in section 12.

5.3. Interrupt handling

On C111C, specific events can generate an interrupt (IRQ) that is notified to the host. Being socket-based, the connection method is rather different from a bus-based connection; therefore, interrupt are handled in a message-based way.

There are three possible IRQ sources:
LAM requests
COMBO Triggers (see **COMBO I/O** chapter)
DEFAULT pushbutton pressure

When an IRQ event is generated, C111C sends a special string to the host computer through the dedicated TCP server at port 2002. The string format is a upper case letter followed by a 32-bit hex value in ascii:
LAM events: **"L_<00hhhhhh>"**, where <hhhhhh> is the ASCII representation of the LAM register content in hex (24-bit).
COMBO events: **"C <bitmask> "**, where:
bit0 = combo1 interrupt pending
bit1 = combo2 interrupt pending
bit2 = dtc combo1 interrupt pending
bit3 = dtc combo2 interrupt pending
bit4-31 = ignored

DEFAULT pushbutton pressure: **"D   defadefa"**

On the host, an IRQ dedicated client receives the messages and launches the proper IRQ-service program. For example, if a LAM or COMBO generated IRQ is received, the host can start a reading sequence of some modules.

The IRQ generated by a DEFAULT pushbutton pressure is a very powerful feature that allows the user to start different programs at each pressure of the button: for example, different module setups can be activated when the button is pressed and different actions can be performed.

Please refer to the C library documentation for further details.

In the following diagram, a typical scenario with 2 separate crates is shown.
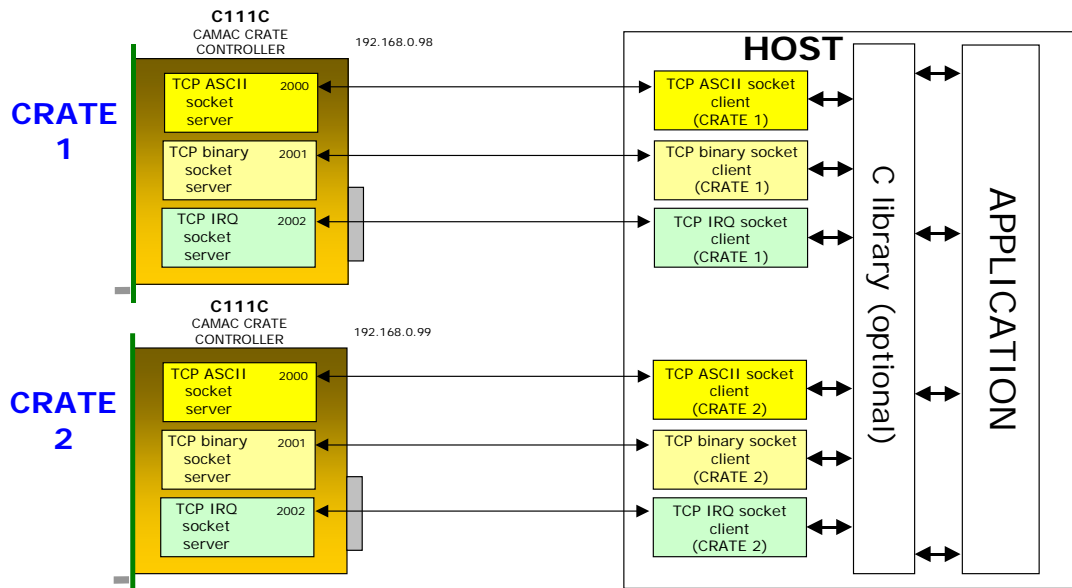
Fig. 5.1: Remote control with 3 separate crates

5.4. **Block transfers**

The C111C TCP protocol has been expanded with block transfer commands. Care has been put in optimizing performance; while the whole protocol on TCP has been designed to be as simple and intuitive as possible, block transfer commands are not following this approach.

The following block transfer modes are implemented:

Address Scan mode

Repeat mode

Stop mode

Following the IEEE standard, the Q line assumes different meanings depending on the selected type of block transfer:

Table 5.3: Q line meaning

Reply	Address Scan mode	Repeat mode	Stop mode
Q = 1	Register is present	Register is ready	Continue block transfer
Q = 0	Register is missing	Register is not ready	End block transfer

Some remarks follow on our solution to block transfer.

- Block data is transferred on the same TCP socket connection used for commands.
- During a block transfer, no other commands can be sent over the socket connection; therefore, block transfer must be completed before other commands can be sent. The user must take care of this limit, as any command can cause a block transfer abort.
- If multiple clients connect to the same TCP control socket server on C111C, they might violate the condition detailed above. As already mentioned on the User's Manual, don't use multiple clients unless you are debugging or you really know what you're doing.
- To allow transfer of large block data in read operations, local buffering is implemented. Read data are transferred to the host in buffers, every time the local buffer is filled. Buffer size is programmable to adapt different requirements and calibrate the tradeoff between optimal transfer efficiency and response time.
- Read operations are available in ASCII and binary mode. ASCII mode is perfect for quick debugging and verification, while binary mode offers higher performance at the cost of increased protocol complexity (being a mixed ASCII-binary protocol).
- Write operations are only available in in ASCII mode.
- Write operations are consumed in streaming mode: C111C does not wait for the whole block data set to begin writing.

A brief summary of available commands follows:

Table 5.4: Block transfer available commands

Utility	BLKBUFFS	Block transfer buffer size set
	BLKBUFFG	Block transfer buffer size get
Q-stop	BLKSS	Block transfer, 16-bit, Q-stop mode
	BLKFS	Block transfer, 24-bit, Q-stop mode
Q-repeat	BLKSR	Block transfer, 16-bit, Q-stop mode
	BLKFR	Block transfer, 24-bit, Q-repeat mode
Address Scan	BLKSA	Block transfer, 16-bit, address scan mode
	BLKFA	Block transfer, 24-bit, address scan mode
In general the command is expressed as **BLKsm** where s = S (short), F(full) m = S (Q-stop), R (Q-repeat), A (address scan) Read or write mode is determined by the Function code passed as a parameter, as follows: F = 0,……,7 ➔ READ mode F = 16,…,27 ➔ WRITE mode		

All block transfer commands have the same behavior. C111C replies to the command itself immediately after reception, before executing the actual block transfer, with one of the following possible replies (compliant with the standard command response of the TCP control socket protocol):

Table 5.5: Block transfer possible replies

Reply	
0	OK, operation in progress
-1	error, wrong parameters
-2	error, non existing command

The general format of a data block is

hdr data1 data2 ….. dataK

where:

K is the current buffer size
in ASCII mode, cmd is formatted as %03X
in ASCII mode, dataj is formatted as %06X (for both 16-bit and 24-bit access types)
in ASCII mode, the data block is terminated by a "\r" character
in binary mode, cmd and dataj are all 32-bit values

in binary mode, the data block is (K+1)*4 bytes
if there are non significant data values (if cmd < K, or cmd = 0), data block size is always the same as above

hdr is a header that can assume one of the following values:

Table 5.6: Cmd values

hdr		notes
0	End of block transfer	data1= actual datasize data2,…dataK = non significant
N > 0	Number of data words being transferred	If N<K, then dataN,dataN+1,…..,dataK are non significant
-3	Timeout error	data1= actual datasize data2,…dataK = non significant
-4	Abort error	data1= actual datasize data2,…dataK = non significant

A complete reference of block transfer commands is available in § 10.

5.5. C library

Although the host programs can be written in any language, a C library is available to simplify code generation: host clients, IRQ handling and binary commands can be handled in a very easy and transparent way that releases the programmer from taking care of low-level details.
The **C library** is an ANSI C library, delivered in source form, providing an ESONE-like abstraction to the socket protocol, including multiple crate support. Documentation specific to the C library is on a separate document available on the C111C support web site.
The C library is compatible with C++ compilers like g++.
Please note that the C library on host is provided "as is", in source code form, without any form of warranty of support. You are allowed to modify it freely, but under any circumstance you are responsible for its use (o misuse).

5.6. Remote Reset

An hardware reset can be given either by pressing the "RESET" pushbutton on the front panel or, by remote, closing an external switch connected to the "RESET" input on the front panel.
A remote reset can also be sent from the network:From a socket connection on port 2000: send the command "reset". If working from a terminal window, just type "reset".
6.From a telnet connection (port 23): type "reboot".

5.7. Local web server

The local web server is a <u>dynamic server</u>, in the sense that gathers relevant data and information directly from the machine; therefore, depending on page contents, page refresh may be slower than expected if compared to a fully static web site.

