Documentation
for :

---

qCNC control unit

---

by
Trilog Studios (Ron Ablinger)
mailto://trilog_studios@nerdshack.com

Programming languages:
C/C++
Qt4

Target Platform :
Linux/Unix

Requirements :
qextserialport
Linux development headers

Developer Documentation

Table of Contents

1 Introduction

This document houses the developers documentation for *qCNC control unit*, from which the program code can be understood and expanded. I will only cover the main features of the program as most of the code is self explanatory.

QCNC control unit was written for the sole purpose of controlling a *ethernut* development board, which is running a *ucos* operating system. Its main communication is via Ethernet connecting through the Linux supplied *socket.h* library. On the *ethernut* board, the TCP/IP stack was supplied by M. Zauner. For more information on the exact functionalities of the *ucos* stack, please consult either the *ucos* Documentation, or that of the respective Ethernet Stack.

I will cover :

1. G-code phrasing
2. Ethernet
    1. PC/Client tasks
    2. Server tasks
    3. Communication structure
    4. protocol opCode
3. Serial debugging
4. Interface/GUI

2 Concept

It was given, that we had a *ethernut* development board, which needed to be connected via Ethernet to a PC control program. The *ethernut* boards would have a *ucos* OS running on it, which in turn then controlled a CNC machine.

The Task of the PC program, was to initiate a TCP/IP connection to the server of the *ucos* system, then communicate with it, sending coordinates read from a g-code file.

Concentrating on the GUI interface, it can connect to a TCP/IP server and a rs232 interface, read g-code files, periodically polls socket connection, if connected and draw a preview of the resulting job.

It was written with Qt4 and C/C++. It is unthreaded, since previous attempts caused de-synchronisations.

3 Ethernet (TCP/IP)

3.1 Communication structure

Here we had to fathom out what the CNC needs, and what capabilities the *ethernut* board has.

We decided on a *struct type* so that one set of coordinates can be sent at once, also having the ability, to send system commands and instructions, without having to change the data-type. All used structures and data protocols can be found within *Interfaces.h*.

