

Introduction

The C111C CAMAC Crate Controller can be controlled by a remote host by means of socket connections. Two different ports at the same TCP/IP address are used for exchanging commands and data between C111C and the host computer. Port 2000 is dedicated to *ASCII commands*, while port 2001 is dedicated to *Binary commands*.

All commands are available in ASCII form. An *ASCII command* is a text string that the host computer sends to C111C on port 2000. A complete reference of all ASCII commands can be found in the User's Manual, section "ASCII command reference". ASCII commands are easy to handle because they are simple text strings, but they can be quite slow for some applications. For this reason a command subset has been introduced, the *Binary commands* subset.

NOT all commands are available in binary form, but only those CAMAC commands for which speed can be an issue. Binary commands are faster, but more complex, because they use a special protocol, described in sections "TCP binary control socket" and "Binary commands reference" of the User's manual.

A control program can be written in any language, regardless of the platform and operating system running on the host computer. What is needed is just a piece of code to send and receive data on a socket connection. If binary commands are used, the programmer must also write some code to implement the binary protocol.

The purpose of the Library is to simplify and speed up code development for C/C++ programmers, hiding all details regarding socket connections and the binary protocol. No knowledge of socket programming and of the binary protocol is required and the user can avoid reading the manual's chapters about binary protocol.

The C Library always uses binary commands. Three special commands (CMDS, CMDR, CMDSR) are available to send and receive commands and data in ASCII format.

Library files

The C Library is distributed in source code form (no precompiled binary).

The library consists in two files:

- 1) crate_lib.c
- 2) crate_lib.h

Compiling and Makefile

Using Linux platforms

The library files must be added to your project for use the library functions.

Example 1 (Application and library files are in the same folder)

1) Assuming that your source code application is in /home/jenet/jenet_appli.c
2) Assuming that the library source code application is in /home/jenet/jenet_appli.c
>From the shell prompt type:
> gcc jenet_appli.c crate_lib.c -o jenet_appli
This will build a jenet_appli application

If your application's source code is in a different folder you have to customize the gcc command to reflect your settings.

Example 2 (Application and library files are in different folders)

1) Assuming that your source code application is in /home/jenet/jenet_appli.c
2) Assuming that the library source is in /home/jenet/library/crate_lib.c
3) Assuming that the working directory of the shell is /home/jenet
>From the shell prompt type:
/home/jenet> gcc jenet_appli.c /home/jenet/library/crate_lib.c -I/home/jenet/library/ -o jenet_appli

You can use the make tools and customize a Makefiles to build your project.

Example makefile:

```
##*-makefile-*-
binaries = jenet_appli

all: compile

compile: $(binaries)

CC = gcc
LINK = gcc

JENET_APPLI_SRC = crate_lib.c jenet_appli.c
JENET_APPLI_OBJ = $(JENET_APPLI_SRC:.c=.o)

jenet_appli: $(JENET_APPLI_OBJ) $(addsuffix .o, $(common))
$(LINK) -o $@ $^ -L. -lpthread

clean:
rm -f core *.o $(binaries) $(addsuffix .gdb, $(binaries))
```

Using Windows platforms

In the default software distribution there is a Workspace file (**crate_lib_win32.dsw**) created with Microsoft Visual Studio 6.0. The workspace includes some examples showing the correct use of the C Library.

Defines

```
////////////////////////////////////
//      Return Values
////////////////////////////////////

#define CRATE_OK                0
#define CRATE_ERROR            -1
#define CRATE_CONNECT_ERROR    -2
#define CRATE_IRQ_ERROR        -3
#define CRATE_BIN_ERROR        -4
#define CRATE_CMD_ERROR        -5
#define CRATE_ID_ERROR         -6
#define CRATE_MEMORY_ERROR     -7

////////////////////////////////////
//      BLK_TRANSF Opcode Defines
////////////////////////////////////

#define OP_BLKSS                0x0
#define OP_BLKFS                0x1
#define OP_BLKSR                0x2
#define OP_BLKFR                0x3
#define OP_BLKSA                0x4
#define OP_BLKFA                0x5

////////////////////////////////////
//      IRQ Type Defines
////////////////////////////////////

#define LAM_INT                 0x1
#define COMBO_INT              0x2
#define DEFAULT_INT            0x3

////////////////////////////////////
//      Miscellaneous
////////////////////////////////////

#define NO_BIN_RESPONSE        0xA0
```


Results:

CRATE_OK: Operation completed successfully; in this case the blk_info.totsize will be filled with the actual data size effectively transferred by the CAMAC Controller

CRATE_ERROR: Operation failed

Example:

```
int main(int argc, char *argv[])
{
    short crate_id;
    int i, j;
    int resp;
    // Block transfer operation sends 16-bit data separated in lines
    // This parameter sets the amount of data per line
    int block_data_size;

    int total_data_size; // This is the total 16-bit data to be sent

    char blk_ascii_buf[2048];
    unsigned int blk_transf_buf[300];

    BLK_TRANSF_INFO blk_info;
    CRATE_OP op;

    printf("Block Transfer Test\n");

/*
=====
Open connection with a Camac controller
=====
*/
    printf("Initializing communication parameters...\n");

    crate_id = CROPEN("192.168.0.98");
    if (crate_id < 0) {
        printf("Error %d opening connection with CAMAC Controller \n", crate_id);
        return 0;
    }

/*
=====
Invoke Block transfer Q-stop mode
(write operation on Crate Module N address 0)
=====
*/
    block_data_size = 16;
    total_data_size = (block_data_size * 5); //Sent 80 16-bit data items

    printf("Start block transfer write\n");

    // Prepare send test pattern
    for (j = 0; j < 5; j++) {
        for (i = 0; i < block_data_size; i++) {
            if (i & 1)
                blk_transf_buf[i + (j * block_data_size)] = 0x5555;
            else
                blk_transf_buf[i + (j * block_data_size)] = 0xAAAA;
        }
    }

    blk_info.opcode = OP_BLKSS;
    blk_info.F = 16;
    blk_info.N = 6;
    blk_info.A = 0;
    blk_info.blksize = block_data_size;
    blk_info.totsize = total_data_size;
}
```

```

blk_info.timeout = 0;

resp = BLKTRANSF(crate_id, &blk_info, blk_transf_buf);

if (resp != CRATE_OK) {
    printf("ERROR: Negative response from socket server\n");
    return 0;
}

printf("Total data written: %d\n", blk_info.totsize);

printf("End of block transfer write\n");

/*
=====
Invoke Block transfer Q-stop mode
(read operation on Crate Module N address 0)
=====
*/

block_data_size = 32;
total_data_size = (block_data_size * 5);

printf("Start block transfer read\n");

blk_info.opcode = OP_BLKSS;
blk_info.F = 0;
blk_info.N = 6;
blk_info.A = 0;
blk_info.totsize = total_data_size;
blk_info.blksize = block_data_size;
blk_info.timeout = 0;

resp = BLKTRANSF(crate_id, &blk_info, blk_transf_buf);
if (resp != CRATE_OK) {
    printf("ERROR: Negative response from socket server\n");
    return 0;
}

printf("Total data read: %d\n", blk_info.totsize);
for (i = 0; i < blk_info.totsize; i++) {
    // Show received buffer
    if ((i > 0) && ((i % 10) == 0)) {
        printf("\n");
    }
    printf("%06X ", blk_transf_buf[i]);
}
printf("\nEnd of block transfer read\n");

return 0;
}

```