A username and a password are required to access the local web server. Default values are "*jenet", "jenet*". Other usernames and passwords can be added, modified or deleted by means of the commands *"user_add", user_del", user_list"*. See the **Command reference** section for details.
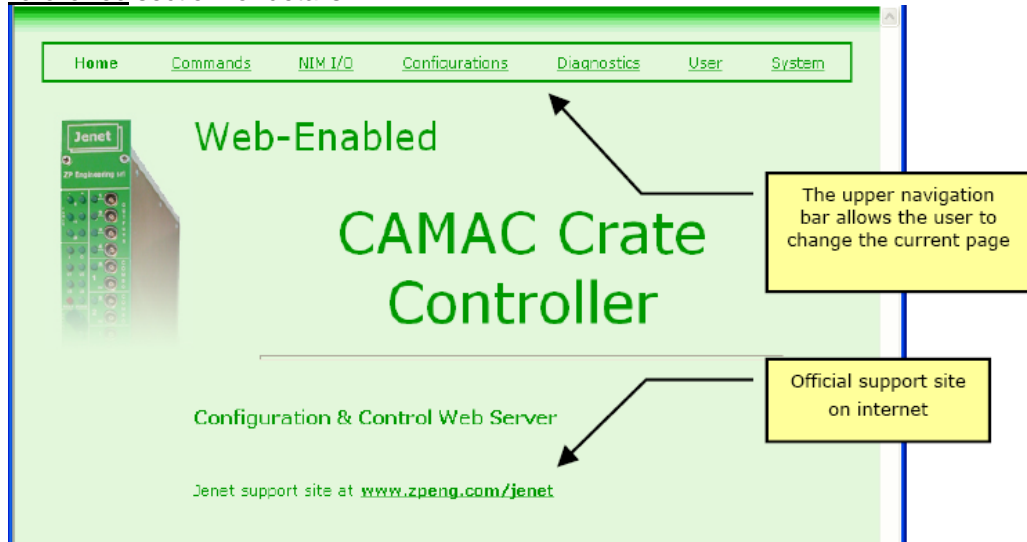
Fig. 5.1: Web server page

Note: in order to trigger update operations in the optimal way, it is best to click on the link available on the navigation bar, instead of hitting the refresh button of the browser (i.e., F5 on Internet Explorer). This is due to the difference in HTTP requests that are sent by various web browsers when refreshing the page.
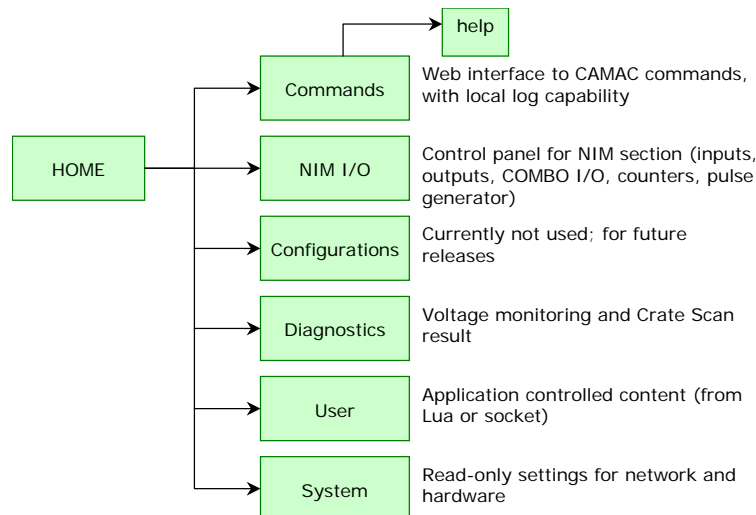
Fig. 5.2: Web server structure

C.A.E.N.

Document type:	Title:	Revision date:	Revision:
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

6.1.5.8. Commands page

The local web server contains a page dedicated to CAMAC commands, in order to allow immediate testing of crate functions. There is a log capability, currently limited to actions performed on the web page. A drop-down selection box allows the choice of one of 7 CAMAC commands; on the entry box the relevant parameters have to be entered; when pressing the EXECUTE button, the commands will be executed. For read functions, the result is available on the data field of the log section; for testing functions, the result is available in the data section, with values 0 or 1.

Please remember that in the current implementation logging is enabled only for commands executed from this web page. Logging is ten events deep; it is also possible to clear the log directly on the web page.

Table 5.1: Commands table

Syntax definition	Description	Notes
CSSA <function> <slot> <subaddr> <data>	execute a CAMAC command with 16-bit data	response in Q function=0..31, slot=1…23 subaddr=0…15
CFSA <function> <slot> <subaddr> <data>	execute a CAMAC command with 24-bit data	response in Q function=0..31, slot=1...23 subaddr=0...15
CCCC	generate dataway initialize	
CCCZ	generate crate clear	
CCCI <value>	set/clear dataway inhibit	value=0 (reset), value=1 (set)
CTCI	test dataway inhibit	response in Q field
CTLM <which>	test LAM	response in Q field which=1…23
LACK	LAM acknowledge	Must be called to clear lam pending interrupts
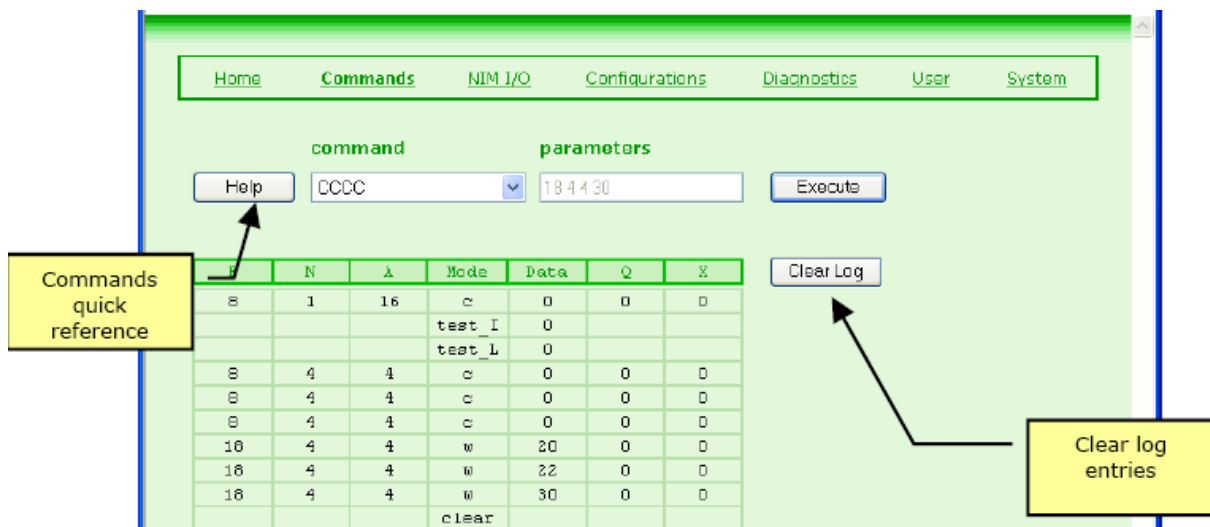
Fig. 5.3: CAMAC commands window

6.2.5.9. NIM I/O page

The NIM I/O page is arranged like a real control panel, in order to allow immediate interaction with the I/O section available on the front panel of the unit.

In addition, it is possible to retrieve default settings by pushing the DEFAULT front panel button (just above the RESET button). Default settings are stored with a specific socket command.

Note that not all interactions are allowed (a fuller control is available from socket or scripting), as browser access is considered unsafe from the remote control point of view.
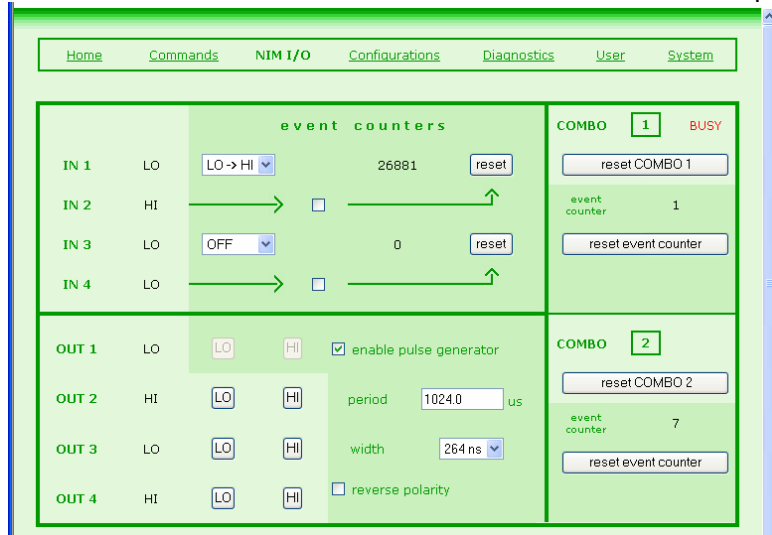
Fig. 5.4: NIM I/O settings