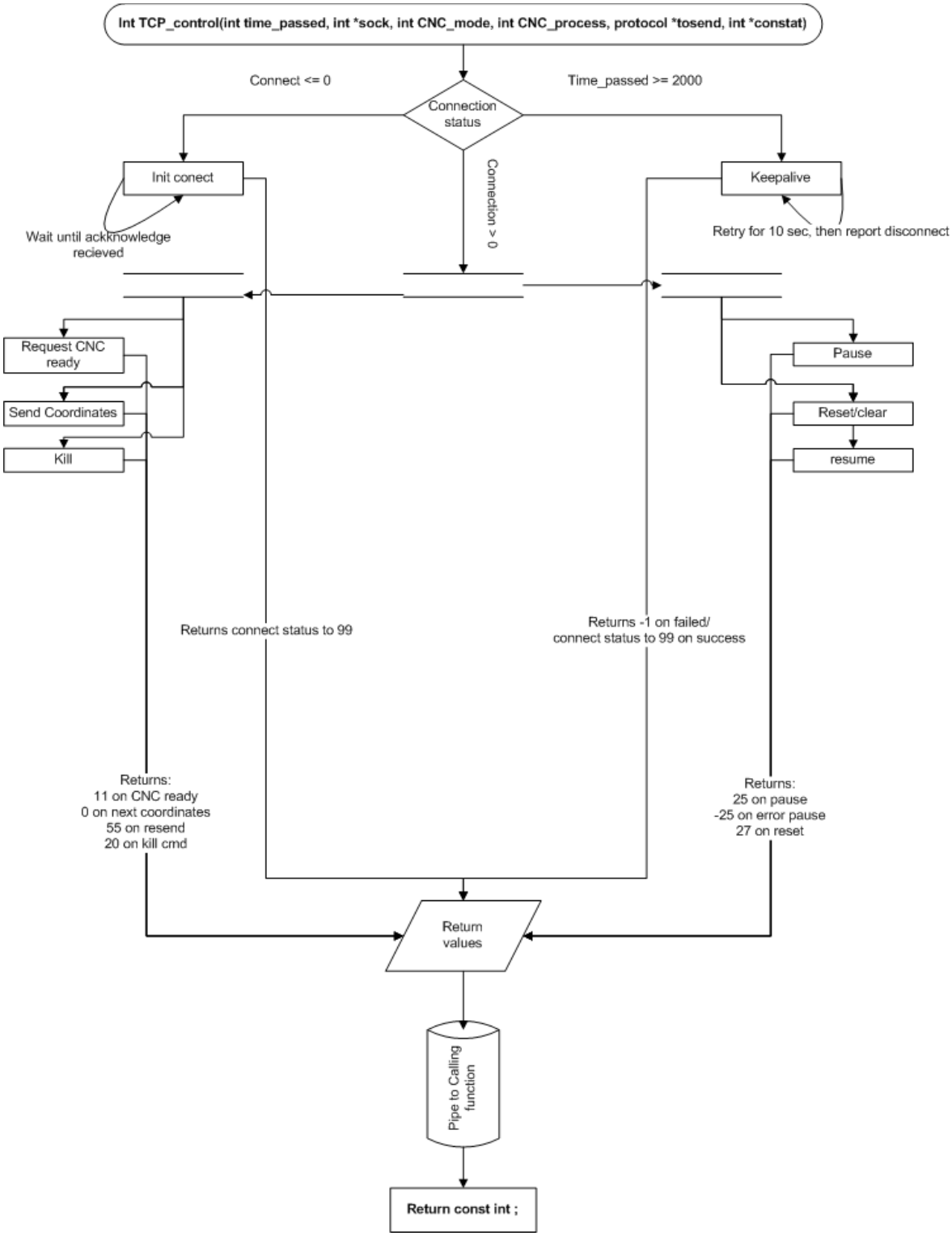
For the TCP/IP connection, following structure was used :

```
typedef struct {
        float x;
        float y;
        float z;
        float status;
        float error;
}protocol;
```

Code snippet 1: TCP/IP communication structure

This code snippet, shows a *struct*, that houses float variables. Data-type *float* was chosen because of the limited capability of the ATMEGA128 MCU, that powers the *ethernut* board.

3.2 PC/client Task



Function flow 1: int TCP_control() flow

Function flow 1 show the workings of the function *TCP_control()* which is called periodically if connected to socket.

For one, it try 's to keep the connection in a *ISALIVE* state if a specific amount of time has passed. If a connection is lost, it is reported to the calling function, which then evaluates the next step.

Also if a first connect is made, a reply from the server is waited on. If the reply is received, the system is set to *READYTOSEND*. If *READYTOSEND* is achieved, it can send different opcode's, CNC modes and the coordinates.

3.2.1 Pause job

The ability to pause the process is not of major importance, but if supported, the process can be paused immediately within any state. If a *resume* is called, the program falls back into its previous state.

3.2.2 Kill job

The *kill* opcode will stop all activity on the *ethernet* board and disconnects active socket connection. If called, it will take up to one timer-periode to achieve the the state.

3.2.3 Reset/Clear job

Both opcodes are in principle the same, but *Reset* also terminates all active connections.

3.2.4 Send coordinates

If the client is in a *READYTOSEND* state, coordinates will be sent. Two different states can be returned. *NEXT* or *RESEND* can be returned to the caller function. A *NEXT* will move ahead in the linked list, a *RESEND* will resend previous coordinates.

3.2.5 Init readytosend

This is the first function called. It requests a *READYTOSEND* from the server. If server replies with *READYTOSEND* coordinates will be sent periodically.

More information can be found within the *opCode* section.