6.2.1.5.9.1. Input section
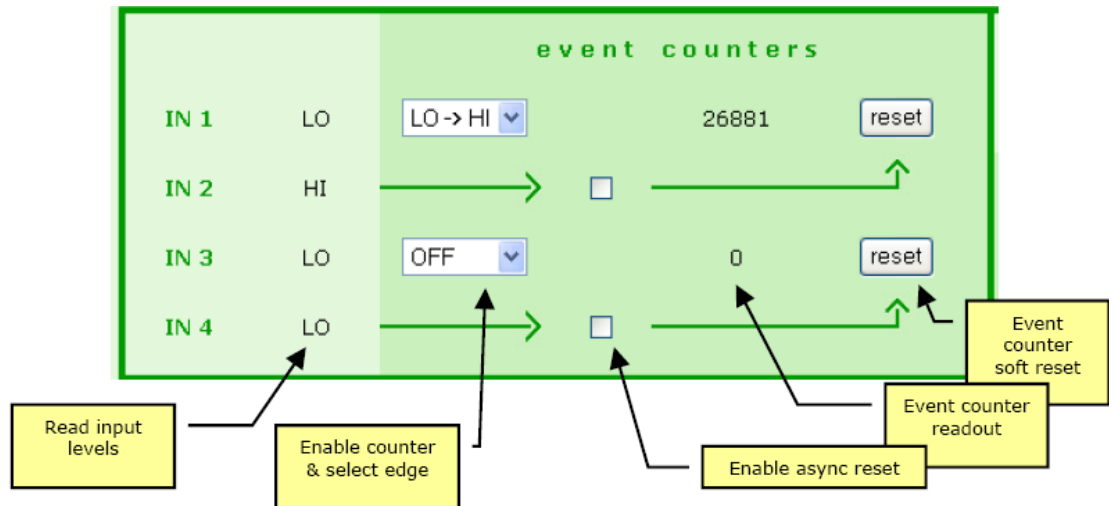
Fig. 5.5: Input section settings

– two <u>event counters</u> can be enabled on inputs 1 and 3 independently, by selecting the required triggering transition (HI-to-LO or LO-to-HI) in the drop-down box

– event counter on input 1 can be asynchronously reset by input 2 (if the checkbox is flagged); there is also a button on the page that allows a software reset of the counter

Document type:
User's Manual (MUT)

Title:
Mod. C111C Ethernet CAMAC Crate Controller

Revision date:
03/12/2007

Revision:
6

– event counter on input 3 can be asynchronously reset by input 4 (if the checkbox is flagged); there is also a button on the page that allows a software reset of the counter

6.2.2.5.9.2. Output section
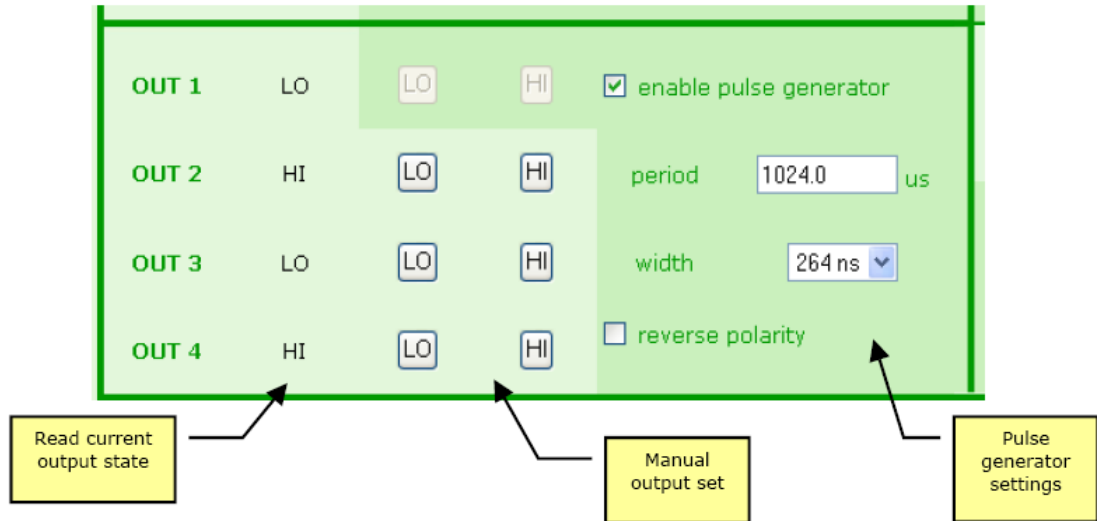
Fig. 5.6: Output section settings

– output 1 can be set as a programmable pulse generator, for which the period, pulse width and polarity can be specified
– numerical entry of pulse generator period is automatically adjusted to fit the available resolution when the page is reloaded

6.2.3.5.9.3. COMBO section
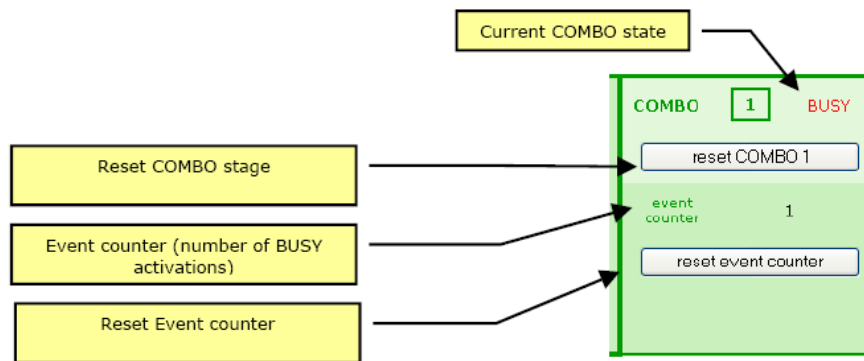
Fig. 5.7: COMBO section settings

– COMBO input 1 can be reset with the dedicated button; the usual way to reset a COMBO input would be from software
– COMBO input 2 can be reset with the dedicated button; the usual way to reset a COMBO input would be from software
– Dead Time Counter for both COMBO inputs is not shown on web page, as it has relevant meaning only immediately before resetting the COMBO input.

NPO:
00108/04:C111C.MUTx/06

Filename:
C111C_REV6.DOC

Number of pages:
41

Page:
21

6.3.5.10. System settings page

Fig. 5.8: System settings

This is a read-only page that displays relevant network settings and local ID details; please always check that your documentation is referring to the same firmware version (release date is NOT relevant).

The MAC address is a read-only property that cannot be changed in any way, it may be useful in certain network environments; the local IP address, subnet mask, default gateway and DHCP enable flag can be modified from the serial port console (see **Serial Port Control** section) or form the control socket (take into account that these settings are effective only after rebooting). If DHCP = 1, then the local DHCP client is enabled. Please refer to your system administrator for additional information relevant to these settings.

Note that wrong or conflicting IP address settings are the most typical issue that creates connection problem when first using the system.

Before connecting a board with static IP (DHCP = 0), you can perform an additional check by issuing a PING command, to verify that the IP address on the unit is really available. If another system is answering at the same address, you may experience intermittent failure (i.e., the web page sometimes does not reload and all other communications will fail).

6.4.5.11. Diagnostics page
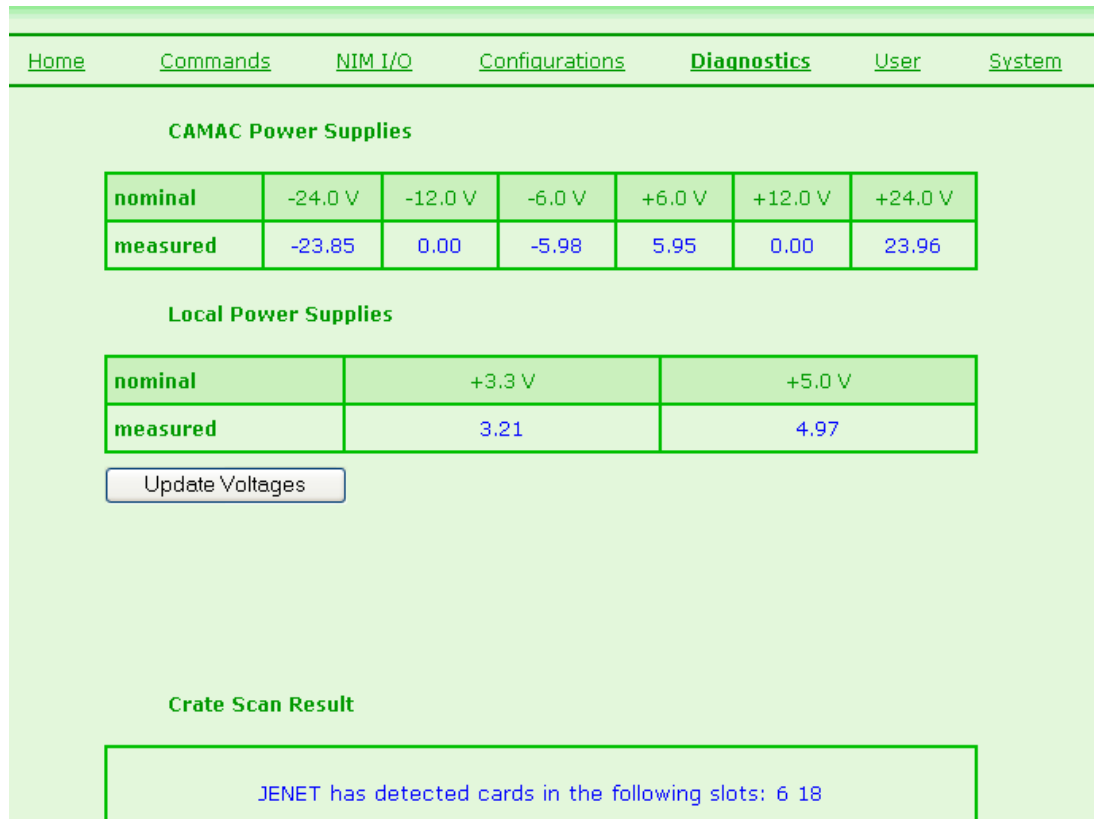
Fig. 5.9: Diagnostics readout

This page shows current voltage readouts, divided into crate power supplies and local (digital-only) power supplies.
Note that C111C requires the following voltages to be present in order to work properly: -24V, -6V, +6V, -24V.

On the bottom of the page, results of the Crate Scan (performed only at power-up if the CSCAN flag is enabled, see **Startup Options** section) are shown in textual form. Remember that if Crate Scan is problematic (depending on which cards are inserted into the crate) it can be disabled either by serial or control socket.

7.6. Local scripting

7.1.6.1. Lua scripting language

An on-board script interpreter is available, allowing complex interactions with the CAMAC bus and local NIM I/O section. The scripting language is Lua, extended with a TCP/IP library and with specific commands that allow management of the underlying hardware.

Lua is a powerful, lightweight programming language designed for extending applications, frequently used as a general-purpose, stand-alone language. More information is available at www.lua.org.

Please take not that, while being free software, it is property of Tecgraf (http://www.tecgraf.puc-rio.br/), of which we acknowledge the excellent work.

A brief description of the scripting engine and its extensions follows.

Table 6.1: Lua description

Lua	Version 4.0	This is the base scripting engine. It has been conceived as an efficient, compact add-on scripting library for various applications.
Luasocket library	Version 1.4	Extension to Lua 4.0 (developed by Diego Nehab) that adds TCP and UDP functionality to the Lua scripting language.
Bit manipulation extension	--	Added by CAEN srl to offer bit-wise AND, OR, XOR functions
NIM I/O extension	--	Added by CAEN srl to offer full control of the local NIM I/O section
System extension	--	Added by CAEN srl to offer full control of system functions
CAMAC extension	--	Added by CAEN srl to offer full control over CAMAC commands and functions.

Note that a complete reference of Lua is available on www.lua.org and on the C111C support site (courtesy of the Lua community) at
http://www.caen.it/nuclear/product.php?mod=C111C

7.2.6.2. Lua engine in C111C

One of the desirable features of Lua is that the language can be easily extended with new commands; this technique has been applied to allow full control of C111C from a script.

A detailed reference of available extension commands is presented in chapter 9 (**Commands Reference**).

A dedicated application that connects to the control socket server allows the user to load the script and control its execution.

The script is launched and executed directly; in case of error, the FAULT red LED on the front panel is lit and script execution is halted. By issuing a **stop** command it is possible to restore the script engine to idle state.

The special function **doevents()** has been added to deal with the single-threaded nature of the scripting engine; it allows the system to terminate execution of the script itself. If this call is missing or called rarely, then it may not be possible to halt execution of the current script.

The ROB (Run-On-Boot) dedicated flag is available on the local EEPROM to indicate whether the FLASH script should be executed at startup. Note that there is no assumption of the temporal evolution coded into the script; therefore, many different uses may be made of the scripting capabilities.

C.A.E.N.

Document type:	Title:	Revision date:	Revision:
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

7.2.1.6.2.1. Bit manipulation extension

As the base Lua library does not provide bit manipulation operands, CAEN added some commands to allow binary operations, an essential feature for any embedded system. As an example, the CLMR Camac function returns the current LAM mask, and a binary AND operator is almost mandatory to allow individual bit testing.
The following additional functions are provided, all with two operands and one result:

Table 6.2: Lua additional functions

function	Description	C equivalent
band(a,b)	32-bit binary AND	a & b
bor	32-bit binary OR	a | b
bxor()	32-bit binary XOR (exclusive-OR)	a ^ b
bmod()	binary module	a % b
bsl	binary shift left	a << b
bsr	binary shift right	a >> b

7.2.2.6.2.2. Socket commands for Lua control

A subset of control socket commands is dedicated to management of the local Lua interpreter engine; these commands are used by the jsm application (C111C Script Manager, see below). Note that these socket commands are the only messages to violate the principle to have one command per line. The following commands are available:

Table 6.3: Lua Socket commands

lua_setfile	It transfers a script from host to C111C; the file is placed in RAM and can be saved on FLASH with the lua_store command. The following procedure is required:
	    • host sends to control socket the following command:
	    lua_setfile <filesize> where <filesize> is expressed in bytes
	    • control sockets answers with "OK"
	    • host sends the file directly
	    • control socket answers with "OK"

lua_getfile	It transfers a script from C111C to host; the following procedure is required:
	    • host sends to control socket the following command:
	    lua_getfile
	    • control sockets answers with <filesize> (expressed in bytes)
	    • host sends "OK"
	    • control socket sends the file directly

lua_store	It saves the current script on FLASH
lua_getrun	It returns the current execution state of the script interpreter
lua_setrun	It changes the executionstate: lua_setrun <value>, where 1 = run, 0 = stop
lua_geterr	It returns the error message (if any) returned by the script interpreter; being a multi-line string, it follows the same protocol of lua_getfile
lua_getlog	It returns the stdout log file returned by the script interpreter; it's useful as a debugging aid as print() messages are sent to the log. Being a multi-line output, it follows the same protocol of lua_getfile. WARNING: using print() on Lua may crash the system if the log file gets too long. Please use it only for debug !!!!

You can review the JSM source code to gain more insight into script file transfers. Please note that JSM is provided in source code form "as is", without support or guarantee.

7.2.3.6.2.3. C111C Script Manager

It is a dedicated application, available for Windows and Linux, that allows full control over all operations related to script management; all operations are specified with command line parameters. The following syntax is implemented (version 1.0):

Table 6.4: Script Manager commands

jsm -h	displays program version and a list of allowed parameters
jsm -ip <IP addr> -u <filename.ext>	uploads script text from specified file on host to C111C
jsm -ip <IP addr> -run	starts execution of current script
jsm -ip <IP addr> -stop	halts execution of current script
jsm -ip <IP addr> -s	stores current script on non volatile memory for Run-on-Boot option
jsm -ip <IP addr> -rob <value>	sets rob (Run-on-Boot) flag to specified value (0 or 1)
jsm -ip <IP addr> -d <filename.ext>	downloads script text from C111C to specified file on host
jsm -ip <IP addr> -d stdout	downloads script text from C111C to stdout on host
jsm -ip <IP addr> -e <filename.ext>	stores error message (if any) from C111C to specified file on host
jsm -ip <IP addr> -e stdout	stores error message (if any) from C111C to stdout on host
jsm -ip <IP addr> -l <filename.ext>	stores log message (if any) from C111C to specified file on host
jsm -ip <IP addr> -l stdout	stores log message (if any) from C111C to stdout on host

The jsm application is available in both source and compiled form on the documentation section of the C111C web site (http://www.caen.it/nuclear/product.php?mod=C111C); it uses dedicated commands to transfer files. Please check periodically for updates.

7.2.4.6.2.4.   Scripting on C111C

A few usage examples follow.

Table 6.5: Script usage examples

Crate initialization	In certain cases, it may be safe to initialize inserted CAMAC target cards as soon as possible after power-up.
	With ROB = 1, the FLASH script performs the required initialization, terminating after completion
	`jn_led(1,1)          -- turn LED U1 on`
	`run_once_init()      -- function somewhere else in the script`
	`jn_led(1,0)          -- turn LED U1 off`

Automatic execution of monitoring loop	After a run-once initialization section, an infinite loop is executed, calling as often as possible the doevents() function:
	`jn_led(1,1)          -- turn LED U1 on`
	`run_once_init()      -- function somewhere else in the script`
	`while (1) do`
	`        doevents()`
	`        run_in_loop() --  function somewhere else in the script`
	`        pause (100)   -- wait 100 msec`
	`end`

COMBO servicing	`jn_led(1,1)          -- turn LED U1 on`
	`run_once_init()      -- function somewhere else in the script`
	`while (1) do`
	`        doevents()`
	`        if (nim_testint(1) == 1) then`
	`            do_something() -- function somewhere else in the script`
	`            nim_cack(1)`
	`        end`
	`end`

8.7. Firmware Upgrade

It is possible to upgrade a section of FLASH memory contents in order to allow firmware upgrades on the field, either to correct any problems that may show up during usage or to load application-specific executables.

Internal FLASH is structured into two banks: one for the operating system, and one for the application; the application bank includes all the executables related to C111C.