3.3 Server task

The Server has to accept socket connections and respond to different opcodes, that are passed to it. It also has to issue specific commands to the entire system and decide on were the received data has to be sent. EG. If data is received with a .status = 1 the data needs to be sent to the motor/MCP task.

For a closer look at the Server task, please refer to the TCP/IP server task Documentation by M. Ziervogel.

3.4 OpCode

The opCode within this system is based on a unique combination of the values that the variables *status* and *x* hold. These opcodes can either be used to issue a command to the *ethernut* board, to acknowledge received and executed command and keep connections alive.

3.4.1 Opcode table

	value of protocol::status	value of protocol::x	Origin	Destination	Function	Implementation status	Response
1	1	N/A	GUI	MCP	a set of coordinates	OK	16
2	2	N/A	GUI	oLED	sync coordinates to oLED	TODO	16
3	3	N/A	GUI	TCP/DEBUG	TCP/Serial specific command	TODO	16
4	4	N/A	*ethernut*	GUI	sync coordinates from oLED	TODO	16
5	15	15	GUI	TCP	file transaction mode	OK	16
6	15	6	GUI	TCP	coordinates are taken from control within gui	PARTIALLY	16
7	15	9	GUI	TCP	standalone mode disconnect TCP clients data from oLED	OK	16
8	99	0	GUI	TCP	reset, clear memory set CNC to 0/0/0	OK	16
9	99	99	GUI	TCP	issue kill command	OK	16
10	99	199	GUI	TCP	pause request must return (11)	OK	11
11	99	201	GUI	TCP	pause achieved, must be responded to (10)	OK	N/A
12	200	30	TCP	GUI	coordinates OK, send next set	OK	1
13	200	50	GUI	TCP	*READYTOSEND* request	OK	16
14	200	60	TCP	GUI	coordinates error, resend request	OK	1
15	200	78	GUI	TCP	Init request/connection ok?	OK	16
16	200	101	N/A	N/A	global acknowledge	OK	N/A
17	200	199	GUI	TCP	alive request must return (18)	OK	18
18	200	999	TCP	GUI	connection still alive response from (17)	OK	N/A

NONCRITICAL task assignment
NORMAL operation
KILL Commands (needs to activated within preferences)

Table 1: Opcode Table

It is important to be careful with the kill commands. To use them, you have to activate them within the preferences settings. These will be active by default, but once deactivated, a bug occurs, that will disable the whole control.