Note:

When using a NFS-mapped disk, please remember to add the following line:

`<nfs_directory> <IP address JNT01>(rw,all_squash)`

*to the **/etc/exports** file on the computer where disk is located.*

Firmware upgrade (limited to the application bank) is performed using a telnet connection to the unit, as follows:

- copy the new binary file on a known location on a network disk
- establish a telnet connection on default port 23 (please note that, depending on fw version, it may be possible that the internal telnet server must be enabled by means of HW jumpers); of course, you must know the unit IP address
- type the following commands:

```
cd app (or "cd www" for old firmware versions)
sh flash <host:/nfs_directory> <jffs2 filename>
```

and wait until reprogramming completes

- reboot the unit (either by pressing the RESET button on the front panel or by typing "*reboot*" from the telnet terminal window).

An example of a typical fw upgrade command line is the following:

```
sh flash 192.168.0.91:/home/jenet2/fwupgrade jenet2.img
```

9.8. NIM subsection

Please note that on the present document the following notation is used for NIM signals:

LO   No current flowing into load

HI   Current flowing into load (corresponding to a –0.8 V voltage level across a 50 ohm load)

This notation is maintained also on the NIM I/O web page.

A command parameter equal to 1 means HI.

9.1.8.1. Default button

A DEFAULT pushbutton, located on the front panel above the reset section, allows immediate manual reload of a default configuration for the NIM I/O subsection, thus allowing a quick reconfiguration of the system.

The same settings are applied at power-up, and can be set through the control socket.

The green LED located just below the DEFAULT pushbutton is turned on when default settings are applied, and turned off whenever any of the relevant settings is altered, providing an immediate visual feedback of the validity of default settings.

The DEFAULT pushbutton can also generate a special interrupt request and notify the host computer through a dedicated socket connection on port 2002. This is a very powerful feature that allows the user to start different programs at each pressure of the button: for example, different module setups can be activated when the button is pressed and different actions can be performed.

9.2.8.2. Inputs

The NIM INPUT subsection implements four independent NIM inputs that may be read asynchronously to retrieve input status or configured to perform event counting; more specifically, inputs 1 and 3 can be set as event counter, triggering on rising or falling transition. Counter reset is performed either with a dedicated control command or with an external reset, derived from input 2 (for counter on input 1) and from input 4 (for counter on input 3). Note that external reset capability must be enabled with the proper commands. External reset is active when relevant input is HI; while HI, counter is kept to zero and will not count further.
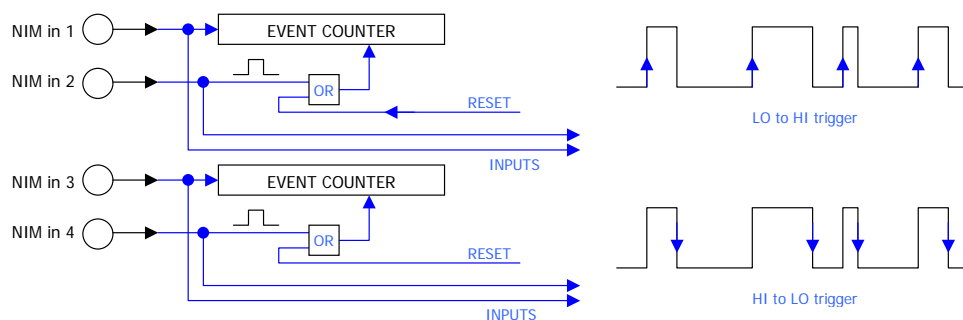
Fig. 8.1: NIM INPUT subsection diagram

Table 8.1: NIM INPUT subsection

Read from inputs		Notes
TCP socket commands	Lua code snippet	Basic reading of input values (j=1,2,3,4)
`nim_getin` `nim_getin j`	`A = nim_getin();` `B = nim_getins(j)`	
Set event counter on IN 3 to falling edge		
TCP socket commands	Lua code snippet	

nim_setievcnt 3 1 1 0	nim_setievcnt(3,1,1,0)	
Enable async reset for counter on IN 1		
TCP socket commands	Lua code snippet	
nim_setievcnt 1 1 1 0	a,b,c=nim_getievcnt(1); nim_setievcnt(1,a,b,1)	

9.3.8.3. Outputs

The OUTPUT subsection implements four independent NIM output that can be independently set and reset; in addition, a programmable pulse generator that can be enabled on the first output.

Fig. 8.2: NIM OUTPUT subsection diagram

PULSE GENERATOR	
Period	186.2 μs to 95325.2 μs (in 93.1 μs steps, 10791 Hz base clock)
Width	88 ns to 616 ns
Output polarity	Direct / Reverse
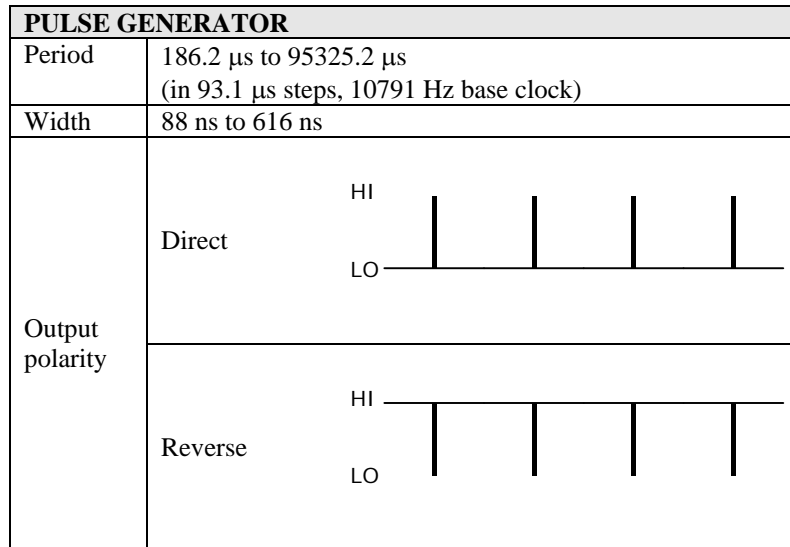
Please note that duty cycle is never higher than 0,05 % (with reversed polarity flag disabled), therefore it may be difficult in certain cases to see the pattern on a scope.

Table 8.2: NIM OUTPUT subsection

Write to outputs		**Notes**
TCP socket commands	Lua code snippet	Basic output control out=1,2,3,4 val=0,1
nim_setout 1 1 1 1	nim_setout(1,1,1,1)	
nim_setouts <out> <val>	nim_setouts(2,0)	
Set pulse generator to 1 ms period, 440 ns width		
TCP socket commands	Lua code snippet	11*93.1 = 1,024 ms 5*88 = 440 ns
nim_setpulse 11 5 0	nim_setpulse(11,5,0)	
Turn pulse generator off and activate outputs 1,3		
TCP socket commands	Lua code snippet	
nim_pulseoff	nim_pulseoff()	
nim_setouts 1 1	nim_setouts(1,1)	
nim_setouts 3 1	nim_setouts(3,1)	

9.4.8.4. COMBO I/O

The COMBO I/O subsection implements a commonly requested Trigger/Busy functional block. The diagram below details one of the two COMBO subsections.
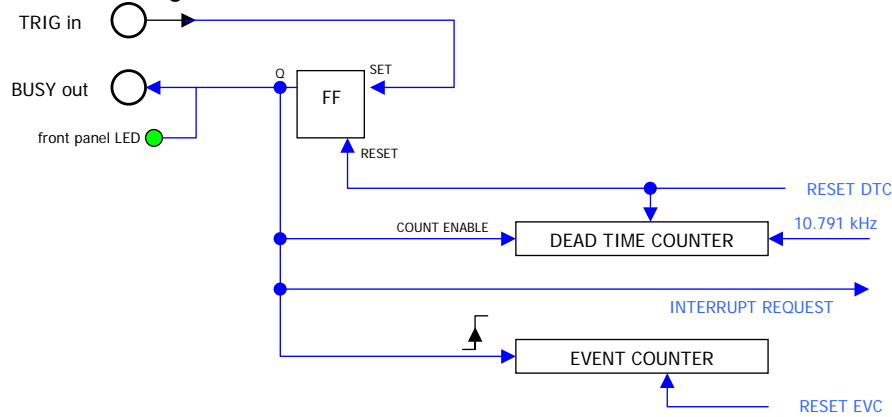
Fig. 8.3: COMBO I/O subsection diagram

Whenever a LO-to-HI transition occurs on the TRIG input, an internal Flip-Flop is set to capture the event; the Flip-Flop output is connected to the BUSY output and its associated LED. The **Dead Time Counter** starts counting at a 10.791 kHz rate, and it is reset when the software application resets the Flip-Flop; therefore, the Dead Time Counter provides an indication of the service time of the TRIG input. It is automatically reset when BUSY is reset (therefore it must be read before the BUSY reset command).

An independent **Event Counter** is incremented at each LO-to-HI transition of the BUSY output, providing an indication of the number of pulses. The Event Counter can be reset by means of an explicit command (through control socket) or from the NIM I/O web page. Please note that, in order to be ready to accept new events, the COMBO section must be explicitly served by the user, in one of the following ways: with a script section that resets the relevant COMBO FF; within the application running on host, with a routine that in response to the COMBO event resets the relevant COMBO FF; by clicking on the RESET button on the NIM I/O web page.

Table 8.3: COMBO I/O subsection

COMBO I/O	
Event Counter	Incremented on BUSY output rising edge
Trigger event	TRIG input rising edge
Dead Time Counter	1 (92.67 usec) to 65535 (6.7 sec)
TRIG-to-BUSY response	35 ns

Table 8.4: COMBO I/O subsection examples

Acknowledge COMBO 1 trigger		Notes
TCP socket commands	Lua code snippet	
`nim_cack 1`	`nim_cack(1)`	
Wait for TRIG on 1 and then turn NIM outs 3,4 ON		
TCP socket commands	Lua code snippet	
---	nim_waitint(1); nim_cack(1) jn_led(3,1); jn_led(4,1)	Blocking mode (wait) not available on socket
nim_testint 1 {repeat then} nim_cack 1 jn_led 3 1 jn_led 4 1	if (nim_waitint(1) == 1) then nim_cack(1); jn_led(3,1); jn_led(4,1) end	Test mode

C.A.E.N.

Document type:	Title:	Revision date:	Revision:
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

10.9.   ASCII Commands reference

A reference of available commands on both the TCP control socket server and the local Lua scripting language is detailed in this section.

Note that whenever possible, TCP control commands are as short as possible in order to minimize Ethernet traffic; on the other hand, Lua commands are more descriptive.

Some general rules follow:

– Control socket commands ALWAYS return an error code as first parameter; it is equal to 0 if the commands executed with success, -1 if command parameters are wrong, -2 if the command is non existent; in the table below this error code is NOT indicated when describing return values, as it must be always 0 to yield a correct answer.

– TCP control sockets are not case sensitive, while Lua commands are <u>case sensitive</u> (this is a specific property of the language).

– As much as possible, all commands have the same name for Lua and for TCP control socket.

Table 9.1: TCP control socket / Lua Commands reference

TCP control socket	Lua	Description
CAMAC bus control		
CCCZ	CCCZ()	Generate Dataway Init
CCCC	CCCC()	Generate Crate Clear
CFSA <function> <slot> <addr> <data>	CFSA(function,slot,addr,data)	24-bit CAMAC command; returns Q and data
CSSA <function> <slot> <addr> <data>	CSSA(function,slot,addr,data)	16-bit CAMAC command; returns Q and data
CCCI <value>	CCCI(value)	Change Dataway Inhibit to specified value (0 or 1)
CTCI	CTCI()	CAMAC test Inhibit; returns 0 or 1
CTLM <slot>	CTLM(slot)	CAMAC test LAM on specified slot = 1…..23
CCLWT <slot>	CCLWT(slot)	CAMAC wait for LAM on specified slot (only for Lua)
CLMR	CLMR()	Returns current LAM register, in hex
LACK	LACK	LAM acknowledge
CTSTAT	CTSTAT()	Returns Q and X values (from last access on bus)
CSCAN		Executes a Crate scan and returns a bit mask with bitn = 1 if in the slot n a CAMAC board is detected
DIAGNOSTIC SECTION		
vn24	vn24()	Returns the measured voltage on –24V power supply, in float format
vn12	vn12()	Returns the measured voltage on –12V power supply, in float format
vn6	vn6()	Returns the measured voltage on –6V power supply, in float format
vp6	vp6()	Returns the measured voltage on +6V power supply, in float format

C.A.E.N.

Document type:	Title:		Revision date:	Revision:
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller		03/12/2007	6

`vp12`	`vp12()`	Returns the measured voltage on +12V power supply, in float format
`vp24`	`vp24()`	Returns the measured voltage on +24V power supply, in float format
`vp33`	`vp33()`	Returns the measured voltage on +3.3V power supply, in float format
`vp5`	`vp5()`	Returns the measured voltage on +5V power supply, in float format
SYSTEM PARAMETERS (stored in EEPROM)		
`ee_getcomspeed`	`---`	Returns the speed of the RS232 COM port
`ee_setcomspeed <baudrate>`	`---`	Sets the speed of the RS232 COM port
`ee_getcscan`	`---`	Returns current Crate Scan flag value (0 or 1)
`ee_getdhcp`	`ee_getdhcp()`	Returns 0 if DHCP client is not enabled, 1 otherwise
`ee_getdns`	`ee_getdns()`	Returns current DNS, in dotted notation
`ee_getgw`	`ee_getgw()`	Returns current Gateway, in dotted notation
`ee_getip`	`ee_getip()`	Returns current IP address, in dotted notation
`ee_getmac`	`ee_getmac()`	Returns current MAC address, in hex format with '-' delimiters, like in "00-50-C2-26-E0-0A"
`ee_getmask`	`ee_getmask()`	Returns current IP mask, in dotted notation
`ee_getname`	`---`	Returns current Crate Name as it appears in the local web pages
`ee_getrob`	`---`	Returns current Run-on-Boot flag value (0 or 1)
`ee_getserial`	`---`	Returns C111C serial number
`ee_setcscan`	`---`	Sets Crate Scan flag to specified value (0 or 1)
`ee_setname <name string>`	`---`	Sets the Crate Name shown in the local web pages
`ee_setrob <value>`	`---`	Sets Run-on-Boot flag to specified value (0 or 1)
`ee_storeconf`	`---`	Stores current NIM section configuration into EEPROM default settings
NIM I/O SECTION		
`nim_getin`	`nim_getin()`	Returns NIM input values; it returns "a(1) a(2) a(3) a(4)", where a(n) = 0 or 1
`nim_getins <which>`	`nim_getins(which)`	Returns a single NIM input value (0 or 1); <which> = 1, 2, 3 or 4
`nim_setievcnt <which> <en> <pol> <ext_reset>`	`nim_setievcnt(which,en ,pol,ext_reset)`	Sets Input Event Counter; <which>=1,3; <en>=0,1; <pol>=0,1; <ext_reset >=0,1
`nim_getievcnt <which>`	`nim_getievcnt(which)`	Returns Input Event Counter settings
`nim_geticnt <which>`	`nim_geticnt(which)`	Returns current Input Event Counter value
`nim_reseticnt <which>`	`nim_reseticnt(which)`	Reset specified Input Event Counter: <which> = 1 or 3

C.A.E.N.

Document type:	Title:	Revision date:	Revision:
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

`nim_getouts <which>`	`nim_getouts(which)`	Returns current value of specified output; <which> = 1…4
`nim_getout`	`nim_getout()`	Returns current value of all outputs
`nim_setouts <which> <value>`	`nim_setouts(which,value)`	Sets value of specified output; <which> = 1…4
`nim_setout <v1> <v2> <v3> <v4>`	`nim_setout(v1,v2,v3,v4)`	Sets current value of all output; <which> = 1…4
`nim_setoutp <which> <pulse_width>`	`---`	Generates a pulse event on a selected output with specified width in multiples of 20 ms. See section **Outputs** for details
`nim_setpulse <period> <width> <polarity>`	`nim_setpulse(period,width,polarity)`	Sets Pulse Generator, values in decimal; <period>=1…1023; <width>=1…7; <polarity>= 0 or 1. See section **Outputs** for details
`nim_getpulse`	`nim_getpulse()`	Returns Pulse Generator settings
`nim_pulseoff`	`nim_pulseoff()`	Disables Pulse Generator
`nim_setcthr <which> <value>`	`nim_setcthr(which, value)`	Sets threshold value for COMBO section; <which> = 1 or 2
`nim_getcthr <which>`	`nim_getcthr(which)`	Returns current threshold value for COMBO section; <which> = 1 or 2
`nim_getcdtc <which>`	`nim_getcdtc(which)`	Returns current COMBO Dead Time Counter value; <which> = 1, 2
`nim_cack <which>`	`nim_cack(which)`	Acknowledges COMBO event,; <which> = 1 or 2
`nim_getcev <which>`	`nim_getcev(which)`	Returns current COMBO Event Counter value; <which> = 1, 2
`nim_resetcev <which>`	`nim_resetcev(which)`	Reset specified COMBO Event Counter: <which> = 1 or 2
`nim_enablecombo <which> <value>`	`nim_enablecombo(which, value)`	Enables or disables specified COMBO section, preventing unwanted triggering; <which> = 1, 2; <value>= 0,1
`---`	`nim_waitcombo(ch)`	Wait for COMBO interrupt; it returns only when COMBO is busy (blocking call) ; ch = 1, 2 or 3 (= both)
`---`	`nim_waitdtc(ch)`	Wait for DTC interrupt; it returns only when DTC has crossed the threshold (blocking call) ; ch = 1, 2 or 3 (= both)
`nim_testcombo <ch>`	`nim_testcombo (ch)`	Test COMBO interrupt; it immediately returns current values (1 if COMBO busy, 0 otherwise); ch = 1, 2 or 3 (= both)
`nim_testdtc <ch>`	`nim_testdtc(ch)`	Test DTC interrupt; it immediately returns current value (1 if DTC has crossed the threshold, 0 otherwise) ; ch = 1, 2 or 3 (= both)
Control socket only commands		
`lua_setfile <data...........>`	`--`	Lua-related command; enables direct transfer of script file
`lua_store`	`--`	Lua-related command; stores transferred filew into non-volatile memory
`lua_getfile`	`--`	Lua-related command; enables direct retrieval of script file
`lua_getrun`	`--`	Lua-related command; returns 0 if script is stopped, 1 if running

C.A.E.N.