4 Serial debug

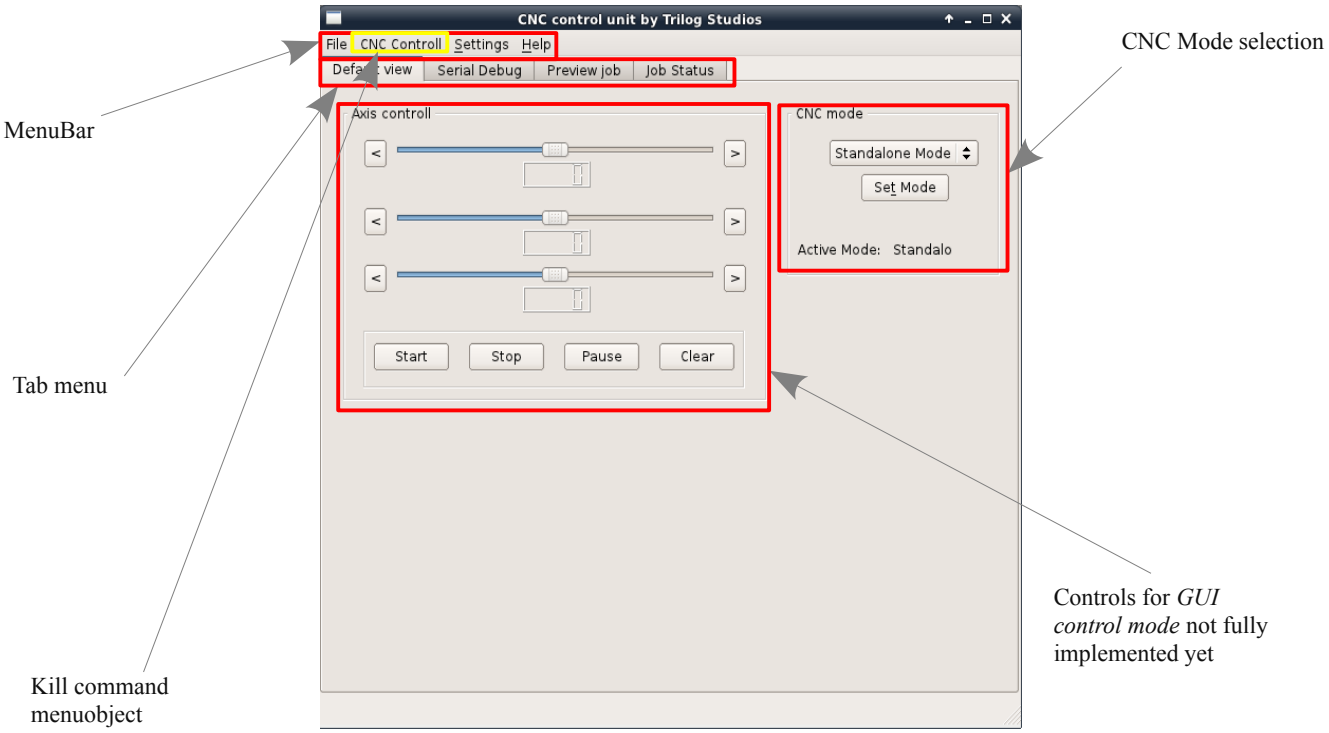
The Serial debug interface was conceptualised to also be able to send commands to the *ethernut* board. But due to lack of time, this was reduced to only receive messages via the rs232 interface.

Current status:

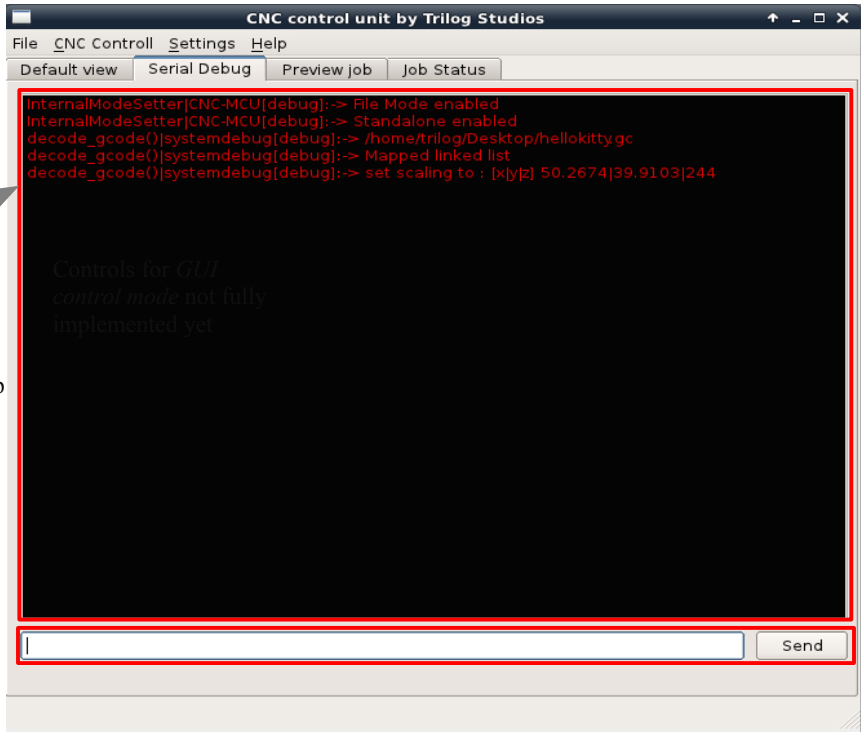
The interface is poled continuously until data is received, which then emits a signal, that data is available. It the is relayed to the debugging window and added to the listbox.