Document type:	**Title:**	**Revision date:**	**Revision:**
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

lua_setrun <value>	--	Lua-related command; <value>=1 runs the script; <value>=0 stops the script
lua_geterr	--	Lua-related command; returns last Lua error message
lua_getlog	--	Lua-related command; returns log file (in place of stdout)
SYSTEM IDENTFICATION and VARIOUS		
jn_fpgaver	jn_fpgaver()	Returns a string containing FPGA details (read-only information from FPGA registers)
jn_fwver	jn_fwver()	Returns a string containing the current firmware version running on the system
jn_led <which> <value>	jn_led(which,value)	Set front panels LEDs (U1, U2, U3, U4); <which> = 1 to 4, <value> = 0 or 1
reset	----	Perform a board reset
user_add <username>:<password>	----	Add a new user to the local web server
user_del <username>:<password>	----	Delete a user from the local web server
user_list	----	List all users of the local web server
ver	----	Returns a string containing firmware and FPGA versions
web_getuser	web_getuser()	Returns text appearing in User page (see web server section)
web_setuser <text>	web_setuser(text)	Sets text that appears in User page (see web server section)
Lua only commands		
--	doevents()	Enable system control of Lua loop
--	pause(ms)	Wait for specified interval (in ms)
	band(a,b) bor(a,b) bxor(a,b) mod(a,b) bsl(a,b) bsr(a,b)	Binary operators: AND: $a\ \&\ b$ OR: $a\ |\ b$ XOR: $a\ \wedge\ b$ MOD: $a\ \%\ b$ BSL: $a \ll b$ BSR: $a \gg b$

12.10. Block transfer reference

A brief summary of available commands follows:

Table 10.1: Block transfer commands

Utility	BLKBUFFS	Block transfer buffer size set
	BLKBUFFG	Block transfer buffer size get
Q-stop	BLKSS	Block transfer, 16-bit, Q-stop mode
	BLKFS	Block transfer, 24-bit, Q-stop mode
Q-repeat	BLKSR	Block transfer, 16-bit, Q-stop mode
	BLKFR	Block transfer, 24-bit, Q-repeat mode
Address Scan	BLKSA	Block transfer, 16-bit, address scan mode
	BLKFA	Block transfer, 24-bit, address scan mode

In general the command is expressed as **BLKsm**
where s = S (short), F(full)
       m = S (Q-stop), R (Q-repeat), A (address scan)
Read or write mode is determined by the Function code passed as a parameter, as follows:
F = 0,......,7 ➜ READ mode
F = 16,...,27 ➜ WRITE mode

All block transfer commands have the same behavior. C111C replies to the command itself immediately after reception, before executing the actual block transfer, with one of the following possible replies (compliant with the standard command response of the TCP control socket protocol):

Table 10.2: Block transfer replies

Reply	
0	OK, operation in progress
-1	error, wrong parameters
-2	error, non existing command

The general format of a data block is

cmd data1 data2 ….. dataK

where:

- **K** is the current buffer size
- in ASCII mode, cmd is formatted as %03X
- in ASCII mode, dataj is formatted as %06X (for both 16-bit and 24-bit access types)
- in ASCII mode, the data block is terminated by a "\r" character
- in binary mode, cmd and dataj are all 32-bit values
- in binary mode, the data block is (K+1)*4 bytes
- if there are non significant data values (if cmd < K, or cmd = 0), data block size is always the same as above

hdr can assume one of the following values:

Table 10.3: HDR possible values

hdr		notes
0	End of block transfer	data1= actual datasize data2,…dataK = non significant
N > 0	Number of data words being transferred	If N<K, then dataN,dataN+1,…..,dataK are non significant
-3	Timeout error	data1= actual datasize data2,…dataK = non significant
-4	Abort error	data1= actual datasize data2,…dataK = non significant

12.1.10.1. Block transfer abort

Any block read operation can be aborted by sending an arbitrary character to C111C; C111C will answer with hdr = -4 followed by actual_datasize (the number of datawords effectively transferred). Any block write operation can be aborted by sending a data block with hdr = -4, taking care to maintain data block formatting and size. C111C will answer (always in ASCII, as it has returned to ordinary command mode) with hdr=-4 followed by actual_datasize (the number of datawords effectively transferred).

Note: please take care to avoid multiple clients connected to the control socket server, as any command sent by other clients will abort the data transfer currently ongoing.

Table 10.4: Block transfer commands

Buffer size get/set (only for read operations) Default buffer size is 16 Buffer size is expressed in terms of data values (not in bytes)		
command	Reply (by C111C)	Notes
`BLKBUFFG`	`0 <buffsize>`	Get current buffer size
`BLKBUFFS <buffsize>`	`0`	Set buffer size; valid range is 1…256

STOP mode reply = see table on page 2 K = Block Transfer buffer size	
ASCII read	Notes

`HOST> BLKFS <F> <N> <A> <maxsize>`
`JENET> reply`
`JENET> hdr data1 data2 data3 ..... dataK`
`JENET> hdr data1 data2 data3 ..... dataK`
`.....`
`JENET> 000 data1 data2 data3 ..... dataK`
	24-bit operation
	F = 0…7

`HOST> BLKSS <F> <N> <A> <maxsize>` `------- same as above -------`	16-bit operation F = 0…7
ASCII write	Notes

`HOST> BLKFS <F> <N> <A> <maxsize>`
`JENET> reply`
`HOST> hdr data1 data2 data3 ..... dataK`
`HOST> hdr data1 data2 data3 ..... dataK`
`      .....`
`HOST> hdr data1 data2 data3 ..... dataK`
`JENET> 0 <actual_datasize>`
	24-bit operation
	F = 16…27

`HOST> BLKSS <F> <N> <A> <maxsize>` `------- same as above -------`	16-bit operation F = 16…27
BINARY read	Notes

`HOST> BLKFS <F> <N> <A> <maxsize> bin`
`JENET> reply`
`JENET> <binary data>`
`JENET> <binary data>`
`.....`
`JENET> <binary data>`
	24-bit operation
	F = 0…7

`HOST> BLKSS <F> <N> <A> <maxsize> bin` `------- same as above -------`	16-bit operation F = 0…7

REPEAT mode reply = see table on page 2 K = Block Transfer buffer size <timeout> in seconds, range = 0,….,32767	
ASCII read	Notes

`HOST> BLKFR <F> <N> <A> <maxsize> <timeout>`
`JENET> reply`
`JENET> hdr data1 data2 data3 ..... dataK`
`JENET> hdr data1 data2 data3 ..... dataK`
`.....`
`JENET> 000 data1 data2 data3 ..... dataK`
	24-bit operation
	F = 0…7
	
	Check for <u>timeout</u> possible reply from Jenet (hdr = -3)

`HOST> BLKSR <F> <N> <A> <maxsize> <timeout>` `------- same as above -------`	16-bit operation F = 0…7

ASCII write	Notes

`HOST> BLKFR <F> <N> <A> <maxsize> <timeout>`
`JENET> reply`
` HOST> hdr data1 data2 data3 ..... dataK`
` HOST> hdr data1 data2 data3 ..... dataK`
`      .....`
` HOST> hdr data1 data2 data3 ..... dataK`
`JENET> 0 <actual_datasize>`
	24-bit operation
	F = 16…27
	
	Check for <u>timeout</u> possible reply from C111C (hdr = -3)

`HOST> BLKSR <F> <N> <A> <maxsize> <timeout>` `------- same as above -------`	16-bit operation F = 16…27

BINARY read	Notes

` HOST> BLKFR <F> <N> <A> <maxsize> <timeout> bin`
`JENET> reply`
`JENET> <binary data>`
`JENET> <binary data>`
`.....`
`JENET> <binary data>`
	24-bit operation
	F = 0…7
	
	Check for <u>timeout</u> possible reply from C111C (hdr = -3)

`HOST> BLKSR <F> <N> <A> <maxsize> <timeout> bin` `------- same as above -------`	16-bit operation F = 0…7

ADDRESS SCAN mode
`    reply   = see table on page 2`
 	
`K       = Block Transfer buffer size`
`Nstart  = station from which address scan begins`
`Nwords = maximum size of data block to transfer`	

ASCII read	Notes

` HOST> BLKFA <F> <Nstart> <Nwords>`
`JENET> reply`
`JENET> hdr data1 data2 data3 ..... dataK`
`JENET> hdr data1 data2 data3 ..... dataK`
`.....`
`JENET> 000 data1 data2 data3 ..... dataK`
	24-bit operation
	F = 0…7

` HOST> BLKSA <F> <Nstart> <Nwords>` `------- same as above -------`	16-bit operation F = 0…7

ASCII write	Notes

` HOST> BLKFA <F> <Nstart> <Nwords>`
`JENET> reply`
` HOST> hdr data1 data2 data3 ..... dataK`
` HOST> hdr data1 data2 data3 ..... dataK`
`      .....`
` HOST> hdr data1 data2 data3 ..... dataK`
`JENET> 0 <actual_datasize>`
	24-bit operation
	F = 16…27

` HOST> BLKSA <F> <Nstart> <Nwords>` `------- same as above -------`	16-bit operation F = 16…27

BINARY read	Notes

` HOST> BLKFA <F> <Nstart> <Nwords> bin`
`JENET> reply`
`JENET> <binary data>`
`JENET> <binary data>`
`.....`
`JENET> <binary data>`
	24-bit operation
	F = 0…7

` HOST> BLKSA <F> <Nstart> <Nwords> bin` `------- same as above -------`	16-bit operation F = 0…7

C.A.E.N.