5 Interface/GUI

5.1 Main Window



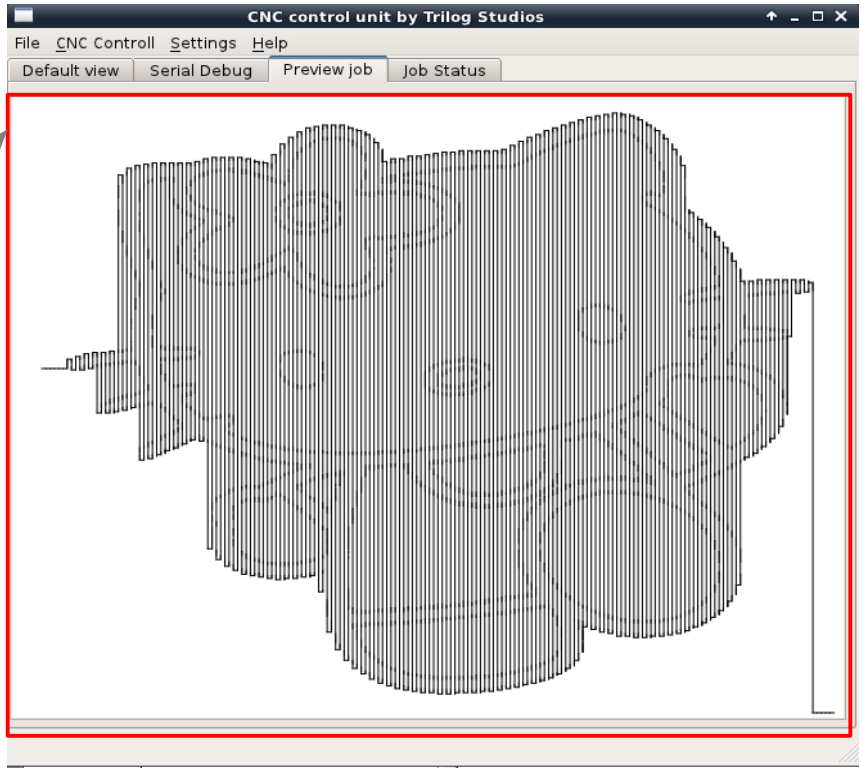
5.2 Serial debug window



Debug tracker message consists of issuing function| destination[messagetype]:->Message

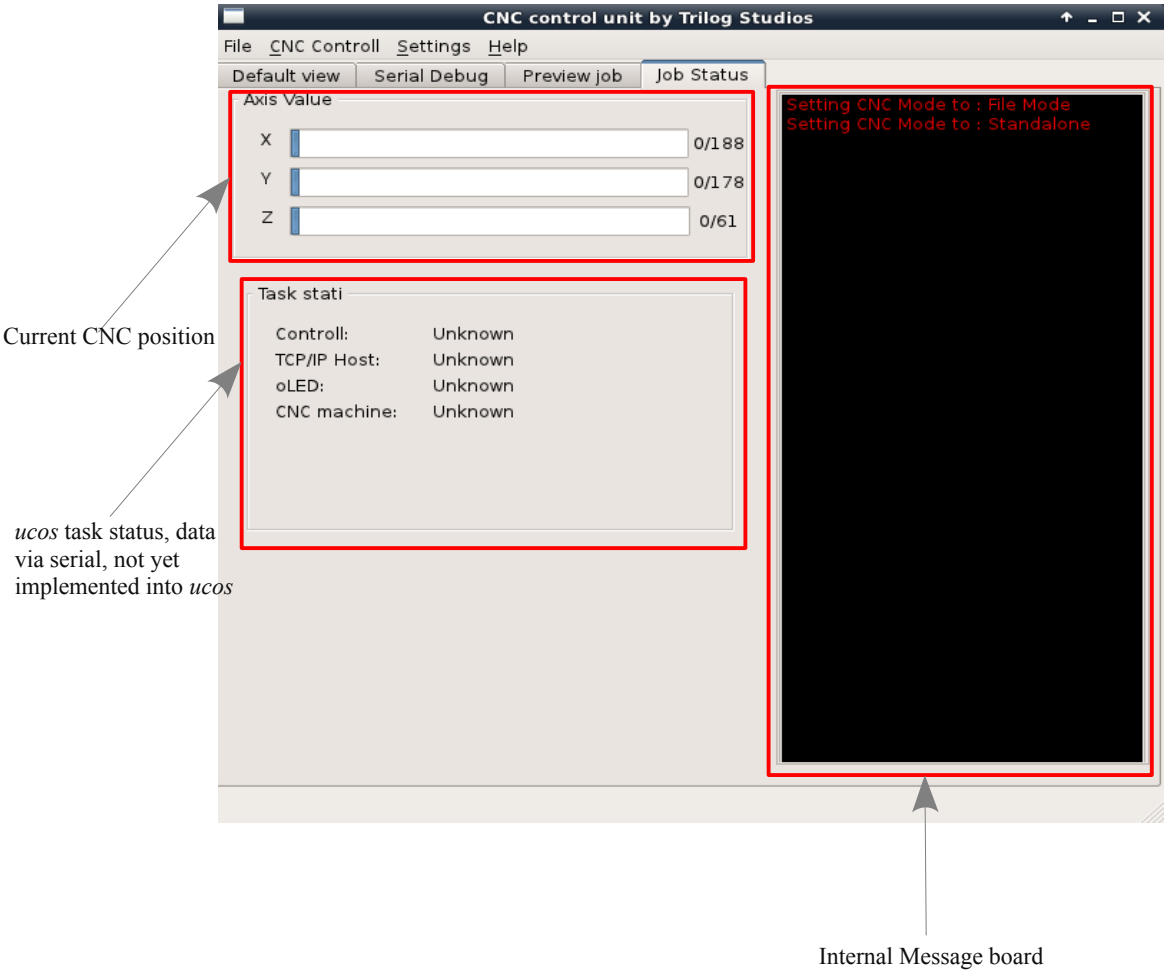
Serial command issuing prompt. Commands not yet implemented

5.3 Job preview



Previews the g-code file. Also shows the path of the CNC machine

5.4 Job Status



6 Gcode phrasing

G-code files, are files that hold axis and instruction for different types of constructions. It could hold information for different points within a 3D design of a house or, as in our case, the information to draw a hello kitty.

6.1 Important instruction sets

M30 – end of data

M03/M04 – spindle on

M05 – spindle off

M01 – pause

G90 – absolute coordinates

G91 – incremental coordinates

G01 – linear cutting

Nx – Line numbering

X – x-coordinates

Y – y-coordinates

Z – z-coordinates

6.2 General information

For the sake of ease, we have found that our conversion program always wrote **G90** and **G01** int the gcode file. Since there is no standard for that file-type, we assumed that our conversion tool generated the gcode's that we then use. In other words, default is **G90** and **G01** and is therefore not extracted by the decoder.

6.3 Phrased gcode information

The coordinates that the gcode supplies, are stored in a linked list consisting of a custom struct.

```
struct gcode_linkedL {
   gcode_linkedL *next;
   double x;
   double y;
   double z;
   int abs; //absolute incremental
};
```

Codesnippet 2: gcode linked list structure

Since the file is read from top to bottom, ie. Begin to end, the coordinates are saved in reverse and will also be drawn from the official end to its beginning, which does not matter, in my humble opinion.