Document type:	Title:	Revision date:	Revision:
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

13.11. Binary commands reference

In general, the binary command has the following format:
byte(0) = STX;
byte(1) = CMD_CODE;
byte(2) = databyte(0)
byte(3) = databyte(1)
....
byte(n) = databyte(k)
byte(n+1) = REQ_RESPONSE;
byte(n+2) = ETX;
where:
STX is the hexdecimal value 0x02
ETX is the hexdecimal value 0x04
CMD_CODE may be one of the followings value:

BIN_CFSA_CMD = 0x20 (equivalent to the ascii command cfsa)

BIN_CSSA_CMD = 0x21 (equivalent to the ascii command cssa)

BIN_CCCZ_CMD = 0x22 (equivalent to the ascii command cccc)

BIN_CCCC_CMD = 0x23 (equivalent to the ascii command ccci)

BIN_CCCI_CMD = 0x24 (equivalent to the ascii command ctci)

BIN_CTCI_CMD = 0x25 (equivalent to the ascii command ctci)

BIN_CTLM_CMD = 0x26 (equivalent to the ascii command ctlm)

BIN_CCLWT_CMD = 0x27 (equivalent to the ascii command cclwt)

BIN_LACK_CMD = 0x28 (equivalent to the ascii command lack)

BIN_CTSTAT_CMD = 0x29 (equivalent to the ascii command ctstat)

BIN_CLMR_CMD = 0x2A (equivalent to the ascii command clmr)

BIN_CSCAN_CMD = 0x2B (equivalent to the ascii command cscan)

BIN_NIM_SETOUTS_CMD = 0x30 (equivalent to the ascii command nim_setouts)

databyte(0) ..databyte(k) is of variable length according to the command code
REQ_RESPONSE may be:

NO_BIN_RESPONSE = 0xa0 (no response requested)

Any other value (response requested)

If one of the databyte(0)..databyte(k) contains 0x2, 0x4 e 0x10, then the databyte must be converted in two bytes accordingly to the following rule (a simple escape sequence to avoid out-of-sync transmissions):

if databyte(n) = 0x2 => converted in => databyte(n) = 0x10; databyte(n+1) = 0x80 + 0x02;

if databyte(n) = 0x4 => converted in => databyte(n) = 0x10; databyte(n+1) = 0x80 + 0x04;

if databyte(n) = 0x10 => converted in => databyte(n) = 0x10; databyte(n+1) = 0x80 + 0x10;

Table 11.1: Binary commands

Command			
CFSA		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CFSA_CMD	resp[1]	BIN_CFSA_CMD
bin_cmd[2]	F	resp[2]	Q
bin_cmd[3]	N	resp[3]	X
bin_cmd[4]	A	resp[4]	(DATA & 0xFF)
bin_cmd[5]	(DATA & 0xFF)	resp[5]	((DATA >> 8) & 0xFF)
bin_cmd[6]	((DATA >> 8) & 0xFF)	resp[6]	((DATA >> 16) & 0xFF)
bin_cmd[7]	((DATA >> 16) & 0xFF)	resp[7 + delta]	ETX
bin_cmd[8 + delta]	RESPONSE		
bin_cmd[9 + delta]	ETX		

Command			
CSSA		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CSSA_CMD	resp[1]	BIN_CSSA_CMD
bin_cmd[2]	F	resp[2]	Q
bin_cmd[3]	N	resp[3]	X
bin_cmd[4]	A	resp[4]	(DATA & 0xFF)
bin_cmd[5]	(DATA & 0xFF)	resp[5]	((DATA >> 8) & 0xFF)
bin_cmd[6]	((DATA >> 8) & 0xFF)	resp[6 + delta]	ETX
bin_cmd[7 + delta]	RESPONSE		
bin_cmd[8 + delta]	ETX		

Command			
CCCZ		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CCCZ_CMD	resp[1]	BIN_CCCZ_CMD
bin_cmd[2]	RESPONSE	resp[2]	ETX
bin_cmd[3]	ETX		

Command			
CCCC		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CCCC_CMD	resp[1]	BIN_CCCC_CMD
bin_cmd[2]	RESPONSE	resp[2]	ETX
bin_cmd[3]	ETX		

Command			
CCCI		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CCCI_CMD	resp[1]	BIN_CCCI_CMD
bin_cmd[2]	DATA_IN	resp[2]	ETX
bin_cmd[3]	RESPONSE		
bin_cmd[4]	ETX		

Command			
CTCI		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CTCI_CMD	resp[1]	BIN_CTCI_CMD
bin_cmd[2]	ETX	resp[2]	test_res
		resp[3]	ETX

Command			
CTLM		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CTLM_CMD	resp[1]	BIN_CTLM_CMD
bin_cmd[2]	slot	resp[2]	test_res
bin_cmd[3 + delta]	ETX	resp[3]	ETX

Command			
CCLWT		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CCLWT_CMD	resp[1]	BIN_CCLWT_CMD
bin_cmd[2]	slot	resp[2]	ETX
bin_cmd[3 + delta]	ETX		

C.A.E.N.

Document type:	Title:	Revision date:	Revision:
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

Command			
CTSTAT		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CTSTAT_CMD	resp[1]	BIN_CTSTAT_CMD
bin_cmd[2]	ETX	resp[2]	Q
		resp[3]	X
		resp[4]	ETX

Command			
CLMR		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CLMR_CMD	resp[1]	BIN_CLMR_CMD
bin_cmd[2]	ETX	resp[2]	(reg & 0xFF)
		resp[3]	((reg >> 8) & 0xFF)
		resp[4]	((reg >> 16) & 0xFF)
		resp[5]	((reg >> 24) & 0xFF)
		resp[6+delta]	ETX

Command			
CSCAN		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_CSCAN_CMD	resp[1]	BIN_CSCAN_CMD
bin_cmd[2]	ETX	resp[2]	(bitmask & 0xFF)
		resp[3]	((bitmask >> 8) & 0xFF)
		resp[4]	((bitmask >> 16) & 0xFF)
		resp[5]	((bitmask >> 24) & 0xFF)
		resp[6+delta]	ETX

Command			
LACK		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_LACK_CMD	resp[1]	BIN_LACK_CMD
bin_cmd[2]	RESPONSE	resp[2]	ETX
bin_cmd[3]	ETX		

Command			
NIM_SETOUTS		Response	
bin_cmd[0]	STX	resp[0]	STX
bin_cmd[1]	BIN_NIM_SETOUTS_CMD	resp[1]	BIN_NIM_SETOUTS_CMD
bin_cmd[2]	nimo	resp[2]	ETX
bin_cmd[3]	value		
bin_cmd[4 + delta]	RESPONSE		
bin_cmd[5 + delta]	ETX		

The server response to a non-existing command is the following:
  resp[0] = STX;
  resp[1] = CMD_ERROR = 0xCE;
  resp[2] = ETX;
If the command has the wring number of parameters, or the wrong length, the server response is the following:
  resp[0] = STX;
  resp[1] = PAR_ERROR = 0xCF;
  resp[2] = ETX;

C.A.E.N.

Document type:	Title:	Revision date:	Revision:
User's Manual (MUT)	Mod. C111C Ethernet CAMAC Crate Controller	03/12/2007	6

14.12. Board Specifications

Table 12.1: Mod. C111C Specifications

Power Supply	
Required voltages on crate	+24 V (100 mA) - 24 V (100 mA) - 6 V (100 mA) + 6 V (700 mA)
Ethernet interface	
Settings	10/100 Mbit autonegotiating
Default configuration	**IP address**: 192.168.0.98 **subnet mask**: 255.255.255.0 **default gateway**: 0.0.0.0 **DHCP client**: 0 (disabled) (default configuration can be changed through serial port)
Serial port	
Settings	38400 baud, 8-N-1, no flow control
Function	for configuration changes: - system settings - startup options
CAMAC bus interface	
Compliance	to ANSI-IEEE std. 583-1982
NIM interface	
Connectors	SUHNER type
CPU section	
Processor	ARM7TDMI running at 44 MHz
Operating System	uClinux ver. 2.4.17
FLASH	4 Mbytes
SDRAM	16 Mbytes