7 General information to code

Within the code directory, numerous files can be found:

<rootofprogramsouce> -/settings/

-/*.tss

-/resources/*.png

-/lib/

-/qextserialport/

-/*c && *h

-/build/*.so.*

-/*.cpp && *.h

Interfaces.h holds all the custom structures for the program.

The rest of the source files, are self-explanatory by name.

To run *qCNC* we have to export the LD_LIBRARY_PATH to point to either your current build, or to the pre-compiled ones found in <*extractedpath*>*/lib/qextserialport/build*.

Run the following in the terminal from where you want to start it:

$ export LD_LIBRARY_PATH=../qCNC/lib/qextserialport/build/

or add the following to ~/.bashrc

$ export LD_LIBRARY_PATH=<fullpathto>/qCNC/lib/qextserialport/build/

8 Settings files

The settings will be read every time the program starts. If any changes have occurred, program needs to be restarted to activate those changes.

'#' - comment.

'TSS' – has to present somewhere within settings file to confirm validity.

'|' - separates the name of the setting <on the left> with its respective value <on the right>

8.1 Program settings

'PROG_enLOG' – 0/1 – sets logging support (not yet implemented)

'PROG_LOGP' – string with all printable characters except for '|' and '#' - shows path to logfile

'PROG _UARTDB' – 0/1 – enables rs232 debugging commands (not yet implemented)

'PROG_CNCkill' – 0/1 – enables CNC kill commands – enabled by default due to bug

8.2 TCP/IP settings

'INET_ADD' – xxx.xxx.xxx.xxx – IP address to which to connect to.

'INET_PORT' – 6666 – integer for port number corresponding to the address specified at 'INET_ADD'

'INET_DEL' – x ms – integer defines default timeout before reporting connection as dead (not yet implemented)

8.3 Serial settings

'SERIAL_DEV' -string with all printable characters except for '|' and '#' - rs232 device mapped on system

9 References

Qextserialport – http://qextserialport.sourceforge.net – Brandon Fosdick & Michal Polich

QT4 – http://qt.nokia.com – Qt developers Trolltech & Nokia

10 Disclaimer

I used a self-build hybrid of Fedora <http://fedoraproject.org/> and Gentoo <http://www.gentoo.org>. I build most of my software from source and use the Gentoo *Portage* package manager which also compiles from Source tarball's. This paper has been written and everything was done on this system, it is possible that the exact command sequences may vary from system to system, but the concepts are the same, and most of the commands are included in every standard Linux distribution. Furthermore, I will not be held responsible for damage to you system. USE at own risk.

11 Interfaces.h

```
#define DEBUG 1
#define STANDALONE 0
#define INTERNAL 0
#define ERROR 0

typedef struct {
        float x;
        float y;
        float z;
        float status; //0-9 used for status-tagging
        float error;  //0-9 used for internal error managment
}protocol;

struct gcode_linkedL {
    gcode_linkedL *next;
    double x;
    double y;
    double z;
    int abs; //absoluto incremential
};

struct Suart_debug {
    int type;
    char sender[10];
    char destination[10];
    char opcode[10];
    int opcodev;
};

struct gui_settings {
    //serial device settings
    QString SerialDevice; //device

    //TCP settings
    QString TCPaddress; //address
    qint32 TCPport; //port
    qint16 TCPStdtimeout; //std timeout

    //Program settings
    QString Proglogpath;
    qint16 Progenlog;
    qint16 Progenuartdb;
    qint16 Progenkill;
    qint16 ProgDevCNCmode;
};
struct tcpset {
    //TCP settings
    QString TCPaddress; //address
    qint32 TCPport; //port
    qint16 TCPStdtimeout; //std timeout
};
struct progset {
    QString Proglogpath;
    qint16 Progenlog;
    qint16 Progenuartdb;
    qint16 Progenkill;
    qint16 ProgDevCNCmode;
};
struct serialset {
    QString SerialDev;
};
```

Codesnippet 3: extract from interfaces.h